Implementación del Mecanismo de Acceso al Medio (MAC) IEEE 802.15.4e CSL (Coordinated Sampled Listening) sobre OpenWSN y Plataforma OpenMote

(Memoria Final)

Master Universitario de Ingeniería de Telecomunicación

(Interuniversitario UOC-URL)

Autor: Sergio Gonzalo San José

Consultor: Pere Tuset Peiró

Fecha de Entrega: **11 de Enero de 2015.**

FICHA DEL TRABAJO FINAL

Título del Trabajo:	Implementación del Mecanismo de Acceso al Medio (MAC) IEEE 802.15.4e CSL (*Coordinated Sampled Listening*) Sobre OpenWSN y Plataforma OpenMote
Nombre del Autor:	Sergio Gonzalo San José
Nombre del Consultor:	Pere Tuset Peiró
Fecha de Entrega (mm/aaaa):	01/2015
Área del Trabajo Final:	Telemática
Titulación:	*Master Universitario de Ingeniería de Telecomunicación (Interuniversitario UOC-URL)*

Resumen del Trabajo (máximo 250 palabras):

El estándar IEEE 802.15.4 es el responsable de la definición del nivel físico y control de acceso al medio de redes inalámbricas de área personal (WPANs), conformando la base para el desarrollo del concepto IoT (*Internet of Things*) donde cualquier objeto podrá ser dotado de la capacidad de comunicarse con su entorno.

Dentro de este marco, el presente trabajo tiene como objetivo el **desarrollo, implementación y validación del mecanismo de acceso al medio CSL (*Coordinated Sampled Listening*) especificado en el estándar 802.15.4 a través de su enmienda 802.15.4e para la capa MAC, empleando para ello la plataforma OpenWSN**.

La plataforma OpenWSN es una de las principales implementaciones de referencia *open-source* del estándar IEEE802.15.4, enraizada en la nueva enmienda IEEE802.15.4e TSCH (*Time Slotted Channel Hopping*) y encargada de proporcionar alta fiabilidad a través del mecanismo de salto en frecuencia (*channel hopping*) así como bajo consumo a través de la sincronización de tiempos.

El desarrollo e incorporación del mecanismo de acceso al medio CSL dota a

OpenWSN de mayor grado de cumplimiento del estándar así como una alternativa de acceso al medio más preparada para entornos móviles que el mecanismo TSCH, más apropiado para redes deterministas. De este modo, la supresión de las planificaciones (*schedule*) así como la incorporación de procedimientos de escucha periódica basados en técnicas *preamble sampling* permite una mayor aplicabilidad de OpenWSN así como una adecuación más eficiente de las motas a redes móviles.

Finalmente, el trabajo comprende la realización y validación del nuevo mecanismo de acceso al medio CSL mediante la generación y despliegue de OpenWSN y CSL sobre motas OpenMote-CC2538, cuyo funcionamiento ha sido validado tanto con OpenWSN como con otras implementaciones del estándar como Contiki o FreeRTOS. Para ello, el trabajo incluye el uso de hardware OpenMote adicional como OpenBattery, para interconexión de sensores y funcionamiento autónomo de las motas, OpenBase para interconexión de redes en modo puerta de enlace, y la depuración de código mediante la herramienta Segger JTAG.

Abstract (in English, 250 words or less):

The IEEE 802.15.4 standard is responsible of defining the physical layer and medium access control for wireless personal area networks (WPANs), forming the basis for IoT new concept (Internet of Things) where any object can be provided with the ability to communicate with their environment.

In this context, this paper considers the **development, testing and validation of medium access radio duty-cycle mechanism CSL (Coordinated Sampled Listening) specified in 802.15.4e MAC amendment, using for this purpose the platform OpenWSN**.

The OpenWSN platform is a major IEEE802.15.4 open-source reference implementation, rooted in the new amendment IEEE802.15.4e TSCH (Time Slotted Channel Hopping) and responsible for providing high reliability through frequency hopping (channel hopping) and low power consumption based on time synchronization.

The development of the CSL mechanism endows OpenWSN with a higher degree of compliance with the standard as an alternative for medium access, more prepared for mobile environments than TSCH mechanism which is most appropriate for deterministic networks. Thus, suppression of schedules as well as incorporation of

periodic listening procedures (based on preamble-sampling techniques) allow a wider applicability of OpenWSN and a more efficient adjustment of motes to mobile networks.

Finally, the work includes the deployment and validation of the new CSL mechanism on OpenMote-CC2538 motes, whose operation has been validated both OpenWSN as other implementations of the standard as Contiki and FreeRTOS. In this sense, the work includes the use of additional hardware like OpenBattery for networking sensors and autonomous operation of nodes, OpenBase for networking in gateway mode, and Segger JTAG debugging tool.

Palabras Clave (entre 4 y 8):

CSL (Coordinated Sampled Listening)

IEEE 802.15.4e

OpenWSN (IEEE802.15.4e open-source reference implementation).

OpenMote-CC2538

Low-Rate Wireless Personal Area Networks (LR-WPAN)

Medium Access Control (MAC) sublayer

Preamble Sampling.

ÍNDICE

LISTA DE TABLAS Y FIGURAS

1. INTRODUCCIÓN

1.1. CONTEXTO Y JUSTIFICACIÓN DEL TRABAJO

Las redes de sensores han surgido en la última década como un nuevo concepto en adquisición e intercambio de datos generando sobre ellas un amplio espectro de aplicaciones y posibilidades de uso en múltiples campos y áreas de desarrollo tales como entornos industriales, domótica, entornos militares, o detección ambiental, entre otros.

Estas redes de sensores inalámbricos (WSN, *Wireless Sensor Network*) constituyen un conjunto de elementos autónomos (motas) interconectados de manera inalámbrica cuyos principales retos de desarrollo se encuentran centrados en la minimización del consumo de potencia (derivado de su alimentación por baterías y por tanto relacionado directamente con la autonomía de los nodos) y la optimización de sus prestaciones expresado principalmente en capacidad de procesamiento y memoria disponible.

El estándar IEEE 802.15.4 fue creado para cubrir la necesidad del mercado de estándares inalámbricos de baja tasa para aplicaciones en redes de sensores, impulsando la creación de distintos grupos de trabajo, (*i.e. IEEE 802.15 WPAN Task Group 4*), con el objetivo de investigar y definir protocolos y mecanismos de comunicaciones con baja tasa de transmisión.

De este modo, el estándar IEEE 802.15.4 es responsable de definir el nivel físico y el control de acceso al medio de redes inalámbricas de área personal con bajas tasas de transmisión de datos (LR-WPAN) siendo la base de desarrollo del concepto IoT (*Internet of Things*) donde cualquier dispositivo podrá disponer de la capacidad de comunicarse con su entorno mediante la transmisión y recepción de datos, posibilitando el desarrollo de servicios adicionales de valor añadido sobre sus funciones básicas.

Dentro de todo este marco, el Trabajo Final de Master (TFM) tiene como objetivo el desarrollo, implementación y validación del mecanismo de acceso al medio CSL (*Coordinated Sampled Listening*) definido y especificado en el estándar 802.15.4, concretamente en la enmienda 802.15.4e para la capa MAC, empleando para ello una de las implementaciones de referencia *open-source* del estándar IEEE802.15.4 más importantes como es OpenWSN.

El proyecto OpenWSN es una implementación de código abierto de una pila de protocolos basada en estándares para redes capilares, enraizada en la nueva norma IEEE802.15.4e TSCH (*Time Slotted Channel Hopping*), la cual proporciona alta fiabilidad mediante el

mecanismo de salto en frecuencia (*channel hopping*) y bajo consumo a través de la sincronización de tiempos.

IEEE802.15.4e, junto con otras normas IoT (*Internet Of Things*) como 6LoWPAN (*IPv6 over Low power Wireless Personal Area Networks*), RPL (*Routing Protocol for Low power and Lossy Networks*) y COAP (*Constrained Application Protocol*) posibilita el establecimiento de redes malladas de bajo consumo, altamente fiables e integradas plenamente en Internet, conformando la piedra angular para la próxima revolución M2M (*Machine-to-Machine*).

En este sentido, el desarrollo e incorporación del mecanismo de acceso al medio CSL dotará a OpenWSN de un mayor grado de cumplimiento del estándar así como incorporará una alternativa de acceso al medio más preparada para entornos de movilidad de motas que el actual mecanismo de acceso al medio basado en TSCH, más adecuado para redes deterministas. De este modo, la supresión de las planificaciones (*schedule*) así como la incorporación de procedimientos de escucha periódica basados en técnicas *preamble sampling* permitirá que la aplicabilidad de la implementación sea mayor y la adecuación a redes móviles de motas sea más eficiente.

1.2. MOTIVACIÓN

Las principales motivaciones que han llevado al desarrollo del presente Trabajo Final de Master incluyen el cumplimiento de los requisitos de Investigación e Innovación que conllevan los trabajos final de master de esta naturaleza así como la contribución al desarrollo del software libre y a la colaboración en la implementación de los estándares del protocolo IEEE802.15.4 en una arquitectura de referencia como es OpenWSN.

La **investigación**, definida como la adquisición de nuevos conocimiento que puedan resultar de utilidad para la creación de nuevos productos, procesos o servicios, o bien contribuir a mejorar considerablemente los ya existentes, se encuentra reflejada dentro del proyecto en el conocimiento del estándar IEEE 802.15.4 así como en sus distintas implementaciones *open-source* actualmente existentes (principalmente OpenWSN y Contiki). Del mismo modo, la fase de investigación contempla igualmente el conocimiento de los procesos de desarrollo, despliegue y depuración de los desarrollos en un sistema empotrado con hardware real.

La **innovación**, definida como la aplicación de un método de producción o suministro nuevo o significativamente mejorado, incluyendo cambios significativos en cuanto a técnicas, equipos y/o programas informáticos, se encuentra reflejada en el proyecto en el análisis,

diseño, implementación y pruebas del mecanismo de acceso al medio CSL especificado en la norma 802.15.4e y no existente hasta el momento en OpenWSN, suponiendo con ello un cambio y mejora significativa en cuanto a sus posibilidades de aplicabilidad, desarrollo y evolución futura.

Finalmente, la **contribución al desarrollo del software libre** se encuentra reflejada en el proyecto a través de la participación e involucración en el marco de desarrollo de una implementación de referencia del estándar 802.15.4 como es OpenWSN, aumentando la cobertura funcional y posibilitando el desarrollo, aplicabilidad y evolución futura.

1.3. OBJETIVOS DEL TRABAJO

Los principales objetivos perseguidos en el desarrollo del presente Trabajo Final de Master son los siguientes:

- Adquisición de conocimiento de la metodología de desarrollo y pruebas aplicable a sistemas empotrados.
- Adquisición de conocimiento de la arquitectura hardware de un sistema empotrado.
- Contribución en el desarrollo de la estandarización de tecnologías 802.15.4 en el ámbito de proyectos de software libre (OpenWSN).
- Conocimiento del estado del arte en cuanto al desarrollo de estándares LR-WPAN de código abierto.
- Adquisición de conocimiento de la estructura, modo de funcionamiento y operatividad a nivel funcional y nivel radio de OpenWSN así como estado de los desarrollos y futuras evoluciones de la plataforma (*roadmap*).
- Estudio y análisis del la especificación para CSL, así como la determinación de su modo de funcionamiento en transmisión y recepción, y el uso de técnicas *preamble sampling*.
- Implementación del mecanismo de acceso al medio CSL (*Coordinated Sampled Listening*) sobre OpenWSN, incrementando el cumplimiento del estándar por parte de éste y posibilitando nuevas vías de estudio, desarrollo y aplicabilidad.
- Despliegue y pruebas de verificación y validación sobre conjunto de motas reales de tipo OpenMote-CC2538.
- Adquisición y aplicación de conocimientos de depuración de código en sistema empotrado con programador Segger JTAG.

1.4. ENFOQUE Y MÉTODO SEGUIDO

El **enfoque metodológico para la monitorización y gestión de tareas** seguido durante la ejecución del Trabajo Final de Master (TFM) se encuentra articulado en torno a los siguientes conceptos principales:

- **Calidad**. La metodología de trabajo seguida en el desarrollo del TFM está orientada al desarrollo e implementación del mecanismo MAC-CSL sobre OpenWSN así como su posterior validación y verificación respecto al comportamiento esperado con las máximas garantías y calidad.
- **Orientación a Funcionalidad y Servicio**. La metodología de trabajo seguida contempla los procesos y actividades orientados a garantizar el cumplimiento de los requisitos de funcionamiento, operatividad y rendimiento definidos para la implementación MAC-CSL objeto del trabajo.

De forma genérica, el enfoque metodológico aplicado persigue además la consecución de los siguientes objetivos:

- **Gestionar y controlar la ejecución de las tareas**, garantizando el cumplimiento de la planificación, detectando los posibles riesgos que puedan suponer una desviación de la programación prevista, y aplicando las medidas correctivas que se considere oportunas.
- **Gestionar y controlar la calidad del proyecto**, asegurando la correcta aplicación de la metodología definida, verificando y validando el producto final, así como el control documental del proyecto.
- **Diseñar una solución técnica óptima** para el desarrollo e implantación de la solución, garantizando el cumplimiento de los requisitos establecidos durante la fase de análisis del TFM.

Así mismo, la metodología aplicada contempla la cobertura de las siguientes actividades principales:

- Seguimiento y supervisión del trabajo.
- Identificación de requisitos funcionales y no-funcionales relacionados con el desarrollo de la solución (sistema, seguridad, y rendimiento).
- Fundamentación teórica y conceptual del entorno así como análisis de la problemática.

- Definición, especificación y diseño de la solución.
- Codificación y pruebas.
- Verificación y validación funcional.
- Análisis de resultados y conclusiones finales.

Para más detalles respecto a la organización de los trabajos, en el **Anexo A – Estructura de Paquetes de Trabajo** se encuentra recogida la estructura de paquetes de trabajo identificadas para la elaboración del presente trabajo final de master.

Respecto a la metodología de desarrollo propuesta para la implementación de los trabajos, consultar el **apartado 6.3 - Metodología de Desarrollo**.

1.5. PLANIFICACIÓN DEL TRABAJO

En la siguiente tabla se presenta a continuación un **desglose de los principales hitos parciales del trabajo final de master así como su correspondencia con las entregas** (PEC) definidas durante su desarrollo.

Nombre (Fecha Entrega)	Hitos Parciales
PAC1 (08/10/2014)	WP1 - Definición del Marco General del TFM
PAC2 (19/11/2014)	WP2 - Fundamentación Teórica y Conceptual WP3 - Diseño de la Solución
PAC3 (23/12/2014)	WP4 - Implementación de la Solución WP6 - Análisis de Resultados y Conclusiones WP5 - Validación y Pruebas
Entrega Final - Memoria (11/01/2015)	WP7 - Elaboración Documento Final del TFM (Memoria)
Entrega Final – Defensa (18/01/2015)	WP8 - Elaboración Presentación Final del TFM (Defensa)

Tabla 1: Desglose de Hitos Principales del Proyecto y Correspondencia con Entregas (PEC)

Para más información relativa a la planificación del proyecto, en el **Anexo B – Planificación Detallada** se proporciona el diagrama de Gantt completo en el cual se encuentra recogida la

planificación temporal de realización de los distintos paquetes de trabajo así como los hitos establecidos en el desarrollo del trabajo.

Respecto a los **recursos necesarios para el desarrollo del proyecto**, a continuación en la siguiente tabla se recogen las características mínimas necesarias para el desarrollo de los trabajos objeto del presente trabajo de acuerdo a los distintos entornos.

Entorno	Recursos y Requerimientos
Entorno de Desarrollo	Ordenador Personal (PC o Portátil) Requisitos mínimos: • CPU dual-core • 4 GB RAM • 1 GB espacio en disco • Conectividad USB. Eclipse IDE CDT (C/C++ Developers)
Entorno de Despliegue, Pruebas y Depuración	2 dispositivos (motes) de tipo OpenMote CC2538 1 dispositivo OpenMote OpenBase 1 dispositivo OpenMote OpenBattery 1 programador Segger JTAG EDU

Tabla 2: Recursos Necesarios para el Desarrollo del Proyecto

1.6. BREVE SUMARIO DE PRODUCTOS OBTENIDOS

Como resultado de la realización del presente Trabajo Final de Máster (TFM), se han generado los siguientes productos:

- Codificación en lenguaje C del mecanismo de acceso al medio (MAC) CSL para la gestión del ciclo de radio, definido en el estándar 802.15.4 (enmienda802.15.4e), sobre la implementación de referencia OpenWSN.
- Diseño del diagrama de estados y plan de trabajo de CSL de acuerdo a la estructura actual de componentes de OpenWSN y mecanismo de implantación de CSL sobre los desarrollos actuales de OpenWSN.
- Roadmap de evolución de los trabajos y líneas de evolución futura para la ampliación funcional y optimización de los desarrollos realizados y las capacidades de CSL.

1.7. BREVE DESCRIPCIÓN DE LOS OTROS CAPÍTULOS DE LA MEMORIA

El presente documento se encuentra estructurado de acuerdo a los siguientes capítulos:

- **Capítulo 1.- Introducción**, en el cual se definen los objetivos del trabajo así como el contenido del mismo, y la metodología y planificación seguidas en su elaboración.
- **Capítulo 2.- Estándar IEEE 802.15.4**, en el cual se procederá a realizar una descripción general del estándar 802.15.4 y más concretamente de la enmienda 802.15.4e para el nivel MAC en la cual se encuentra definido el modo de trabajo CSL.
- **Capítulo 3.- CSL (Coordinated Sampled Listening)**, en el cual se detallan las características, operación y modos de comportamiento definidos en el estándar para el mecanismo CSL.
- **Capítulo 4.- OpenWSN**, en el cual se realizará una introducción descriptiva del proyecto OpenWSN y su implementación de referencia del estándar 802.15.4 y 802.15.4e.
- **Capítulo 5.- Plataforma OpenMote CC2538**, en el cual se describirá la estructura y componentes de las plataformas (motas) reales utilizadas para el despliegue y pruebas de la implementación de CSL realizada.
- **Capítulo 6.- Implementación MAC-CSL (Coordinated Sampled Listening)**, en el cual se recogerán los aspectos relativos al análisis, diseño y desarrollo realizado en el ámbito del trabajo para la implementación del mecanismo CSL en OpenWSN.
- **Capítulo 7.- Despliegue y Pruebas**, en el cual se detallará el proceso de despliegue de los desarrollos realizados para la implementación CSL así como los planes de prueba establecidos para su verificación y validación funcional.
- **Capítulo 8.- Resultados y Valoraciones**, en el cual se recoge el análisis de los resultados obtenidos en la ejecución de los planes de prueba definidos así como una valoración global de éstos respecto al comportamiento deseado y los objetivos establecidos.
- **Capítulo 9.- Conclusiones**, en el cual se recogen las principales lecciones aprendidas en la elaboración del trabajo así como una reflexión crítica sobre el grado de consecución de los objetivos y un análisis crítico del seguimiento llevado a cabo en la planificación y metodología en el desarrollo del trabajo.

- **Capítulo 10.- Líneas Futuras**, en el cual se detallan las líneas de trabajo futuro que nos se han podido explorar en este trabajo y que han quedado pendientes.
- **Capítulo 11.- Glosario**, en el cual se establecen las definiciones de los términos y acrónimos más relevantes utilizados dentro de la memoria.
- **Capítulo 12.- Bibliografía**, en el cual se recogen de manera numerada, todas las referencias bibliográficas utilizadas dentro de la memoria.
- **Capítulo Anexos**, en los cuales se proporciona información adicional y de referencia sobre el resto de contenido del trabajo.

2. ESTÁNDAR IEEE 802.15.4

2.1. INTRODUCCIÓN

El estándar IEEE 802.15.4 define el nivel físico y el control de acceso al medio de una interfaz de bajo consumo. Desarrollado por el IEEE, está principalmente diseñado para redes LR-WPAN (*Low-Rate Wireless Personal Area Network*) debido a que el volumen de datos por unidad de tiempo que suelen generar los dispositivos de una red de sensores es muy bajo comparado con otras redes, como por ejemplo redes con tecnología IEEE 802.11. Lo habitual es encontrarse con transmisiones poco frecuentes, y de baja cantidad de datos.

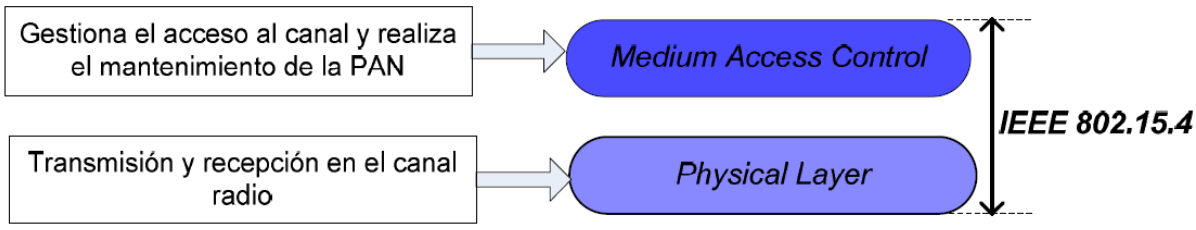


Ilustración 1: IEEE 802.15.4 Protocol Stack

El principal objetivo de este estándar es lograr un bajo consumo ya que algunos de los dispositivos que usan esta tecnología se alimentan con una batería o mediante otras fuentes de energía limitadas, por lo tanto es necesario garantizar un bajo consumo para permitir un tiempo de vida largo del nodo.

2.2. TIPOS DE NODOS

El estándar IEEE 802.15.4 define dos tipos de nodos, *Full Function Device* (FFD) y *Reduced Function Device* (RFD).

Los FFD son dispositivos con la capacidad de desempeñar todos los roles que define el estándar. Se trata de tres roles distintos: dispositivo final (*device*), coordinadores de otros nodos de menor capacidad (*coordinator*) y coordinadores de toda una red PAN (*PAN coordinator*).

Los RFD son dispositivos de menor capacidad a nivel de hardware y/o software que los FFD. En una red sólo pueden actuar como dispositivos debido a sus limitadas prestaciones. En la práctica, en muchos despliegues no existe una distinción entre nodos FFD y RFD. Sin

embargo, se suele diseñar la red de sensores de forma que algunos nodos se comportan de forma similar a un RFD, mientras que otros realizan tareas que pueden atribuirse a un FFD.

2.3. TOPOLOGÍAS

En función de los requisitos de una aplicación, se definen dos topologías de red: topología en estrella o topología peer-to-peer.

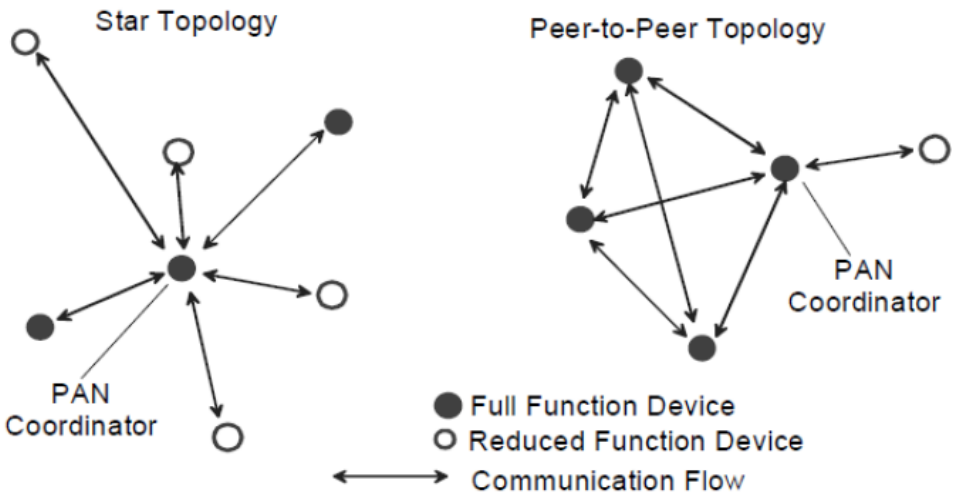


Ilustración 2: Topologías de Red IEEE 802.15.4

La **topología en estrella** se basa en que un dispositivo FFD actúa como coordinador de todos los dispositivos de la red, es decir, actúa como *PAN Coordinator* (PANC). La comunicación siempre es directa desde el dispositivo hasta el PANC o viceversa. Normalmente el PANC está alimentado mediante la red eléctrica ya que siempre está operativo, mientras que el resto de dispositivos están alimentados por baterías.

La **topología peer-to-peer** permite formar redes más complejas, como por ejemplo, redes con topología *mesh*. En este escenario también hay un PANC pero ahora los dispositivos pueden estar conectados a otros dispositivos, no necesariamente al PANC tal y como ocurre en la topología en estrella. Para formar este tipo de red se necesita un protocolo de encaminamiento dado que las comunicaciones pueden requerir realizar más de un salto (comunicaciones *multihop*) entre origen y destino.

2.4. ARQUITECTURA DE CAPAS

La arquitectura 802.15.4 se encuentra definida en términos de un número de bloques con el fin de simplificar el estándar. Estos bloques son llamados capas de tal manera que cada capa es responsable de la implementación de una parte de los servicios del estándar, ofreciendo además servicios a las capas superiores.

Un dispositivo LR-WPAN comprende al menos un nivel físico (PHY) el cual es encargado de contener el transceptor de radiofrecuencia (RF) junto con el mecanismo de control de bajo nivel, y una subcapa de control de acceso al medio (MAC) responsable de proporcionar acceso al canal físico para todos los tipos de transferencia.

A continuación, en la siguiente ilustración se muestra gráficamente los bloques indicados anteriormente.

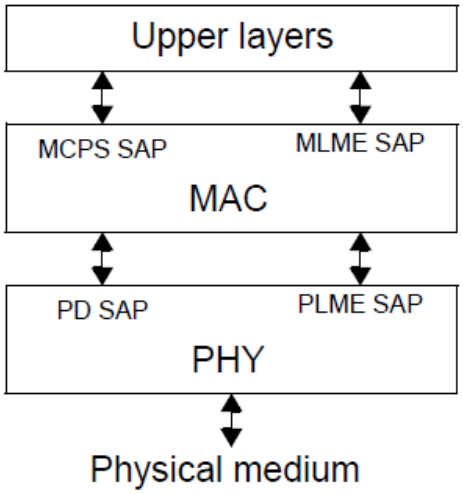


Ilustración 3: Arquitectura de Capas IEEE 802.15.4

En este caso, las capas superiores (*upper layers*) mostradas en la ilustración anterior están formadas por una capa de red encargada de proporcionar configuración de red, manipulación, y enrutamiento de mensajes, así como una capa de aplicación responsable de ofrecer el cometido previsto para el dispositivo. En cualquier caso, la definición de estas capas superiores se encuentra fuera del alcance de la norma IEEE 802.15.4, siendo objeto de desarrollo de otras normas y estándares adicionales.

2.4.1. CAPA FÍSICA

La capa física es la responsable de difundir la información por el medio de transmisión, especificando las propiedades físicas y eléctricas de los componentes del hardware.

De este modo, la capa física (PHY) es la responsable de las siguientes tareas:

- Activación y desactivación del transceptor radio.
- Detección de energía (*Energy Detection, ED*) en el canal actual.
- Indicador de calidad del enlace (*Link Quality Indicator, LQI*) para los paquetes recibidos.
- Evaluación de canal libre (*Clear Channel Assessment, CCA*) mediante detección de portadora de acceso múltiple con prevención de colisiones (*Carrier Sense Multiple Access with Collision Avoidance, CSMA-CA*).
- Selección del canal en frecuencia.
- Transmisión y recepción de datos.
- Variación de precisión en enlaces UWB (*ultra-wide band, UWB*).

La capa PHY proporciona la interfaz entre la subcapa MAC y el canal físico de radio a través del hardware y el firmware de radio-frecuencia (RF). La capa PHY incluye conceptualmente una entidad de gestión denominada PLME. Esta entidad proporciona las interfaces del servicio de gestión de la capa a través de la invocación de funciones propias de gestión de la capa física. La entidad PLME es además responsable del mantenimiento de la base de datos de objetos gestionados y asociados a la capa física, denominada PIB (*PHY PAN Information Base*).

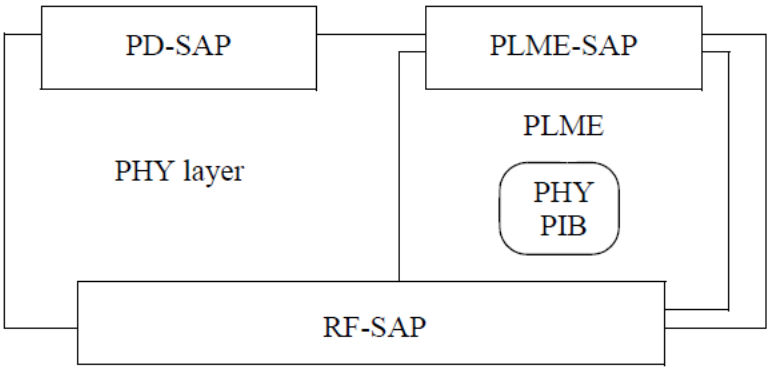


Ilustración 4: Modelo de Referencia de la Capa Física (PHY)

La capa física (PHY) proporciona dos servicios, accesibles a través de dos puntos de acceso al servicio (*Service Access Points, SAP)*:

- Servicio de datos (*PHY Data Service, PD*), accesible a través del punto de acceso al servicio de datos (PD-SAP).

- Servicio de gestión (*PHY Management Service, PLME*), accesible a través del punto de acceso al servicio de gestión (PLME-SAP).

Tanto el punto de acceso al servicio PD-SAP como PLME-SAP no se encuentran, sin embargo definidos en el estándar por lo que es posible que no estén expuestos en una implementación típica.

Finalmente, los atributos de la base de información de la capa física (PHY PIB) son accesibles a través del punto de acceso al servicio MLME SAP usando las primitivas MLME-GET y MLME-SET.

2.4.2. SUBCAPA MAC

La subcapa MAC es la encargada de la gestión de los accesos a los canales físicos de radio, siendo responsable de las siguientes tareas:

- Generación y gestión de tramas *beacon (*baliza*)* en el caso que el dispositivo sea un coordinador.
- Sincronización de *beacons* de red.
- Soporte de asociaciones y disociaciones PAN.
- Soporte de mecanismos de seguridad a nivel de dispositivo.
- Implementación de mecanismo CSMA-CA como mecanismo de acceso al canal.
- Gestión y mantenimiento del mecanismo GTS (*Guaranteed Time Slot*).
- Provisión de enlace fiable entre entidades MAC de dos nodos.
- Validación de tramas y reconocimiento de tramas entregadas (*Acknowledgement*).

La subcapa MAC proporciona una interfaz entre el nivel superior y la capa física (PHY), incluyendo conceptualmente una entidad de gestión denominada MLME (*MAC Sublayer Management Entity*). Esta entidad proporciona interfaces de servicio a través de las cuales es posible invocar las funcionalidades de gestión. Así mismo, MLME es también responsable del mantenimiento de la base de datos de objetos pertenecientes a la subcapa MAC, referida como MAC PIB (*MAC PAN Information Base*).

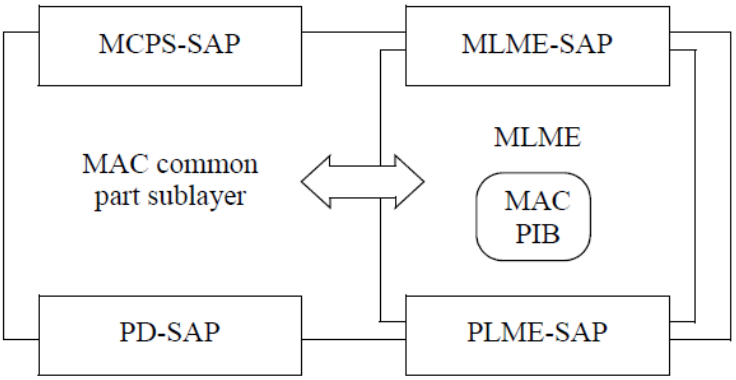


Ilustración 5: Modelo de Referencia de la Subcapa de Acceso al Medio (MAC)

La subcapa MAC proporciona dos servicios, accesibles a través de dos puntos de acceso al servicio (*Service Access Points,* SAP):

- Servicio de datos (*MAC Common Part Sublayer, MCPS*), accesible a través del punto de acceso al servicio de datos (*MAC Common Part Sublayer Service Access Point, MCPS-SAP*). Esta entidad es responsable de la transmisión y recepción de las MPDUs (*MAC Protocol Data Units*) a través del servicio de datos de la capa física (PHY).
- Servicio de gestión (MAC *Management Service, MLME*), accesible a través del punto de acceso al servicio de gestión (MLME-SAP). Esta entidad proporciona los mecanismos para la gestión y mantenimiento de la base de datos de objetos de la subcapa MAC.

Estos dos servicios son los encargados de proporcionar la interfaz entre el nivel superior y la capa física. Adicionalmente a estos interfaces externos, existe un interfaz implícito entre la entidad de gestión MLME y la entidad de datos MCPS con el fin de permitir que la entidad de gestión MLME pueda utilizar el servicio de transferencia de datos (MCPS).

2.4.2.1 Estructura de Supertrama

Un aspecto importante a destacar dentro de la subcapa MAC está relacionado con la sincronización y gestión de tramas. En este sentido, un coordinador de una PAN puede opcionalmente vincular su tiempo de canal utilizando una estructura de supertrama. Una supertrama es delimitada por la transmisión de una trama *beacon* y puede tener una parte activa y una parte inactiva. Esto permite que el coordinador pueda entrar en un modo de bajo consumo durante la parte inactiva.

La estructura de una supertrama incluye intervalos que requieren usar mecanismos de acceso al medio como CSMA/CA ranurado o TDMA mediante el empleo de *Guaranteed Time Slots* (GTS).

En la siguiente ilustración se muestra un ejemplo de estructura de supertrama.

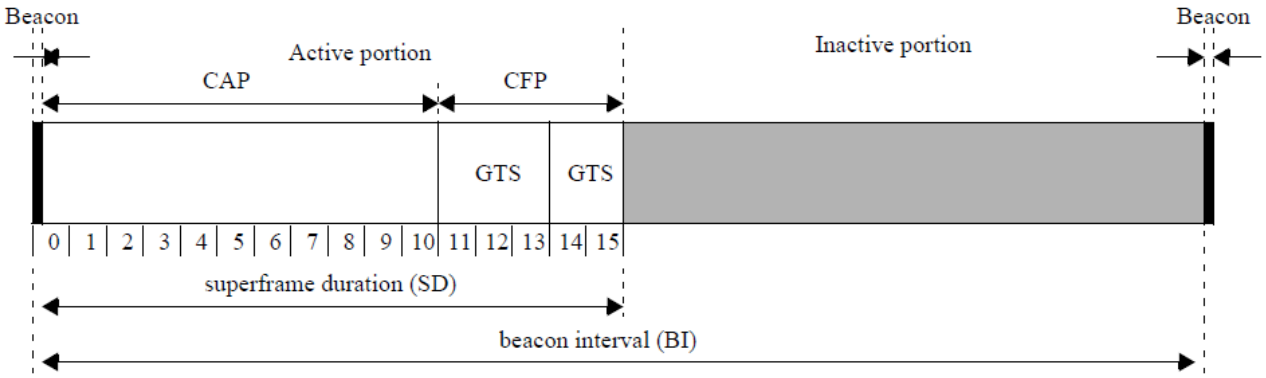


Ilustración 6: Ejemplo de Estructura de Supertrama

La supertrama está delimitada por tramas de tipo *beacon* que son enviadas por el coordinador de la PAN y que contienen información para los nodos que quieran sincronizarse con la red. Básicamente, la supertrama consta de 3 partes:

- *Contention Access Period (CAP)*
- *Contention Free Period (CFP)*
- Periodo inactivo.

El CAP define el intervalo de tiempo durante el cual los nodos competirán para transmitir en slots usando el protocolo CSMA/CA.

El CFP define el periodo durante el cual se ofrecen slots garantizados para nodos que requieran una transmisión con latencia limitada. Es decir, durante este periodo no se utiliza CSMA/CA ya que los slots están pre-asignados.

Durante el periodo inactivo ningún nodo puede transmitir. Todos los nodos entran en un estado de bajo consumo (*sleep*). La supertrama vuelve a repetirse al siguiente envío de un *beacon*, y así sucesivamente.

2.5. ENMIENDA 802.15.4E: SUBCAPA MAC

El grupo de trabajo *IEEE 802.15 WPAN Task Group 4* ha desarrollado una enmienda al estándar para la ampliación funcional y mejora de la especificación MAC de IEEE 802.15.4. Se trata del nuevo estándar IEEE 802.15.4e, publicado en abril del 2012, con el objetivo principal de dar soporte y cobertura a mercados industriales.

A continuación se procederá a describir brevemente los cambios y aspectos más importantes que aporta la enmienda, principalmente con aquello relacionado con los trabajos y tareas desarrolladas en el Trabajo Final de Máster.

A este respecto, se prestará una especial atención a los modos de trabajo de la radio y los mecanismos de acceso al medio (MAC) orientados a la reducción del consumo energético de LR-WPAN (*Low-Rate Wireless Personal Area Networks*) como son CSL (*Coordinated Sampled Listening*) y TSCH (*Time Slotted Channel Hopping*), para el acceso fiable y eficiente de despliegue de motas así como a su capacidad de movilidad en el entorno.

2.5.1. MODOS DE OPERACIÓN

El estándar IEEE 802.15.4e incorpora nuevos modos de operación tales como:

- *Deterministic & Synchronous Multi-channel Extension (DSME)*
- *Low Latency Deterministic Network (LLDN)*
- *Time Slotted Channel Hopping (TSCH)*
- *Radio Frequency Identification Blink (RFID Blink)*
- *Asynchronous Multi-Channel Adaption (AMCA)*

Cada uno de estos modos de operación ofrece un servicio muy específico que beneficia a las distintas aplicaciones de la red.

- El modo DSME se encuentra diseñado para aplicaciones cuyos requisitos son alta disponibilidad, eficiencia, escalabilidad y robustez.
- El modo LLDN está desarrollado para aplicaciones que requieran muy poca latencia tales como robots, grúas, etc.
- El modo TSCH se enfoca principalmente para entornos industriales donde el consumo de energía tiene que ser reducido y la diversidad y robustez frente a interferencias tiene que ser alta.

- El modo *RFID Blink* está diseñado para la comunicación con dispositivos a partir de un identificador que cada uno posee. Se utiliza para aplicaciones de identificación de objetos o personas, localización o seguimiento de los mismos.
- Finalmente, el modo AMCA es utilizado en redes de gran tamaño y dispersión geográfica tales como las redes para *smart utility*, redes de monitorización de infraestructuras y redes de control de proceso.

2.5.2. LOW ENERGY

El estándar IEEE 802.15.4e define el protocolo *Low Energy* (LE) que permite a los dispositivos operar bajo un ciclo de trabajo (*duty cycle*) de la fracción del 1%. Este protocolo incorpora dos mecanismos que garantizan un consumo menor de energía:

- *Coordinated Sampled Listening (CSL)*
- *Receiver Initiated Transmissions (RIT)*

CSL es un mecanismo de ahorro de energía basado en la escucha periódica del canal por parte del receptor en espera de solicitudes de transmisión. De este modo, si durante la escucha recibe una trama *wakeup* con su misma dirección MAC, el mecanismo desactiva el receptor durante un tiempo específico (*rendezvous (RZ) time*) incluido en la trama *wakeup*.

El valor del parámetro *RZ Time* indica por tanto el tiempo restante entre el final de la trama *wakeup* y el principio de la trama de datos, por lo que el receptor sabrá exactamente cuándo despertarse para empezar a recibir los datos del emisor.

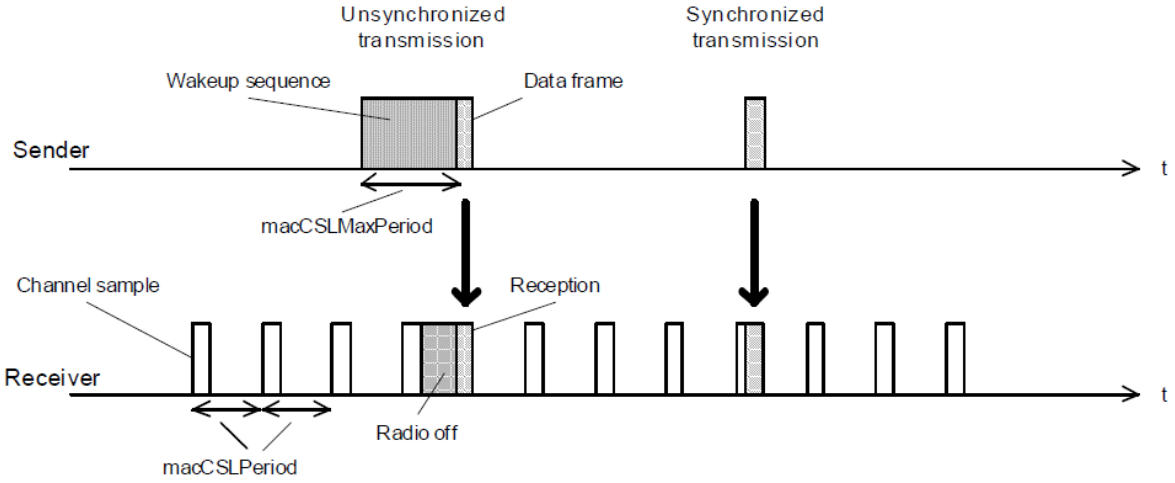


Ilustración 7: Funcionamiento del Mecanismo CSL

El mecanismo RIT, por su parte, es utilizado en redes PAN que no utilizan *beacons*. De este modo, se basa en enviar tramas *datareq* de forma periódica utilizando CSMA/CA no ranurado. Cada vez que envía una de estas tramas, escucha posteriormente el canal durante un tiempo corto para recibir transmisiones. El emisor espera a recibir una trama *datareq* para empezar a transmitir inmediatamente una trama de datos.

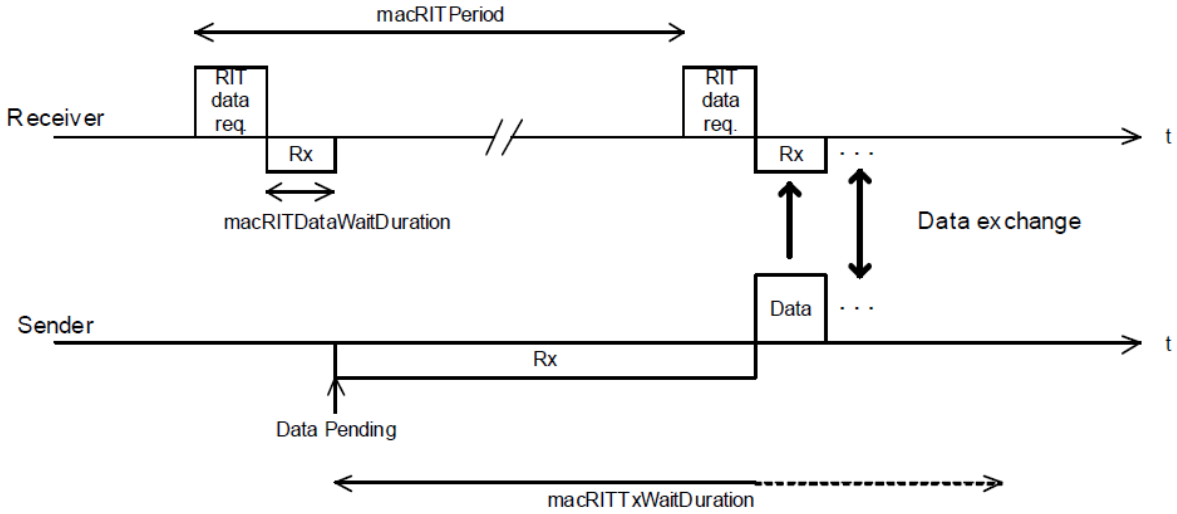


Ilustración 8: Funcionamiento del Mecanismo RIT

2.5.3. FORMATO DE TRAMA MAC IEEE 802.15.4E

La enmienda realiza una serie de ajustes sobre el formato de la trama MAC estándar. A continuación, en la siguiente ilustración se muestra la estructura de la trama definida en la cual se pueden distinguir tres partes principales:

- *MAC Header (MHR)*
- *MAC Payload o MAC Service Data Unit (MSDU)*
- *MAC Footer (MFR)*

<table>
<tr><td>Octets:
1/2</td><td>0/1</td><td>0/2</td><td>0/1/2/8</td><td>0/2</td><td>0/1/2/8</td><td>0/1/5/6/1
0/14</td><td colspan="2">variable</td><td>variable</td><td>2</td></tr>
<tr><td rowspan="2">Frame Control</td><td rowspan="2">Sequence Number</td><td>Destination PAN Identifier</td><td>Destination Address</td><td>Source PAN Identifier</td><td>Source Address</td><td rowspan="2">Auxiliary Security Header</td><td colspan="2">Information Elements</td><td rowspan="2">Frame Payload</td><td rowspan="2">FCS</td></tr>
<tr><td colspan="4">Addressing fields</td><td>Header IEs</td><td>Payload IEs</td></tr>
<tr><td colspan="7">MHR</td><td colspan="2">MAC Payload</td><td>MFR</td></tr>
</table>

Ilustración 9: Formato General de Trama MAC IEEE 802.15.4e

Tal y como podemos observar, la cabecera de la trama MAC (MHR) se compone de varios campos cuyo uso y contenido vendrá determinado por la naturaleza y cometido de cada trama:

- *Frame Control*, encargado de especificar el tipo de trama.
- *Sequence Number*, para especificar un número único a la trama.
- *Adressing Fields*, conjunto de campos que especifican los identificadores y direcciones origen y destino de la PAN.
- Auxiliary Security Header, campo opcional utilizado para la extensión de los mecanismos de seguridad de la información de niveles superiores.
- Cabeceras *Information Elements (IEs)* formado por datos adicionales orientados al funcionamiento y comportamiento específico de determinados elementos y componentes del subnivel.

Por su parte, en la MSDU (*MAC Payload*), se transporta propiamente la carga (*payload*) de los IEs así como la carga (*payload*) de la trama la cual puede corresponder con las PDUs de niveles superiores.

Finalmente, la trama incluye dos octetos para el almacenamiento de la secuencia de control de la trama (*Frame Control Sequence, FCS*) por medio de la cual es posible verificar y validar la integridad de los datos recibidos.

En este sentido, la principal novedad de la trama definida en la enmienda 802.15.4e respecto a la trama definida en el estándar 802.15.4 son los campos variables de IEs, los cuales serán detallados en los siguientes apartados.

2.5.4. INFORMATION ELEMENTS

La enmienda IEEE 802.15.4e define los *Information Elements (IEs)* como un método flexible, extensible y sencillo de implementar para el encapsulamiento de información.

El formato general de un IE consiste en un campo identificador (ID), un campo de longitud (*length*) y un campo de contenido (*content*). Los IEs proporcionan contenedores de información utilizados par encapsular información relacionada con la sincronización, estado de la red, etc. y que permiten añadir, de manera sencilla y flexible, nuevas definiciones de IEs en futuras versiones del estándar con total compatibilidad con las normas y desarrollos existentes.

Tal y como se ha podido ver en el apartado anterior, los IEs forman parte tanto de la cabecera MAC (*MAC Header*) como de la carga MAC (*MAC Payload*). Las cabeceras IEs son utilizadas por la capa MAC para ser procesadas inmediatamente, evitando retardos derivados del propio procesamiento. Las carga IEs (*Payload IEs*) forma siempre parte de la carga MAC (*MAC Payload*) y son destinados directamente al nivel superior o el punto de acceso al servicio.

Cada IE comienza siempre con un descriptor IE cuyo formato es diferente para la cabecera y carga del IE. A continuación en la siguiente ilustración se muestra la estructura de la cabecera de un elemento de información (IE).

Bit: 0-6	7–14	15	Octets: 0 … 127
Length	Element ID	Type = 0	IE Content

Ilustración 10: Formato de Cabecera del Elemento de Información (*IE Header*)

La cabecera se compone de un campo para indicar la longitud del contenido (*length*), un identificador único que identifica unívocamente el IE (*elementID*), y un campo tipo (*type*) que permite identificar el tipo de IE.

Respecto al contenido de un IE, se trata de un campo variable en función del modo de funcionamiento (DSME, TSCH, LLDN, etc.), funcionalidades y modos de comportamiento concretos (CSL, RIT, fiabilidad con ACKs, *beacons*, etc.), definiendo en este caso y para ello IEs específicos.

2.5.5. ENHANCED BEACON Y ENHANCED BEACON REQUESTS

Los *Enhanced Beacon* (EB) son extensiones basadas en las tramas *Beacon* definidas en el estándar IEEE 802.11 con el fin de proporcionar una mayor flexibilidad en el contenido que los *Beacons* del estándar IEEE 802.15.4. Para ello, se hace uso del campo de versión de trama (*Frame Version*) para diferenciar entre un *Beacon* del estándar 802.15.4 y un *Enhanced Beacon* 802.15.4e, presentando en este ultimo caso un valor de campo igual a "0b10".

Los *Enhanced Beacon Request* (EBR) son por su parte una extensión de los comandos MAC *Beacon Request*, diferenciándose de éstos a partir del campo *Frame Version* con valor “0b10”. Estos comandos son empleados para realizar la solicitud de un *Enhanced Beacon,* permitiendo definir y especificar ciertos filtros en la respuesta de tal manera que

sólo se incluya en el *Enhanced Beacon* la información solicitada por el *Enhanced Beacon* Request, reduciendo de este modo el tamaño de la trama enviada.

Finalmente indicar que las extensiones presentadas son usadas en los modos de funcionamiento DSME y TSCH, estando su contenido constituido por un conjunto de IEs.

A continuación en la siguiente ilustración se muestra el formato de trama de un *Enhanced Beacon*, el cual y como se puede observar, su cabecera MHR contiene un campo específico para IEs mientras que su carga o *MAC Payload* contiene un campo de datos para la carga de los IEs (*IEs Payload*).

<table>
<tr><th>Octets: 1/2</th><th>0/1</th><th>variable</th><th>0/1/5/6/10/ 14</th><th colspan="2">variable</th><th>variable</th><th>2</th></tr>
<tr><td rowspan="2">Frame Control</td><td rowspan="2">Sequence Number</td><td rowspan="2">Addressing fields</td><td rowspan="2">Auxiliary Security Header</td><td colspan="2">Information Elements</td><td rowspan="2">Beacon Payload</td><td rowspan="2">FCS</td></tr>
<tr><td>Header IEs</td><td>Payload IEs</td></tr>
<tr><td colspan="4">MHR</td><td colspan="2">MAC payload</td><td>MFR</td></tr>
</table>

Ilustración 11: Formato de Trama Enhanced Beacon (EB)

2.5.6. TSCH

Aunque el modo de funcionamiento TSCH no es objeto de desarrollo del presente Trabajo Final de Máster, se considera necesario realizar una breve descripción general dada la relevancia que presenta en la definición de la enmienda 802.15.4e.

De este modo, en los siguientes apartados se procederá a realizar una introducción a los conceptos y funcionamiento generales de este modo de operación con el fin de proporcionar una visión acerca de su comportamiento y relevancia en la evolución del estándar.

2.5.6.1 Descripción

El modo de funcionamiento *Time Slotted Channel Hopping* (TSCH) utiliza comunicaciones sincronizadas en el tiempo con saltos de canal en frecuencia con el fin de dotar a la red de una mayor robustez frente a fenómenos espectrales tales como interferencias o *multipath fading*.

Este fenómeno puede resultar muy grave en determinados entornos dado que puede llegar a destruir por completo las señales en el receptor por lo que la utilización de este modo de operación dota al sistema de una mayor fiabilidad de comunicación en este tipo de entornos.

Así mismo, TSCH divide el tiempo en slots de uso predefinido, lo cual garantiza transmisiones sin colisiones, permitiendo incrementar el rendimiento global de las comunicaciones.

TSCH puede ser utilizado independientemente de la topología de la red. Funciona tanto en una red en estrella como en una topología peer-to-peer. Este modo de funcionamiento se basa en una estructura de *slotframe* (supertrama). El *slotframe* contiene un conjunto de *timeslots* que se repite en el tiempo. Cada *timeslot* tiene una duración suficiente como para que un emisor envíe una trama de máximo tamaño y que el receptor pueda enviar una trama de reconocimiento (ACK). El número total de *timeslots* que componen el *slotframe* determina la longitud del mismo. Una vez finaliza el *slotframe*, esta estructura y su proceso es repetido y por lo tanto también lo hacen los *timeslots*.

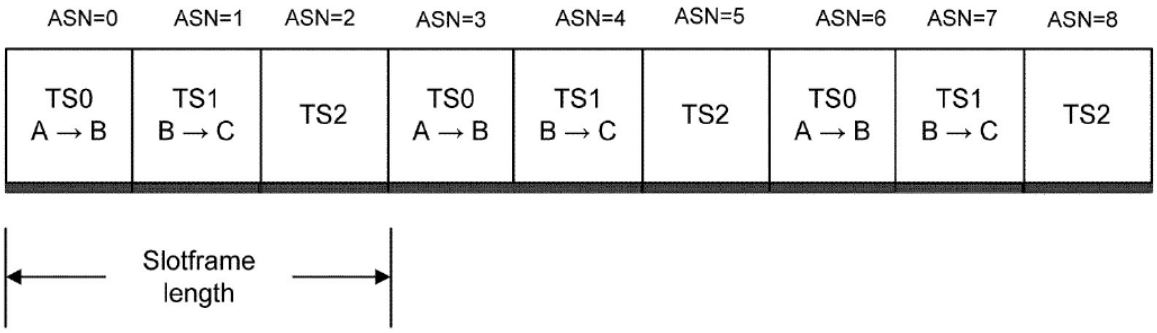


Ilustración 12: Ejemplo de SlotFrame de TSCH que utiliza tres timeslots

Tal y como podemos ver en la ilustración anterior, se muestra un ejemplo de *slotframe* con 3 *timeslots* (TS0, TS1 y TS2). En cada *timeslot* hay asignada una transmisión (A→B, B→C, etc.). La configuración del *slotframe* y *timeslots* es determinada por una capa superior.

Un atributo destacable de los *timeslots* es el *Absolute Slot Number* (ASN) formado por un número incremental con cada *timeslot*. A pesar de que el *slotframe* se repita, el ASN siempre se incrementa de tal manera que este valor determina el número total de *timeslots* que han transcurrido desde que la red se puso en marcha o desde un tiempo arbitrario determinado por el coordinador de la PAN (*PAN Coordinator*, PANC).

Los nodos deben sincronizarse con la estructura de *slotframe* de la red. Si no están sincronizados con el resto de la red, no podrán comunicarse con otros nodos. Cada uno de los nodos sigue una planificación (*schedule)* la cual le indica qué debe hacer en cada *timeslot* (sus posibilidades son transmitir datos, recibir datos o dormir).

En un *timeslot* en el que el nodo deba permanecer dormido (*sleep*), no será necesario encender ninguna interfaz radio lo cual conllevará un gran ahorro de energía dado que el uso de las interfaces radio en modo de transmisión o recepción representa el mayor consumo de energía dentro del nodo. En los *timeslots* activos, el *schedule* indica de qué nodo vecino debe recibir los datos o transmitir, así como en qué canal.

Cuando un protocolo de capa superior genera un paquete, éste es enviado a la capa MAC del dispositivo a través del punto de acceso al servicio (SAP), donde se guarda el paquete en una cola para ser transmitido. En cada *timeslot* de transmisión, la capa MAC comprueba si tiene algún paquete en la cola de transmisión para algún nodo vecino en ese preciso *timeslot*. En tal caso, la capa transmite el paquete. En caso opuesto, el nodo permanece dormido sin encender sus interfaces radio.

En cada slot de recepción, el nodo enciende su radio antes del tiempo esperado de llegada del paquete. Una vez comprobado que el paquete tiene como destinatario el mismo nodo receptor, se envía un ACK al emisor y se apaga la interfaz radio. Por último se envía el paquete a la capa superior correspondiente para que sea procesado.

A continuación, en la siguiente ilustración se muestra de nuevo la estructura del *slotframe* con sus respectivos *timeslots* (color rojo) representando en este caso además los distintos canales en frecuencia en cada *timeslot* (color azul). Por tanto y como se puede observar, el *slotframe* no sólo está dividido en el tiempo sino además se encuentra dividido en frecuencia con lo que las transmisiones deben operar en distintos canales (*channel hopping*).

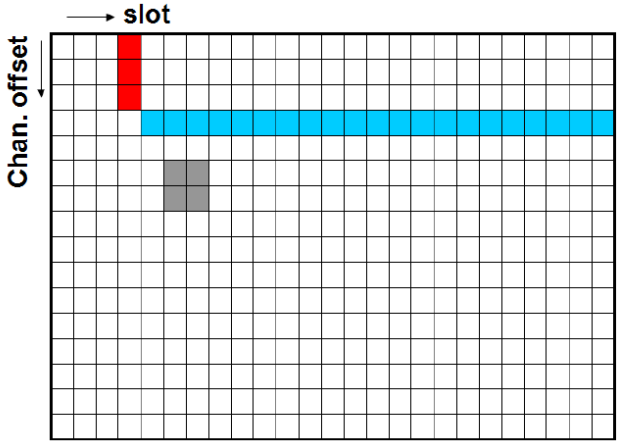


Ilustración 13: Transmisión en Distintos Canales en Estructura de Slotframe (*Channel Hopping*).

Además, es posible que un dispositivo tenga varias interfaces radio y que en un mismo canal pueda transmitir en uno o más canales (frecuencias). Este es el caso de las tres casillas marcadas en rojo en la ilustración anterior las cuales hacen referencia a un mismo slot en el que se producen tres transmisiones en distintos *offsets* de canal.

También es posible que, dentro del *slotframe*, el dispositivo tenga varios *timeslots* (con sus *channel offset* correspondientes) para transmitir. Este caso, pintado en color gris, se conoce como asignación de bloques de *timeslot* y *offset*.

Por último, si el dispositivo sólo dispone de una interfaz radio, sólo podrá realizar una transmisión en el mismo *timeslot*, siendo posible que dentro del *slotframe* se le conceda más de un *timeslot*, como es el caso de las casillas en azul.

Finalmente, la determinación o cálculo del canal por el cual se debe transmitir o recibir es realizado a partir de la siguiente fórmula:

$$canal = ASN + channel\ offset\ \%\ 16$$

Por ejemplo, en la banda de trabajo de 2,4 GHz, el canal es un valor entre 0 y 15 el cual es calculado a partir de la suma del ASN y del *channel offset*. El rango entre 0 y 15 es debido a que los canales en la banda de 2.4GHz son de 2MHz con una separación de 5MHz lo cual supone una cantidad de 16 canales útiles en dicha banda.

2.5.6.2 Organización y Estructura del Timeslot

Tal y como se ha indicado en los apartados anteriores, el *slotframe* es un grupo de *timeslots* donde cada uno de estos *timeslots* presenta una estructura definida.

A continuación en la siguiente ilustración se muestra la estructura del *timeslot* así como los distintos tiempos implicados dentro del modo de operación.

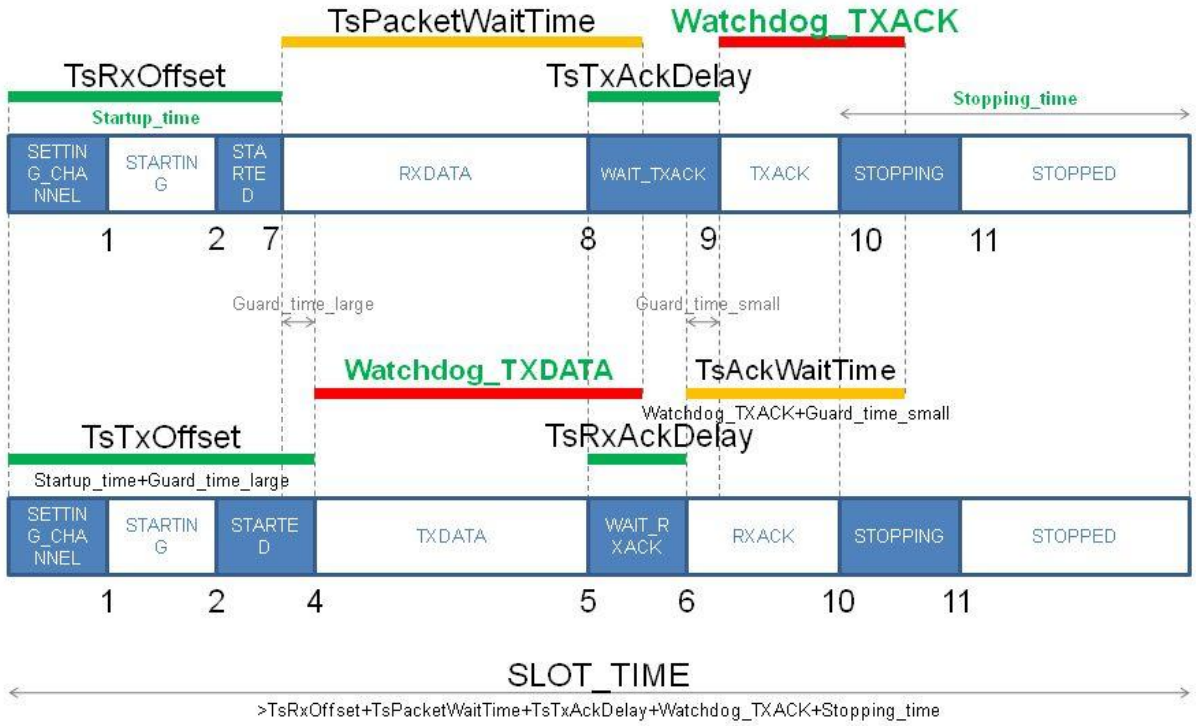


Ilustración 14: Organización y Estructura del *TimeSlot*

Como se puede observar en la ilustración anterior, el *timeslot* comienza en el instante T=0 desde la perspectiva del dispositivo transmisor (parte inferior). El transmisor espera un tiempo **TsTxOffset** (microsegundos) y comienza a transmitir el paquete de información. El transmisor a continuación espera un tiempo **TsRxAckDelay** (microsegundos) para cambiar a modo recepción y esperar el reconocimiento (*acknowledgement*) del receptor remoto.

En el caso que el reconocimiento (*acknowledgement*) no se reciba dentro de un tiempo **TsAckWaitTime** (microsegundos), el dispositivo puede apagar la radio dado que no existirá confirmación de recepción remota.

Desde el comienzo del slot, el receptor (parte superior) espera un tiempo **TsRxOffset** (microsegundos) y activa su interfaz radio. En este estado permanece un tiempo máximo de **TsPacketWaitTime** (microsegundos) o bien hasta la recepción del paquete. Tras realizar la recepción del paquete, espera un tiempo **TsTxAckDelay** (microsegundos) para contestar a continuación con la confirmación o reconocimiento de recepción (*acknowledgement*).

Por tanto, durante el tiempo en un timeslot, se observa que un nodo transmisor envía una trama y el receptor envía un ACK tras su recepción correcta. Así mismo, antes de

una transmisión hay un tiempo de offset donde el nodo se prepara para la transmisión y recepción de los datos. Durante este periodo se configura la interfaz radio para configurar el canal de transmisión y posteriormente se activa esta interfaz. Una vez las radios están listas, el receptor empieza la escucha en el canal antes de que el emisor comience a transmitir los datos.

Una vez se ha acabado la transmisión, el emisor espera un tiempo para recibir el ACK. Durante este tiempo debe cambiar la interfaz a modo de escucha y lo hace antes de que el receptor haya empezado a transmitir sus datos. De igual manera, el receptor espera un tiempo para empezar a transmitir el ACK, en el que también configura la interfaz radio. Para finalizar hay un periodo de parada en el cual ambos nodos apagan las interfaces radio. Estas pasan a modo *sleep* hasta que no tengan otro *timeslot* asignado en el que deban transmitir o recibir datos.

2.5.6.3 Retransmisión

Cuando un dispositivo envía una trama, espera durante un tiempo máximo para la recepción de la confirmación de entrega (*acknowledgement, ACK*). En el caso que no se reciba dicho ACK en un cierto periodo (tiempo *TsAckWaitTime* descrito anteriormente), o bien el ACK recibido no presenta la estructura o contenido correctos, el dispositivo transmisor determinará que la transmisión ha sido fallida.

Cuando esto sucede, el proceso de transmisión de la trama es repetido en el siguiente *timeslot* asignado, volviendo de nuevo a esperar el tiempo de recepción de ACK. En caso erróneo, este proceso será repetido un máximo de N veces, siendo N un parámetro interno dependiente de la implementación del estándar. En el caso que se supere dicho número máximo de retransmisiones, el dispositivo asumirá que la transmisión no se ha podido realizar, notificando en consecuencia a las capas superiores.

2.5.7. SINCRONIZACIÓN

La sincronización entre los nodos de la red es un aspecto muy importante dado que en el caso que un nodo pierda la sincronización y no sea capaz de seguir correctamente la estructura de la supertrama, no podrá comunicarse con ningún otro nodo ni podrá recibir datos.

En este sentido, los dispositivos o nodos constan normalmente de un cristal de cuarzo con el cual es posible disponer de una estimación aproximada como base de tiempo en base a

contadores de las oscilaciones producidas en el cristal. Sin embargo y dado que dicho cristal es un material físico que puede presentar defectos de fábrica, verse afectado por condiciones atmosféricas, o bien otras situaciones, es muy posible que presente derivas. Por este motivo se deben implementar mecanismos para mantener la sincronización.

El estándar define dos métodos para realizar esta sincronización:

1. **Sincronización basada en ACKs**: El receptor calcula el tiempo delta (la diferencia de tiempos) entre el tiempo esperado de llegada de la trama y el tiempo real en la que llega la trama. Este valor calculado lo proporciona el receptor en la trama ACK que envía al emisor. Esto permite al emisor mantener la sincronización con el reloj del receptor.
2. **Sincronización basada en Trama**: El receptor calcula el tiempo delta entre el tiempo esperado de llegada de la trama y el tiempo real en la que recibe la trama. En este caso es el receptor quien a partir de este valor delta ajusta su propio reloj para estar sincronizado con el emisor.

De este modo y empleando estos mecanismos, mientras exista tráfico en la red, los nodos podrán estar sincronizados. En el caso que los nodos no hayan realizado ninguna transmisión o recibido datos en un periodo muy largo de tiempo (normalmente a partir de 30 segundos), enviarán un paquete de datos vacío simplemente para volver a iniciar el proceso de sincronización.

Considerando una topología en árbol, formada por RPL, un nodo sólo se sincronizará con el reloj de un nodo que está por encima de él. Esto permite asegurarse que todos los nodos tienen una noción del tiempo en la cual el nodo raíz es quien acaba dictando el valor del tiempo de la red.

2.6. SOPORTE A PROTOCOLOS PARA REDES DE SENSORES

Hasta ahora se ha hablado de la capa física y MAC, capas en las que se ha utilizado el estándar IEEE 802.15.4 (PHY) y IEEE 802.15.4e (MAC). El objetivo de este apartado es proporcionar brevemente una visión de los principales protocolos de nivel superior utilizados en redes de sensores IP.

2.6.1. VISIÓN GENERAL

La utilización de una pila de protocolos basada en IP posibilita la interconexión de la red de sensores a Internet así como un acceso sencillo a cada uno de los elementos. A

continuación, se muestra la principal pila de protocolos definida para la realización de este cometido:

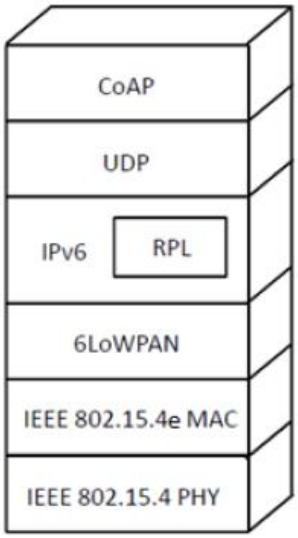


Ilustración 15: Pila de Protocolos definida para Redes IP

Tal y como se puede observar en la ilustración anterior, por encima de la capa MAC definida por el estándar 802.15.4e, se define una capa de adaptación **6LoWPAN**, desarrollada por el 6LoWPAN *Working Group* de la IETF. Esta capa de adaptación entre MAC e IPv6 proporciona una serie de mecanismos para la compresión de cabeceras, fragmentación y encaminamiento a nivel 2 las cuales son funcionalidades imprescindibles dado el alto coste en cabeceras de los paquetes usado en IPv6 que provoca que los datos a nivel de aplicación sean mínimos.

Así mismo, la unidad máxima de transmisión (*Maximum Transfer Unit, MTU)* de IPv6 e IEEE 802.15.4 son diferentes (1280 bytes en IPv6 por un centenar de bytes en 802.15.4) con lo cual se requiere de un mecanismo de adaptación intermedio. 6LoWPAN implementa un algoritmo de fragmentación encargado de la compatibilidad entre ambas capas.

Por encima de 6LoWPAN se encuentra la capa IPv6, con el añadido de RPL (*IPv6 Routing Protocol for Low-Power and Lossy Networks*), un protocolo de encaminamiento diseñado por el *ROLL Working Group* de la IETF para redes *Low-Power* and *Lossy Networks* (LLNs).

Este protocolo define una topología de red lógica similar a una topología en árbol. Cada jerarquía en la topología tiene un valor llamado *rank* donde el nodo raíz presenta el *rank* igual a 1.

Al presentar una topología en árbol, las rutas *upward* (es decir, las que van de un nodo hoja hasta el nodo raíz) son sencillas de determinar dado que el siguiente salto (*next-hop*) de los

nodos será siempre su nodo padre (*default parent)*. Sin embargo, si es el nodo raíz quien quiere transmitir datos a un nodo, es necesario crear una ruta *downward* específica lo cual es posible realizarlo de dos maneras distintas: 1) mediante *storing mode* o 2) *non-storing mode*.

Primero se deben enviar mensajes DAO desde los nodos hoja. En el caso de utilizar configuración *storing mode*, los nodos intermedios deben guardar información de todos los hijos que tienen para poder encaminar los paquetes *downward*. En el caso que se utilice configuración *non-storing mode*, será el nodo raíz quien guarde todas las rutas posibles hasta los nodos hoja, empleando *source routing*.

Estos dos modos de funcionamiento tienen características positivas y negativas, con lo que el uso de una u otra deberá adecuarse a los distintos tipos de aplicaciones y escenarios que se pueden dar en una red de sensores.

Por último, se ha definido un protocolo de aplicación llamado *Constrained Application Protocol (CoAP)*, desarrollado por el *CoRE Working Group* de la IETF, con las mismas bases que HTTP pero adaptado para redes de sensores aunque con una menor sobrecarga (*overhead*).

CoAP define dos subcapas: la capa de petición/respuesta y la capa de mensaje. En la primera se definen los tipos de peticiones que puede solicitar el cliente (GET, POST, PUT, DELETE) y las posibles respuestas del servidor (2.00 OK, 2.01 *Created*, 2.02 *Deleted*, 4.04 *Not Found*, etc.). La capa de mensaje especifica el tipo de mensaje que será enviado. Las diferentes opciones son: CON, NON, ACK, RST. Si el mensaje es CON significa que se debe confirmar mediante un ACK. Si es NON no se requiere una confirmación. Un mensaje ACK es una confirmación de un mensaje CON recibido anteriormente. Por último, el mensaje RST hace referencia a *Reset*, y se usa cuando un nodo no puede procesar adecuadamente los mensajes (p.ej. porque se acaba de reiniciar tras una caída).

2.6.2. INICIATIVA 6TSCH

La iniciativa 6TSCH es una iniciativa que pretende convertirse en *Working Group* del IETF y cuyo objetivo es utilizar IPv6 sobre el modo de funcionamiento TSCH del estándar IEEE 802.15.4e.

IEEE 802.15.4e no define ningún método para construir y mantener el *schedule* que utiliza TSCH. Dado que TSCH requiere mecanismos para gestionar el *schedule*, este grupo de trabajo está definiendo una capa intermedia entre la capa MAC IEEE 802.15.4e con el modo

TSCH y la capa 6LoWPAN. Esta capa recibe el nombre de *6top* y ofrece interfaces de gestión y datos para capas superiores.

2.6.3. OTRAS PILAS DE PROTOCOLOS

Adicionalmente a los protocolos descritos anteriormente, existen en el mercado otras pilas de protocolos ampliamente utilizadas en redes de sensores basadas en IEEE 802.15.4 como capa física y MAC.

En este sentido, cabe destacar la pila de protocolos ZigBee, definida por la ZigBee Alliance, la cual establece una serie de especificaciones propias a partir del nivel MAC, pero que utiliza IEEE 802.15.4 en los niveles físico y acceso al medio.

Otro ejemplo de pila de protocolos es la pila de *Bluetooth Low Energy* (BLE). Esta tecnología, definida por el Bluetooth SIG en la especificación Bluetooth v4.0 puede hallar aplicación en el ámbito de la salud, seguridad, deportes y hogar. Su objetivo es reducir el consumo de energía en los dispositivos, el tamaño y el coste, además de ser compatibles con el máximo número de dispositivos posibles. Un aspecto clave de esta tecnología es el hecho de que los *smartphones* que ya disponen de dispositivos Bluetooth podrán fabricarse con soporte también para BLE con bajo coste adicional, dado que BLE reaprovecha parte de la circuitería de Bluetooth.

3. CSL (COORDINATED SAMPLED LISTENING)

El presente apartado tiene como objetivo describir el modo de funcionamiento y características principales del modo de funcionamiento CSL (*Coordinated Sampled Listening*) descrito en la enmienda 802.15.4e del estándar.

El mecanismo CSL se trata de un modo de funcionamiento de bajo consumo, tal y como se describe en el **apartado 2.5.2 - Low Energy**, definido dentro de la enmienda 802.15.4e para la subcapa MAC dentro del estándar, y orientado a la operación de dispositivos bajo un ciclo de trabajo (*duty cycle*) de la fracción del 1%.

A continuación en los siguientes capítulos se procederá a realizar una descripción funcional y operativa del modo CSL así como de los principales aspectos de consideración necesarios para su implementación.

3.1. DESCRIPCIÓN FUNCIONAL

El modo CSL es un modo de trabajo de bajo consumo (*low energy*) dirigido a la operación bajo ciclos de trabajo (*duty cycle*) muy reducidos y orientados al mínimo consumo, posibilitando una mayor autonomía de éstos.

De este modo, en CSL los dispositivos receptores muestrean periódicamente los canales para la detección de posibles solicitudes de transmisión. Los dispositivos receptor y transmisor están coordinados con el fin de reducir la sobrecarga de transmisión.

A continuación en la siguiente ilustración se recoge el modo de funcionamiento de CSL en transmisión y recepción en cada una de sus modalidades de trabajo:

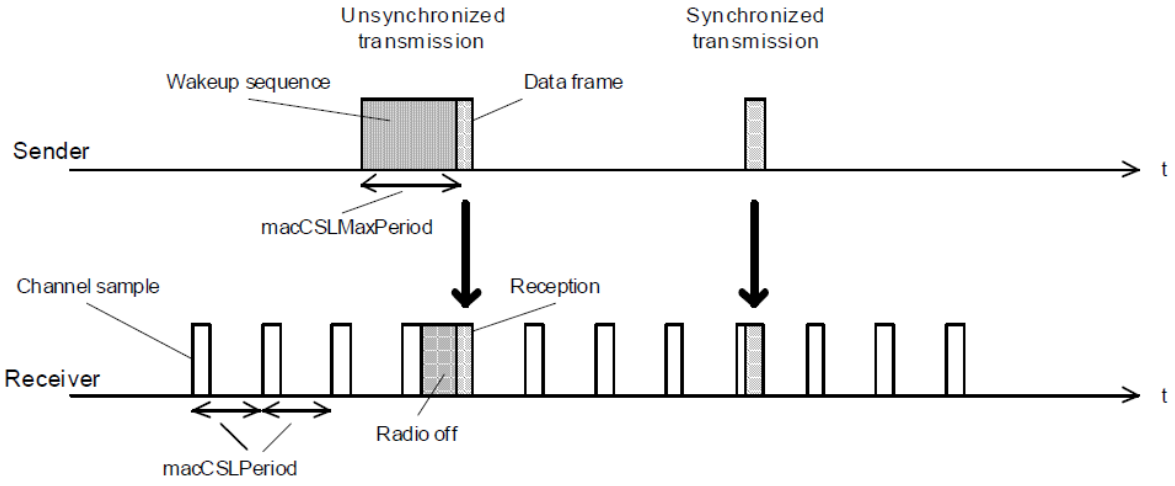


Ilustración 16: Funcionamiento del Mecanismo CSL

Tal y como se puede observar en la ilustración anterior, el periodo de muestreo en recepción para detección de solicitudes de transmisión es realizado a intervalos regulares definidos por el atributo *macCSLPeriod.* Por tanto, la habilitación del modo de funcionamiento CSL es realizada a través del atributo ***macCSLPeriod*** de la PIB (*PAN Information Base*) de tal modo que cuando este parámetro es distinto de cero, el modo CSL se encuentra habilitado mientras que una asignación de valor cero a dicho atributo implicará la desactivación del modo de funcionamiento.

A continuación se procederá a describir los modos de funcionamiento y operación en las fases de escucha periódica del canal así como en recepción y transmisión.

3.1.1. FASE DE ESCUCHA INACTIVA

Durante la fase de escucha inactiva (*idle listening*), CSL lleva a cabo un muestreo del canal con una periodicidad igual a la definida por el parámetro *macCSLPeriod.* En el caso que el muestreo no detecte energía en el canal, CSL deshabilitará el receptor hasta el siguiente muestro del canal donde volverá a comprobar la existencia o no de energía.

Por el contrario, si el muestro del canal detecta energía y se recibe una trama de tipo *wake-up*, CSL comprobará la dirección destino de la trama para comprobar si el receptor es el destinatario. En el caso que coincida con la dirección destino, cuya longitud vendrá definida por *macShortAddress*, CSL deshabilitará de nuevo el receptor durante un tiempo igual al tiempo indicado en la trama *wake-up* en el campo *Rendezvous Time (RZ Time)*, habilitando de nuevo el receptor para recibir la trama de datos o información (*payload frame*) pasado dicho tiempo.

En cualquier otro caso, CSL deshabilitará el receptor un tiempo igual a *RZ Time* más el tiempo de transmisión de una trama *payload* de longitud máxima más el tiempo de transmisión de una trama de reconocimiento (*acknowledgment)*, reiniciando a continuación de nuevo el proceso periódico de muestreo del canal.

3.1.2. FASE DE TRANSMISIÓN

La fase de transmisión de una trama de datos o información (*payload frame*) se caracteriza por ir precedida en todos los casos por una secuencia de tramas de tipo *wake-up* denominada secuencia *wake-up (wake-up sequence)*.

En este punto, es importante distinguir entre transmisiones punto a punto (*unicast*) y transmisiones *broadcast* dado que el comportamiento en ambos casos es ligeramente distinto.

3.1.2.1 Transmisión Unicast

En el caso de transmisiones punto a punto o *unicast*, la longitud de la secuencia de tramas *wake-up* (*wake-up sequence*) puede ser mayor o menor en función de si se trata de una transmisión sincronizada o no-sincronizada.

- **Transmisión No-Sincronizada** (*Unsynchronyzed)*: En este caso, la capa MAC no conoce la fase y periodo CSL del dispositivo destinatario por lo que la longitud de la secuencia *wake-up* (*wake-up sequence*) se corresponderá con el valor del atributo *macCSLMaxPeriod.*
- **Transmisión Sincronizada** (*Synchronyzed)*: En este caso, la capa MAC tiene conocimiento de la fase y periodo del dispositivo destinatario y por tanto la longitud de la secuencia *wake-up* (*wake-up sequence*) será únicamente el tiempo de guarda entre el desplazamiento del reloj (*clock drift*) basado en el último instante en que la fase y periodo CSL fue actualizado desde el dispositivo destinatario.

En ambos casos y en el supuesto que el nivel superior tenga que transmitir varias tramas al mismo destino, se establece la posibilidad de establecer a uno el bit *frame pending* dentro del campo *Frame Control* salvo en la última trama con el fin de maximizar el rendimiento.

La secuencia de acciones o pasos llevada a cabo por la capa MAC durante la transmisión unicast es por tanto la siguiente:

a) Adquisición del acceso al canal utilizando el mecanismo CSMA-CA.

b) Si la última trama de datos reconocida (*acknowledged previous frame*) tiene el bit *frame pending* activado (valor 1) y se encuentra dentro del tiempo indicado por el atributo *macCSLFramePendingWaitT*, entonces saltar al paso e).

c) Si se trata de una transmisión sincronizada, esperar hasta el siguiente muestreo del canal en el dispositivo destino.

d) Durante la duración de la secuencia de tramas *wake-up* (de mayor o menor duración):

1. Construcción de la trama *wake-up* con la dirección corta del destino (*short address*) y el tiempo restante hasta la transmisión de la trama de datos (*payload frame*).
2. Transmisión de la trama *wake-up.*

e) Transmisión de la trama de datos o información (*payload frame*).

f) Esperar durante un tiempo máximo definido por el atributo *maxEnhAckWaitDuration* por la trama de reconocimiento (*enhanced acknowledgement frame*) en el caso que el campo *Acknowledge Request* en la trama de datos (*payload frame)* es establecido a uno.

g) En el caso de recibir la trama de reconocimiento (*enhanced acknowledgement frame*), actualizar la información sobre la fase y periodo CSL del dispositivo destino a partir del campo *Acknowledgement CSL Sync*.

h) En el caso de no recibir la trama de reconocimiento (*enhanced acknowledgement frame*), iniciar el proceso de retransmisión.

3.1.2.2 Transmisión Broadcast

En el caso de transmisiones *broadcast*, el proceso es completamente análogo al empleado en las transmisiones *unicast* con las siguientes salvedades:

- Se trata siempre de transmisiones no sincronizadas (*unsynchronyzed*).
- La dirección destino en las tramas *wake-up* es siempre 0xFFFF.
- Opcionalmente, puede incluir elementos de información (IE) de tipo LE CSL IE (*Low Energy CSL Information Elements*).

Selectivamente, el nivel superior puede añadir campos LE CSL IE en la cabecera de la trama con el fin de propagar información de la fase y periodo CSL entre todos los dispositivos vecinos.

3.1.3. FASE DE RECEPCIÓN

La fase de recepción se encuentra referida al proceso de tratamiento de la trama de datos o información (*payload frame*) recibida en el canal.

En este sentido, la capa MAC lleva a cabo las siguientes tareas:

- Envía inmediatamente una trama de reconocimiento (*enhanced acknowledgement frame*) con la dirección de destinatario igual al dispositivo transmisor así como su propia fase y periodo CSL como elementos de información (campos de tipo LE CSL IE). La trama de reconocimiento puede opcionalmente ser autenticada y/o cifrada en base al modelo y modo de seguridad establecido.

- Si el elemento de información (LE CSL IE) está presente en la trama de datos o información (*payload frame*), la información de fase y periodo CSL del dispositivo transmisor es actualizada con la información contenida en dicho elemento.
- Si el bit *frame pending* del campo *Frame Control* está activado (valor igual a 1) en la trama de datos recibida, el receptor se mantiene encendido un tiempo igual al indicado por el atributo *macCSLFramePendingWaitT* antes de volver de nuevo a la fase de escucha inactiva. En cualquier otro caso, se inicia inmediatamente la fase de escucha inactiva.

3.1.4. CSL SOBRE MÚLTIPLES CANALES

Cuando el atributo *macCSLChannelMask* tiene un valor distinto de cero, las operaciones CSL son extendidas a todos los canales seleccionados en el mapa de bits (*bitmap*). En este caso, la escucha inactiva CSL (*CSL idle listening*) realiza un muestreo en cada canal desde el número inferior al superior en modalidad *round-robin*.

En el caso de transmisión no-sincronizada, la transmisión CSL envía una secuencia de tramas *wake-up* de longitud igual al producto entre el número de canales y el valor del atributo *macCSLPeriod* antes del envío de la trama de datos o información (*payload frame*).

En el caso de transmisión sincronizada, la transmisión CSL calcula el siguiente tiempo de muestreo y el número de canal, enviando a continuación en el siguiente muestreo sobre el canal correcto con una secuencia corta de tramas *wake-up*. En este caso, la fase CSL es la duración desde el instante actual hasta el siguiente muestreo del canal del primer canal seleccionado dentro de la máscara de canales definida por *macCSLChannelMask.*

3.1.5. REDUCCIÓN DE LATENCIA MEDIANTE APAGADO DEL MODO CSL

El nivel superior tiene la opción de desactivar o apagar el muestreo de canales y la escucha inactiva, eliminando con ello el envío periódico de secuencias de tramas *wake-up* con el objetivo de reducir la latencia en el caso de mensajes urgentes.

Esto considera, por tanto, que el nivel superior es quien gestiona la coordinación entre el emisor y el receptor en los procesos de encendido y apagado del proceso de muestreo periódico y escucha inactiva.

El proceso de apagado o desactivación requiere que el nivel superior establezca el valor del atributo *macCSLPeriod* a valor cero mientras que el proceso de encendido es llevado a cabo restaurando los valores distinto de cero sobre este mismo atributo.

De manera similar, para detener el envío de secuencias de tramas *wake-up,* el nivel superior establece el valor del atributo *macCSLMaxPeriod* a valor cero, restaurando el comportamiento de nuevo restableciendo el atributo a su valor previo.

Finalmente, para solicitar a un dispositivo vecino la detención del proceso de muestreo periódico y escucha inactiva, el nivel superior debe enviar una trama al dispositivo destino con el bit *frame pending* a valor uno (1). Esta acción evitará que CSL apague la radio antes de que la petición sea procesada.

3.2. ATRIBUTOS CSL

Tal y como se ha indicado en la descripción funcional, el modo de funcionamiento CSL considera los siguientes atributos dentro de la PIB (*PAN Information Base*) del nivel MAC.

Atributo	Descripción
macCSLPeriod	Periodo de muestreo de canales y escucha inactiva. En el caso que tenga valor cero, la escucha del canal será continua.
macCSLMaxPeriod	Periodo máximo de muestreo de canales y escucha inactive. Determina la longitud de la secuencia de tramas *wake-up* en el proceso de comunicación con el dispositivo cuyo periodo de escucha CSL es desconocido. El nivel superior puede establecer este atributo a valor cero para detener el envío de secuencias de tramas *wake-up* con la correspondiente coordinación entre los dispositivos vecinos.
macCSLChannelMask	Mapa de bits relacionado con los canales disponibles. Representa el listado de canales en los cuales CSL está active.
macCSLFramPendingWaitT	Periodo de tiempo en el cual se mantiene activo el receptor tras la recepción de una trama de datos o información (*payload frame)* con el bit *frame pending* del campo *Frame Control* a valor uno.

Ilustración 17: Definición de Atributos CSL

A continuación, en la siguiente ilustración se recogen los atributos descritos anteriormente tal y como se encuentran recogidos en la enmienda 802.15.4e al estándar.

Attribute	Type	Range	Description	Default
macCSLPeriod	Integer	0 … 65535	CSL sampled listening period in unit of 10 symbols. Zero means always listening, i.e., CSL off.	0
macCSLMax-Period	Integer	0 … 65535	Maximum CSL sampled listening period in unit of 10 symbols in the entire PAN. This determines the length of the wake-up sequence when communicating to a device whose CSL listen period is unknown. NHL may set this attribute to zero to stop sending wake-up sequences with proper coordination with neighboring devices.	*macCSLPeriod*
macCSLChannel Mask	Bitmap	0x00000000 … 0xffffffff	32-bit bitmap relative to phyCurrentPage of channels. It represents the list of channels CSL operates on. Zero means CSL operates on phyCurrentChannel of phyCurrentPage.	0x00000000
macCSLFrame-PendingWaitT	Integer	*macMinLIFSPeriod* + max number of symbols per PPDU … 65535	Number of symbols to keep the receiver on after receiving a payload frame with Frame Control field frame pending bit set to one.	—

Ilustración 18: Listado de Atributos PIB para CSL

3.3. MODIFICACIONES EN FORMATOS DE TRAMA

El presente apartado tiene como objetivo presentar aquellos aspectos relativos a los formatos y estructuras de las tramas relacionadas funcionalmente con el modo CSL.

3.3.1. TRAMA MAC GENERAL (GENERAL MAC FRAME)

A continuación se muestra el formato de la estructura general de la trama MAC dentro de la cual cabe destacar el campo *Frame Control,* el cual y como ya se ha ido describiendo en los apartados anteriores, contiene el bit *frame pending.*

<table>
<tr><th>Octets:
1/2</th><th>0/1</th><th>0/2</th><th>0/1/2/8</th><th>0/2</th><th>0/1/2/8</th><th>0/1/5/6/1
0/14</th><th>variable</th><th>variable</th><th>2</th></tr>
<tr><td rowspan="2">Frame Control</td><td rowspan="2">Sequence Number</td><td>Destination PAN Identifier</td><td>Destination Address</td><td>Source PAN Identifier</td><td>Source Address</td><td rowspan="2">Auxiliary Security Header</td><td colspan="2">Information Elements</td><td rowspan="2">Frame Payload</td><td rowspan="2">FCS</td></tr>
<tr><td colspan="4">Addressing fields</td><td>Header IEs</td><td>Payload IEs</td></tr>
<tr><td colspan="7">MHR</td><td colspan="2">MAC Payload</td><td>MFR</td></tr>
</table>

Ilustración 19: Formato General de la Trama MAC (*MAC General Frame*)

3.3.1.1 Campo *Frame Control*

El campo *Frame Control* contiene información relativa al tipo de trama, direccionamiento y otros *flags* de control. A continuación se muestra la estructura y contenido del campo Frame Control para tramas de tipo *beacon, data, acknowledgement* y *MAC commands*.

Bits: 0–2	3	4	5	6	7	8	9	10–11	12–13	14–15
Frame Type	Security Enabled	Frame Pending	AR	PAN ID Compression	Reserved	Sequence Number Suppression	IE List Present	Dest. Addressing Mode	Frame Version	Source Addressing Mode

Ilustración 20: Estructura del Campo *Frame Control*

Tal y como se refleja en la ilustración anterior, el campo *Frame Control* contiene el bit *frame pending* el cual tiene especial relevancia en el modo de operación CSL.

De esta manera y bajo el modo de operación CSL, el bit *frame pending* puede ser establecido a valor uno con el fin de indicar que el dispositivo transmisor tiene múltiples tramas que enviar al mismo receptor por lo que espera que dicho receptor mantenga la radio activa hasta recibir una trama de la secuencia que contenga el bit *frame pending* a cero.

3.3.2. TRAMA WAKE-UP MULTIPROPÓSITO (LE-MULTIPURPOSE WAKE-UP)

La trama *wake-up* es una trama de tipo multipropósito (*multipurpose*) empleada para notificar al nodo receptor el próximo envío de una trama de datos o información (*payload frame*) conteniendo para ello el elemento de información (IE) con la cabecera del tiempo *Rendezvous* (*RZ Time*) indicando el tiempo restante hasta el envío de dicha trama de datos.

La cabecera MAC de una trama *wake-up* contiene el campo *Frame Control,* el campo *Sequence Number,* el *Destination PAN ID,* y el campo *Destination Address.*

A continuación en la siguiente ilustración se muestra el formato y estructura de la trama *wake-up* multipropósito.

Octets: 1	1	2	2	4	2
Frame Control	Sequence Number	Dest. PAN ID	Dest. Address	RZ Time Header IE	IE List Terminator

Ilustración 21: Estructura de la Trama *Wake-Up*

En el campo *Frame Control,* el tipo de trama (*Frame* Type) debe ser igual a 0b101 para indicar que se trata de una trama multipropósito. En el caso de querer añadir protección a la trama *wake-up,* será preciso establecer el campo *Security Enabled* a valor uno.

El campo *Sequence Number* deberá contener el valor actual del *macDSN*.

El campo *Destination PAN ID* contendrá el identificador de la PAN (*Personal Area Network)* mientras que el campo *Destination Address* almacenará la dirección corta (2 octetos) del dispositivo receptor de la trama *wake*-up.

Finalmente y respecto a la carga (*payload*) de la trama de *wake-up*, este tipo de trama no tiene asociado ningún tipo de carga.

3.4. ELEMENTOS DE INFORMACIÓN CSL (IE)

Los elementos de información (IE) proporcionan un método potente, flexible y sencillo para el encapsulamiento de información tal y como se describe en el **apartado 2.5.4 - Information Elements** .

En este sentido, en este apartado se procederá a describir los elementos de información (IE) asociados a CSL y necesarios para su correcto funcionamiento.

3.4.1. LE CSL IE

El elemento de información LE CSL IE debe ser utilizado en todos los *enhanced acknowledgements* en el caso que la propiedad *macLEenabled* tenga un valor igual a true.

A continuación se muestra en la siguiente ilustración la estructura del elemento de información LE CSL:

Octets: 2	2
CSL Phase	CSL Period

Ilustración 22: Formato del Elemento de Información LE CSL

El campo de fase CSL (*CSL Phase*) especifica la información de la fase de CSL. Este campo puede tomar uno de los valores siguientes:

Range	Description
0x0000–0xffff	CSL phase, in 10 symbols

Ilustración 23: Rango de Valores para el campo de Fase CSL (*CSL Phase*) en LE CSL IE

El campo de periodo CSL (*CSL Period*) especifica la información del periodo de CSL. Este campo puede tomar uno de los valores siguientes:

Range	Description
0x0000–0xffff	CSL period, in 10 symbols

Ilustración 24: Rango de Valores para el campo de Periodo CSL (*CSL Period*) en LE CSL IE

Por último, podemos resumir las características y estructura de este elemento de información en los siguientes puntos principales:

- *Header IE* en tramas de tipo (*enhanced*) *acknowledgment*.
- *Element ID* = 0x1A.
- Longitud de contenido (*content length*) = 4 octetos
    - 2 octetos para campo *CSL Phase* con valores entre 0x0000 y 0xFFFF en unidades de 10 símbolos.
    - 2 octetos para campo CSL Period con valores entre 0x0000 y 0xFFFF en unidades de 10 símbolos.

3.4.2. RENDEZVOUS TIME IE

El tiempo de espera hasta la recepción de la trama de datos o información (*payload frame*) es conocido como *rendezvous* (*RZ Time*) el cual es establecido en este elemento de información.

Consta de dos octetos y representa el intervalo de tiempo esperado en unidades de diez símbolos entre el final de la transmisión de la trama de *wake-up* y el comienzo de la transmisión de la trama de datos (*payload frame*).

Este tiempo debe ser establecido por el nivel superior en la solicitud de envío realizada a la capa MAC. La última trama *wake-up* dentro de la secuencia de tramas *wake-up* debe contener el valor del tiempo igual a cero.

Por último, podemos resumir las características y estructura de este elemento de información en los siguientes puntos principales:

- *Header IE* en tramas de tipo *wake-up*.
- *Element ID* = 0x1D.
- Longitud de contenido (*content length*) = 2 octetos
    - 2 octetos para campo *RZ time* con valores entre 0x0000 y 0xFFFF en unidades de 10 símbolos.

4. OPENWSN

4.1. INTRODUCCIÓN

El presente apartado tiene como objetivo describir el proyecto **OpenWSN** (https://openwsn.berkeley.edu) sobre el cual se desarrollará la implementación del modo de funcionamiento CSL, describiendo sus características y cometidos principales a nivel funcional y operativo.

El **proyecto OpenWSN** es una implementación de código abierto de una pila de protocolos plenamente funcional, basada en estándares y aplicable a redes capilares enraizadas dentro del nuevo estándar TSCH (*Time Synchronized Channel Hopping*) definido en la enmienda 802.15.4e.

IEEE 802.15.4e, junto a otros estándares IoT (*Internet of Things*) tales como 6LoWPAN, RPL y CoAP, posibilita el establecimiento de redes malladas altamente fiables y de mínimo consumo, totalmente conectadas e integradas en Internet, conformando la piedra angular para la próxima revolución de las interacciones y comunicaciones máquina-a-máquina.

En este sentido, **OpenWSN** es la primera implementación de código abierto del estándar IEEE802.15.4e.

En los siguientes apartados, se proporcionará una visión general de la pila de protocolos de **OpenWSN** así como detalles clave de su integración y las plataformas y herramientas desarrolladas alrededor del proyecto.

La pila de protocolos de **OpenWSN**, plenamente desarrollada en lenguaje C, ha sido portada a numerosas plataformas hardware actualmente en uso, las cuales contemplan desde antiguos microcontroladores de 16 bits hasta arquitecturas Cortex-M con tecnología de última generación de 32 bits.

Respecto a las herramientas desarrolladas alrededor de las redes de baja potencia, se incluye software de visualización y depuración, un simulador para emular redes **OpenWSN** en un ordenador personal, y el entorno necesario para interconectar estas redes a Internet.

4.2. PILA DE PROTOCOLOS (PROTOCOL STACK)

La revolución producida por el concepto *Internet de las Cosas* (IoT) y las interacciones M2M (*Machine-to-Machine)* está produciendo cambios en los modos por medio de los cuales las personas interactúan con los objetos de su alrededor. En este sentido, los organismos de

estandarización están jugando un rol clave a través de la definición de los distintos protocolos involucrados en los diferentes niveles de comunicación, estableciendo con ello una pila de protocolos *de facto* para el desarrollo de las futuras redes capilares.

En la siguiente ilustración se muestra la pila de protocolos implementada en **OpenWSN**.

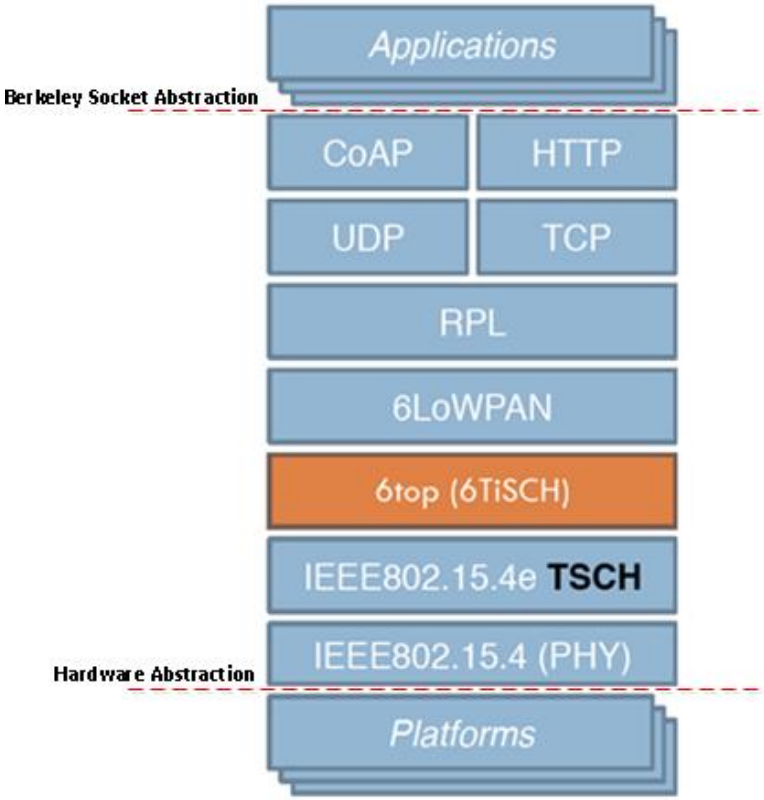


Ilustración 25: Pila de Protocolos de OpenWSN (OpenWSN Protocol Stack)

Así mismo, se muestra a continuación la organización y mapeos de la pila de protocolos realizada en OpenWSN, mostrando claramente las necesidades existentes alrededor de IEEE802.15.4e en términos de capas y módulos.

En este sentido, la capa de enlace (*link layer*) está formada por dos sub-capas:

- **02a-MAC**. Es la capa inferior del nivel MAC, consistente en la implementación IEEE802.15.4e y los temporizadores asociados. Se alimenta de la cola de paquetes a enviar (modulo *OpenQueue*), y el planificador. Se ejecuta enteramente en modo interrupción o ISR (*interrupt service routine*).
- **02b-RES**. Es la capa superior del nivel MAC, siendo responsable de la gestión de la planificación. Se ejecuta enteramente en modo tarea

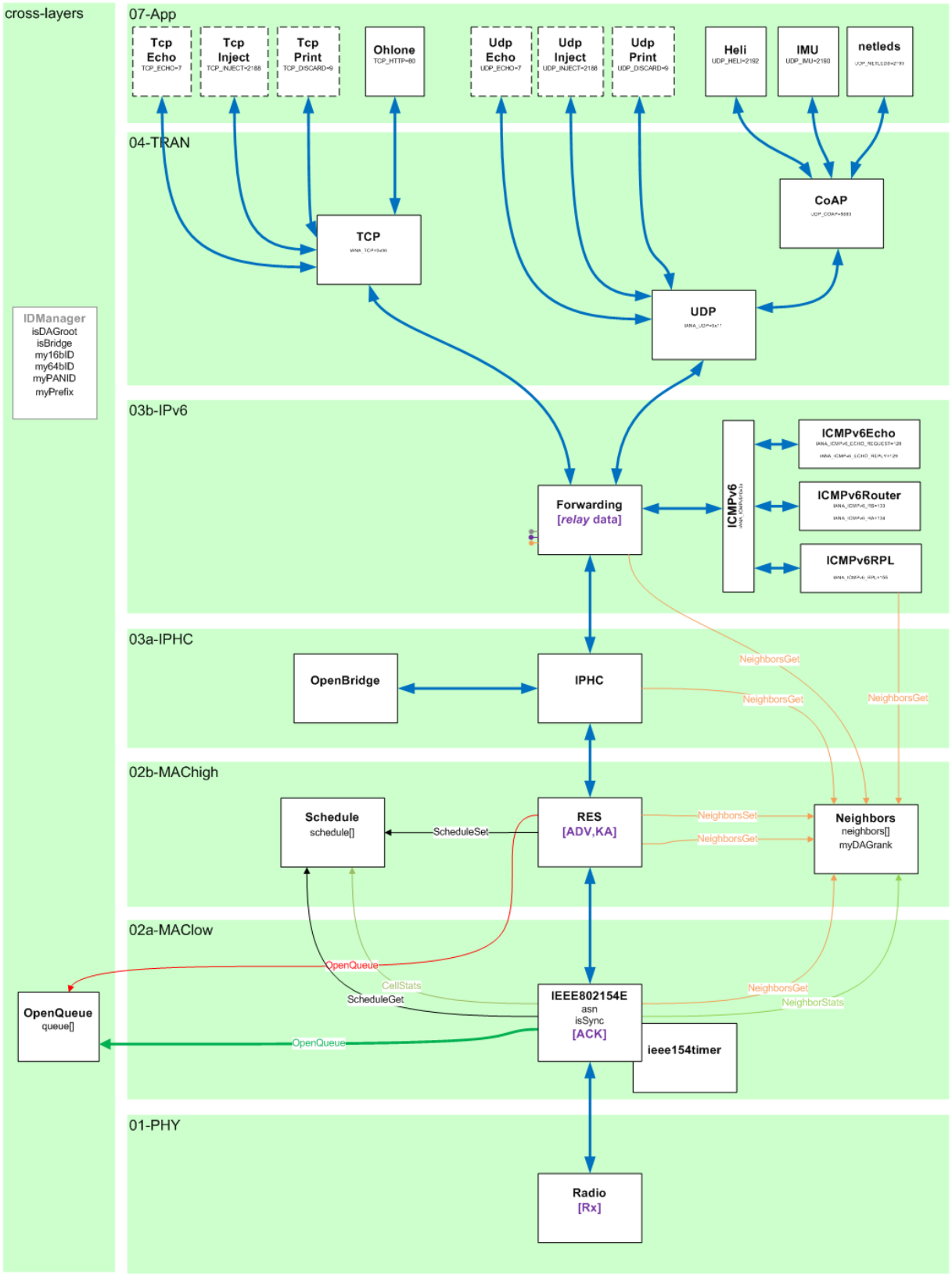


Ilustración 26: Organización de la Pila de Protocolos de OpenWSN

Tal y como se puede observar en las ilustraciones anteriores, la pila de protocolos está basada enteramente en estándares IoT (*Internet of Things*), utilizando **abstracciones** a dos niveles:

- ***Berkeley Socket Abstraction***, la cual fue implementada como parte del desarrollo del sistema operativo de la *Berkeley Software Distribution*, siendo adoptado por todos los sistemas operativos y constituyendo la base de comunicación de Internet. Esta abstracción considera que las aplicaciones en dos servidores se comunican entre sí a través de un *socket* el cual es unívocamente identificado por las direcciones IP de ambos servidores y los dos puertos asociados a cada aplicación.
  La pila de protocolos OpenWSN respeta esta abstracción por lo que el desarrollo de cualquier aplicación sobre ella es totalmente análogo al desarrollo de una aplicación en un servidor de Internet.
- ***Hardware Abstraction***, el cual consiste fundamentalmente en agrupar todas las funcionalidades relacionadas con el acceso al hardware (fundamentalmente funciones de acceso y escritura de los registros) dentro de un grupo de ficheros denominados BSP (*Board Support Package*). Como resultado, esto permite la compartición de prácticamente la totalidad de código entre todas las plataformas de tal modo que existirá un BSP para cada plataforma soportada, permaneciendo el resto del código de manera compartida entre todas las implementaciones.

Analizando los distintos niveles que componen la pila de protocolos de OpenWSN y comenzando por los niveles inferiores hacia los niveles superiores, tenemos:

- ***IEEE 802.15.4 (PHY)***, el cual representa la implementación de la capa de nivel físico definida por el estándar. El cometido principal de esta capa es establecer los mecanismos adecuados para la intercomunicación entre los distintos elementos por medio del enlace radio dentro de los cuales se contemplan aspectos de modulación, tasa de transferencia, o las potencias de transmisión y recepción, entre otros.
- ***IEEE 802.15.4e TSCH****,* el cual representa la implementación de la capa MAC bajo la cual se arbitra el acceso al medio de comunicación inalámbrico. OpenWSN incorpora TSCH como mecanismo de acceso al medio de tal manera que el tiempo es particionado en porciones denominadas *time-slots* por medio de las cuales las motas pueden comunicarse y sincronizarse.
  En este sentido, se define el concepto de supertrama (*superframe*) formado por un número de time-slots (típicamente de decenas a pocos miles de slots) que se repiten en el tiempo. A través de un planificador (*schedule*) distribuido y manejado por todas

las motas, se indica la acción que deben tomar en cada slot de la supertrama, es decir, transmitir, recibir o dormir. La modificación de esta planificación permite establecer un equilibrio entre latencia, rendimiento de red y consumo de potencia.

Finalmente, cada slot es asignado a un desplazamiento de canal lo cual se traduce en la traslación de la comunicación a una frecuencia diferente en cada repetición de la supertrama de tal modo que mientras que cada mota mantenga la misma planificación (*schedule*), cada transmisión o retransmisión tendrá lugar a una frecuencia diferente. Esta técnica, conocida como *channel hopping* es ampliamente utilizada para combatir las interferencias externas y el desvanecimiento multi-camino (*multi-path fading*).

- **6top (6TiSCH)**. 6TiSCH hace referencia al nivel de adaptación para el correcto funcionamiento de IPv6 sobre el modo de funcionamiento TSCH de IEEE 802.15.4e. Para ello, define la sub-capa **6top** así como un conjunto de protocolos empleados principalmente para el establecimiento de una planificación (*schedule*) de manera centralizada o distribuida, gestionando la asignación de recursos.

  Así mismo, 6TiSCH define la arquitectura global para el enlazado de todos ellos juntos para su uso den redes basadas en IPV6 y TSCH. Por su parte, la subcapa 6top es la responsable de la adaptación entre TSCH y los niveles superiores como 6LoWPAN y RPL.
- ***6LoWPAN**,* la cual se trata de una especificación consistente en un conjunto de reglas para el análisis y adecuación de cabeceras IPv6 a redes WPAN. Esta capa elimina por tanto campos que no son necesarios (por ejemplo, el campo versión dado que se mantiene inalterable), y comprime otros campos si es posible (por ejemplo, el campo de dirección origen y destino dado que parte de su contenido puede ser inferido del prefijo de la red IPv6).

  De este modo, todos los paquetes que se intercambien en la red WPAN contendrán solamente la cabecera 6LoWPAN resultante, la cual puede ser tan pequeña como dos bytes en el caso más favorable.

  A este respecto y dado que se precisa una cabecera IPv6 completa para soportar la comunicación en Internet, una red OpenWSN implementa la funcionalidad LBR (*Low-Power Border Router*) la cual se sitúa entre la red de motas e Internet y cuyo cometido es transformar las cabeceras 6LoWPAN en cabeceras IPv6 en los paquetes salientes de la red de motas, y compactar las cabeceras IPv6 en los paquetes entrantes.

El resultado es que cada mota puede tener asignada una dirección IPv6 y mostrarse en Internet al igual que cualquier otro servidor de Internet. Esto permite el desarrollo sencillo de aplicaciones cliente, especialmente en casos donde los usuarios no disponen de un conocimiento previo sobre tecnologías WSN de bajo nivel.

- ***RPL**,* el cual es utilizado sobre 6LoWPAN para el mantenimiento de la topología y el encaminamiento de paquetes (*routing*). Por tanto, este nivel es responsable del encaminamiento de paquetes en la red a través de varios saltos entre el nodo origen y destino de la comunicación. Se compone de un motor de envío, el cual utiliza una tabla de encaminamiento para decidir el nodo vecino que se corresponde con el siguiente salto del paquete, y un protocolo de encaminamiento encargado del mantenimiento y actualización de la tabla de rutas.

  Para ello, RPL considera el uso de dos mecanismos: colección (*collection routing*) y origen (*source routing*).

  En el caso de colección de información de enrutamiento, es preciso construir un gradiente de red denominado grafo acíclico dirigido por destino (DAG). OpenWSN define diferentes métricas para establecer el gradiente a partir de la inversa de la relación de probabilidad de entrega usada por defecto. Para ello, los nodos informan periódicamente medidas a un pequeño número de puntos de colección, no existiendo normalmente comunicación entre los puntos de colección y los nodos individuales.

  En el caso de enrutamiento origen, la información es mantenida por los nodos LBR mediante el almacenamiento de una tabla con una ruta para cada posible destino en la red. Esta tabla es actualizada de manera periódica a través de mensajes de anuncio de destino (*destination advertisement*) enviados por todos los nodos de la red.
- **CoAP**, el cual se trata de un protocolo de aplicación por medio del cual es posible habilitar la interacción por mensajes REST con motas individuales, sin la sobrecarga introducida por TCP y el detalle de información existente en HTTP. El protocolo CoAP está formado por una cabecera de cuatro bytes sobre datagramas UDP.

  Como resultado, CoAP habilita a las motas para actuar como navegador web y como servidor web.

4.3. CASOS DE USO

En la siguiente ilustración se recoge el caso de uso típico referido a la conexión a Internet de una red OpenWSN.

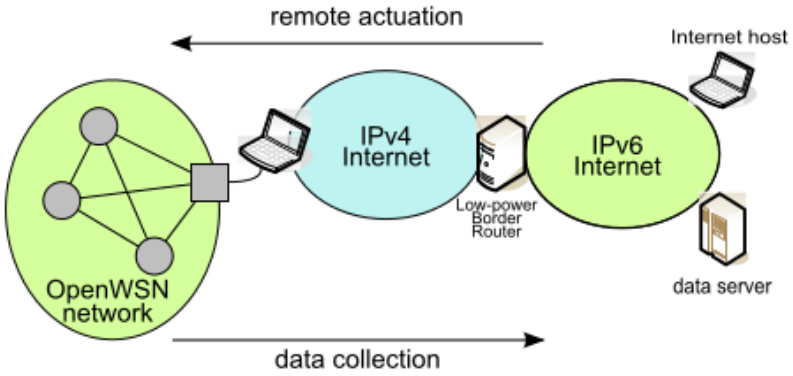


Ilustración 27: Casos de Uso de OpenWSN

Tal y como se ha mostrado en el apartado anterior, gracias a la abstracción a nivel de *socket* (*Berkeley Socket Abstraction*), es sencillo implementar una aplicación sobre la pila de protocolos de OpenWSN para la comunicación con clientes vía Internet tales como sensores o dispositivos de actuación.

En este sentido, se presentan a continuación tres casos de uso típicos encargados de dar cobertura a prácticamente la mayoría del conjunto de aplicaciones y funcionalidades requeridas.

El caso de uso más común es la **captura de información** (*data collection*). En este caso, la mota se encuentra conectada a un sensor físico de tal manera que una aplicación sobre OpenWSN permite muestrear periódicamente el sensor e iniciar una transmisión a un puerto UDP CoAP del servidor de datos en Internet. Los datos del sensor son pasados a la capa CoAP la cual añade la cabecera indicando el recurso origen del cual procede la información. De acuerdo a la pila de protocolos, este mensaje es encapsulado en cada nivel, añadiendo las cabeceras UDP, 6LoWPAN y IEEE802.15.4e respectivamente. Como resultado, las tramas son planificadas para su transmisión empleando IEEE802.15.4e TSCH en la capa MAC.

Cuando este paquete es recibido en la frontera de la red, los paquetes son encaminados hacia el LBR el cual traduce la cabecera 6LoWPAN en una cabecera IPv6 para su correcta transmisión por Internet. El servidor de datos ubicado en Internet responsable de la recepción de los paquetes, escucha en un puerto UDP CoAP conocido por el cual recibirá los datos y los almacenará en su base de datos desde la cual pueden ser mostrados o procesados.

El segundo caso de uso a destacar está referido al **envío de datos desde Internet** a una mota concreta, esto es, una mota equipada con un dispositivo de actuación. En este caso, el

servidor en Internet dispone de una aplicación cliente encargada de formatear y enviar comandos a los recursos CoAP de la mota (*coap://ipv6::addr/resource/*).

Estos comandos son enviados desde el servidor en Internet usando paquetes IPv6 hasta el LBR situado en la frontera de la red OpenWSN en el cual se llevan a cabo las operaciones inversas de traducción y compresión entre IPv6 y 6LoWPAN. Cuando el mensaje llega a la mota, la aplicación CoAP analiza el comando recibido y ejecuta las acciones locales asociadas a dicho comando.

Finalmente y como tercer caso de uso, es posible definir **interacciones cliente-servidor más complejas** entre una mota y un servidor de internet. De este modo, el servidor de Internet puede solicitar a la mota el listado de todos sus recursos disponibles, esto es, la lista de aplicaciones disponibles sobre CoAP, a través de la invocación del recurso "*well-known*" (*coap://ipv6::addr/well-known/*). Esta operación devuelve el listado de recursos disponibles que el cliente puede invocar para obtener las últimas lecturas del sensor o para ejecutar la actuación de un evento.

Así mismo, para una mota es igualmente posible buscar recursos CoAP disponibles en Internet. Por ejemplo, una mota asociada a un sensor inteligente de riego podría solicitar la previsión meteorológica en un servidor CoAP en Internet para determinar inteligentemente si debe o no iniciar dicho proceso de riego.

4.4. PLATAFORMAS HARDWARE

OpenWSN se encuentra actualmente portado a numerosas plataformas hardware, cuyo listado mostrado a continuación no es más que una muestra representativa del tipo de hardware que puede ser encontrado fácilmente hoy en día.

En el momento de la elaboración de la presente memoria, el hardware soportado por OpenWSN es el siguiente:

- TelosB
- GINA
- WSN430
- Zolertia Z1
- OpenMoteCC2538
- OpenMoteSTM
- SAM R21 Xplained Pro
- USP Mote MC13213

- USP Mote CC2538
- IoT-LAB_M3
- AgileFox
- Experimentales
    - OpenMoteHack
    - K20hack
    - USP/SC Motes
    - eZ430-RF2500
    - Current Monitor
    - XpressoHack

Estas plataformas comprenden un amplio abanico desde la plataforma más antigua y con menor rendimiento como es TelosB hasta las plataformas más avanzadas basadas en microcontroladores Cortex de 32 bits. En cualquier caso, todas ellas tienen en común la utilización de una radio externa así como una comunicación con el microcontrolador basada en interfaz serie SPI (*serial peripheral interface*).

En este punto, resulta importante distinguir conceptos como mota o SOC (*System-On-Chip*) dado que esto permitirá clarificar aspectos respecto al soporte actual hardware de OpenWSN.

En este sentido, una **mota** podemos definirla como una placa hardware con varios chips sobre ella entre los que cabe destacar el microcontrolador, la radio y los puertos para la conectividad serie y USB. A continuación se muestra un extracto de motas actualmente soportadas en OpenWSN.

Nombre	MicroControlador	Radio	Batería
deUSB2400	Atmel AT91SAM7S256 256kBROM/64kBRAM	AT86RF231	USB
RAVEN	AT90USB1287	AT86RF230	USB
EPIC	MSP430f1611	CC2420	-
TelosB	MSP430f1611	CC2420	2AA
IRIS	AtMega128L	AT86RF230	2AA
MICAz	AtMega128L	CC2420	2AA
CC2531EMK	8051 (SoC)	CC2520 (SoC)	USB

Tabla 3: Extracto de placas hardware (motas) soportadas en OpenWSN

Por su parte, SOC hace referencia a un único chip encargado de contener en el mismo chip el micro-controlador y la radio, tales como los mostrados a continuación.

Nombre	Descripción
Ember EM357	**ARM Cortex M3, 32-bit, 192kBROM/12kBRAM**
STMicroelectronics STM32W	**ARM Cortex M3, 32-bit, 128kBROM/8kBRAM**
Freescale MC13224V	**ARM7, 32-bit, 128kBROM/96kBRAM**
Jennic JN5148	**32-bit, 128kBROM/128kBRAM + ToF engine!**
Texas Instruments CC2531	**TI 8051 8-bit, 256kBROM/8kBRAM + CC2520**

Tabla 4: Extracto de SoC soportadas en OpenWSN

Respecto a los **microcontroladores** soportados, a continuación se proporciona igualmente un extracto de los distintos tipos de microcontroladores actualmente soportados en OpenWSN.

Nombre	Arch.	Velocidad	Potencia	RAM	ROM	Timers	E/S
ARM7TDMI	32-bit	115-236MHz	2mW/MHz	8kB	-	6 Timers + RTC	3 USART, SPI, 8 10-bit ADC, JTAG
ARM920T	32-bit	180-200MHz	22.4mA/250uA	16kB	128kB	Extensivo	UART, USB, Ethernet, 4 USART, I2C, SPI, JTAG
MSP430f2274	16-bit	16MHz	390uA/100nA	1kB	32kB	2 timers con 2 CCR/timer	UART/LIN/IsDA/SPI y I2C/SPI
MSP430f1611	16-bit	8MHz	500uA/200nA	10kB	48kB	16-bit (3CCR), 16-bit (7CCR)	USART (SPI o UART o I2C), USART (SPI o UART)
ATmega128A	8-bit	16MHz	9.8mA/1mA	4kB	128kB	2 8-bit, 2 16-bit	2USART, SPI

Tabla 5: Extracto de MicroControladores soportados en OpenWSN

Finalmente y respecto a las **interfaces de radio** soportadas, a continuación se proporciona igualmente un extracto de las distintas interfaces actualmente soportadas en OpenWSN.

Fabricante	Modelo	Sensibilidad	Potencia Tx / Rx / Sleep
Atmel	AT86RF230	-101dBm	16.5mA/15.5mA/20nA
Atmel	AT86RF231	-101dBm	14mA/12.3mA/20nA
Texas Instruments	TI CC2420	-95dBm	17.4mA/18.8mA/20nA
Texas Instruments	TI CC2520	-98dBm	25.8mA/18.8mA/30nA

Tabla 6: Extracto de Interfaces de Radio soportadas en OpenWSN

En el **apartado 5 - Plataforma OpenMote CC2538** podemos consultar detalles más específicos referentes al tipo de mota OpenMote CC2538 utilizada para el despliegue y pruebas del desarrollo realizado para la implementación del modo CSL.

4.5. HERRAMIENTAS OPENWSN

El siguiente apartado tiene como objetivo proporcionar una breve descripción de las principales herramientas desarrolladas alrededor del proyecto OpenWSN.

4.5.1. TOOLCHAINS

El código fuente de OpenWSN está completamente desarrollado en lenguaje C lo cual implica que puede ser compilado con cualquier *toolchain*, esto es, cualquier herramienta con capacidad de construir el firmware de OpenWSN, cargarlo en el hardware y ejecutar el proceso de depuración.

En este sentido, la depuración es posible realizarla en todas las plataformas hardware a través del interfaz JTAG (*Joint Test Action Group*) por medio del cual es posible establecer puntos de ruptura en el código (*breakpoints*) y congelar la ejecución, inspeccionando el valor de las variables y registros.

JTAG es fundamentalmente el nombre común utilizado para la norma IEEE 1149.1 denominada *Standard Test Access Port and Boundary-Scan Architecture* utilizada para probar y validar PCBs utilizando escaneo de límites.

JTAG fue estandarizado en 1990 como la norma IEEE 1149.1-1990, agregando posteriormente un suplemento para la incorporación de la descripción del *boundary scan description language* (BSDL). Desde entonces, esta norma fue adoptada por las compañías electrónicas de todo el mundo.

Diseñado originalmente para circuitos impresos, actualmente es utilizado para la prueba de sub-módulos de circuitos integrados, y es muy útil también como mecanismo para depuración de aplicaciones empotradas. Su uso como herramienta de depuración permite emular el circuito que usa JTAG como mecanismo de transporte posibilitando que el programador pueda acceder al módulo de depuración integrado dentro del microcontrolador.

4.5.2. OPENOS

Este componente es el planificador principal (*kernel scheduler*), desarrollado como parte del proyecto OpenWSN. Las interrupciones hardware así como las interrupciones producidas por los temporizadores (*timers*) determinan las tareas a ejecutar en base a su prioridad, almacenando éstas en una lista interna de tareas (*task list*).

En la medida que existan tareas dentro de esta lista, el planificador invocará a la función *callback* asociada a cada tarea, eliminando posteriormente la tarea de la lista. Cuando la lista se encuentre vacía y ya no existan más tareas a tratar, el planificador conmutará el microcontrolador a un estado latente (*deep sleep)* en espera de una nueva interrupción que conlleve la inclusión de tareas en la lista.

En este sentido, OpenOS no es un sistema con derechos preferentes, es decir, las tareas no interrumpen a otras tareas así como OpenWSN no se encuentra ligado al planificador OpenOS de manera directa lo cual permite que OpenWSN pueda ejecutarse como parte de un sistema operativo diferente.

4.5.3. 6LOWPAN LBR

Tal y como se ha visto anteriormente en el **apartado 4.2 - Pila de Protocolos (Protocol Stack)**, OpenWSN implementa el protocolo 6LoWPAN por medio del cual las motas pueden disponer de una IPv6 totalmente direccionable sin la necesidad de tratar con cabeceras IPv6 de 40 bytes en cada trama IEEE802.15.4 de 127 bytes.

En este sentido, todos los paquetes en la red de baja potencia contienen una cabecera 6LoWPAN de tal modo que para comunicar con otros servidores de Internet mediante IPv6, OpenWSN implementa un *router* LBR encargado de convertir las cabeceras 6LoWPAN en

cabeceras IPv6 para paquetes salientes de la red de motas hacia Internet, así como realizar el proceso inverso para los paquetes entrantes a la red de motas.

4.5.4. OPENVISUALIZER

El entorno *OpenVisualizer* es una aplicación de visualización y depuración desarrollada en lenguaje Python, la cual se ejecuta en un ordenador personal e interactúa con las motas OpenWSN conectadas a él.

La comunicación con cada mota es realizada a través del puerto serie, mostrando información relevante de red como el estado interno de cada mota (conectividad, tablas de nodos vecinos, y estado de las colas), gráficos de conectividad multi-salto, y códigos de error y depuración generados por las motas. Así mismo, *OpenVisualizer* permite la interacción con las aplicaciones disponibles en cada mota.

OpenVisualizer, dado su desarrollo en lenguaje Python, ha sido diseñado para ser totalmente independiente del sistema operativo, pudiendo por tanto ser ejecutado en cualquier ordenador personal con soporte para el interfaz serie.

En este sentido y más allá de proporcionar un marco común e integrado de visualización, OpenVisualizer está formado por un *framework* Python modular que posibilita el desarrollo sencillo y ágil de aplicaciones clientes con interacción en la red de motas.

4.5.5. OPENSIM

Tal y como se ha descrito anteriormente en el detalle de la pila de protocolos OpenWSN, las funciones que interactúan directamente con el hardware son agrupadas en un paquete específico para cada plataforma denominado BSP (*Board Support Package*), manteniendo el resto de código de OpenWSN común entre todas ellas.

En este sentido, OpenWSN proporciona un tipo especial de BSP encargado de emular el comportamiento de un hardware genérico en un ordenador personal. De este modo, es posible generar el *stack* así como aplicaciones OpenWSN y emular su comportamiento en una red de motas simulada sobre un entorno Windows o Linux.

La simulación de múltiples motas es llevada a cabo a través de la conexión de cada una de ellas a un elemento central de simulación (*simulation core*) encargado de gestionar la concurrencia entre los dispositivos emulados y la propagación de los paquetes. Este elemento central de simulación es denominado *OpenSim* cuya arquitectura se muestra a continuación.

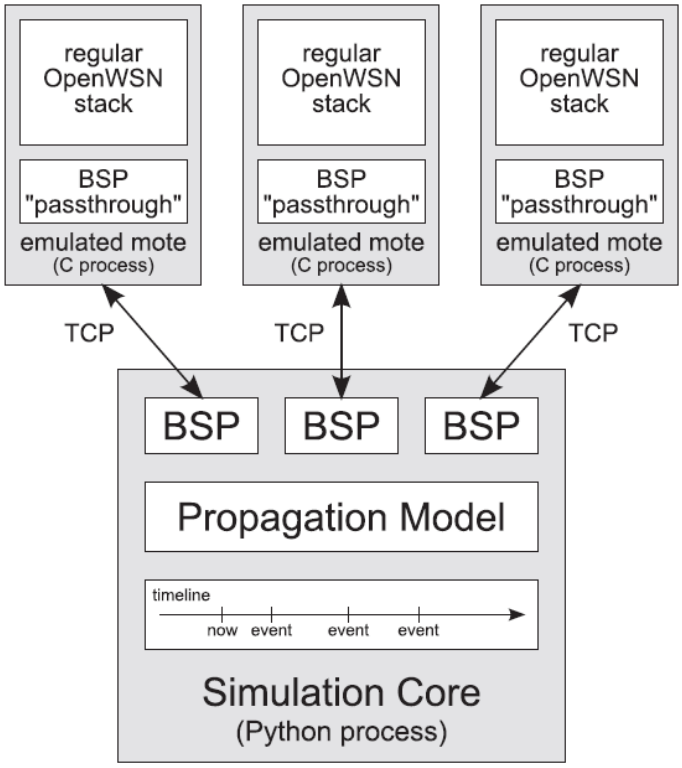


Ilustración 28: Arquitectura del Simulador OpenSIM

Cada mota emulada ejecuta un proceso en el PC, comunicándose con el núcleo central de simulación (*simulation core*) mediante una sesión TCP. Cuando el entorno OpenSIM se inicia, el núcleo central de simulación es inicializado y se inician tanto procesos de emulación de motas como motas existan en la red simulada. En el proceso de arranque de cada una de estas motas, se establece una conexión con el núcleo central de simulación quién instancia un objeto representando el BSP de la mota. Cuando el *stack* en modo emulado invoca alguna función del BSP, esto se traduce internamente en una llamada a un procedimiento remoto desde la mota emulada al núcleo central de simulación en el cual se ejecuta realmente la función BSP.

Tanto el núcleo central de simulación como las motas emuladas ejecutan código de manera síncrona, es decir, hasta que el núcleo central de simulación no devuelva el resultado de la llamada al BSP, la mota emulada permanece a la espera y no continúa ejecutando ningún código. En consecuencia, esto permite que el núcleo central de simulación pueda pausar la ejecución de cualquier mota emulada en cualquier instante de tiempo y coordinar ejecuciones concurrentes entre diferentes motas.

Finalmente, el núcleo central de simulación es un simulador de eventos discretos de tal manera que contiene una línea temporal (*timeline*) formada por un número de eventos a

futuro así como el código a ejecutar en cada uno de ellos. Un evento es típicamente la expiración de un temporizador hardware sobre la mota emulada. De este modo y durante la simulación, la ejecución de una llamada BSP provoca más eventos a incluir sobre la línea temporal para su ejecución futura, los cuales serán atendidos por el núcleo central de simulación uno tras otro.

4.6. MÁQUINA DE ESTADOS IEEE 802.15.4

En una red IEEE802.15.4E, el tiempo es particionado en pequeñas porciones denominadas *time-slots.* En cada uno de estos *slots,* la mota transmite, recibe o permanece latente. Durante los procesos de transmisión y recepción, resulta necesario disponer de una precisión de tiempos con el fin de mantener la precisión y la sincronización en la realización de las distintas acciones.

A continuación en las siguientes ilustraciones se muestra una visión simplificada de la máquina de estados en los procedimientos de transmisión y recepción dentro de un *timeslot*.

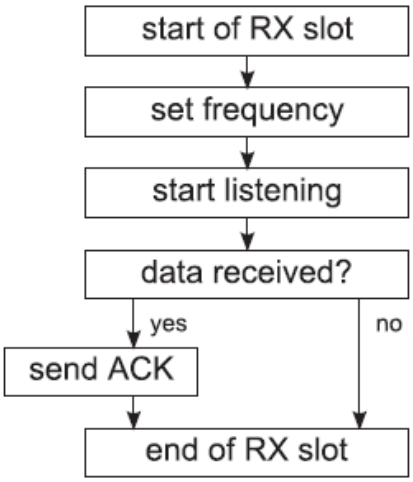


Ilustración 29: Máquina de Estados Finitos (FSM) en Recepción (RX) IEEE802.15.4e

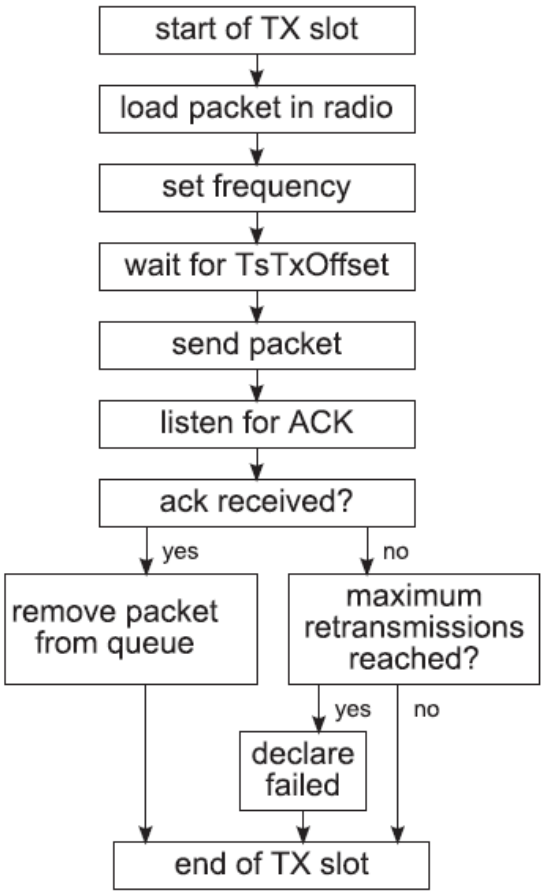


Ilustración 30: Máquina de Estados Finitos (FSM) en Transmisión (RX) IEEE802.15.4e

La mota transmisora debe enviar un paquete de datos exactamente un tiempo *TsTxOffset* después del inicio del *timeslot*, con el fin de que la mota receptora sea capaz de evaluar cómo de sincronizada se encuentra respecto al transmisor.

Así mismo, el *timeslot* debe ser lo suficientemente largo como para permitir al transmisor el envío de una trama de longitud máxima (127 bytes) y recibir el reconocimiento (ACK) asociado desde la mota receptora.

En OpenWSN, la máquina de estados asociada a una transmisión y recepción completa está formada por nueve estados de tal manera que en cada estado, el microcontrolador debe ejecutar tareas y operaciones atómicas tales como la comunicación con la radio y la planificación de las expiraciones del temporizador hardware. El código asociado a estos estados es ejecutado en un contexto de interrupciones sobre el microcontrolador y sin intervención del planificador.

A continuación en la siguiente ilustración se muestra gráficamente el proceso de transmisión y recepción completa en un *timeslot*, indicando los eventos producidos y los tiempos implicados así como las variaciones producidas sobre los estados de la máquina de estados.

Con el fin de entender y comprender mejor el contenido de esta ilustración, se proporciona a continuación una lista de aspectos clave para su entendimiento:

- El fichero representa la actividad de una mota transmisora (TX) y una mota receptora (RX) durante un solo *timeslot.*
- Tanto para el transmisor (TX) como el receptor (RX), la ilustración muestra:
    - La actividad de la mota. Cuando la línea roja se encuentra por encima o por debajo de la línea punteada, la mota está transmitiendo o recibiendo respectivamente.
    - El valor de las variables globales.
    - Los eventos y los temporizadores involucrados.
    - El estado de los cuatro pins de depuración.
- La parte central de la ilustración muestra la nomenclatura de los distintos tiempos involucrados en el proceso.
- Cada temporizador, tarea o actividad de interrupción es dotada de un nombre de acuerdo a la siguiente nomenclatura:
    - La primera letra identifica si se refiere al transmisor (t) o receptor (r).
    - La segunda letra se refiere al tipo: temporizador (t), tarea (a) o interrupción (i).
    - El número es utilizado para la identificación unívoca dentro de cada actividad.
    - Las actividades de interrupción (ISR) son etiquetadas con una “e” cuando se trata de actividades de excepción.

Ilustración 31: Máquina de Estados Finitos (FSM) en Transmisión (TX) y Recepción (RX) en un Timeslot

5. PLATAFORMA OPENMOTE CC2538

5.1. DESCRIPCIÓN

La mota OpenMote-CC2538 representa el núcleo del hardware OpenMote. Cuando la mota se encuentra conectada a una placa *OpenBattery*, es posible interconectar varios sensores y funcionar de manera autónoma como un nodo. En el caso que la mota esté conectada a la placa *OpenBase*, es posible interconectar varias redes y operar como puerta de entrada (*gateway*) siendo posible además depurar los desarrollos realizados. Actualmente la mota OpenMote-CC2538 está soportada por las principales implementaciones de código abierto del estándar IEEE802.15.4 como son Contiki, OpenWSN y FreeRTOS.

En la siguiente ilustración se muestra el aspecto de una mota OpenMote-CC2538.

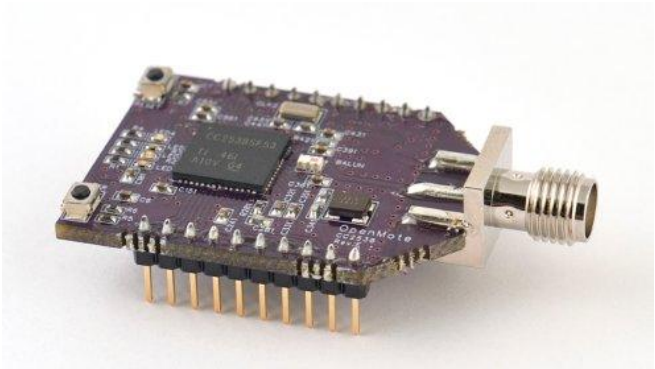


Ilustración 32: Aspecto Visual de la mota OpenMote-CC2538

5.2. COMPONENTES

La mota OpenMote-CC2538 está constituida principalmente por los siguientes componentes hardware:

- **CC2538**: Representa el microcontrolador y la radio de la mota, siendo pieza fundamental dentro de la estructura global de componentes. Se trata de un SoC de Texas Instruments formado por un microcontrolador Cortex-M3 de 32 bits y un transceptor radio tipo CC2520. El microcontrolador opera a una frecuencia de 32 MHz e incluye 32 Kbytes de RAM y 512 Kbytes de memoria Flash así como los periféricos más comunes tales como GPIOs, ADC y temporizadores, entre otros. Por su parte, la radio opera en la banda de 2,4 GHz, siendo totalmente compatible con el estándar IEEE802.15.4-2006.

- **TPS62730**: Se trata de un convertidor-reductor de corriente continua de Texas Instruments con capacidad para funcionar en dos modos de operación: regulado o derivado (*bypass*). En modo *bypass*, el componente directamente conecta la tensión de entrada de la batería (típicamente 3 Voltios) a todo el sistema. En modo regulado, el componente regula la tensión de entrada (típicamente 3 Voltios) a un valor reducido de 2.1 Voltios. La ventaja de este modo de funcionamiento es que la eficiencia global del sistema se ve mejorada tanto bajo condiciones de alta o baja carga, esto es, tanto cuando el sistema se encuentra latente como cuando la radio se encuentra transmitiendo o recibiendo datos.
- **ABM8G**: Este elemento se trata de un cristal de cuarzo de 32 MHz de la compañía *Abracon Corporation*, utilizado para la generación de la señal de reloj del microcontrolador y el transceptor radio. La pieza presenta una precisión de 30 ppm (partes por millón) en el rango de temperaturas comprendido entre -20 ºC y +70 ºC.
- **ABS07**: Este elemento se trata de un cristal de cuarzo de 32.768 kHz de la compañía *Abracon Corporation* utilizado para la generación de la señal de reloj RTC (*Real Time Clock*) del microcontrolador. La pieza presenta una precisión de 10ppm (partes por millón) en el rango de temperaturas comprendido entre -40 ºC y +85 ºC.
- **LEDs**: Se trata de un conjunto de diodos emisores de luz formado por cuatro diodos de color rojo, verde, amarillo y naranja. Es fabricado por la empresa *Rohm Semiconductor* y son utilizados con fines de depuración.
- **Botones**: La mota incluye dos botones de la compañía *Omron*. El primer botón es utilizado para el reinicio de la placa mientras que el segundo botón se encuentra conectado a la línea GPIO habilitando el inicio del microcontrolador desde los estados de latencia (*sleep modes*) a través de una interrupción.
- **Conector de Antena**: Este conector permite conectar la placa a una antena externa.
- **XBee**: La mota es totalmente compatible con el factor de forma XBee (*XBee Form Factor*) lo cual significa que puede ser integrado facilmente desde un ordenador personal utilizando *XBee Explorer Dongle* de *Sparkfun*. Esto no significa sin embargo que sea físicamente compatible con el *XBee Explorer USB* dado que los pins JTAG/USB interfieren con el conector USB.

A continuación en la siguiente ilustración, se muestra el diagrama esquemático de conexiones de la placa:

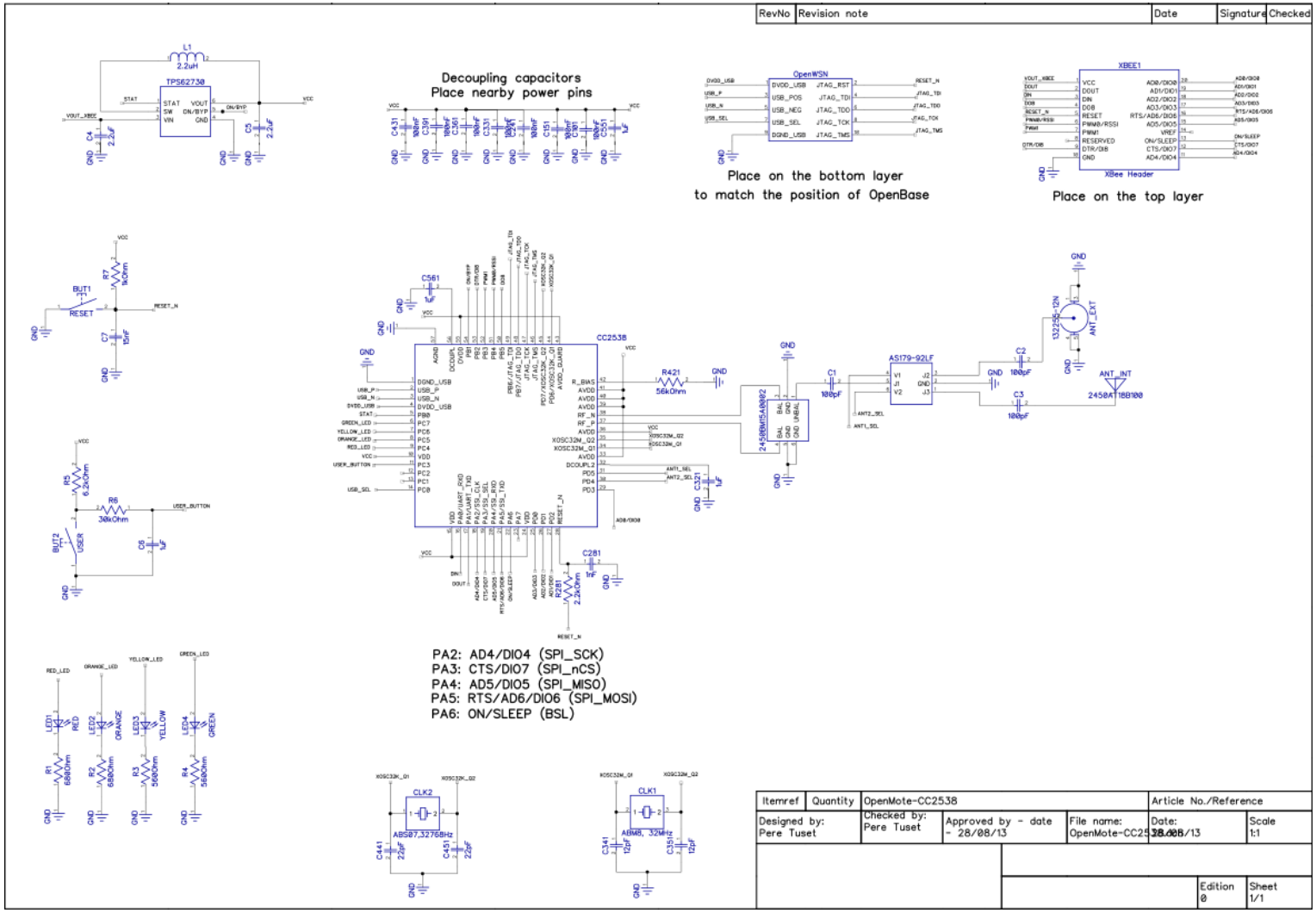


Ilustración 33: Diagrama Esquemático de Conexiones de la mota OpenMote-CC2538

5.3. ELEMENTOS RELACIONADOS

En los siguientes apartados se proporcionará una breve descripción de otros componentes y elementos adicionales relacionados con la mota OpenMote-CC2538, empleados para complementar o ampliar las capacidades y funcionalidades ofrecidas por ella.

5.3.1. OPENBASE

OpenBase es una placa interfaz para la mota OpenMote-CC2538. Para ello, la mota OpenMote-CC3538 es conectada a la placa *OpenBase* a través de una cabecera XBee, disponiendo de una sonda para la medición de la corriente consumida por la mota en sus distintos estados.

En la siguiente ilustración se muestra el aspecto de una placa *OpenBase*.

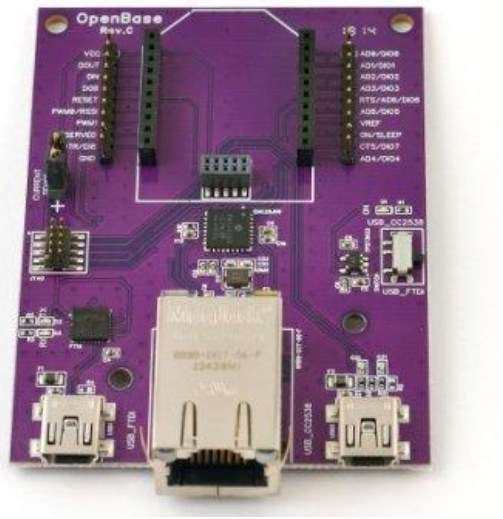


Ilustración 34: Aspecto Visual de la placa *OpenBase*

La placa *OpenBase* dispone principalmente de tres propósitos. Primero, posibilita la programación y depuración de código con soporte completo de puntos de ruptura, utilizando para ello una sonda con el conector estándar ARM JTAG de 10 pines (sonda de depuración *Segger J-Link*). Segundo, permite la comunicación con un ordenador personal a través del puerto serie o USB para su programación y depuración. El puerto serie, el cual es controlado a través del chip FTDI FT232R, permite tanto la programación a través de BSL como la depuración utilizando la operación de lenguaje C "*printf*". El puerto USB, soportado nativamente por el chip CC2538 de Texas Instruments, solo permite la depuración a través de la operación de lenguaje C "*printf*". Tercero, permite la comunicación con una red Ethernet 10/100 Mbps, habilitando con ello que la conexión de la mota hacia Internet sin la

necesidad de intervención de ningún ordenador o servidor. En este caso, la conexión Ethetrnet está soportada a través del chip Microchip ENC28J60 y el conector estándar RJ-45 que incluye tanto la protección magnética como la protección del circuito.

5.3.2. OPENBATTERY

OpenBattery constituye una placa interfaz para la mota OpenMote-CC2538. Para ello, la mota OpenMote-CC3538 es conectada a la placa *OpenBattery* a través de una cabecera XBee, disponiendo de un soporte para dos baterías AAA desde las cuales se proporciona la energía suficiente para operar el dispositivo de manera autónoma.

La placa *OpenBattery* incluye un conmutador de encendido y apagado de la placa así como tres tipos de sensores digitales diferentes: un sensor de temperatura y humedad (SHT21), un sensor de aceleración (ADXL346) y un sensor de luz (MAX44099). Todos estos sensores se encuentran conectados a la mota OpenMote-CC2538 a través de un bus I2C.

En la siguiente ilustración se muestra el aspecto de una placa *OpenBattery*.

Ilustración 35: Aspecto Visual de la placa *OpenBattery*

6. IMPLEMENTACIÓN MAC-CSL (COORDINATED SAMPLED LISTENING)

6.1. DESCRIPCIÓN

Los siguientes apartados tienen como objetivo describir la parte práctica elaborada en el desarrollo del trabajo final de máster, bajo el cual se ha realizado una implementación del mecanismo de bajo consumo CSL (*Coordinated Sampled Listening*) sobre la implementación OpenWSN del estándar IEEE802.15.4e y motas de tipo OpenMote-CC2538.

En este sentido, los objetivos marcados para la realización de esta parte práctica podemos resumirlos a continuación en el siguiente listado:

- **Extender la implementación OpenWSN del estándar IEEE802.15.4e a través del desarrollo del mecanismo de bajo consumo CSL**. En este sentido y tal y como se verá en el siguiente apartado, se ha contemplado un objetivo alcanzable en cuanto al desarrollo dados los plazos disponibles en la elaboración del Trabajo Final de Máster, dejando para líneas futuras de evolución y desarrollo algunos aspectos no contemplados en el presente desarrollo.

  En este sentido, la colaboración en el desarrollo y extensión de la implementación de referencia del estándar 802.15.4e como es OpenWSN tiene el objetivo adicional de la **contribución, colaboración y participación al desarrollo del software libre** aumentando con ello su cobertura funcional y posibilitando el desarrollo, aplicabilidad y evolución futura.

- **Adquisición de conocimiento de la arquitectura de OpenWSN**, las distintas capas y protocolos implicados y el modo de funcionamiento interno en las distintas fases de comunicación, esto es, transmisión, recepción y estado latente o dormición.

- **Experiencia en el trabajo con sistemas empotrados**, expresada en el diseño, validación y optimización de los desarrollos realizados sobre OpenWSN para su funcionamiento en tiempo real sobre las motas OpenMote-CC2538 y en general sobre sistemas embebidos.

La metodología seguida para el desarrollo de los trabajos, descrita en el **apartado 1.4 - Enfoque y Método Seguido**, ha comprendido la realización de las siguientes fases las cuales han sido aplicadas de manera secuencial en la elaboración de esta parte práctica:

1. **Fundamentación Teórica y Conceptual**, dado que previo a la realización de la implementación, es necesario conocer en detalle cada uno de los sistemas implicados y los requerimientos y características funcionales del desarrollo a realizar, en este caso, la implementación de CSL. Esta acción ha requerido por tanto la formación relativa a la revisión de los distintos estándares aplicables (IEEE 802.15.4 y 802.15.4e), el conocimiento en cuanto a la estructura, componente, entorno de desarrollo y despliegue, código fuente y funcionamiento de OpenWSN, los requerimientos de depuración sobre motas OpenMote-CC2538, y la información de análisis pormenorizado acerca del funcionamiento y operación del modo de funcionamiento de bajo consumo CSL.
2. **Diseño de la Solución**, en cuya fase y con el conocimiento adquirido en la fase anterior, se ha elaborado un diseño de la solución a implementar, validando posteriormente su preparación y aplicando las modificaciones y correcciones pertinentes para la determinación del diseño definitivo.
3. **Codificación**, en cuya fase se ha realizado la codificación del diseño del mecanismo de bajo consumo CSL sobre la implementación de referencia OpenWSN y en lenguaje C. Para ello, se ha requerido previamente la instalación y configuración del entorno de desarrollo y despliegue empleando para ello el IDE Eclipse CDT.
4. **Pruebas Preliminares**, en cuya fase se ha definido la arquitectura, ámbito y plan de pruebas preliminar para la verificación y validación del modo de funcionamiento CSL implementado respecto al comportamiento esperado. Este plan de pruebas es el que posteriormente en fase de depuración y pruebas es utilizado para la obtención de los resultados finales y la validación del funcionamiento CSL.

6.2. ALCANCE CONTEMPLADO

Tal y como se ha comentado en el apartado anterior y como consecuencia de las limitaciones de tiempo en el desarrollo del trabajo final de máster, el **objetivo definido como alcanzable** para la implementación del mecanismo de bajo consumo CSL comprende la realización de los siguientes puntos:

- Implementación del **mecanismo de muestreo** (*sampling*) periódico en receptor para la detección de solicitudes de transmisión.

- Implementación del mecanismo de transmisión basado en el **envío de una secuencia previa como preámbulo a la transmisión de datos** (*wake-up sequence*).
- **Recepción de trama de datos y envío de trama de reconocimiento** (ACK) a la trama recibida, **una vez detectada una trama *wake-up*,** espera un tiempo igual al indicado en el elemento de información *RZ-Time y* verificado el destino de recepción.
- **Transmisión de trama de datos y recepción de trama de reconocimiento** (ACK) a la trama enviada, **tras el envío del preámbulo o secuencia de notificación** (*wake-up sequence*). En este caso, se ha implementado una transmisión de tipo no-sincronizada (*unsynchronized*) lo que implica una longitud de secuencia durante un tiempo igual al periodo CSL máximo (*macCSLMaxPeriod*).
- **Definición y análisis de trama multipropósito** (*multipurpose frame*) empleada para el envío de **tramas *wake-up***, generadas y enviadas por el transmisor como preámbulo y notificación al envío de la trama de información.
- **Definición y análisis de elemento de información *RZ Time*** empleado para la indicación del tiempo de espera restante hasta la transmisión de la trama de información. El receptor utilizará esta información posteriormente para determinar el tiempo de espera hasta la disponibilidad de los datos, apagando mientras tanto el transceptor de radio y reduciendo con ello el consumo global.
- **Transmisión broadcast**, la cual y una vez implementada la funcionalidad de transmisión no-sincronizada (*unsynchronized*), solamente afecta a la dirección destino la cual será en este caso 0xFFFF. A este respecto, cabe hacer la puntualización de que no se contempla en el alcance la implementación del elemento de información LE CSL IE por medio del cual el transmisor y receptor intercambian su fase y periodo CSL. El estándar indica a este respecto que este IE es opcional en tramas broadcast por lo que es importante precisar que este apartado no se encuentra actualmente cubierto en el desarrollo llevado a cabo.
- **Extensión y adaptación de la máquina de estados** (FSM) definida actualmente en OpenWSN para el mecanismo TSCH.
- **Definición y establecimiento de una arquitectura de pruebas** para la validación y verificación funcional del modo de funcionamiento CSL en transmisión y recepción.

En este sentido, los **aspectos relativos a la implementación del mecanismo CSL que han quedado fuera del alcance** contemplado en el desarrollo han sido los siguientes:

- **Transmisión sincronizada** (*synchronized)* en la cual la capa MAC conoce la fase y periodo CSL del dispositivo destino, implicando que la longitud de la secuencia *wake-up* sea solamente el tiempo de guarda respecto al desplazamiento del reloj en base al tiempo obtenido en la última actualización de la fase y periodo CSL desde el dispositivo destino.
- **Elemento de Información LE CSL IE** por medio del cual el transmisor y receptor se intercambian el valor de su fase y periodo CSL con el fin de mantener la sincronización entre ellos y reducir las duraciones de las secuencias de tramas *wake-up* (*wake-up sequence*), reduciendo con ello el consumo derivado de la interfaz radio.
- **Soporte para el envío y recepción de varias tramas simultáneas** (*frame pending bit*) por medio de la cual es posible secuenciar varias transmisiones o recepciones de tramas sin necesidad de iniciar nuevos ciclos de apagado y encendido de la interfaz radio, optimizando el tiempo global de transmisión y reduciendo los consumos derivados de encendido y apagado de la interfaz radio.
- **CSL sobre múltiples canales**, en cuyo caso el modo de funcionamiento CSL es extendido a varios canales definidos a través de la propiedad *macCSLChannelMask*. Este caso implica que la operación de muestreo (*sampling*) es realizada para cada canal, de menor a mayor en modalidad *round-robin*.

  De esta manera, para transmisión no-sincronizada, la transmisión CSL enviará una secuencia de tramas *wake-up* de longitud igual al producto entre el número de canales y el valor del atributo *macCSLPeriod* antes del envío de la trama de datos o información (*payload frame*).

  Para transmisión sincronizada, la transmisión CSL calculará el siguiente tiempo de muestreo de canal y el número de canal, y procederá al envío en el siguiente muestreo de canal sobre el canal correcto con una secuencia corta de tramas *wake-up*. En este caso, la fase CSL será la duración desde el instante actual hasta el siguiente muestreo del canal del primer canal seleccionado dentro de la máscara de canales definida por *macCSLChannelMask*.

Estos puntos no contemplados dentro del alcance han sido establecidos como líneas futuras de ampliación del desarrollo tal y como se describe en el **apartado 10 - Líneas Futuras**.

6.3. METODOLOGÍA DE DESARROLLO

La estrategia de desarrollo propuesta para la implementación del Trabajo Final de Master (TFM) se encuentra basada en una **combinación de dos de las metodologías más utilizadas** en el mundo para el desarrollo de proyectos:

- **Metodología en Cascada (*Waterfall*)**: La existencia de requisitos estáticos, unívocos y claramente definidos en el estándar hace más conveniente aplicar una metodología en cascada para el desarrollo. Esto implicará la realización secuencial de las fases de requerimientos, análisis, diseño, codificación y pruebas durante el proceso.
- **Metodología Ágil (Scrum)**: Dentro de la fase de codificación (y solo dentro de esta fase), se propone la utilización de metodología ágil, y más concretamente Scrum, con el objetivo de segmentar el trabajo de codificación en pequeños paquetes que puedan ser ejecutados en modo iterativo en plazos de dos semanas. Los motivos principales que sustentan esta propuesta están basados en la capacidad para proporcionar incrementos funcionales con periodicidad quincenal (dos semanas), siendo posible detectar tempranamente cualquier desviación o problema durante la fase de desarrollo, así como ser capaces de priorizar los aspectos funcionales más relevantes del desarrollo.

En las siguientes secciones se describirán brevemente las características principales de ambas metodologías.

6.3.1. METODOLOGÍA EN CASCADA

Esta metodología, definida originalmente por Winston W. Royce en 1970, se ganó rápidamente el apoyo de los gerentes dado que todo fluye lógicamente desde el principio del proyecto hasta el final.

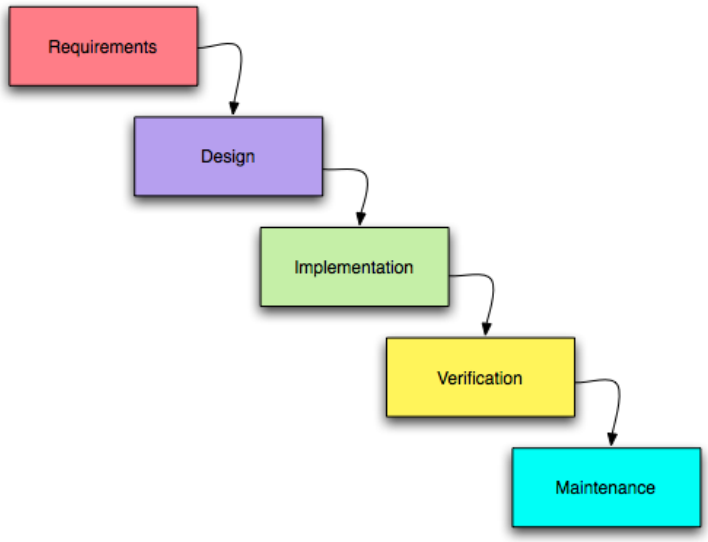


Ilustración 36: Metodología en Cascada (Fases)

Las fases de la metodología incluyen Requerimientos, Diseño, Implementación, Verificación y Mantenimiento. La metodología en cascada se basa en la suposición de que todos los requisitos pueden ser identificados durante la fase de Requerimientos, tal y como es el caso de la implementación del modo CSL.

Por este motivo, la comunicación con el usuario es fundamental en esta fase en la cual el jefe de proyecto debe hacer su mejor esfuerzo para obtener un conocimiento detallado de las necesidades del usuario y determinar el comportamiento y características esperadas del sistema. Una vez cumplida esta etapa, el proceso es realizado secuencialmente de manera natural.

La fase de Diseño por su parte puede describirse mejor a través de las sub-fases Diseño Lógico y Diseño Físico. Durante la fase de Diseño Lógico, el analista hace uso de la información recopilada en la fase Requerimientos para diseñar el sistema de forma independiente a cualquier sistema hardware o software. Una vez que el Diseño Lógico de alto nivel se ha completado, el analista comienza su transformación en un Diseño Físico el cual dependerá de las especificaciones tecnológicas específicas de hardware y software.

La fase de Implementación tiene como cometido llevar a cabo la codificación del sistema. En la metodología en cascada, esta fase es responsabilidad del programador, tomando los requerimientos del proyecto y sus especificaciones de diseño, y generando el código fuente de las aplicaciones.

La Fase de Verificación asegura que el sistema cumple con las expectativas iniciales. Sin embargo, en el análisis-diseño del mundo real y salvo en aplicaciones no críticas, esta etapa

es a menudo ignorada de tal modo que el sistema es puesto en marcha, comenzando directamente la fase de mantenimiento.

Durante la Fase de Mantenimiento, el usuario hace uso del sistema desarrollado. La aparición de problemas a consecuencia de requisitos mal interpretados o errores en el proceso de diseño implica la realización de correcciones en el sistema durante esta fase.

Dado que los requisitos CSL son estáticos y unívocamente definidos en el estándar, la metodología en cascada se determina como la mejor opción para ser utilizada en el desarrollo del Trabajo Final de Master (TFM), aportando con ello una serie de ventajas:

- Los errores de diseño son detectados antes de realizar ninguna codificación lo cual supone un ahorro de tiempo durante la fase de implementación.
- La documentación técnica generada es numerosa, precisa y detallada lo cual simplifica el proceso de aprendizaje por parte de nuevos programadores dentro de la fase de mantenimiento.
- El enfoque está claramente estructurado, siendo más fácil establecer medidas de progreso mediante el establecimiento de hitos parciales claramente definidos.
- El coste total del proyecto puede estimarse con precisión una vez que se han definido los requisitos del sistema (a través de las especificaciones de la interfaz de usuario y funcionales).
- La realización de las pruebas es más sencilla, pudiendo realizarse como referencia a los escenarios y casos de uso definidos en la especificación funcional.

6.3.2. METODOLOGÍA ÁGIL

Las metodologías ágiles permiten una aproximación al desarrollo software basado en la colaboración entre los diferentes roles involucrados en la implementación, más allá de las relaciones contractuales definidas en la documentación del proyecto.

Esto permite un conocimiento completo de la plataforma por parte de los equipos técnicos así como la re-priorización y re-definición de los requerimientos funcionales en cualquier instante.

En nuestro caso concreto, los requisitos funcionales de CSL están bien definidos y no se verán alterados durante el proyecto. Sin embargo, se propone un **enfoque basado en la flexibilidad en la priorización de requisitos CSL y en el desarrollo incremental, basado en la metodología Scrum y un proceso de desarrollo iterativo**, con el fin de **detectar**

cualquier problema o desviación respecto al plan original así como proporcionar incrementos funcionales en el desarrollo:

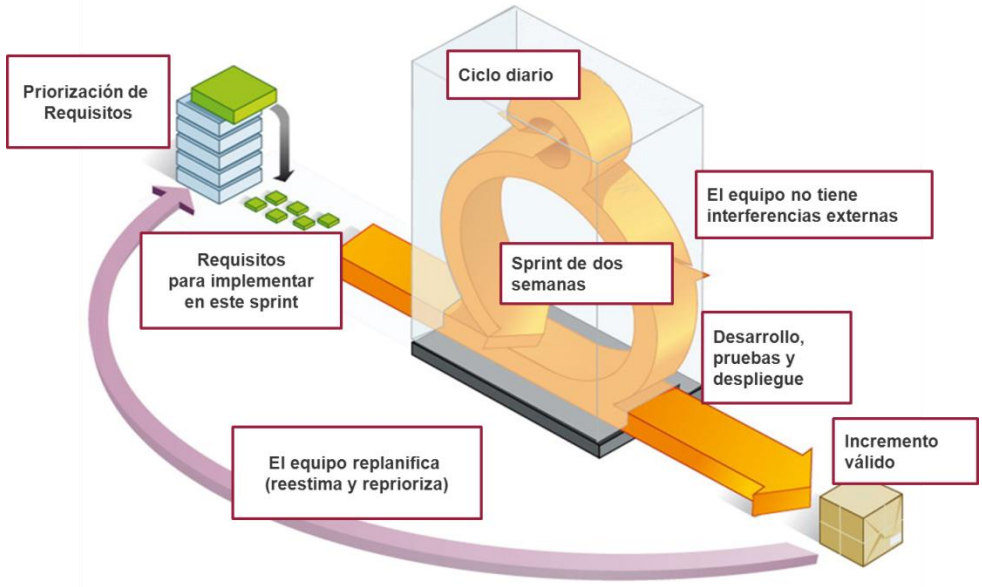


Ilustración 37: Trabajo Basado en Iteraciones

Las metodologías ágiles se apoyan en los siguientes conceptos:

- **Desarrollo iterativo e incremental**: pequeñas mejoras, unas tras otras.
- **Pruebas unitarias continuas**, frecuentemente repetidas y automatizadas, incluyendo pruebas de regresión. En este sentido, es aconsejable escribir el código de la prueba antes de la propia codificación del método.
- **Frecuente integración del equipo de programación con el usuario**. Hacer entregas frecuentes que permitan monitorizar y realizar el seguimiento de la evolución del trabajo.
- **Refactorización del código**, es decir, reescribir ciertas partes del código para aumentar su legibilidad y mantenibilidad, pero sin modificar su comportamiento. Las pruebas han de garantizar que en la refactorización no se ha introducido ningún fallo.
- **Simplicidad en el código**: es la mejor manera de que las cosas funcionen. Cuando todo funcione se podrá añadir funcionalidad si es necesario. La programación extrema apuesta que es más sencillo hacer algo simple y tener un poco de trabajo extra para cambiarlo si se requiere, que realizar algo complicado y quizás nunca utilizarlo.

Las principales herramientas en las que se basará el seguimiento de la evolución de la codificación de CSL son:

- **Product Backlog** (o conjunto de requisitos definidos y priorizados en cada iteración y estimados en complejidad y esfuerzo): Tras cada entrega parcial se hace una revisión de este conjunto de requisitos para ser re-estimados y re-priorizados, hasta el punto de que algunos requisitos pueden descartarse o incluirse en el alcance. Como resultados, algunos requerimientos pudieran ser incluidos o excluidos respecto al alcance CSL inicial.
- **Entregas Frecuentes**: Una entrega es una versión del producto que contiene una serie de funcionalidades identificadas y planificadas a priori totalmente terminadas. Se propone que las entregas se realicen de forma periódica durante la fase de codificación en intervalos de dos (2) semanas denominados *sprints*.
- **Iteraciones Evolutivas**: Cada entrega estará compuesta por un número variable de iteraciones cortas de entre una y dos semanas de duración. Cada iteración producirá una versión del desarrollo CSL con nueva funcionalidad y/o mejoras técnicas y estará disponible para la revisión en el entorno de desarrollo.

Tras cada entrega se realizará una valoración de la evolución en la que se determinará:

- La funcionalidad que debe ser implementada para la siguiente entrega. Para ello, no sólo se re-priorizará el "*product backlog*", sino que se evaluará la conveniencia de eliminar o añadir requisitos a propuesta del equipo.
- Los cambios y correcciones que deben realizarse respecto a las funcionalidades implementadas en la entrega anterior.
- Problemas detectados en esta iteración y cómo corregirlos en la siguiente.

6.4. ANÁLISIS Y DISEÑO DE LA SOLUCIÓN

La fase de análisis para la implementación del mecanismo de bajo consumo CSL ha implicado la identificación de las distintas líneas de actuación y funcionalidades dentro del código fuente de OpenWSN.

Como resultado de esta fase de análisis, ha sido posible identificar aquellos elementos sobre los cuales aplicar el diseño de la solución, expresado como extensiones y modificaciones necesarias para el cumplimiento de los requisitos, determinando en cada caso el mejor modo de actuación para las posteriores evoluciones y mantenimientos del código desarrollado.

A continuación, en los siguientes apartados, se procederá a describir cada uno de estos elementos, funcionalidades y puntos de acción requeridos para la implementación de CSL sobre OpenWSN, detallando en cada caso el diseño realizado sobre cada uno de ellos.

6.4.1. IDENTIFICACIÓN Y ORGANIZACIÓN DE CÓDIGO CAPA MAC

La primera cuestión a resolver en la fase de análisis ha sido la identificación del código fuente de la capa MAC dentro de la organización y estructura en OpenWSN en base al cual determinar posteriormente el modo más adecuado de proceder para el desarrollo de las modificaciones y extensiones requeridas para CSL.

En este sentido, el proyecto OpenWSN proporciona la configuración para su inclusión dentro del entorno de desarrollo Eclipse IDE desde el cual es posible compilar, construir y desplegar el firmware con los cambios realizados. El procedimiento para la importación, compilación y despliegue del proyecto OpenWSN puede ser consultado en el **apartado 7.2 - Procedimiento de Compilación y Despliegue**.

A continuación se recoge el listado de los **principales ficheros y recursos de código fuente identificados durante la fase de análisis así como el enfoque de diseño aplicado** para cada uno de ellos.

6.4.1.1 IEEE802154E.c / IEEE802154E.h

Estos ficheros se encuentran ubicados bajo la carpeta “02a-MAC” y representa el nivel inferior de la capa MAC en el cual se encuentra realizada la implementación del modo TSCH de IEEE802.15.4e así como los temporizadores asociados.

La funcionalidad de este nivel hace uso de la cola de paquetes a enviar (módulo *OpenQueue*) así como del planificador (*Schedule*). Su funcionamiento es enteramente en modo interrupción (ISR).

La funcionalidad de este módulo está basada en una estructura de slots que se repiten en el tiempo y sobre la cual se realizan las acciones de envío, recepción o dormición en función del planificador, así como la mota remota con la cual realizar la comunicación y el desplazamiento a realizar sobre el canal.

En este sentido, el diseño realizado sobre este módulo implica la realización de las siguientes acciones:

- **Renombrado de los ficheros del módulo a IEEE802154Ecsl.c y IEEE802154Ecsl.h** con el fin de indicar claramente la funcionalidad contenida y separar su implementación de la realizada para TSCH.
- **Modificación de referencias a la cabecera del módulo antiguo** con el fin de validar que siempre se está incluyendo la definición del nuevo módulo en lugar del módulo original con la implementación TSCH.
- **Supresión del modo de funcionamiento TSCH**. Esta operación será realizada a través de la definición de la nueva máquina de estados para CSL, sustituyendo la actualmente definida para TSCH. Esto implicará adicionalmente la modificación de algunos otros elementos necesarios tales como funciones auxiliares, definiciones de constantes y tipos, y código asociado a temporizadores e interrupciones.
- **Modificación y adaptación de las funciones *callback*** definidas para cada tipo de interrupción.
- **Soporte de tramas *Wake-Up*** empleadas para el aviso de un próximo envío de datos y minimizar con ello el consumo de energía de la radio en el receptor. En este sentido, se ha optado por definir dos métodos nuevos dedicados a la construcción y análisis de las tramas *wake-up*, evitando extender la implementación de análisis y creación de cabeceras estándar IEEE802.15.4 con el fin de independizar el comportamiento añadido así como simplificar su tarea de administración y mantenimiento futuro.
- **Definición de atributos CSL** encargados de definir el comportamiento global de CSL. A este respecto, se definirán solamente los atributos *macCSLPeriod* y *macCSLMaxPeriod* necesarios para determinar la frecuencia del muestreo para detección de transmisiones de mensajes y la duración de la secuencia de tramas *wake-up* en el proceso de comunicación en transmisión CSL no sincronizada. Para más información acerca de la definición de atributos CSL, consultar el **apartado 6.4.3 - Definición y Valores de los Atributos CSL**.
- **Adaptación y extensión de la máquina de estados** definida para su adecuación al modo de comportamiento de CSL. Para más información acerca de la máquina de estados CSL, consultar el **apartado 6.4.6 - Maquina de Estados CSL.**

6.4.1.2 IEEE802154.c / IEEE802154.h

El diseño contempla la **ampliación de la definición de los tipos de trama** (*frame type*) definidos en el sistema, necesarios para la correcta construcción y análisis de las tramas *wake-up*.

De este modo, es necesario **extender el tipo enumerado IEEE802154_fcf_type_enums** para que contemple los nuevos tipos LLDN y MULTIPURPOSE, redefiniendo además el valor asignado al tipo UNDEFINED. De este modo, los valores asignados al tipo enumerado quedarán de la siguiente manera:

- IEEE154_TYPE_BEACON = 0
- IEEE154_TYPE_DATA = 1
- IEEE154_TYPE_ACK = 2
- IEEE154_TYPE_CMD = 3
- IEEE154_TYPE_LLDN = 4
- IEEE154_TYPE_MULTIPURPOSE = 5
- IEEE154_TYPE_UNDEFINED = 6

El tipo IEEE154_TYPE_LLDN ha sido definido aunque no será utilizado dentro del marco del actual desarrollo, siendo definido simplemente con objeto de actualizar y contemplar la totalidad de tipos de trama actualmente definidos en el estándar.

Por su parte, el tipo IEEE154_TYPE_MULTIPURPOSE se corresponde al tipo de trama multipropósito definido como valor 101 en binario (5 en digital) para el campo *frame type* del elemento *frame control* de las tramas MAC.

Dado que las tramas *wake-up* son tramas de tipo multipropósito, este será el valor de campo *frame type* que tendrán asociado en la cabecera del mensaje.

6.4.1.3 Opendefs.h

Este fichero se trata de un fichero trasversal a todos los niveles y capas de la implementación de OpenWSN en el cual se establecen definiciones de variables, constantes y tipos generales.

El diseño contempla la realización de las siguientes acciones sobre este módulo:

- **Extensión de los códigos de error y mensajes asociados** con el fin de definir los correspondientes a la implementación CSL. Los nuevos códigos de error así como los mensajes asociados a cada uno de ellos son los siguientes:
    - ERR_WRONG_STATE_IN_START_CSL_SAMPLING
        - wrong state {0} in start CSL channel sampling, at CSL sample {1}
    - ERR_WRONG_STATE_IN_CSL_TIMERFIRE
        - wrong state {0} in timer fires, at CSL sample {1}
    - ERR_MAXRXWAKEUPPREPARE_OVERFLOWS

        - maxRxDataPrepare overflows while at state {0} in CSL sample {1}
    - ERR_WDWAKEUPDURATION_OVERFLOWS
        - wdDataDuration overflows while at state {0} in CSL sample {1}
    - ERR_WRONG_STATE_IN_CSL_SAMPLE
        - wrong state {0} in start of frame on CSL sampling at CSL sample {1}
    - ERR_WRONG_STATE_IN_CSL_ENDOFFRAME:
        - wrong state {0} in end of frame, at CSL sample {1} {0}
    - ERR_MAC_OPERATION_IN_PROGRESS
        - MAC Operation {0} still in progress in CSL sample {1}. Abort actions and wait to finish.

6.4.2. MODOS DE TRABAJO CSL

El diseño considera la definición de dos modos de trabajo de CSL:

- **Modo Transmisión**, en el cual se procederá al envío de una trama de datos a un nodo remoto. En este caso y aunque no es una cuestión propiamente de CSL, se hará uso del sistema de planificación (*schedule*) definido en OpenWSN para el establecimiento de una asociación y control de relaciones entre nodos vecinos (direcciones vecinas) y datos pendientes de envío en la cola de salida.
- **Modo Recepción**, en el cual se llevará a cabo el proceso periódico de escucha de avisos de envío y la posterior recepción de los mensajes.

Ambos modos de comportamiento serán excluyentes entre sí en el desarrollo realizado lo cual implicará que cuando el sistema se encuentre trabajando en uno de estos modos, los eventos producidos en el otro modo serán ignorados hasta la finalización del modo en curso.

6.4.3. DEFINICIÓN Y VALORES DE LOS ATRIBUTOS CSL

Los atributos CSL definen el modo de funcionamiento CSL respecto a su comportamiento y nivel de actuación tal y como se describe en el **apartado 3.2 - Atributos CSL**.

A este respecto, dentro del alcance de la implementación CSL realizada en el trabajo, se considerará únicamente la definición de los siguientes atributos:

- ***macCSLPeriod***, el cual definirá el periodo de muestreo de canales y escucha inactiva.
- ***macCSLMaxPeriod***, el cual definirá Periodo máximo de muestreo de canales y escucha inactiva, así como la longitud de la secuencia de tramas *wake-up* (*wake-up sequence*) en transmisiones no sincronizadas.

Estas variables serán definidas en el fichero **IEEE802154Ecsl.h** referenciando su valor a variables definidas en el fichero **board_info.h** definido para cada tipo de mota con el fin de generalizar el desarrollo realizado en este módulo y particularizar el comportamiento a nivel de las motas concretas de acuerdo a sus características y especificaciones.

Respecto a los valores seleccionados para cada una de ellas, el criterio aplicado ha estado basado en las frecuencias de muestreo empleadas en otras implementaciones similares como el mecanismo ContikiMAC de la implementación IEEE 802.15.4 Contiki, en la cual se consideran periodos de muestreo comprendidos entre 5 y 400 milisegundos, en función de las características de cada plataforma hardware y los requerimientos a nivel de aplicación.

En este sentido, se ha aplicado un criterio intermedio, determinando **200 milisegundos como tiempo de muestreo CSL** aplicable, el cual podrá ser fácilmente ajustado a través de la modificación de esta variable.

Por otro lado, OpenWSN hace uso internamente de contadores de tiempo basados en *ticks* para determinar la generación de las distintas interrupciones en el sistema y los cambios de estado, para lo cual será preciso determinar la correspondencia entre dicho periodo de muestreo y su correspondencia en *ticks.*

Para ello, debemos partir primeramente del tiempo de duración de un *tick* sabiendo que la mota OpenMote-CC2538 trabaja a una frecuencia de 32 KHz o lo que es lo mismo 32.768 Hercios (32 * 1024 Hz). De este modo, el tiempo de un *tick* será el siguiente:

$$tick = \frac{1}{32768\ Hz} = 30{,}52\ \mu s$$

Por tanto, el número de *ticks* asociados a un intervalo de 50 milisegundos será:

$$\boldsymbol{macCSLPeriod\ (ticks)} = \frac{200 \cdot 10^{-3}\ segs}{30{,}52 \cdot 10^{-6} segs} = 6553{,}07 \approx \mathbf{6554}$$

Respecto al valor del atributo ***macCSLMaxPeriod***, se configurará a su valor por defecto el cual coincidirá de acuerdo al estándar con el mismo valor que el atributo ***macCSLPeriod***.

6.4.4. DEFINICIÓN DE TRAMA WAKE-UP

La definición, estructura y campos de la trama *wake-up* puede ser consultada en el **apartado 3.3.2 - Trama Wake-Up Multipropósito (LE-Multipurpose Wake-Up)**.

La trama *wake-up* es una trama de tipo multipropósito (*multipurpose*) empleada para avisar al nodo receptor el próximo envío de una trama de datos o información adjuntando para ello el elemento de información (IE) *rendezvous time* (*RZ Time*) con el tiempo restante hasta el envío de dicha trama de datos.

Se muestra a continuación en la siguiente ilustración la estructura y campos de la trama *wake-up*:

Octets: 1	1	2	2	4	2
Frame Control	Sequence Number	Dest. PAN ID	Dest. Address	RZ Time Header IE	IE List Terminator

Ilustración 38: Estructura de la Trama *Wake-Up*

Tal y como se puede observar en la ilustración anterior, la estructura de la trama *wake-up* se compone de los siguientes elementos:

- ***Frame Control*** (1 byte): La estructura de este campo en las tramas multipropósito es la siguiente:

Bit: 0–2	3	4–5	6–7	8	9	10	11	12-13	14	15
Frame Type	Long Frame Control	Destination Address Mode	Source Address Mode	PAN ID Present	Security Enabled	Seq Number Suppression	Frame Pending	Frame Version	Ack Request	IEs List Present

Ilustración 39: Estructura FCF de Tramas Mutipropósito (*multipurpose frame*)

Para el caso de la trama *wake*-up, el campo FCF es de 1 byte (controlado mediante el campo *Long Frame Control*), presentando por tanto un valor para este campo de 0x55 o 01010101 en binario, obtenido a partir del análisis del primer byte de la estructura anterior:

- **Bits b0-b2 *(Frame Type)***: Para trama de tipo *wake-up* debe tener un valor 101b (5 en digital).
- **Bit b3 *(Long Frame Control)***: En este caso, debe ser valor 0 para indicar que se trata de un FCF de 1 byte tal y como es el caso de tramas *wake-up*).

- **Bits b4-b5 (*Dest Addr Mode*)**: Debe ser 10 (binario) para indicar que la dirección del destino en trama *wake-up* es una dirección corta (*short addr)* tal y como se indica en el estándar y se recoge en la siguiente ilustración.

Dst Addr Mode b_5 b_4	Description
00	No Destination Address in MHR
01	1-octet Destination Address
01	2-octet Destination Address
11	8-octet Destination Address

Ilustración 40: Valores de *Dst Addr Mode* en Tramas Mutipropósito (*multipurpose frame)*

- **Bits b6-b7 (Src Addr Mode)**: Debe ser 10 (binario) para indicar que la dirección del origen en trama *wake-up* es una dirección corta (*short addr).* En este caso, el listado de valores para el campo anterior aplica igualmente para este campo.

- ***Sequence Number*** (1 byte): Este campo se corresponderá con el valor del DSN del nivel MAC.
- ***PAN Identifier*** (2 bytes): Este campo se corresponderá con el identificador de la red.
- ***Destination Address*** (2 bytes): Este campo se corresponderá con la dirección corta del dispositivo destino del mensaje (*short address*) o en el caso de mensajes *broadcast* un valor igual a 0xffff en hexadecimal.
- ***RZ Time IE*** (2 bytes cabecera y 2 bytes de carga): Este campo contendrá el valor del elemento de información *RZ Time* encargado de informar del tiempo pendiente en el transmisor hasta el comienzo del envío de la transmisión de datos. Este tiempo será utilizado en los dispositivos receptores para determinar el tiempo de dormición y por tanto, de desactivación del interfaz radio, reduciendo con ello el consumo de energía. La estructura y contenido de este elemento puede consultarse en el **apartado 6.4.4.1 - Definición del Elemento de Información RZ-Time (RendezVous Time)**.
- ***IE List Terminator*** (2 bytes): En este caso, la trama debe finalizar con un elemento de información de finalización denominado *IE List Terminator* con identificador (ID) igual a 0x7e o 0x7f y con una longitud de contenido igual a cero. La terminación es necesaria cuando tras un elemento de información de cabecera (*Header IE*) haya uno o más elementos de información en la carga (*Payload IE)* en cuyo caso el ID

será 0x7e, o bien carga de nivel MAC (*MAC Payload*) en cuyo caso el valor será 0x7f. En cualquier otro caso, este campo puede omitirse.

En nuestro caso y desde un punto de diseño, se considerarán como opciones válidas dentro de la trama wake-up estas tres opciones (IDs 0x7e, 0x7f o bien, omisión del campo).

En este sentido y en los casos considerados, la estructura del elemento estará formada por 2 bytes de acuerdo a la estructura dada para los elementos de información de cabecera (*Header IE*):

Bit: 0-6	7–14	15	Octets: 0 … 127
Length	Element ID	Type = 0	IE Content

Ilustración 41: Estructura de elemento de información *Header IE* (IE List Terminator IE)

- **Bits b0-b6 (*Length*)**: Debe ser igual a cero en el caso de este elemento. Esto implica que no tiene contenido asociado (*IE Content*).
- **Bits b7-b14 (*Element* ID)**: Debe ser igual a 0x7e o 0x7f en el caso de este elemento, en el caso de estar presentes.
- **Bits b15 (*Type*)**: Debe ser igual a cero en el caso de este elemento.

6.4.4.1 Definición del Elemento de Información RZ-Time (RendezVous Time)

La estructura del elemento de información *RZ-Time* sigue el mismo patrón que la estructura de un elemento de información de cabecera (*Header IE*) indicado en el apartado anterior.

En este sentido, los valores asociados a cada uno de los campos serán los siguientes:

- **Bits b0-b6 (*Length*)**: En este caso la longitud del campo de contenido (*IE Content*) debe ser igual a dos (numero de bytes de la carga) en el cual se proporcionará el tiempo restante hasta el inicio de la transmisión de la información.
- **Bits b7-b14 (*Element ID*)**: Para este elemento de información, el valor de este campo debe ser igual a 0x1D.
- **Bits b15 (*Type*)**: Debe ser igual a cero en el caso de este elemento.
- **Bits b16-b31 (*IE Content*)**: Tiempo restante hasta el inicio de la transmisión, utilizado en el receptor para determinar el tiempo de espera y desactivación de la radio hasta el comienzo de la recepción de los datos (en caso de ser el destinatario) o el inicio de nuevo del periodo de muestreo (en caso de no ser el destinatario).

6.4.5. INTERRUPCIONES Y TEMPORIZACIONES

6.4.5.1 Arquitectura Actual en OpenWSN

Antes de entrar en detalle acerca de los aspectos de diseño asociados a las interrupciones y temporizaciones consideradas, resulta necesario introducir unos conceptos generales. En este sentido, la temporización hardware en un entorno microcontrolador consiste básicamente en los siguientes elementos:

- **Fuente de reloj** (*clock source*) que genera a su salida una onda cuadrada con periodo constante. Típicamente esta fuente de reloj es externa al microcontrolador, estando normalmente formada por un cristal para mejor precisión. El flanco de subida de la señal generada es lo que normalmente denotamos como "*tick*".
- **Contador** (*counter*) el cual se incrementa con cada nuevo "*tick*".
- **Registro de comparación** (*compare registers*) cuyo valor es establecido por la aplicación de tal manera que en cada incremento del contador, la lógica de temporización compara el nuevo valor con el valor almacenado en el registro. En el caso que el valor coincida, el temporizador generará una interrupción para su procesamiento asociado, siendo por este motivo utilizados como eventos "despertador".
- **Registro de captura** (*capture registers*) los cuales copian el valor actual del contador a través de disparadores (*triggers*). Normalmente, la aplicación configura el temporizador de tal manera que los registros de captura registran el tiempo cuando se produce una señal o conmutación en algún pin externo.

En el caso de OpenWSN, la temporización anterior se encuentra por defecto implementada de la siguiente manera:

- **Fuente de reloj** (*clock source*) procede de un cristal externo de 32 KHz lo cual supone un *tick* de duración igual a 30,52 microsegundos.
- **Contador** (*counter*) el cual está configurado para realizar una cuenta ascendente entre 0 y 492 tras lo cual genera una interrupción al microcontrolador, volviendo de nuevo a valor cero. Esta duración se corresponde a un tiempo real de 15 ms (492 * 30,52 us) empleado para la definición de la duración de un *slot (TsSlotDuration*).
- **Registro de comparación** (*compare registers*) el cual es utilizado para temporizar las diferentes duraciones asociadas a los distintos estados de la máquina de estados. De este modo, cada vez que el contador alcanza el valor del registro de comparación, se ejecuta otra de las actividades de la máquina de estados.

- **Registro de captura** (*capture registers*) los cuales son empleados para el registro del tiempo de inicio y fin de cada paquete enviado y recibido.

A este respecto, los eventos producidos por el cumplimiento del contador así como por los registros de comparación tienen asociada una función denominada ***callback*** la cual será invocada en la generación de la interrupción (ISR).

6.4.5.2 Modificaciones CSL

Los tipos de eventos así como las funciones de *callback* asociadas que serán utilizadas en la implementación del modo CSL serán las siguientes:

- **Evento Contador** (*counter*). Actualmente, OpenWSN está configurado para generar el evento cada *slot* de acuerdo al particionado del tiempo realizado en TSCH. En nuestro caso, modificaremos esta configuración para que realice la cuenta durante un tiempo igual a la frecuencia de muestreo CSL (atributo *macCSLPeriod*) tal y como se describe en el **apartado 6.4.3 - Definición y Valores de los Atributos CSL**.
  Esto permitirá invocar regularmente (cada *macCSLPeriod*) a la función de *callback* ***isr_ieee154ecsl_newChannelSample*** para establecer el comportamiento de escucha periódica e iniciar la máquina de estados finitos asociada al proceso de recepción (ver **apartado 6.4.6.1 - Maquina de Estados CSL en Modo Recepción**).
- **Evento Comparación** (*compare registers*) el cual permitirá temporizar las diferentes duraciones asociadas a los distintos estados de la máquina de estados. Para ello, se invocará a la función ***isr_ieee154ecsl_timer*** desde donde se determinará la siguiente acción a realizar en función del estado actual, el vencimiento producido y las expectativas funcionales realizadas o no realizadas durante el estado anterior.

Adicionalmente a los eventos anteriores, existen dos eventos de interrupción asociados a los siguientes acontecimientos:

- **Evento de Inicio de Trama** (*start of frame*) el cual es generado por el interfaz radio cuando se detecta señal (SFD) en el medio. Esto permite invocar a la función *callback* ***ieee154ecsl_startOfFrame*** en la cual y en función del estado actual de la máquina de estados, se procederá a la realización de las tareas asociadas.
- **Evento de Fin de Trama** (*end of frame*) el cual es igualmente generado por el interfaz radio cuando se detecta el envío del fin de la trama al medio. Esto permite invocar a la función *callback* ***ieee154ecsl_endOfFrame*** donde de nuevo y en función

del estado actual de la máquina de estados, se realizarán las tareas correspondientes.

6.4.5.3 Temporizador de Transmisión CSL

Adicionalmente a los tipos de temporizadores e interrupciones definidas en OpenWSN a nivel MAC, se considera necesario definir un nuevo **temporizador destinado a verificar periódicamente la existencia de datos pendientes de envío** siguiendo para ello una planificación establecida (*schedule*).

Tal y como se ha comentado anteriormente, el planificador (*schedule*) definido en OpenWSN para TSCH no es propiamente un elemento necesario en CSL dado el carácter asíncrono presentado por este modo.

Sin embargo y a pesar de no ser requerido, **se ha optado por su aprovechamiento de utilización dentro del esquema de transmisión** por los siguientes motivos:

- **El planificador actual permite la vinculación de cada celda con una dirección MAC de otra mota lo cual permite simplificar el proceso de comunicación**. En cada ejecución del temporizador de transmisión (función de *callback*), es preciso verificar la existencia de datos pendientes de envío para alguno de los nodos remotos. Actualmente, el sistema de cola de mensajes no incluye para cada mensaje (*OpenQueueEntry*) la dirección MAC del nodo remoto, estableciéndose este vinculo actualmente a nivel del planificador. Con el fin de no extender esta estructura de mensaje de la cola, se ha optado por utilizar el planificador (*schedule*) como elemento vinculante entre la dirección MAC y la cola de mensajes.
- **La utilización del planificador evita la necesidad de implementar un sistema de gestión de prioridades** dado que cada celda del planificador estará asociada a un nodo remoto en el cual se determinará si hay datos pendientes de envío para dicho nodo. Es decir, un algoritmo por prioridades que permitiera determinar qué datos de la cola deben enviarse en cada invocación a la *callback* de verificación de datos pendientes de envío dado que podrían existir distintos paquetes pendientes en un instante dado.
- La cola será avanzada en cada invocación del temporizador lo cual conllevará que **el tiempo máximo que esperará una trama para ser enviada** será igual al (**periodo del temporizador del chequeo de transmisiones) x (el número de motas de la red)**.

Por tanto, en el caso de detectarse que la celda del planificador es de tipo transmisión y que hay datos pendientes de envío para el nodo remoto asociado a la celda, se conmutará el estado a modo transmisión para proceder a la transmisión de los datos.

La duración de este temporizador se establece con carácter periódico cada 60 milisegundos, optando por este valor por ser un número razonablemente inferior al valor establecido para la escucha periódica del canal (*macCSLPeriod*) pero superior al valor de duración de un slot previamente definido en OpenWSN y verificado como suficiente para la transmisión y recepción de los datos y su reconocimiento (ACK) asociado.

6.4.6. MAQUINA DE ESTADOS CSL

La máquina de estados CSL define el comportamiento del sistema mediante un número determinado de estados y un número determinado de transiciones entre dicho estados, donde las transiciones de un estado a otro se generan en respuesta a eventos de entrada externos e internos mientras que estas transiciones y/o subsecuentes estados pueden generar otros eventos de salida.

En este sentido, a continuación se presentará la máquina de estados de CSL, tanto en modo recepción como en modo transmisión.

6.4.6.1 Maquina de Estados CSL en Modo Recepción

La máquina de estados CSL en modo recepción hace referencia al comportamiento del sistema en las siguientes fases de operación:

1. **Fase de Escucha Activa** (*CSL idle listening)* en la cual el dispositivo escucha periódicamente la interfaz radio en espera de recepción de una trama *wake-up* indicando la próxima transmisión de una trama de datos. Los estados considerados en esta fase son los siguientes:
    - S_CSLRXWAKEUPOFFSET
        - Espera de recepción de trama *wake-up.*
    - S_CSLRXWAKEUPPREPARE
        - Preparando para recepción trama *wake-up*.
    - S_CSLRXWAKEUPREADY
        - Preparado para recibir trama *wake-up*, esperando señal inicio (“go”).
    - S_CSLRXWAKEUPLISTEN
        - Esperando la recepción de trama *wake-up.*
    - S_CSLRXWAKEUP

- Recepción de trama *wake-up.*

- S_CSLRXWAKEUPVALIDATE
  - Validación de la trama *wake-up* recibida.

2. **Fase de Recepción** (*data reception)* en la cual el dispositivo se prepara para la recepción de una trama de datos dirigida a él. Los estados considerados en esta fase son los siguientes:

   - S_CSLRXDATAOFFSET
     - Espera de recepción de trama de datos.
   - S_CSLRXDATAPREPARE
     - Preparando para recepción de trama de datos.
   - S_CSLRXDATAREADY
     - Preparado para recibir trama datos, esperando señal de inicio (“go”).
   - S_CSLRXDATALISTEN
     - Esperando la recepción de trama de datos.
   - S_CSLRXDATA
     - Esperando la recepción de trama de datos.

3. **Fase de Reconocimiento** (*ack sending)* en la cual el dispositivo confirma la recepción del mensaje al dispositivo origen en el supuesto que éste así lo haya requerido a través del campo de cabecera AR (*acknowledgment request*). Los estados considerados en esta fase son los siguientes:

   - S_CSLTXACKOFFSET
     - Espera para la transmisión de trama de reconocimiento ACK.
   - S_CSLTXACKPREPARE
     - Preparando para la transmision de trama de reconocimiento ACK.
   - S_CSLTXACKREADY
     - Preparado para transmitir trama ACK, esperando señal inicio (“go”).
   - S_CSLTXACKDELAY
     - Señal de inicio “go” completada, esperando SFD de trama ACK.
   - S_CSLTXACK
     - Detectado envío SFD de trama ACK, enviando resto de bytes.
   - S_CSLRXPROC
     - Detectado envío del fin de la trama ACK, procesando datos.

A continuación en la siguiente ilustración se recoge la máquina de estados CSL de modo recepción con sus transiciones y eventos, así como los métodos asociados para el tratamiento de acciones asociadas a cada uno de los estados.

Así mismo, se identifican cada una de las tres fases anteriores, representando **la fase de escucha activa** (color verde, lateral izquierdo), **la fase de recepción** (color azul, centro) y **la fase de reconocimiento** (color naranja, lateral derecho).

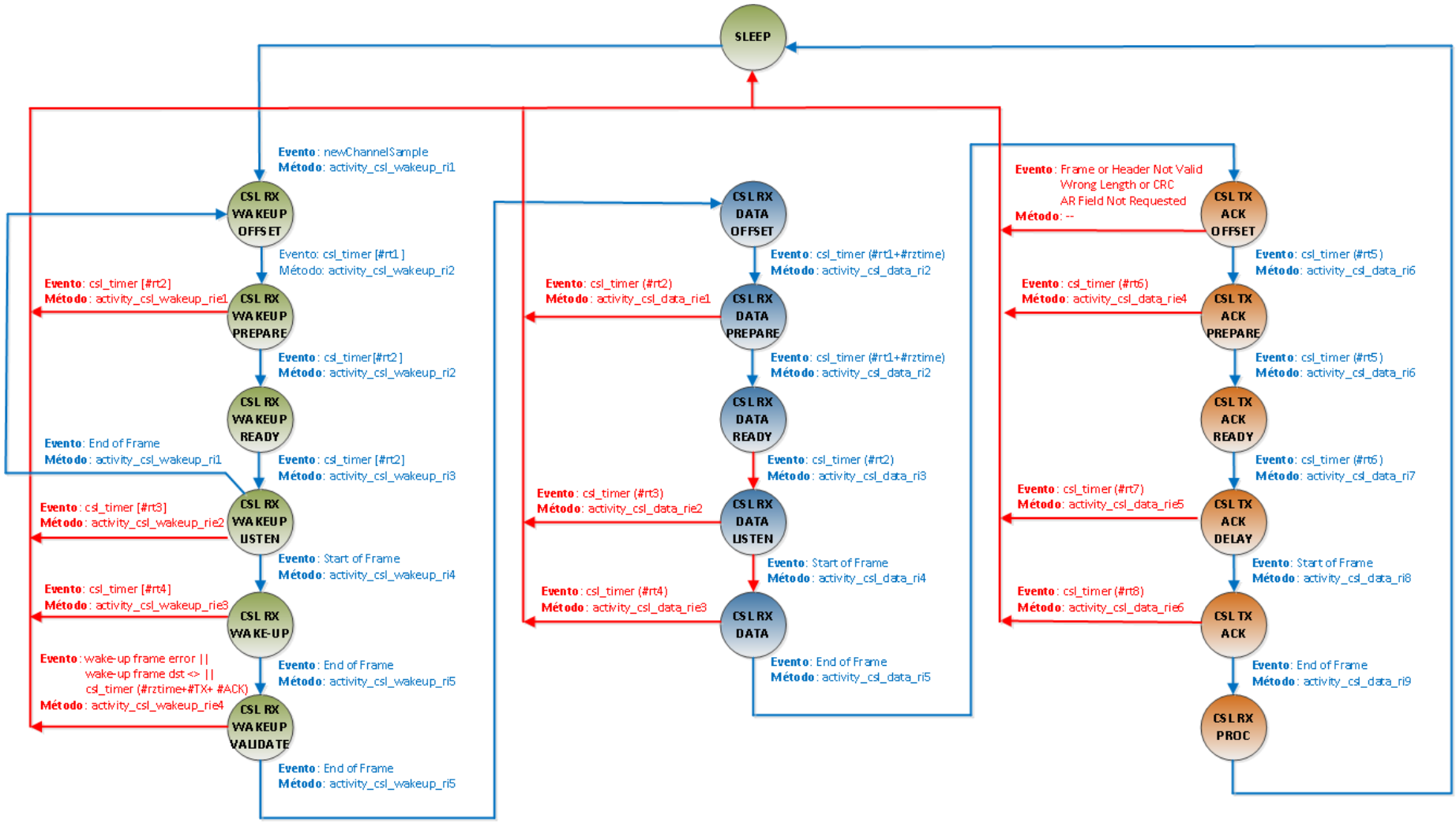


Ilustración 42: Máquina de Estados CSL en Modo Recepción

6.4.6.2 Maquina de Estados CSL en Modo Transmisión

La máquina de estados CSL en modo transmisión hace referencia al comportamiento del sistema en las siguientes fases de operación:

1. **Fase de Aviso de Próximo Envío** (*CSL wake-up sequence)* en la cual el dispositivo transmisor envía una secuencia de tramas de tipo *wake-up* previo al envío de la trama de datos para notificar al receptor el momento del envío y minimizar su consumo de radio. Los estados considerados en esta fase son los siguientes:
    - S_CSLTXWAKEUPOFFSET
        - Espera de transmisión de trama *wake-up.*
    - S_CSLTXWAKEUPPREPARE
        - Preparando para transmisión de trama *wake-up*.
    - S_CSLTXWAKEUPREADY
        - Preparado para enviar trama *wake-up*, esperando señal inicio (“go”).
    - S_CSLTXWAKEUPDELAY
        - Esperando la transmisión de trama *wake-up.*
    - S_CSLRXWAKEUP
        - Transmisión de trama *wake-up.*
2. **Fase de Transmisión** (*data sending)* en la cual el dispositivo envía la trama de datos al nodo destino. Los estados considerados en esta fase son los siguientes:
    - S_CSLTXDATAPREOFFSET
        i. Espera de tiempo restante hasta el comienzo del envío de la trama de datos en el caso que dicho tiempo restante sea inferior a la duración para la transmisión de una nueva trama wake-up.
    - S_CSLTXDATAOFFSET
        - Espera de transmisión de trama de datos.
    - S_CSLTXDATAPREPARE
        - Preparando para transmisión de trama de datos.
    - S_CSLTXDATAREADY
        - Preparado para enviar trama datos, esperando señal de inicio (“go”).
    - S_CSLTXDATADELAY
        - Esperando la transmisión de trama de datos.
    - S_CSLTXDATA
        - Transmisión de trama de datos.

3. **Fase de Reconocimiento** (*ack reception)* en la cual el dispositivo remoto confirma la recepción del mensaje al dispositivo origen. Los estados considerados en esta fase son los siguientes:

    a. S_CSLRXACKOFFSET
        - Espera para la recepción de trama de reconocimiento ACK.
    b. S_CSLRXACKPREPARE
        - Preparando para la recepción de trama de reconocimiento ACK.
    c. S_CSLRXACKREADY
        - Preparado para recibir trama ACK, esperando señal inicio (“go”).
    d. S_CSLRXACKLISTEN
        - Señal de inicio “go” completada, esperando SFD de trama ACK.
    e. S_CSLRXACK
        - Detectado envío SFD de trama ACK, recibiendo resto de bytes.
    f. S_CSLTXPROC
        - Detectado envío del fin de la trama ACK, procesando datos.

A continuación en la siguiente ilustración se recoge la máquina de estados CSL de modo transmisión con sus transiciones y eventos, así como los métodos asociados para el tratamiento de acciones asociadas a cada uno de los estados.

Así mismo, se identifican cada una de las tres fases anteriores, representando **la fase de aviso de próximo envío** (color verde, lateral izquierdo), **la fase de transmisión** (color azul, centro) y **la fase de reconocimiento** (color naranja, lateral derecho).

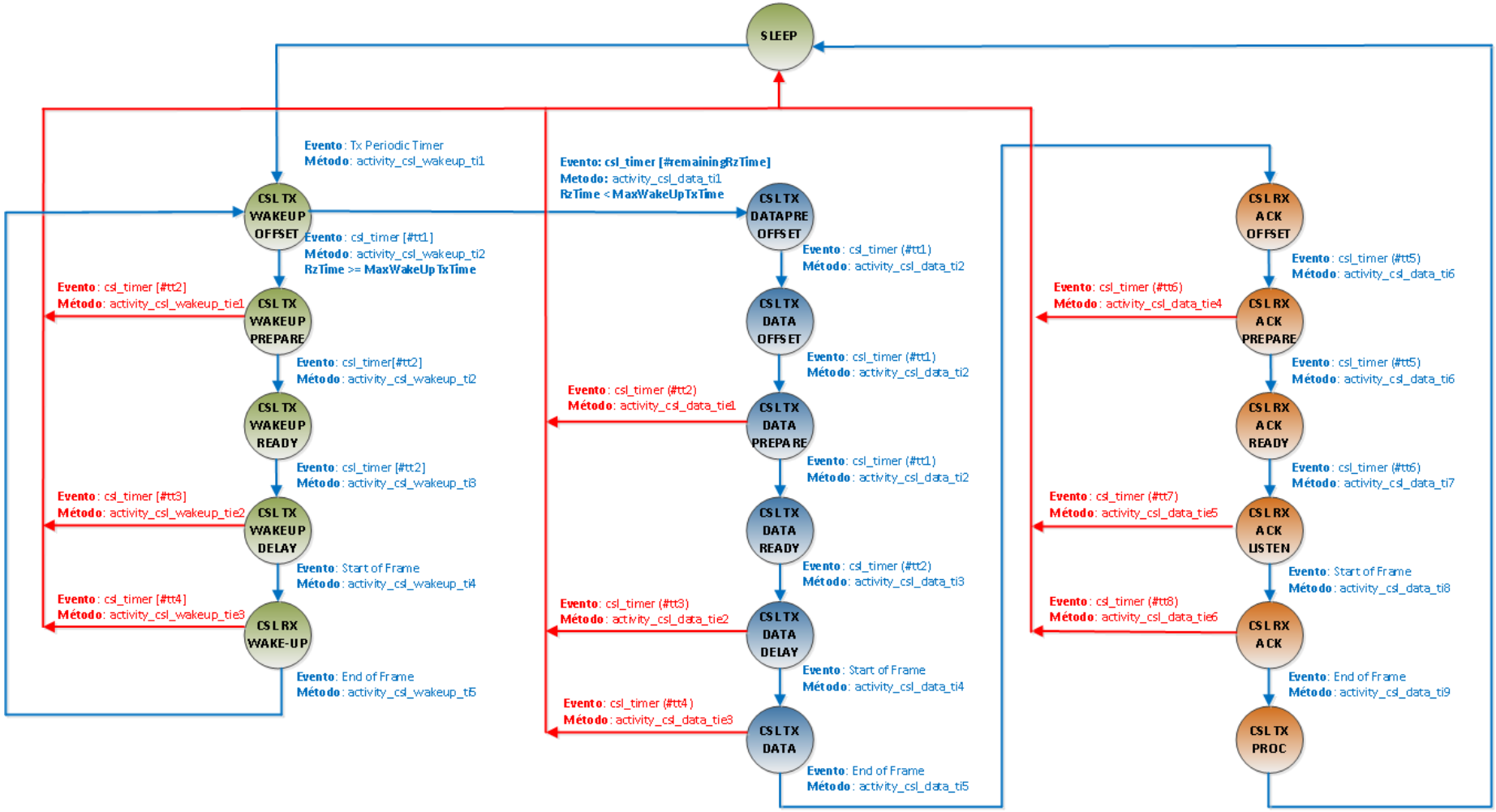


Ilustración 43: Máquina de Estados CSL en Modo Transmisión

6.4.7. CRONOGRAMA DE TRANSMISIÓN Y RECEPCIÓN

Una vez detallado el análisis y diseño realizado para la implementación del modo CSL en OpenWSN, resulta necesario presentar el cronograma de transmisión y recepción propuesto en el cual se detallan las relaciones temporales existentes entre uno y otro.

A este respecto, cabe decir que el cronograma presentado únicamente contempla la secuencia de eventos producidos hasta el inicio de la transmisión de la trama de datos dado que la secuenciación para esta transmisión así como para la trama de reconocimiento (ACK) es idéntica a la existente en OpenWSN sobre TSCH.

De este modo y tal y como se puede observar en la siguiente ilustración, podemos distinguir los siguientes elementos:

a) **Secuencia de Transmisión**: La secuencia de transmisión, representada en la parte superior, muestra el comportamiento del sistema en el modo transmisor de CSL. De este modo y como se ha visto en los apartados anteriores, existirá una verificación de la existencia de datos en la cola pendientes de envío al nodo remoto con una periodicidad igual a *macCSLTxChkFreq*. En el caso de no existir ninguna trama de datos pendiente, el sistema dormirá hasta la próxima verificación. Por el contrario, si se detecta una trama y el sistema no se encuentra ya inmerso en otro proceso de transmisión o recepción previo, se iniciará el procedimiento para el envío de la trama de datos.

   Este proceso implicará primeramente la creación de la trama *wake-up* en la cual se definirá el tiempo inicial *rendezvous* con valor igual al parámetro *macCSLMaxPeriod* dado que únicamente se están contemplando transmisiones no sincronizadas dentro del alcance del trabajo.

   La secuencia de tramas *wake-up* será enviada de manera continua, actualizando en cada una de ellas el tiempo *rendezvous* obtenido como la resta del último tiempo capturado respecto al tiempo *macCSLMaxPeriod*.
   Dentro de este proceso de verificación del nuevo tiempo *rendezvous* se verificará además que el tiempo restante (*remainingRzTime*) no es menor que el tiempo máximo de transmisión de una trama *wake-up* (*maxWakeUpTxTime*) definido como:

```
// [CSL]: tiempo limite de transmisión de una trama wake-up CSL
#define MaxWakeUpTxTime TsTxOffset-delayTx+wdRadioTx+wdDataDuration
```

Ilustración 44: Definición de Tiempo Máximo de Transmisión de Trama Wake-Up (**IEEE802154Ecsl.h**)

Los tiempos implicados en la definición de este tiempo máximo son los siguientes:

- **TsTxOffset**, tiempo entre el inicio de la transmisión y la emisión por la radio del SFD en el proceso de envío
- **delayTx**, tiempo de retardo producido entre la invocación a la función "go" de la radio y la emisión real del SFD por la interfaz de radio.
- **wdRadioTx**, tiempo conservador de duración mayor al peor caso contemplado para *delayTx*.
- **wdDataDuration**, tiempo conservador de duración mayor al tiempo máximo de emisión del paquete más largo (payload de 127 bytes).

En el caso que el tiempo restante sea inferior al tiempo máximo de emisión de la trama *wake-up*, se procederá a dormir un tiempo igual al tiempo *rendezvous* restante dado que dicho tiempo no sería suficiente para enviar una nueva trama *wake-up*.

Finalizada la dormición durante el tiempo restante (*remainingRzTime*), se procederá al envío de la trama de datos siguiendo el mismo procedimiento y temporización a la definida actualmente en OpenWSN sobre TSCH.

b) **Evolución del Tiempo Rendezvous**: La evolución del tiempo rendezvous, representada en la parte central, muestra la actualización del tiempo pendiente hasta el comienzo de la transmisión de la trama de datos.
De este modo, inicialmente el tiempo *rendezvous* será igual al tiempo máximo del periodo CSL (*macCSLMaxPeriod*) al tratarse de una transmisión no sincronizada, para a continuación ir actualizándose en cada envío partir del tiempo capturado en el evento *end-of-frame*.
Tal y como se ha comentado anteriormente, la resta entre el tiempo máximo *macCSLMaxPeriod* y el último tiempo capturado (*lastCaptured*) representará el tiempo restante (*remainingRzTime*).

c) **Secuencia de Recepción**: La secuencia de recepción, representada en la parte inferior, muestra el comportamiento del sistema en el modo receptor de CSL. En este caso, el proceso de escucha periódica (realizado con una periodicidad

macCSLPeriod), procederá a la escucha del canal (en el caso de no haber una operación de transmisión o recepción ya en curso) a la espera de recibir una posible solicitud de transmisión notificada a través de una trama *wake-up*. El procedimiento de escucha es totalmente análogo en tiempos y estados al seguido en el actual OpenWSN sobre TSCH para las tramas de datos y reconocimiento, con las siguientes adaptaciones:

- El **tiempo de escucha** (LISTEN) definido para la recepción del paquete (DURATION_rt3) tendrá una duración igual a *maxWakeUpTxTime* o lo que es lo mismo, el tiempo máximo de transmisión de una trama *wake-up* con el fin de asegurar su recepción.
- La **validación de la trama *wake-up*** recibida se realizará dentro de un nuevo estado dentro de la FSM de recepción denominado CSL-RX-WAKEUP VALIDATE tal y como se ha definido en el **apartado 6.4.6.1 - Maquina de Estados CSL en Modo Recepción**.

Recibida la trama de *wake-up* y una vez validada y verificada, el receptor desactivará la radio y permanecerá dormido durante un tiempo igual al tiempo *rendezvous* recibido, con las siguientes consideraciones:

- Si **el destinatario de la trama *wake-up* es la mota receptora**, el tiempo de espera será igual al tiempo *rendezvous* contenido en dicha trama.
- Si **el destinatario de la trama *wake-up* es otra mota receptora**, el tiempo de espera será igual al tiempo *rendezvous* contenido en dicha trama más el tiempo máximo de envío de una trama de datos más el tiempo máximo de envío de una trama de reconocimiento (ACK), tras lo cual se iniciará de nuevo el procedimiento de escucha periódica.

Finalmente y en el caso que el destinatario sea la propia mota receptora, una vez vencido el temporizador del tiempo *rendez*vous, se iniciará el procedimiento de recepción de la trama de datos siguiendo el mismo procedimiento y temporización definido actualmente en OpenWSN sobre TSCH.

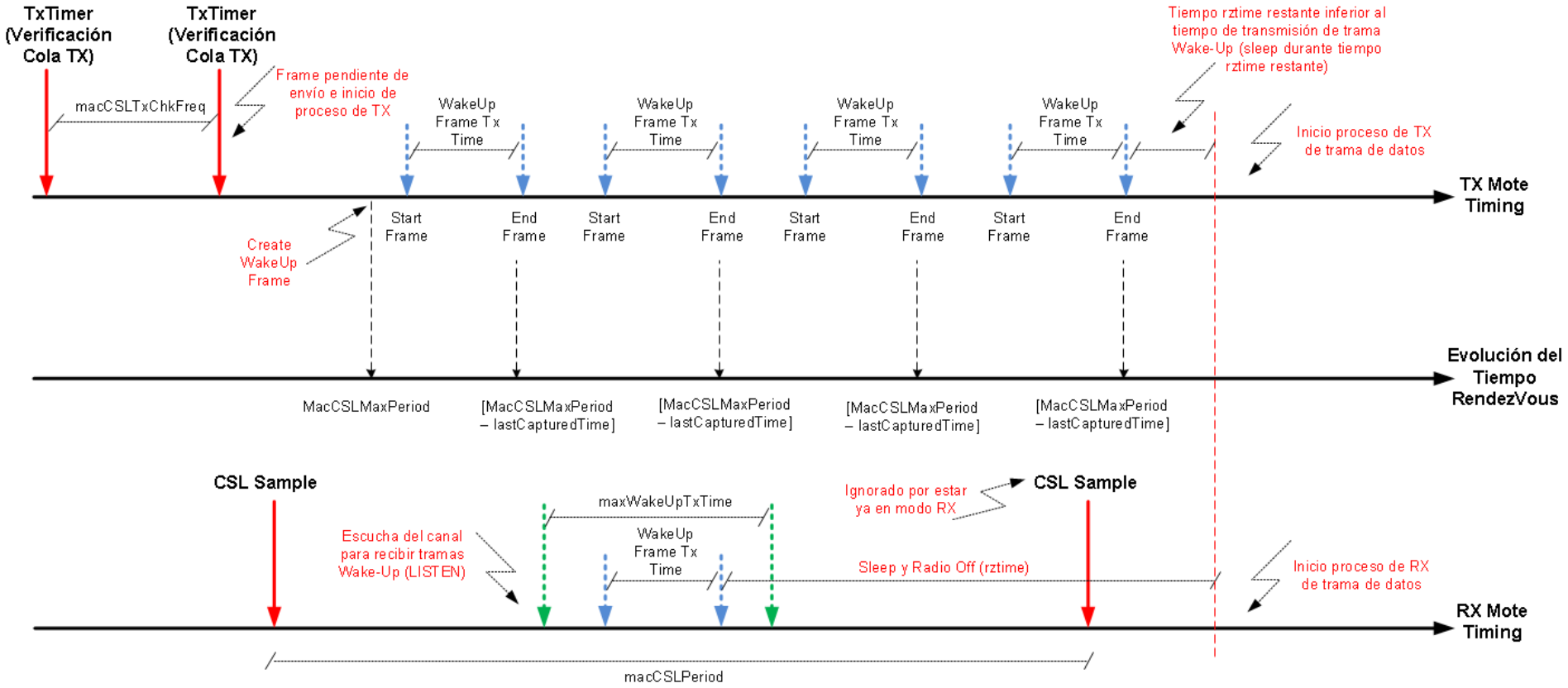


Ilustración 45: Cronograma de Transmisión y Recepción CSL

En este punto y a la vista del cronograma anterior, dado que las motas no se encuentran sincronizadas en el tiempo, cabe la posibilidad que en la mota receptora, durante el periodo de escucha por trama *wake-up* (LISTEN), no reciba correctamente la trama *wake-up*, distinguiendo con ello tres casos distintos tal y como se representa en la siguiente ilustración.

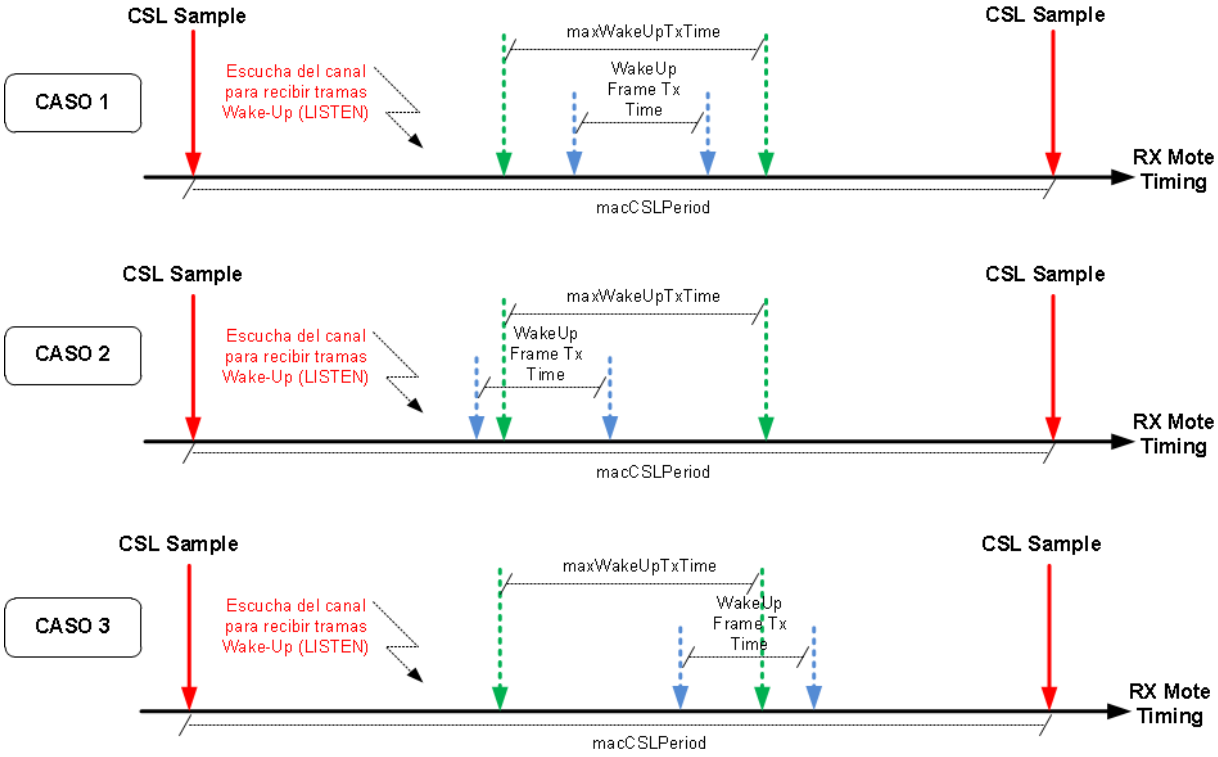


Ilustración 46: Definición de Tiempo Máximo de Transmisión de Trama Wake-Up (**IEEE802154Ecsl.h**)

A continuación se describe cada uno de estos casos, así como las acciones relacionadas con cada uno de ellos para su tratamiento.

- **Caso 1**: Representa el caso en el cual el inicio y fin de la trama se encuentran dentro del periodo de escucha de la trama. Dado que el inicio de trama (*start-of-frame*) se encuentra dentro del periodo de escucha, la trama será detectada procediendo al siguiente estado de la FSM (CSL-RX-WAKEUP) en espera de recibir el fin de trama (*end-of-frame*).
- **Caso 2**: Representa el caso en el cual solamente el fin de trama se encuentra dentro del periodo de escucha de la trama. En este caso, la acción asociada será volver a iniciar el periodo de escucha volviendo para ello al estado CSL-RX-WAKEUP-OFFSET de la FSM.

- **Caso 3**: Representa el caso en el cual solamente el inicio de trama se encuentra dentro del periodo de escucha de la trama. Dado que el inicio de trama (*start-of-frame*) se sigue encontrando dentro del periodo de escucha, el comportamiento será el mismo al definido en el caso 1, esto es, se procederá con el estado CSL-RX-WAKEUP en espera de recibir el fin de trama (*end-of-frame*).

6.5. CODIFICACIÓN

Una vez realizada la fase de análisis y diseño de la solución, se ha procedido a su realización en código fuente cuyos aspectos principales se describen en el presente apartado.

Aquellas funcionalidades o cometidos, tales como las relativas a la implementación de las funciones asociadas a los distintos estados de la máquina de estados, serán comentadas brevemente incluyendo el detalle de ellas dentro de los anexos de la presente memoria.

Así mismo y para cada uno de los apartados, se mostrarán los cambios realizados aplicables al código desarrollado, incluyendo los cambios y modificaciones realizadas así como el fichero en el cual se han aplicado dichos cambios.

6.5.1. ACCESO Y DESCARGA DEL CÓDIGO FUENTE

El código fuente desarrollado en el marco del presente trabajo se encuentra disponible para descarga pública desde el repositorio "**openwsn-fw-csl**" de GitHub.

https://github.com/sgonzalo75/openwsn-fw-csl.git

El procedimiento para la descarga del código fuente desde este repositorio bajo un directorio de nombre ***openwsn-csl*** es la siguiente:

```
$ mkdir openwsn-csl
$ cd openwsn-csl/
$ git clone https://github.com/sgonzalo75/openwsn-fw-csl.git
```

Ilustración 47: Extensión de Códigos de Error y Mensajes Asociados (**opendefs.h**)

6.5.2. IDENTIFICACIÓN DEL CÓDIGO FUENTE

Todos los cambios realizados sobre el código fuente han sido convenientemente comentados junto al código, aplicando para ello la siguiente nomenclatura para su rápida identificación.

- **[CSL]**: Este identificador se encuentra referido a aquellos comentarios asociados a la funcionalidad implementada para el soporte CSL sobre OpenWSN.
- **[CSL-TEST]**: Este indicador se encuentra referido a funcionalidad o modificaciones llevadas a cabo sobre el código fuente para la validación funcional del sistema durante la fase de pruebas.

6.5.3. VARIABLES Y DEFINICIONES GENERALES

A continuación se muestra la definición de nuevos tipos de errores y sus mensajes asociados necesarios para la nueva implementación del modo CSL (***opendefs.h***).

```
#ifndef __OPENDEFS_H
#define __OPENDEFS_H
...
enum {
...
  // [CSL]: l2a csl errors
  ERR_WRONG_STATE_IN_START_CSL_SAMPLING = 0x3a, // wrong state {0} in start CSL channel sampling, at CSL sample {1}
  ERR_WRONG_STATE_IN_CSL_TIMERFIRES = 0x3b, // wrong state {0} in timer fires at CSL sample {1}
  ERR_MAXRXWAKEUPPREPARE_OVERFLOWS = 0x3c, // maxRxDataPrepare overflows while at state {0} in CSL sample {1}
  ERR_WDWAKEUPDURATION_OVERFLOWS = 0x3d, // wdDataDuration overflows while at state {0} in CSL sample {1}
  ERR_WRONG_STATE_IN_CSL_SAMPLE = 0x3e, // wrong state {0} in start of frame while CSL sampling, at CSL sample {1}
  ERR_WRONG_STATE_IN_CSL_ENDOFFRAME = 0x3f, // wrong state {0} in end of frame, at CSL sample {1} {0}
  ERR_MAC_OPERATION_IN_PROGRESS = 0x40, // MAC Operation {0} still in progress in CSL sample {1}. Abort actions and wait to finish.
...
#endif
```

Ilustración 48: Extensión de Códigos de Error y Mensajes Asociados (**opendefs.h**)

Adicionalmente, se ha modificado la estructura creada para el registro de eventos y posterior depuración con el fin de considerar las operaciones de muestreo en recepción en lugar de las operaciones por *slot* definidas en OpenWSN (***IEEE802154csl.h***).

```
#ifndef __IEEE802154E_H
#define __IEEE802154E_H
...
// [CSL]: debugging info
typedef struct {
  PORT_RADIOTIMER_WIDTH    num_newSample;         // CSL - Reference to sample number.
  PORT_RADIOTIMER_WIDTH    num_timer;
  PORT_RADIOTIMER_WIDTH    num_startOfFrame;
  PORT_RADIOTIMER_WIDTH    num_endOfFrame;
  PORT_RADIOTIMER_WIDTH    num_cslSamples;        // CSL - Counter of CSL samples
} ieee154e_dbg_t;
...
#endif
```

Ilustración 49: Modificación de Estructura de Eventos y Depuración (**IEEE802154csl.h**)

Finalmente, se ha realizado un cambio de firma del método ***endSlot*** con el fin de hacer más consistente la nomenclatura global del módulo dado que CSL no hace uso de *slots* tal y como realiza TSCH implementado en OpenWSN. De este modo, el método ha sido modificado a **endOps** para indicar el fin de las operaciones y con ello el inicio de la máquina de estados al estado inicial (SLEEP).

```
/**
\brief Housekeeping tasks to do at the end of CSL operations.

This functions is called once in each CSL mode, when there is nothing more
to do. This might be when an error occured, or when everything went well.
This function resets the state of the FSM so it is ready for the next slot.
*/

// [CSL] – Modificación de firma del método endSlot.
void endOps() {
...
#endif
```

Ilustración 50: Modificación de Firma de Método *EndSlot* por *EndOps* (**IEEE802154csl.c**)

6.5.4. MODOS DE TRABAJO

Los modos de trabajo se encuentran definidos en el fichero de cabecera (**IEEE802154Ecsl.h**) como tipo enumerado.

```
#ifndef __IEEE802154E_H
#define __IEEE802154E_H
...
// [CSL]: CSL Working Mode (CWM)
typedef enum {
  CSL_SLEEP_MODE     // CSL - State where mote is sleeping or idle and it is not involved in a current RX o TX.
  CSL_RX_MODE,       // CSL - State where mote is performing a channel sampling and a (possible) frame RX.
  CSL_TX_MODE        // CSL - State where mote is performing a frame TX.
} ieee154ecsl_mode_t;
...
#endif
```

Ilustración 51: Definición de Modos de Trabajo CSL (**IEEE802154Ecsl.h**)

Así mismo, se ha modificado la estructura interna de variables (**ieee154e_vars_t**) con el fin de registrar la información del modo de trabajo actual.

```
#ifndef __IEEE802154E_H
#define __IEEE802154E_H
...
typedef struct {
  ...
  // [CSL]: Variables CSL
  ieee154ecsl_mode_t      cslMode;          // CSL – csl mode (sleep, transmission o reception).
} ieee154e_vars_t;
...
#endif
```

Ilustración 52: Registro del Modo de Trabajo en Variables MAC (**IEEE802154Ecsl.h**)

6.5.5. DEFINICIÓN Y VALORES DE LOS ATRIBUTOS CSL

6.5.5.1 Periodo de Muestreo de Canales y Periodo Máximo

A continuación se muestra la definición del periodo de muestreo de canales y escucha inactiva (*macCSLPeriod*) así como el periodo máximo de muestreo necesario para la longitud de la secuencia de tramas *wake-up* (*wake-up sequence*) en transmisiones no sincronizadas (***IEEE802154Ecsl.h***).

```
#ifndef __IEEE802154E_H
#define __IEEE802154E_H
...
enum ieee154e_atomicdurations_enum {
  // [CSL]: timing related
  macCSLPeriod          = PORT_macCSLPeriod,      // CSL - 200000us (200 ms) by default.
  macCSLMaxPeriod       = PORT_macCSLMaxPeriod,  // CSL - By default equal to macCSLPeriod
...
#endif
```

Ilustración 53: Periodos de Muestreo Escucha Inactiva y Periodo Máximo (**IEEE802154Ecsl.h**)

Tal y como se puede observar en la ilustración anterior, el valor de ambos atributos dependerá de cada tipo de mota, por lo que su valor en el código estará referido a su valor específico indicado en cada tipo de plataforma (***board_info.h***).

```
#ifndef __BOARD_INFO_H
#define __BOARD_INFO_H
...
//===== [CSL]: IEEE802154E CSL timing
#define PORT_macCSLPeriod          6554                  // CSL - 200000us (200ms) (csl idle listening)
#define PORT_macCSLMaxPeriod       PORT_macCSLPeriod     // CSL - by default, equal to macCSLPeriod.
...
#endif
```

Ilustración 54: Periodo de Muestreo y Periodo Máximo en OpenMote (**board_info.h**)

6.5.6. DEFINICIÓN DE TRAMA WAKE-UP

Primeramente, es necesario extender los tipos de trama a través de la modificación de los posibles valores del campo FCF *(Frame Control Field Type)* (**IEEE802154.h**).

```
#ifndef __IEEE802154_H
#define __IEEE802154_H
...
// [CSL]: Added new FCF types (LLDN and MULTIPURPOSE)
enum IEEE802154_fcf_type_enums {
  IEEE154_TYPE_BEACON          = 0,
  IEEE154_TYPE_DATA            = 1,
  IEEE154_TYPE_ACK             = 2,
  IEEE154_TYPE_CMD             = 3,
  IEEE154_TYPE_LLDN            = 4,   // CSL - Defined for LLDN frames.
  IEEE154_TYPE_MULTIPURPOSE    = 5,   // CSL - Defined for Multipurpose frames, (frame type = 101b (5) )
```

```
  IEEE154_TYPE_UNDEFINED          = 6,   // CSL - Modified previous default type
};
...
#endif
```

Ilustración 55: Ampliación de Tipos de Trama (Tramas Multipropósito) (**IEEE802154.h**)

Adicionalmente, se han añadido las siguientes variables en la estructura interna de variables (**ieee154e_vars_t**) con el fin de almacenar las referencias en la cola de mensajes a la trama wake-up enviada o recibida (según el modo de trabajo sea transmisión o recepción), el DSN de la capa MAC y el valor del tiempo *rendezvous* necesarios para la posterior creación de la trama *wake*-up.

```
#ifndef __IEEE802154E_H
#define __IEEE802154E_H
...
typedef struct {
  ...
  OpenQueueEntry_t*     wakeupToSend;       // CSL - Pointer to the wakeup to send.
  OpenQueueEntry_t*     wakeupReceived;     // CSL - Pointer to the wakeup received.
  uint8_t               cslDSN;             // CSL - DSN for wake-up sequence frames.
  uint16_t              remainingRzTime;    // CSL - Stop sending wake-up and start  sending data?.
} ieee154e_vars_t;
...
#endif
```

Ilustración 56: Registro de DSN y RzTime en Variables MAC (**IEEE802154Ecsl.h**)

Una vez ampliados los tipos de trama para considerar la trama multipropósito y las variables necesarias, debemos añadir los métodos asociados a la creación y análisis de las cabeceras de las tramas de tipo *wake-up.*

Para ello y como se ha visto en la fase de análisis y diseño, se han creado dos métodos encargados precisamente de estos cometidos con el fin de independizar el comportamiento añadido del comportamiento existente (utilizado para tramas de datos y reconocimiento), simplificando sus tareas de administración y mantenimiento futuro.

- ***ieee802154_retrieveWakeUpHeader***
- ***ieee802154_createWakeUpHeader***

A continuación se adjunta el código fuente de ambas funciones (***IEEE802154.c***):

```
...
/**
\brief Retrieve the IEEE802.15.4 MAC Wake-Up Frame header from a (just received) packet.
Note We are writing the fields from the beginning of the header to the end.
\param[in,out] msg           The message just received.
\param[out] ieee802514_header The internal header to write the data to.
\param[out] rztime                    Rendezvous time to wait for data packet.
*/
// [CSL]: Retrieve a wake-up header from the received packet.
void ieee802154_retrieveWakeUpFrame(OpenQueueEntry_t*      msg,
                                    ieee802154_header_iht* ieee802514_header,
                                    uint16_t* rztime) {
  uint8_t temp_8b, temp_8b1, temp_8b2;
  uint8_t src_addr_mode, dst_addr_mode;
  uint16_t temp_16b;

  // La estructura de una trama Wake-Up es la siguiente:
  //   - FRAME CONTROL: 1 byte
  //   - SEQ NUMBER (macDSN): 1 byte
  //   - PAN ID: 2 bytes
  //   - DEST ADDR: 2 bytes
  //   - RZ TIME IE: 2 bytes
  //   - IE LIST TERMINATOR: 2 bytes
  //
  // |   FRAME    |     SEQ     | PAN | DEST  | RZ TIME  |      IE LIST       |
  // | CONTROL | NUMBER |   ID   | ADDR |  HDR IE   | TERMINATOR |

  // Por defecto, se asume que la cabecera no es válida en el caso de abandonar la función al ser el
  // paquete más corto que la cabecera.
  ieee802514_header->valid=FALSE;
  ieee802514_header->headerLength = 0;

  // Validamos la existencia de datos que leer.
  if (ieee802514_header->headerLength > msg->length) { return; }

  // Frame Control Field (1 byte)
  temp_8b = *((uint8_t*)(msg->payload)+ieee802514_header->headerLength);

  // La estructura del FCF en la trama de wake-up es 0x55 = 01010101b:
  // b0-b2 -- Frame Type (debe ser 101 para trama wake-up)
  // b3 ------ Long Frame Control (0 para indicar que es un FCF de 1 byte como es para tramas wake-up).
```

```
// b4-b5 -- Dest Addr Mode (en este caso, short address por lo que tiene que valer 10).
// b6-b7 -- Src Addr Mode (en este caso, short address por lo que tiene que valer 10).

//  b7 | b6 | b5 | b4 | b3 | b2 | b1 | b0 |
//   0  |  1 |  0 |  1 |  0 |  1 |  0 |  1 |
ieee802514_header->frameType = (temp_8b >> IEEE154_FCF_FRAME_TYPE) & 0x07;//3b

// Comprobamos que efectivamente se trata de una trama de tipo MULTIPURPOSE.
if (ieee802514_header->frameType != IEEE154_TYPE_MULTIPURPOSE) {return; }

// Verificamos que el campo Long Frame Control tiene valor cero para indicar que el FCF solo tiene 1 byte.
if ((temp_8b >> 3) & 0x01) { return; }

// Recuperamos los Destination y Source Address Mode
// Destination Address Mode
dst_addr_mode = (temp_8b >> 4) & 0x03;
if(dst_addr_mode == 1) { // 01 en binario en bits b5-b4
  // Debería ser siempre este caso en las tramas wake-up por lo que se trata como error cualquier otra opción.
  ieee802514_header->dest.type = ADDR_16B;
}
else {
   openserial_printError(COMPONENT_IEEE802154,ERR_IEEE154_UNSUPPORTED,
                         (errorparameter_t)1, (errorparameter_t)(dst_addr_mode));
  return; // this is an invalid packet, return
}

// Source Address Mode
src_addr_mode = (temp_8b >> 6) & 0x03;
if(src_addr_mode == 1) { // 01 en binario en bits b7-b6
   // Debería ser siempre este caso en las tramas wake-up por lo que se trata como error cualquier otra opción.
   ieee802514_header->src.type = ADDR_16B;
}
else {
  openserial_printError(COMPONENT_IEEE802154,ERR_IEEE154_UNSUPPORTED,
                         (errorparameter_t)2, (errorparameter_t)src_addr_mode);
   return; // this is an invalid packet, return
}

// En este punto ya hemos analizado el contenido del byte 1 correspondiente al Frame Control Field (FCF)
// A partir de este punto, la estructura del paquete es la siguiente:
//   - SEQ NUMBER (macDSN): 1 byte
```

```
   //   - PAN ID: 2 bytes
   //   - DEST ADDR: 2 bytes
   //   - RZ TIME IE: 2 bytes
   //   - IE LIST TERMINATOR: 2 bytes

   // Avanzamos 1 byte en el procesamiento de la cabecera.
   ieee802514_header->headerLength += 1;

   // Posicionamos dentro del paquete para leer el resto de elementos.

   // 1.- SequenceNumber
   if (ieee802514_header->headerLength > msg->length) { return; } // no more to read!
   ieee802514_header->dsn  = *((uint8_t*)(msg->payload)+ieee802514_header->headerLength);
   ieee802514_header->headerLength += 1;

   // 2.- panID
   if (ieee802514_header->headerLength > msg->length) { return; } // no more to read!
   packetfunctions_readAddress(((uint8_t*)(msg->payload)+ieee802514_header->headerLength),
                               ADDR_PANID, &ieee802514_header->panid, OW_LITTLE_ENDIAN);
   ieee802514_header->headerLength += 2;

   // 3.- Dest Addr
   if (ieee802514_header->headerLength > msg->length) { return; } // no more to read!

   // En este punto ya sabemos que el tipo de dirección tiene que ser ADDR_16B dado que cualquier otro caso
   // ya fue tratado antes en la lectura del frame control field por lo que leemos los 2 bytes de la dirección del
   // destino.
   packetfunctions_readAddress(((uint8_t*)(msg->payload)+ieee802514_header->headerLength), ADDR_16B,
&ieee802514_header->dest, OW_LITTLE_ENDIAN);
   ieee802514_header->headerLength += 2;

   // 4.- RZ Time Header IE, formado por 4 bytes con la siguiente estructura (0x0E82)
   //  - b0-b6 (Length) = 2
   //  - b7-b14 (Element ID) = 0x1D
   //  - b15 (Type) = 0
   //  - b16-b31 (IE Content) = time (2 bytes)
   //
   //  b15 | b14 | b13 | b12 | b11 | b10 | b9 | b8 | b7 | b6 | b5 | b4 | b3 | b2 | b1 | b0 |
   //   0  |  0  |  0  |  0  |  1  |  1  | 1  | 0  | 1  | 0  | 0  | 0  | 0  | 0  | 1  | 0  |

   if (ieee802514_header->headerLength > msg->length) {  return; } // no more to read!
```

```
// Verificamos longitud = 2 y elementID = 0x1D, leyendo los primeros 2 bytes del RZ Time IE, uno a uno.
temp_8b1 = *((uint8_t*)(msg->payload)+ieee802514_header->headerLength);
ieee802514_header->headerLength += 1;

temp_8b2 = *((uint8_t*)(msg->payload)+ieee802514_header->headerLength);
ieee802514_header->headerLength += 1;

temp_16b = (temp_8b2 << 8) | temp_8b1;

// Comprobación de longitud a valor 2.
if ((temp_16b & 0x007f) != 2) { return; }

// Comprobación de element ID = 0x1D.
if ((temp_16b >> 7) != 0x001d) { return; }

// Comprobación del type = 0.
if ((temp_16b >> 15) & 0x0001) { return; }

// Obtención del IE Content (rztime).
if (ieee802514_header->headerLength > msg->length) { return; } // no more to read!

temp_8b1 = *((uint8_t*)(msg->payload)+ieee802514_header->headerLength);
ieee802514_header->headerLength += 1;

temp_8b2 = *((uint8_t*)(msg->payload)+ieee802514_header->headerLength);
ieee802514_header->headerLength += 1;

(*rztime) = (temp_8b2 << 8) | temp_8b1;

// 5.- IE List Terminator, formado por 2 bytes con la siguiente estructura (0x3F80 o 0x3F00)
//  - b0-b6 (Length) = 0
//  - b7-b14 (Element ID) = 0x7e o 0x7f
//  - b15 (Type) = 0
//
//  b15 | b14 | b13 | b12 | b11 | b10 | b9 | b8 | b7 | b6 | b5 | b4 | b3 | b2 | b1 | b0 |
//   0  |  0  |  1  |  1  |  1  |  1  | 1  | 1  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  |

if (ieee802514_header->headerLength > msg->length) { return; } // no more to read!

// Verificamos longitud a valor 0 y el elementID a valor 0x7E o 0x7F. Lo hacemos leyendo los 2 bytes IE.
```

```
   temp_8b1 = *((uint8_t*)(msg->payload)+ieee802514_header->headerLength);
   ieee802514_header->headerLength += 1;

   temp_8b2 = *((uint8_t*)(msg->payload)+ieee802514_header->headerLength);
   ieee802514_header->headerLength += 1;

   temp_16b = (temp_8b2 << 8) | temp_8b1;

   // Comprobación de longitud a valor 0.
   if (temp_16b & 0x007f) { return; }

   // Comprobación de element ID = 0x7E o 0x7F.
   if (((temp_16b >> 7) != 0x7e) && ((temp_16b >> 7) != 0x7f)) { return; }

   // Comprobación del type = 0.
   if ((temp_16b >> 15) & 0x0001) { return; }

   // Por ultimo y en el caso de haber llegado aquí, consideramos la cabecera como válida.
   ieee802514_header->valid=TRUE;
}
...
```

Ilustración 57: Función de Análisis *ieee802154_retrieveWakeUpHeader* ((**IEEE802154Ecsl.c**)

```
/**
/**
\brief Prepend the IEEE802.15.4 MAC Wake-Up header to a (to be transmitted) packet.
Note that we are writing the field from the end of the header to the beginning.
\param[in,out] msg             The message to append the header to.
\param[in]     sequenceNumber   Sequence number of this frame.
\param[in]     nextHop          Address of the next hop
\param[in]     rztime           Rendezvoud time
*/
// [CSL]: Prepend  a wake-up header to a packet.
void ieee802154_createWakeUpFrame(OpenQueueEntry_t*          msg,
                                  uint8_t                    sequenceNumber,
                                  open_addr_t*               nextHop,
                                  uint16_t                   rz_time) {
   open_addr_t    nextHop16b;
   // La estructura de una trama Wake-Up es la siguiente:
   //   - FRAME CONTROL: 1 byte
   //   - SEQ NUMBER (macDSN): 1 byte
```

```
   //   - PAN ID: 2 bytes
   //   - DEST ADDR: 2 bytes
   //   - RZ TIME IE: 2 bytes (header) + 2 bytes (body)
   //   - IE LIST TERMINATOR: 2 bytes
   //
  // |   FRAME    |    SEQ      | PAN  | DEST  | RZ TIME  |     IE LIST       |
  //  | CONTROL | NUMBER |    ID    | ADDR  |  HDR IE   | TERMINATOR |
  //

   // Empezamos a escribir el payload de la trama wake-up empezando del final hacia adelante.

   // IE LIST TERMINATOR, formado por 2 bytes con la siguiente estructura (0x3F80 o 0x3F00)
   //  - b0-b6 (Length) = 0
   //  - b7-b14 (Element ID) = 0x7e o 0x7f
   //  - b15 (Type) = 0
   //
   //  b15 | b14 | b13 | b12 | b11 | b10 | b9 | b8 | b7 | b6 | b5 | b4 | b3 | b2 | b1 | b0 |
   //    0   |   0   |   1   |   1   |   1   |   1   | 1 | 1   |   0   |   0   |   0   |   0   |   0   |   0   |   0   |   0   |
   packetfunctions_reserveHeaderSize(msg,sizeof(uint16_t));
   *((uint16_t*)(msg->payload)) = 0x3F00;  // Element ID = 0x7e

   // RZ TIME IE, formado por 4 bytes con la siguiente estructura (0x0E82). La cabecera es igual a:
   //  - b0-b6 (Length) = 2
   //  - b7-b14 (Element ID) = 0x1D
   //  - b15 (Type) = 0
   //  - b16-b31 (IE Content) = time (2 bytes)
   //
   //  b15 | b14 | b13 | b12 | b11 | b10 | b9 | b8 | b7 | b6 | b5 | b4 | b3 | b2 | b1 | b0 |
   //    0   |   0   |   0   |   0   |   1   |   1   | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |

   // rz time ie body (time)
   packetfunctions_reserveHeaderSize(msg,sizeof(uint16_t));
   *((uint16_t*)(msg->payload)) = rz_time;

   // rz time ie header
   packetfunctions_reserveHeaderSize(msg,sizeof(uint16_t));
  *((uint16_t*)(msg->payload)) = 0x0E82;

  // DEST ADDR
  if (nextHop->type == ADDR_16B) {
     packetfunctions_writeAddress(msg,nextHop,OW_LITTLE_ENDIAN);
```

```
  } else if (nextHop->type == ADDR_64B) {
     packetfunctions_mac64bToMac16b(nextHop, &nextHop16b);
     packetfunctions_writeAddress(msg,&nextHop16b,OW_LITTLE_ENDIAN);
  }

  // PAN ID
  packetfunctions_writeAddress(msg,idmanager_getMyID(ADDR_PANID),OW_LITTLE_ENDIAN);

  // SEQ NUMBER
  packetfunctions_reserveHeaderSize(msg,sizeof(uint8_t));
  *((uint8_t*)(msg->payload)) = sequenceNumber;

  // La estructura del FCF en la trama de wake-up es 0x55 = 01010101b:
  // b0-b2 -- Frame Type (debe ser 101 para trama wake-up)
  // b3 ----- Long Frame Control (0 para indicar que es un FCF de 1 byte como es en el caso de tramas wake-up).
  // b4-b5 -- Dest Addr Mode (en este caso, short address por lo que tiene que valer 10).
  // b6-b7 -- Src Addr Mode (en este caso, short address por lo que tiene que valer 10).

  //  b7 | b6 | b5 | b4 | b3 | b2 | b1 | b0 |
  //   0  |  1  |  0  |  1  |  0  |  1  |  0  |  1  |

  // FCF
  packetfunctions_reserveHeaderSize(msg,sizeof(uint8_t));
  *((uint8_t*)(msg->payload)) = 0x55;
}
...
```

Ilustración 58: Función de Preparación *ieee802154_createWakeUpFrame* (**IEEE802154Ecsl.c**)

6.5.7. INTERRUPCIONES Y TEMPORIZACIONES

Tal y como se ha visto en el apartado correspondiente de análisis y diseño, las interrupciones y temporizaciones juegan un papel clave dentro del correcto funcionamiento del sistema a través de las distintas transiciones de estados de la FSM y la respuesta a eventos especiales producidos en el sistema.

En este sentido, la primera modificación realizada sobre el sistema se encuentra localizada en la función de inicialización del módulo sobre la cual se han modificado las referencias a las funciones de callback así como el valor de temporización para producir un evento cada *macCSLPeriod* e iniciar el proceso de escucha periódica para la detección de nuevas solicitudes de transmisión (*CSL idle listening*).

A continuación se muestra el contenido de esta función modificada:

```
...
/**
  \brief This function initializes this module.
  Call this function once before any other function in this module, possibly during boot-up.
*/
// [CSL]: Modificaciones sobre método inicial referidas a los temporizadores y al estado inicial.
void ieee154e_init() {
...
   // [CSL]: update CSL Mode to SLEEP in order to allow RX o TX.
   ieee154e_vars.cslMode = CSL_SLEEP_MODE;
   // [CSL]: Set initial DSN counter for wake-up sequence frames.
   ieee154e_vars.cslDSN    = 0;
...
   // set callback functions for the radio
   radio_setOverflowCb(isr_ieee154ecsl_newChannelSample); // CSL - Fires every macCSLPeriod for sampling.
   radio_setCompareCb(isr_ieee154ecsl_timer);                    // CSL - Fires for FSM state changing.
   radio_setStartFrameCb(ieee154ecsl_startOfFrame);          // CSL – Fires on start of frame detected on radio.
   radio_setEndFrameCb(ieee154ecsl_endOfFrame);              // CSL – Fires on end of frame detected on radio.

   // [CSL]: set timer for checking frames on local queue to transmit.
  ieee154e_vars.txTimer = opentimers_start(macCSLTxChkFreq, TIMER_PERIODIC, TIME_TICS,
                                           isr_ieee154ecsl_txtimer_cb);

   // [CSL-TEST]: set timer for callback to add packet to queue for testing CSL TX (every 5 seconds)
  ieee154e_vars.cslTxTestTimer = opentimers_start(2000, TIMER_PERIODIC,  TIME_MS,
                                                  isr_ieee154ecsl_addPacketToQueueForTestingCslTx_cb);

   // [CSL]: have the radio start its timer for channel sampling (macCSLPeriod)
   radio_startTimer(macCSLPeriod);
}
...
```

Ilustración 59: Función *ieee154e_init* para Inicialización del Módulo (**IEEE802154Ecsl.c**)

A continuación se muestra el contenido de las distintas funciones de *callback* definidas.

La primera de ellas (***isr_ieee154csl_newChannelSample***) será invocada a intervalos regulares (cada *macCSLPeriod*) para la realización de un nuevo muestreo de canal en espera de detección de nuevas solicitudes. Esta función iniciará por tanto la máquina de estados finitos en modo recepción.

```
...
/**
\brief Indicates a new CSL Channel Sample has just started.
This function executes in ISR mode, when the new CSÑ Channel Sample timer fires.
*/
// [CSL]: Callback indicating the start of a new CSL channel sample.
void isr_ieee154ecsl_newChannelSample() {
  // Establish the new timer for the next channel sample
  radio_setTimerPeriod(macCSLPeriod);

  // Verificamos que no estamos ya en un proceso de TX o RX previo.
  if(ieee154e_vars.cslMode == CSL_SLEEP_MODE) {
     // We call the first activity on FSM for channel sampling (CSL FSM RX mode)
     activity_csl_wakeup_ri1();
  } else {
     // log the error
     openserial_printInfo(COMPONENT_IEEE802154E,ERR_MAC_OPERATION_IN_PROGRESS,
          (errorparameter_t)ieee154e_vars.cslMode, (errorparameter_t)ieee154e_dbg.num_cslSamples);
  }

  // Increment the number of cslSamples.
  ieee154e_dbg.num_cslSamples++;
}
...
```

Ilustración 60: Función de callback *isr_ieee154ecsl_newChannelSample* (**IEEE802154Ecsl.c**)

La siguiente función de *callback* (***isr_ieee154ecsl_timer***) es la encargada de determinar el siguiente estado dentro de la máquina de estados de recepción y transmisión, el cual vendrá determinado en función del estado actual y las entradas y eventos producidos.

```
...
/**
\brief Indicates the FSM timer has fired.
This function executes in ISR mode, when the FSM timer fires.
*/
// [CSL]: Callback to select the next FSM action.
void isr_ieee154ecsl_timer() {
  if (ieee154e_vars.cslMode == CSL_RX_MODE)  { // CSL channel sampling and possible frame RX.
     switch (ieee154e_vars.state) {
       // RX-MODE
      case S_CSLRXWAKEUPOFFSET:          activity_csl_wakeup_ri2();          break;
      case S_CSLRXWAKEUPPREPARE:         activity_csl_wakeup_rie1();         break;
      case S_CSLRXWAKEUPREADY:           activity_csl_wakeup_ri3();          break;
      case S_CSLRXWAKEUPLISTEN:          activity_csl_wakeup_rie2();         break;
      case S_CSLRXWAKEUP:                activity_csl_wakeup_rie3();         break;
      case S_CSLRXWAKEUPVALIDATE:        activity_csl_wakeup_rie4();         break;

      case S_CSLRXDATAOFFSET:            activity_csl_data_ri2();            break;
      case S_CSLRXDATAPREPARE:           activity_csl_data_rie1();           break;
      case S_CSLRXDATAREADY:             activity_csl_data_ri3();            break;
      case S_CSLRXDATALISTEN:            activity_csl_data_rie2();           break;
      case S_CSLRXDATA:                  activity_csl_data_rie3();           break;

     case S_CSLTXACKOFFSET:              activity_csl_data_ri6();            break;
     case S_CSLTXACKPREPARE:             activity_csl_data_rie4();           break;
     case S_CSLTXACKREADY:               activity_csl_data_ri7();            break;
     case S_CSLTXACKDELAY:               activity_csl_data_rie5();           break;
     case S_CSLTXACK:                    activity_csl_data_rie6();           break;
     default:
       // log the error
      openserial_printError(COMPONENT_IEEE802154E,ERR_WRONG_STATE_IN_CSL_TIMERFIRES,
              (errorparameter_t)ieee154e_vars.state, (errorparameter_t)ieee154e_dbg.num_cslSamples);
       // abort
      endOps();
      break;
     }
  } else if (ieee154e_vars.cslMode == CSL_TX_MODE)  { // Current CSL operation mode is frame TX.
     switch (ieee154e_vars.state) {
      // TX-MODE
     case S_CSLTXWAKEUPOFFSET:           activity_csl_wakeup_ti2();          break;
     case S_CSLTXWAKEUPPREPARE:          activity_csl_wakeup_tie1();         break;
```

```
      case S_CSLTXWAKEUPREADY:          activity_csl_wakeup_ti3();      break;
      case S_CSLTXWAKEUPDELAY:          activity_csl_wakeup_tie2();     break;
      case S_CSLTXWAKEUP:               activity_csl_wakeup_tie3();     break;

      case S_CSLTXDATAPREOFFSET:        activity_csl_data_ti1();        break;
      case S_CSLTXDATAOFFSET:           activity_csl_data_ti2();        break;
      case S_CSLTXDATAPREPARE:          activity_csl_data_tie1();       break;
      case S_CSLTXDATAREADY:            activity_csl_data_ti3();        break;
      case S_CSLTXDATADELAY:            activity_csl_data_tie2();       break;
      case S_CSLTXDATA:                 activity_csl_data_tie3();       break;
      case S_CSLRXACKOFFSET:            activity_csl_data_ti6();       break;
      case S_CSLRXACKPREPARE:           activity_csl_data_tie4();       break;
      case S_CSLRXACKREADY:             activity_csl_data_ti7();        break;
      case S_CSLRXACKLISTEN:            activity_csl_data_tie5();       break;
      case S_CSLRXACK:                  activity_csl_data_tie6();       break;

     default:
      // log the error
     openserial_printError(COMPONENT_IEEE802154E,ERR_WRONG_STATE_IN_CSL_TIMERFIRES,
           (errorparameter_t)ieee154e_vars.state, (errorparameter_t)ieee154e_dbg.num_cslSamples);
     // abort
     endOps();
     break;
    }
  }
 ieee154e_dbg.num_timer++;
}
...
```

Ilustración 61: Función de callback *isr_ieee154ecsl_timer* (**IEEE802154Ecsl.c**)

Por último, las dos funciones de *callback* siguientes (***isr_ieee154ecsl_startOfFrame*** y ***isr_ieee154ecsl_endOfFrame***) son invocada cuando la interfaz radio detecta un inicio de trama (SFD) o un fin de trama en el medio respectivamente. En este caso y en función del estado actual de la máquina de estados, procederá a la determinación de la siguiente acción a realizar a través de la actualización del estado.

```
...
// [CSL] – Callback fired after receive a start of frame on radio.
void ieee154ecsl_startOfFrame(PORT_RADIOTIMER_WIDTH capturedTime) {
    if(ieee154e_vars.cslMode == CSL_RX_MODE)  { // CSL channel sampling and possible frame RX.
      switch (ieee154e_vars.state) {
       // RX MODE
        /* It is possible to receive in this state for radio where there is no way of differentiated between
           "ready to listen" and "listening"  (e.g. CC2420). We must therefore expect to the start of a packet
           in this "ready" state.
        */
       case S_CSLRXWAKEUP:              // no break!
       case S_CSLRXWAKEUPLISTEN:        activity_csl_wakeup_ri4(capturedTime); break;

       case S_CSLRXDATAREADY:           // no break!
      case S_CSLRXDATALISTEN:           activity_csl_data_ri4(capturedTime);   break;

      case S_CSLTXACKDELAY:             activity_csl_data_ri8(capturedTime);   break;
      default:
           // log the error
           openserial_printError(COMPONENT_IEEE802154E,ERR_WRONG_STATE_IN_CSL_SAMPLE,
                      (errorparameter_t)ieee154e_vars.state, (errorparameter_t)ieee154e_dbg.num_cslSamples);
          // abort
          endOps();
          break;
     }
 } else if (ieee154e_vars.cslMode == CSL_TX_MODE)  { // Current CSL operation mode is frame TX.
   switch (ieee154e_vars.state) {
    // TX MODE
       /* It is possible to receive in this state for radio where there is no way of differentiated between
          "ready to listen" and "listening"  (e.g. CC2420). We must therefore expect to the start of a packet in
          this "ready" state.
       */
      case S_CSLRXACKREADY:              // no break!
      case S_CSLRXACKLISTEN:             activity_csl_data_ti8(capturedTime);  break;

      case S_CSLTXWAKEUPDELAY:           activity_csl_wakeup_ti4(capturedTime); break;
      case S_CSLTXDATADELAY:             activity_csl_data_ti4(capturedTime);   break;
       default:
        // log the error
       openserial_printError(COMPONENT_IEEE802154E,ERR_WRONG_STATE_IN_CSL_SAMPLE,
           (errorparameter_t)ieee154e_vars.state, (errorparameter_t)ieee154e_dbg.num_cslSamples);
```

```
      // abort
      endOps();
      break;
    }
  }
  ieee154e_dbg.num_startOfFrame++;
}
...
```

Ilustración 62: Función de callback *isr_ieee154ecsl_startOfFrame* (**IEEE802154Ecsl.c**)

```
...
// [CSL] – Callback fired after receive a end of frame on radio.
void ieee154ecsl_endOfFrame(PORT_RADIOTIMER_WIDTH capturedTime) {
  if(ieee154e_vars.cslMode == CSL_RX_MODE)  { // CSL channel sampling and possible frame RX.
    switch (ieee154e_vars.state) {
     // RX MODE
    case S_CSLRXWAKEUP:              activity_csl_wakeup_ri5(capturedTime);  break;
    case S_CSLRXDATA:                activity_csl_data_ri5(capturedTime);    break;
    case S_CSLTXACK:                 activity_csl_data_ri9(capturedTime);    break;
    // [CSL]: En caso que el periodo de LISTEN solo detectemos el fin de trama al no haber sincronismo con Tx.
    case S_CSLRXWAKEUPLISTEN:        activity_csl_wakeup_ri1();              break;
    default:
      // log the error
     openserial_printError(COMPONENT_IEEE802154E,ERR_WRONG_STATE_IN_CSL_ENDOFFRAME,
           (errorparameter_t)ieee154e_vars.state, (errorparameter_t)ieee154e_dbg.num_cslSamples);
     // abort
    endOps();
    break;
  }
 } else if (ieee154e_vars.cslMode == CSL_TX_MODE)  { // Current CSL operation mode is frame TX.
   switch (ieee154e_vars.state) {
    // TX MODE
    case S_CSLTXWAKEUP:              activity_csl_wakeup_ti5(capturedTime);  break;
    case S_CSLTXDATA:                activity_csl_data_ti5(capturedTime);    break;
    case S_CSLRXACK:                 activity_csl_data_ti9(capturedTime);    break;
    default:
     // log the error
     openserial_printError(COMPONENT_IEEE802154E,ERR_WRONG_STATE_IN_CSL_ENDOFFRAME,
        (errorparameter_t)ieee154e_vars.state, (errorparameter_t)ieee154e_dbg.num_cslSamples);
     // abort
      endOps();
```

```
      break;
  }
}
  ieee154e_dbg.num_endOfFrame++;
}
...
```

Ilustración 63: Función de callback *isr_ieee154ecsl_endOfFrame* (**IEEE802154Ecsl.c**)

Finalmente, se muestra a continuación la función (*callback*) asociada al temporizador periódico para la verificación de datos pendientes de transmitir.

Esta función (**isr_ieee154ecsl_txtimer_cb)**, invocada periódicamente a intervalos regulares verificará el modo actual de funcionamiento para a continuación verificar la existencia de datos pendientes de envío en la siguiente celda del planificador.

Previo a la revisión del contenido de la función de *callback* asociada a este temporizador, a continuación se muestra la definición del periodo asociado (***IEEE802154Ecsl.h***).

```
#ifndef __IEEE802154E_H
#define __IEEE802154E_H
...
enum ieee154e_atomicdurations_enum {
  // [CSL]: timing related
  // How often should we check if it is something pending on queue to transmit?.
 macCSLTxChkFreq   = PORT_macCSLTxChkFreq,  // CSL - 60000us (60ms) by default.
...
#endif
```

Ilustración 64: Periodo del Temporizador de Verificación de Datos para Transmisión (**IEEE802154Ecsl.h**)

Tal y como se puede observar en la ilustración anterior, el valor de este atributo dependerá de cada tipo de mota, por lo que su valor en el código estará referido a su valor específico indicado en cada tipo de plataforma (***board_info.h***).

```
#ifndef __BOARD_INFO_H
#define __BOARD_INFO_H
...
//===== [CSL]: IEEE802154E CSL timing
#define PORT_macCSLTxChkFreq      1966    // CSL - 60000us (60ms) (check for new frames on queue to tx).
...
#endif
```

Ilustración 65: Periodo de Muestreo y Periodo Máximo en OpenMote (**board_info.h**)

Vista la configuración de la periodicidad de este temporizador, a continuación se muestra el contenido de esta función de *callback*.

```
...
// [CSL]: Callback to check pending data to be sent.
void isr_ieee154ecsl_txtimer_cb() {
  // Verificamos que no estamos ya en un proceso de TX o RX previo.
  if(ieee154e_vars.cslMode == CSL_SLEEP_MODE) {
    // We call the first activity on TX FSM (CSL FSM TX mode)
    activity_csl_wakeup_ti1();
  }
  else {
    // log the error
   openserial_printInfo(COMPONENT_IEEE802154E,ERR_MAC_OPERATION_IN_PROGRESS,
          (errorparameter_t)ieee154e_vars.cslMode, (errorparameter_t)ieee154e_dbg.num_cslSamples);
  }
}
...
```

Ilustración 66: Función de callback *isr_ieee154ecsl_txtimer_cb* (**IEEE802154Ecsl.c**)

6.5.8. MAQUINA DE ESTADOS CSL

En este apartado, se procederá a detallar las funciones con las acciones correspondientes a cada uno de los estados de la máquina de estados CSL para los modos recepción y transmisión.

6.5.8.1 Maquina de Estados CSL en Modo Recepción

Tal y como se ha descrito en la fase de análisis y diseño, la máquina de estados CSL en modo recepción se encuentra dividida en tres fases de operación: escucha activa, recepción y reconocimiento (ver **apartado 6.4.6.1 - Maquina de Estados CSL en Modo Recepción**).

Para cada una de estas fases, se han definido y agrupado una serie de estados y funciones asociadas, representadas gráficamente en la **Ilustración 42: Máquina de Estados CSL en Modo Recepción**, resumida a continuación en la siguiente tabla:

Fase	Estado	Función
Escucha Activa	S_CSLRXWAKEUPOFFSET	activity_csl_wakeup_ri1
	S_CSLRXWAKEUPPREPARE	activity_csl_wakeup_ri2
	S_CSLRXWAKEUPREADY	activity_csl_wakeup_ri2
	S_CSLRXWAKEUPLISTEN	activity_csl_wakeup_ri3
	S_CSLRXWAKEUP	activity_csl_wakeup_ri4
	S_CSLRXWAKEUPVALIDATE	activity_csl_wakeup_ri4
Recepción	S_CSLRXDATAOFFSET	activity_csl_wakeup_ri5
	S_CSLRXDATAPREPARE	activity_csl_data_ri2
	S_CSLRXDATAREADY	activity_csl_data_ri2
	S_CSLRXDATALISTEN	activity_csl_data_ri3
	S_CSLRXDATA	activity_csl_data_ri4
Reconocimiento	S_CSLTXACKOFFSET	activity_csl_data_ri5
	S_CSLTXACKPREPARE	activity_csl_data_ri6
	S_CSLTXACKREADY	activity_csl_data_ri6
	S_CSLTXACKDELAY	activity_csl_data_ri7
	S_CSLTXACK	activity_csl_data_ri8
	S_CSLRXPROC	activity_csl_data_ri9

Tabla 7: Relación de Estados de FSM CSL Modo Recepción y Funciones de Actividades

Adicionalmente, en algunos estados se gestionan distintas situaciones de excepción las cuales son tratadas a través de una función asociada y que tienen como objetivo principal el tratamiento del evento concreto, el registro de la situación, la limpieza de recursos utilizados, y el reinicio de la máquina de estados a su valor por defecto (SLEEP).

Fase	Estado	Función
Escucha Activa	S_CSLRXWAKEUPOFFSET	-
	S_CSLRXWAKEUPPREPARE	activity_csl_wakeup_rie1
	S_CSLRXWAKEUPREADY	-
	S_CSLRXWAKEUPLISTEN	activity_csl_wakeup_rie2
	S_CSLRXWAKEUP	activity_csl_wakeup_rie3
	S_CSLRXWAKEUPVALIDATE	activity_csl_wakeup_rie4
Recepción	S_CSLRXDATAOFFSET	-
	S_CSLRXDATAPREPARE	activity_csl_data_rie1
	S_CSLRXDATAREADY	-
	S_CSLRXDATALISTEN	activity_csl_data_rie2
	S_CSLRXDATA	activity_csl_data_rie3

Reconocimiento	S_CSLTXACKOFFSET	-
	S_CSLTXACKPREPARE	activity_csl_data_rie4
	S_CSLTXACKREADY	-
	S_CSLTXACKDELAY	activity_csl_data_rie5
	S_CSLTXACK	activity_csl_data_rie6
	S_CSLRXPROC	-

Tabla 8: Relación de Estados de FSM CSL Modo Recepción y Funciones de Actividades de Excepción

Dada la extensión en el código fuente de las distintas actividades, se adjunta el código fuente como anexo a la presente memoria para su consulta (ver **Anexo C – Código Fuente de Actividades CSL FSM**).

6.5.8.2 Maquina de Estados CSL en Modo Transmisión

Al igual que se ha realizado en el modo de recepción y tal y como se ha descrito en la fase de análisis y diseño, la máquina de estados CSL en modo transmisión se encuentra dividida en tres fases de operación: aviso de envío, transmisión y reconocimiento (ver **apartado 6.4.6.2 - Maquina de Estados CSL en Modo Transmisión**).

Para cada una de estas fases, se han definido y agrupado una serie de estados y funciones asociadas, representadas gráficamente en la **Ilustración 43: Máquina de Estados CSL en Modo Transmisión**, resumida a continuación en la siguiente tabla:

Fase	Estado	Función
Aviso de Envío	S_CSLTXWAKEUPOFFSET	activity_csl_wakeup_ti1
	S_CSLTXWAKEUPPREPARE	activity_csl_wakeup_ti2
	S_CSLTXWAKEUPREADY	activity_csl_wakeup_ti2
	S_CSLTXWAKEUPDELAY	activity_csl_wakeup_ti3
	S_CSLRXWAKEUP	activity_csl_wakeup_ti4
Transmisión	S_CSLTXDATAPREOFFSET	activity_csl_wakeup_ti5
	S_CSLTXDATAOFFSET	activity_csl_data_ti1
	S_CSLTXDATAPREPARE	activity_csl_data_ti2
	S_CSLTXDATAREADY	activity_csl_data_ti2
	S_CSLTXDATADELAY	activity_csl_data_ti3
	S_CSLTXDATA	activity_csl_data_ri4
Reconocimiento	S_CSLRXACKOFFSET	activity_csl_data_ti5
	S_CSLRXACKPREPARE	activity_csl_data_ti6
	S_CSLRXACKREADY	activity_csl_data_ti6
	S_CSLRXACKLISTEN	activity_csl_data_ti7
	S_CSLRXACK	activity_csl_data_ti8
	S_CSLTXPROC	activity_csl_data_ti9

Tabla 9: Relación de Estados de FSM CSL Modo Transmisión y Funciones de Actividades

Adicionalmente, en algunos estados se gestionan distintas situaciones de excepción las cuales son tratadas a través de una función asociada y que tienen como objetivo principal el tratamiento del evento concreto, el registro de la situación, la limpieza de recursos utilizados, y el reinicio de la máquina de estados a su valor por defecto (SLEEP).

Fase	Estado	Función
Aviso de Envío	S_CSLTXWAKEUPOFFSET	-
	S_CSLTXWAKEUPPREPARE	activity_csl_wakeup_tie1
	S_CSLTXWAKEUPREADY	-
	S_CSLTXWAKEUPDELAY	activity_csl_wakeup_tie2
	S_CSLTXWAKEUP	activity_csl_wakeup_tie3
Transmisión	S_CSLTXDATAPREOFFSET	-
	S_CSLTXDATAOFFSET	-
	S_CSLTXDATAPREPARE	activity_csl_data_tie1
	S_CSLTXDATAREADY	-
	S_CSLTXDATADELAY	activity_csl_data_tie2
	S_CSLTXDATA	activity_csl_data_tie3
Reconocimiento	S_CSLRXACKOFFSET	-
	S_CSLRXACKPREPARE	activity_csl_data_tie4
	S_CSLRXACKREADY	-
	S_CSLRXACKLISTEN	activity_csl_data_tie5
	S_CSLRXACK	activity_csl_data_tie6
	S_CSLTXPROC	-

Tabla 10: Relación de Estados de FSM CSL Modo Transmisión y Funciones de Actividades de Excepción

Dada la extensión en el código fuente de las distintas actividades, se adjunta el código fuente como anexo a la presente memoria para su consulta (ver **Anexo C – Código Fuente de Actividades CSL FSM**).

7. DESPLIEGUE Y PRUEBAS

Una vez realizada la implementación del modo de funcionamiento CSL de acuerdo al diseño realizado, es necesario llevar a cabo la compilación del código fuente, la generación del firmware y su despliegue en las motas para el inicio del proceso de validación y verificación funcional.

En este sentido, en los próximos apartados se describirá brevemente los requerimientos de entorno necesarios para la correcta realización de estas acciones y llevar a cabo con éxito tanto la compilación y despliegue del firmware como la depuración del comportamiento funcional respecto al comportamiento esperado.

Para ello, se definirá y aplicará un plan de pruebas orientadas a la verificación y validación funcional de dicho comportamiento.

7.1. REQUERIMIENTOS DE ENTORNO

Los requerimientos de entorno hacen referencia a los elementos necesarios para la realización de las labores de desarrollo, compilación y despliegue, adicionales a los recursos de desarrollo del proyecto descritos en la **Tabla 2: Recursos Necesarios para el Desarrollo del Proyecto**.

Estos recursos adicionales comprenden básicamente los siguientes elementos:

- **Código Fuente OpenWSN**: La implementación CSL realizada en el trabajo final de máster se encuentra integrada dentro del código fuente de OpenWSN siendo necesario disponer de una copia actualizada desde los repositorios centrales https://github.com/openwsn-berkeley/openwsn-fw usando la herramienta GIT.
  A continuación se indica la secuencia de acciones para la descarga inicial del código fuente asociado al firmware desplegado sobre las motas (siguientes actualizaciones serían realizadas de igual manera pero con la operación *git pull*).

```
# mkdir OpenWSN_csl
# git clone https://github.com/openwsn-berkeley/openwsn-fw
```

Ilustración 67: Descarga de Código Fuente de OpenWSN desde Repositorios GIT

- **Scons**: Se trata de una herramienta de construcción automático de software. Se encuentra implementada en lenguaje Python como un script y un conjunto de módulos.

  A continuación se indica la secuencia de acciones (en un sistema Linux) para la instalación del software *scons* así como sus dependencias.

```
# sudo python-dev python-pip bottle PyDispatcher
# sudo apt-get install scons
```

Ilustración 68: Instalación de Herramienta *Scons* para la Construcción Automático de Software

- **GNU ARM Gcc toolchain**: Una vez instalado el software general, es necesario instalar el *toolchain* requerido por las distintas motas. En el caso de la mota OpenMote-CC2538, contiene un chip con un microcontrolador ARM M3.

  ARM mantiene un *toolchain* completo para la construcción y depuración de *firmware* basado en microcontroladores ARM. Este *toolchain* proporciona la capacidad de construir el firmware OpenWSN, cargarlo en el hardware, e iniciar el depurador.

  Los pasos para su instalación son básicamente la descarga del paquete desde la web https://launchpad.net/gcc-arm-embedded (la versión utilizada ha sido la versión 4.7 2013 Q3), la descompresión y desempaquetado del fichero, y la modificación de la variable PATH para incluir la localización del *toolchain* en la realización de las distintas acciones.

  A continuación se muestra la secuencia de acciones (en un sistema Linux) para la instalación del *toolchain* así como sus dependencias.

```
# cp   gcc-arm-none-eabi-4_7-2013q3-20130916-linux.tar.bz  /opt
# cd /opt
# bunzip gcc-arm-none-eabi-4_7-2013q3-20130916-linux.tar.bz
# tar xvf gcc-arm-none-eabi-4_7-2013q3-20130916-linux.tar
```

Ilustración 69: Instalación de Herramienta *toolchain* ARM Gcc

  En el fichero de inicio (*profille*) del usuario, deberá modificarse la variable PATH tal y como se muestra a continuación:

```
ARM_GCC_HOME=/opt/gcc-arm-none-eabi-4_7-2013q3
export ARM_GCC_HOME

PATH=$JAVA_HOME/bin:$ARM_GCC_HOME/bin:$PATH
export PATH
```

Ilustración 70: Modificación en Fichero de Inicio del Usuario para Actualización de Variable PATH

Adicionalmente a las tareas anteriores, es necesario añadir el usuario del sistema al grupo *dialout* para simplificar el acceso al puerto serie.

- **SEGGER Jlink Software**: Este elemento, disponible para descargar desde la dirección https://www.segger.com/jlink-software.html, representa el servidor de depuración (*gdb server*) así como las herramientas necesarias para el seguimiento y análisis del comportamiento del código fuente.

  A continuación se muestra la secuencia de acciones (en un sistema Linux) para la instalación del software de depuración *Segger Jlink*.

```
# sudo dpkg –i jlink_4.92_i386.deb
```

Ilustración 71: Instalación de Software de Depuración *Segger JLink*

La instalación del producto quedará instalada en el servidor bajo la ruta */opt/SEGGER/JLink*.

- **Eclipse CDT**: Eclipse es un entorno integrado de desarrollo con soporte para distintos lenguajes de programación y disponible para descarga desde la dirección http://www.eclipse.org/downloads. La instalación de este componente es sencilla, la cual podemos resumir en los siguientes pasos:
  - Descarga de *Eclipse for C/C++ developers.*
  - Descompresión en carpeta /opt/eclipse. Opcionalmente, es posible crear un enlace directo en el Escritorio para su rápido acceso.
  - Inicio de Eclipse y acceso a repositorio de descarga e instalación de software a través *Help→ Install New Software*.
  - Instalación del plugin "GNU ARM C/C++ Cross Development Tools" desde la dirección http://gnuarmeclipse.sourceforge.net/updates. Este plugin incluye

los siguientes componentes entre los cuales se encuentran el plugin de depuración J-Link:

- GNU ARM C/C++ Cross Compiler
- GNU ARM C/C++ Documentation (Placeholder)
- GNU ARM C/C++ Freescale Project Templates
- GNU ARM C/C++ Generic Cortex-M Project Template
- GNU ARM C/C++ J-Link Debugging
- GNU ARM C/C++ OpenOCD Debugging
- GNU ARM C/C++ Packs
- GNU ARM C/C++ STM32Fx Project Templates

- Reinicio de Eclipse.

Adicionalmente a los puntos anteriores, será preciso realizar además las siguientes configuraciones adicionales sobre la configuración de Eclipse:

- En el menú ***Window → Preferences → C/C++ → Build → Environment***, añadir una nueva variable de entorno denominada SCONS cuyo valor sea el comando "*scons*".
- En el menú ***Window → Preferences → Run/Debug → String Substitution***, establecer la variable *jlink_path* a la ruta de instalación de SEGGER J-Link, y la variable *jlink_gdbserver* al valor JLinkGDBServer.

7.2. PROCEDIMIENTO DE COMPILACIÓN Y DESPLIEGUE

La implementación del modo de funcionamiento CSL se encuentra configurada bajo un nuevo proyecto Eclipse creado durante el desarrollo del proyecto y configurado adecuadamente para que se importen los ficheros necesarios.

Este proyecto ha sido creado como una copia del directorio "***Eclipse-format***" ubicado en la ruta **${openwsn-fw}/projects/OpenMote-CC2538**, con nombre ***OPENWSNCSL***. Para su importación dentro del espacio de trabajo de Eclipse, es necesario desplazarse a la ventana "Project Explorer" y seleccionar la opción "*Import*" accesible desde el menú disponible tras pulsar botón derecho del ratón.

Seleccionando a continuación la opción ***General → Existing Projects into Workspace*** y desplazándonos hasta el directorio raíz del proyecto (***${openwsn-fw}/projects/OpenMote-CC2538/OPENWSNCSL*)**, importaremos el proyecto dentro de la herramienta desde la cual podremos llevar a cabo los procesos de compilación y despliegue.

Una vez disponible el proyecto dentro del entorno de desarrollo, el **procedimiento de compilación** estará disponible a través de la definición de *Targets* ya disponibles en el proyecto.

De este modo, desde la pestaña "*Make Targets"* disponible en los menús de la ventana derecha del entorno de desarrollo, compilaremos el proyecto global utilizando el *target* ***oos_openwsn***. El resultado de la operación podrá ser consultado a través de la pestaña "*Console*" disponible en la ventana inferior del IDE.

Una vez construido correctamente el proyecto, el **procedimiento de despliegue** implicará la realización de las siguientes acciones:

- Desde el menú ***Run → Debug Configurations → GDB SEGGER J-Link Debugging***, accede a la configuración ***OpenWSN Debug***.
- En la pantalla principal, desplazarse hasta la opción "***C/C++ Application***" y seleccionar "***Search Project***". En este punto, seleccionaremos el binario a desplegar en la mota, tras lo cual pulsaremos la opción "***Debug***".

Realizadas estas operaciones, el binario será subido a la mota (previamente conectada al equipo), pudiendo iniciar el establecimiento de puntos de ruptura y depuración funcional.

7.3. PROCEDIMIENTO DE DEPURACIÓN

El procedimiento de depuración tiene como cometido la ejecución controlada de una aplicación con el fin de analizar cada instrucción ejecutada, inspeccionar variables y localizar deficiencias o bugs existentes en el código fuente, en este caso, en tiempo de ejecución sobre el entorno hardware real (mota OpenMote-CC2538).

Para ello, cabe distinguir dos procedimientos fundamentales de los cuales se ha hecho uso para llevar a cabo la depuración del código fuente CSL desarrollado.

7.3.1. DEPURACIÓN DE CÓDIGO FUENTE (ECLIPSE IDE)

El entorno de desarrollo (Eclipse IDE) posibilita la realización del proceso de depuración de código una vez realizado el procedimiento de despliegue sobre el hardware real tal y como se ha descrito en el **apartado 7.2 - Procedimiento de Compilación y Despliegue**.

La depuración del código fuente es realizada a través de la definición de puntos de ruptura (***breakpoints***) dentro del código fuente. Este punto de ruptura establece la línea específica dentro del código fuente en la cual se desea que se detenga el hilo de ejecución de la aplicación con el fin de verificar e inspeccionar los valores de las variables y registros del sistema en ese instante.

El procedimiento para agregar un ***breakpoint*** es sencillo, requiriendo simplemente el posicionamiento en la parte izquierda de la línea del código fuente en la cual deseamos que se detenga el depurador, realizando doble-click sobre ella. Como resultado, Eclipse mostrará un círculo azul sobre dicha línea indicando su establecimiento.

En la siguiente ilustración se muestra la definición de un ***breakpoint*** asociado a una línea de código así como el listado de todos los ***breakpoints*** definidos y su estado activado o desactivado.

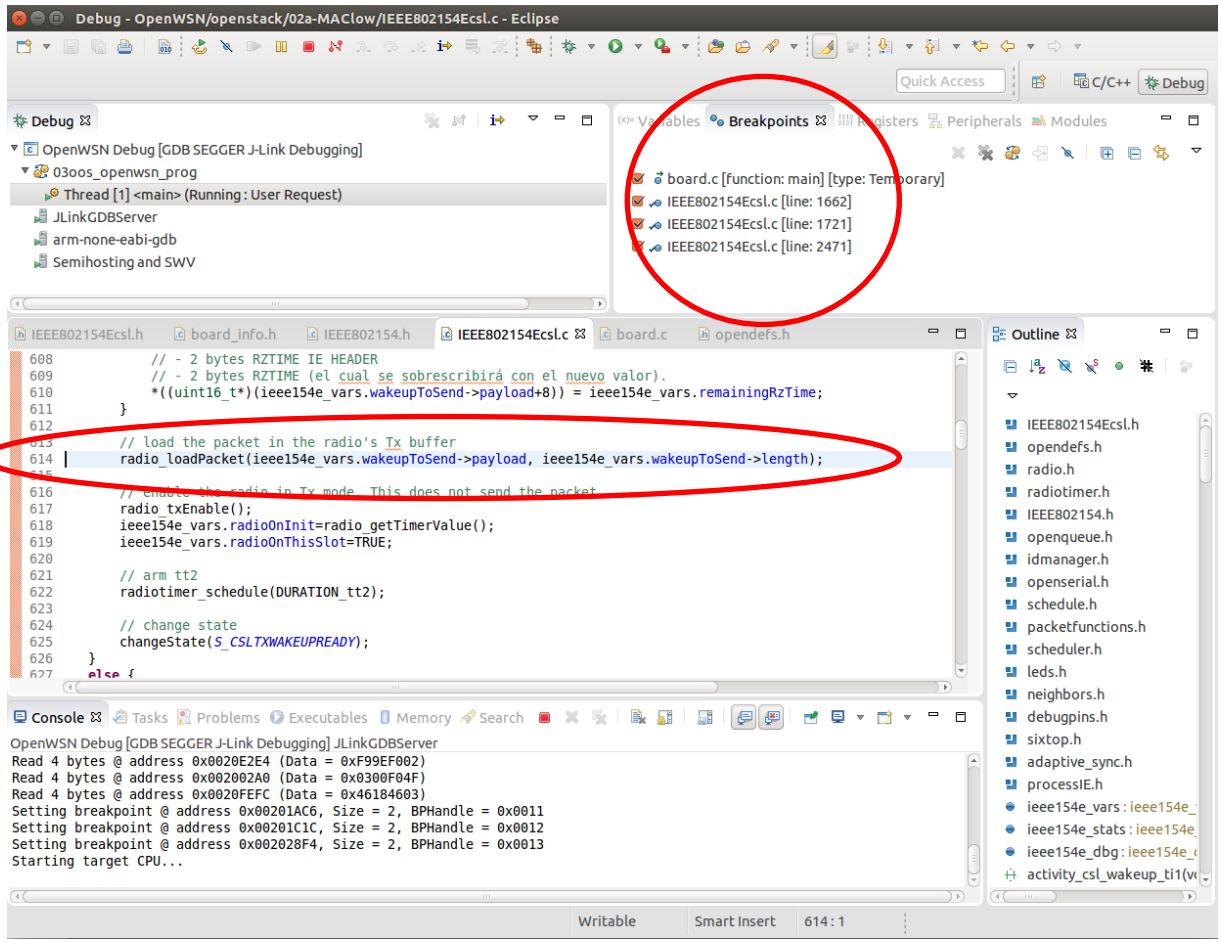


Ilustración 72: Definición de Breakpoints Sobre Código Fuente desde Eclipse IDE

Una vez ejecutado el código fuente y alcanzada la línea de código asociada al ***breakpoint***, el entorno detendrá la ejecución, registrando dicha línea con color verde tal y como se muestra en la siguiente ilustración.

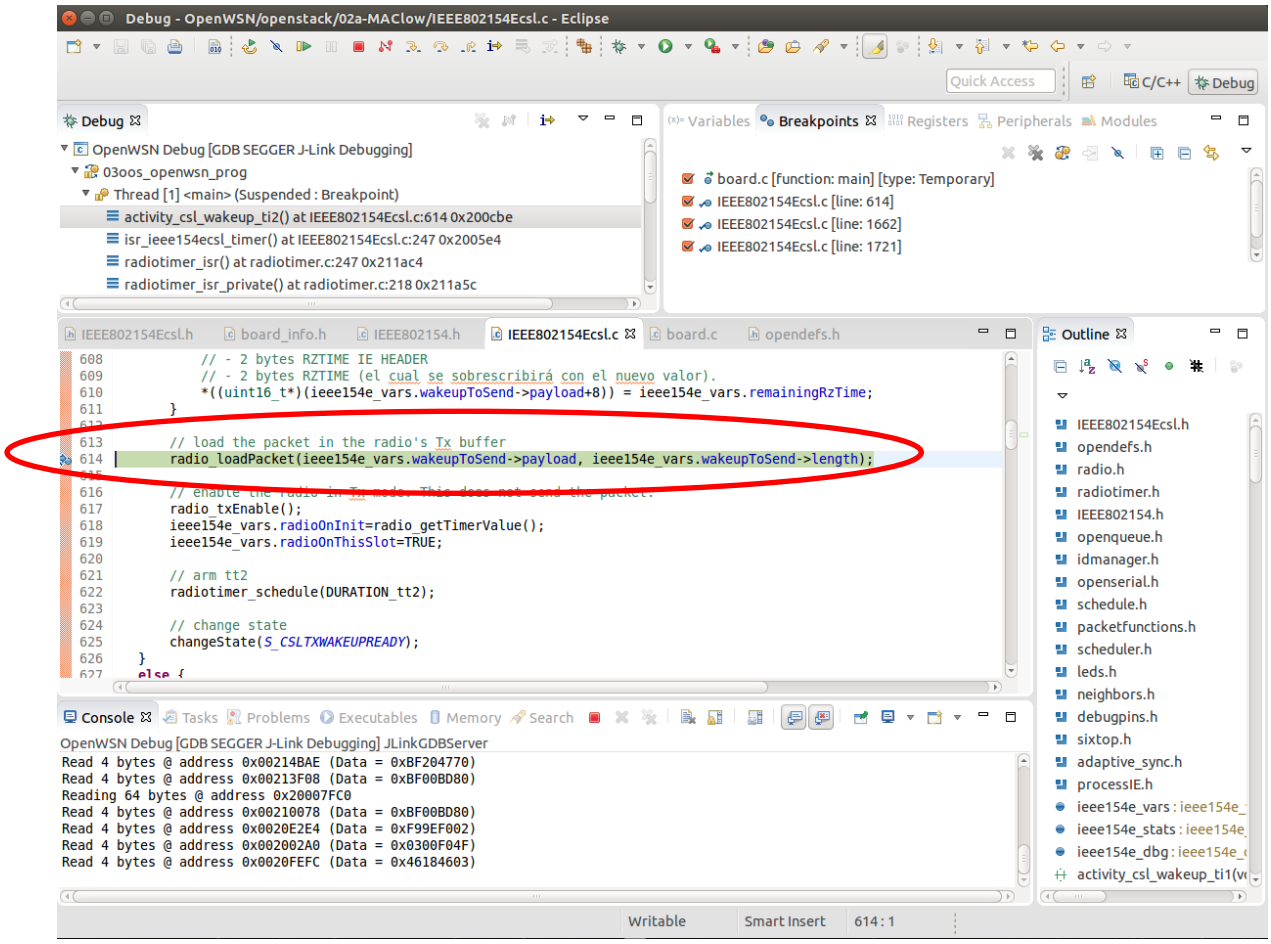


Ilustración 73: Registro de Detención de la Ejecución de Código por Presencia de Breakpoint

En este punto, las acciones más importantes que será posible realizar serán las siguientes:

- ***Step Into*** (o pulsar F5): El código continuará ejecutándose instrucción a instrucción. Si el depurador encuentra la llamada a una función, irá a la primera instrucción de dicha función (entrará en ella).
- ***Step Over*** (o pulsar F6): El código continuará ejecutándose instrucción a instrucción pero en este caso y a diferencia del caso anterior, si el depurador encuentra una función, no entrará en ella y continuará con la siguiente instrucción del código existente tras ella.
- ***Step Return*** (o pulsar F7): Permite retornar de la llamada de una función, saliendo de ella y haciendo que el depurador encuentre automáticamente el final de la función y se sitúe fuera de ella.

- ***Resume*** (o pulsar F8): El depurador continuará ejecutándose, parando únicamente de nuevo en el caso de encontrar un nuevo *breakpoint*.

Finalmente, en el momento que el código se encuentra detenido, es posible inspeccionar los valores de las variables y registros con el fin de verificar y validar el contenido de éstos. Para ello, podemos posicionarnos directamente con el ratón sobre el valor de la variable o bien a través del menú de variables existente en el entorno, tal y como se muestra en la siguiente ilustración:

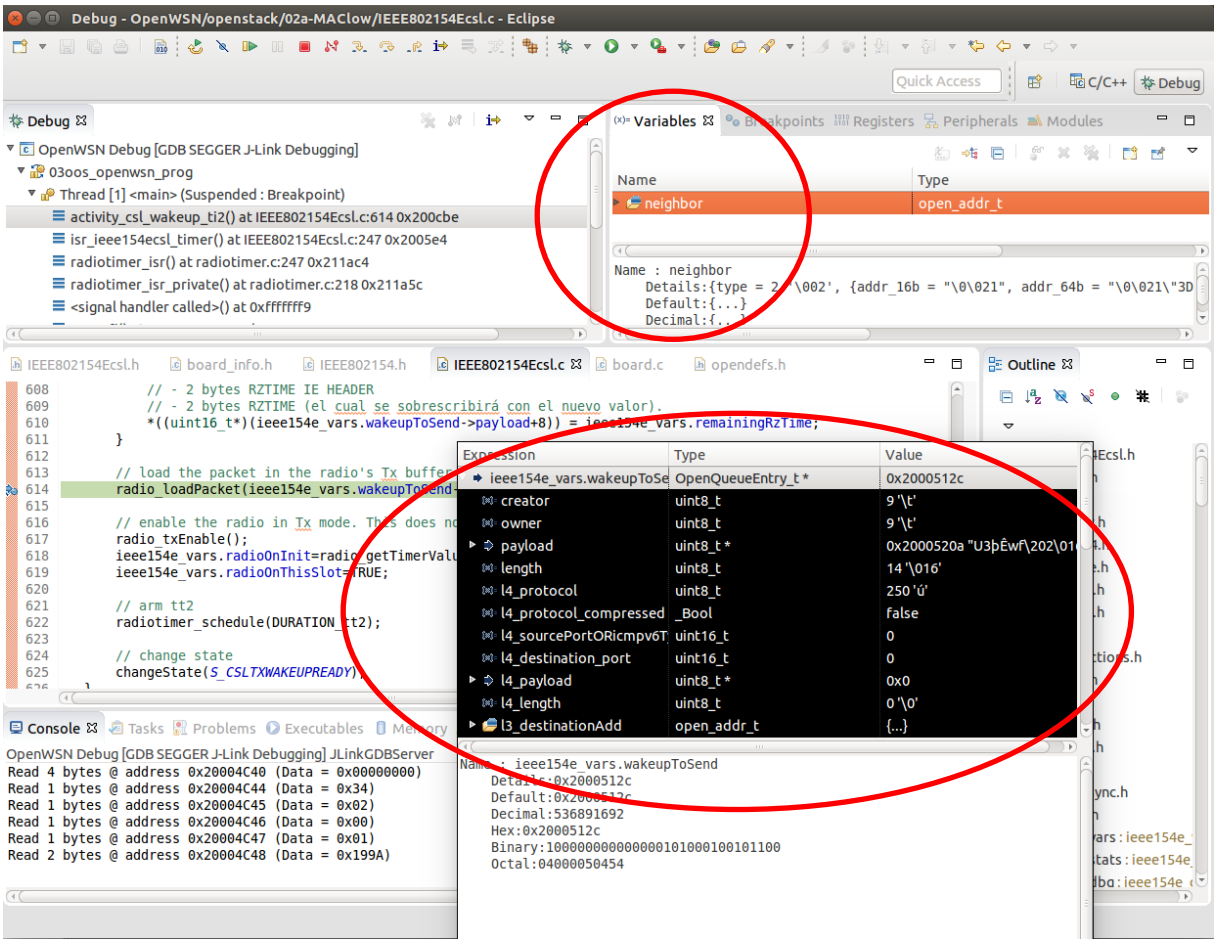


Ilustración 74: Inspección de Variables y Registros en Depuración desde Eclipse IDE

7.3.2. DEPURACIÓN DE LA COMUNICACIÓN (WIRESHARK)

Un aspecto importante dentro del proceso de depuración del trabajo está relacionado con el proceso de comunicación entre las motas y la verificación principal de los datos enviados desde el nodo transmisor.

Para ello, se ha empleado un mecanismo de depuración de la comunicación basado en la utilización de un *sniffer* (concretamente la solución *wireshark).* Como resultado, esto ha

permitido validar que las tramas enviadas por la mota actuando en modo transmisor son válidas y correctas en contenido y estructura.

El procedimiento seguido para la configuración y establecimiento de la depuración de la comunicación ha sido el siguiente:

1. **Configuración de la mota en modo transmisión en la placa OpenBattery**. De acuerdo a la configuración del plan de pruebas descrito en el próximo apartado, esta mota enviará periódicamente (cada 2 segundos) un ciclo completo de tramas, esto es, la trama de datos precedida de la secuencia de tramas *wake*-up.
   A este respecto, es importante asegurarse que la mota transmisora hace uso del canal 26 dado que es el canal en el cual escucha el *sniffer*.
2. **Configuración de la mota en modo recepción en la placa OpenBase y con la aplicación sniffer**. Esto permitirá cargar en la mota la aplicación *sniffer*, disponible bajo el directorio "*projects/test-sniffer/src*" dentro del repositorio de firmware de OpenMote (https://github.com/OpenMote/firmware).
   Previo a la carga de la aplicación *sniffer* en la mota, es necesario iniciar el depurador en modo comando. Para ello, primeramente conectamos el JTAG a la placa OpenBattery, y a continuación ejecutamos la siguiente sentencia desde otra consola:

```
# make TARGET=cc2538 jlink
```

Ilustración 75: Conexión JTAG desde Línea de Comandos

   El proceso de carga de la aplicación *sniffer* en la mota receptora implicará la realización de las siguientes acciones:

```
# cd projects/test-sniffer/src
# make TARGET=cc2538
# make TARGET=cc2538 load
```

Ilustración 76: Generación y Despliegue de la Aplicación Sniffer en OpenMote-CC2538

3. **Conexión Ethernet de la placa OpenBase**. Este paso es necesario con el fin de poder capturar los paquetes recibidos desde *wireshark* dado que la aplicación *sniffer* realiza una retransmisión de cada paquete recibido a través de la interfaz radio por la interfaz Ethernet.

4. **Captura de paquetes desde Wireshark**. Desde otro equipo conectado a la misma red, realizamos la captura y análisis de los paquetes empleando para ello la aplicación *wireshark*.

En la siguiente ilustración se muestra un esquema operativo de conectividades para la captura y validación de tramas *wake-up* enviadas.

En este esquema se observa la mota transmisora en el lado izquierdo de la imagen, desde la cual se realiza el envío de tramas *wake-up*, así como la mota receptora en el centro en la cual se ha desplegado el *sniffer* mediante JTAG y que se encargará de reenviar los paquetes recibidos por la interfaz radio a través de la interfaz Ethernet desde donde es posible capturarlos con *wireshark*.

Ilustración 77: Esquema de Conectividades para Validación de Tramas Wake-Up mediante Wireshark

7.4. PLAN DE PRUEBAS

El presente apartado tiene como objetivo proporcionar una descripción del **escenario de pruebas empleado para la validación y verificación funcional** de los desarrollos realizados así como las modificaciones necesarias que ha sido necesario implementar en el código fuente y los mecanismos de depuración empleados durante el proceso.

7.4.1. ESCENARIO DE PRUEBAS

El escenario de pruebas representa el marco arquitectural encargado de definir las diferentes condiciones bajo las cuales operará el sistema CSL desarrollado, con el objetivo de realizar y validar el conjunto de casos de prueba funcionales y no funcionales asociados.

En este sentido, en la siguiente ilustración se recoge el escenario de pruebas empleado en la evaluación de los desarrollos CSL.

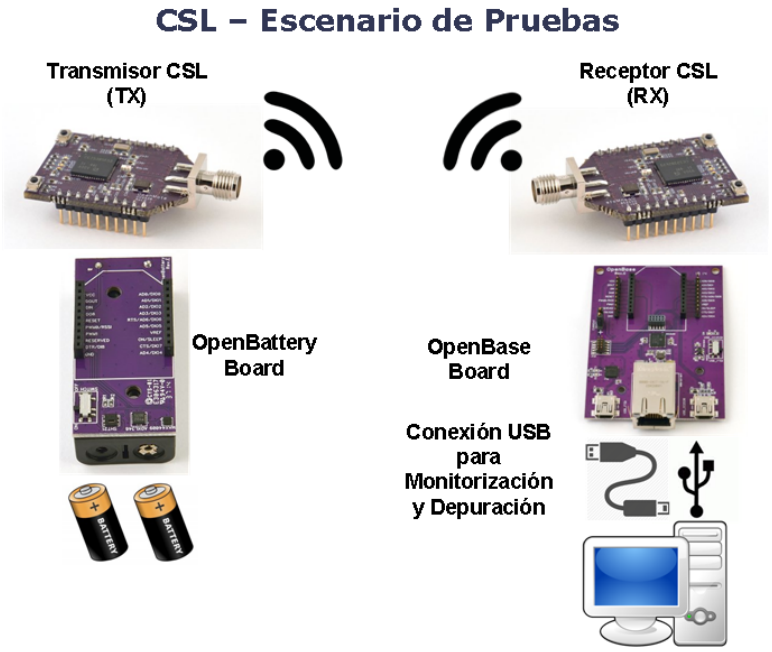


Ilustración 78: Escenario de Pruebas para la Verificación y Validación de la Implementación CSL

Tal y como se puede observar en la ilustración anterior, el escenario de pruebas se compone de dos bloques principales, destinados cada uno de ellos a verificar la transmisión o recepción de datos mediante el modo CSL.

- **Bloque Transmisor (TX)**: Este bloque se encuentra formado por una mota OpenMote-CC2538 sobre la cual se ha realizado previamente el despliegue del firmware desarrollado a través del procedimiento descrito en el **apartado 7.2 - Procedimiento de Compilación y Despliegue**.

  El *firmware* desarrollado ha sido extendido con el fin de incluir el código necesario para el envío periódico de tramas, realizando un encendido del led naranja a la finalización del proceso completo de transmisión de la trama de datos lo cual incluye la correspondiente transmisión previa de la secuencia de tramas *wake-up* (*wake-up sequence*).

Esta mota se encuentra instalada sobre una placa *OpenBattery*, trabajando de manera autónoma a través de su alimentación por pilas.

- **Bloque Receptor (RX)**: Este bloque se encuentra formado por una mota OpenMote-CC2538 conectada a la placa OpenBase sobre la cual se ha desplegado el *firmware* desarrollado, siguiendo el mismo procedimiento que en el caso anterior.

  El *firmware* desarrollado, al igual que en el caso anterior, ha sido modificado y extendido para incluir el código asociado a la recepción completa de la trama de datos lo cual incluye la recepción previa de la trama de wake-up y la espera del tiempo indicado hasta la recepción de la trama de datos.

  En este sentido y dentro del marco de estas pruebas, se ha modificado el contenido del mecanismo de notificación a los niveles superiores (método *notif_receive*) con el fin de registrar la correcta recepción a través del encendido del led naranja.

  Dado que tanto el proceso de transmisión como recepción realizan un encendido del led naranja (el led verde está asociado a la radio y el led rojo ha sido mantenido para notificar errores) y con el fin de evitar posibles errores de interpretación, el *firmware* desplegado en esta mota incluye la desactivación de los temporizadores asociados a la transmisión periódica, eliminando con ello cualquier posible relación del encendido del led con el proceso de transmisión, el cual nunca se ejecutará en la mota receptora.

7.4.2. MODIFICACIONES AL CODIGO FUENTE

A continuación en los siguientes apartados se procederá a describir las principales modificaciones y extensiones realizadas sobre el código fuente para el establecimiento del escenario de pruebas descrito anteriormente.

Aunque estas modificaciones se encuentran todas ellas contenidas bajo un único repositorio de código, se procederá a su detalle de manera agrupada en función de si está relacionado con el proceso de transmisión o recepción.

7.4.2.1 Nodo Transmisor (TX)

Las pruebas para la realización del nodo transmisor se basan en la generación periódica de tramas de datos en la cola de envío con el fin de simular solicitudes de transmisión desde los niveles superiores.

Para ello, se ha definido un nuevo temporizador usando las funcionalidades de *virtual_timer* existentes en OpenWSN

A continuación, en las siguientes ilustraciones se muestra la definición del temporizador, establecido con una periodicidad de 2 segundos, así como el contenido del método *callback* asociado (**IEEE802154Ecsl.c**).

```
..
/**
  \brief This function initializes this module.
  Call this function once before any other function in this module, possibly during boot-up.
*/
// [CSL]: Modificaciones sobre método inicial referidas a los temporizadores y al estado inicial.
void ieee154e_init() {
...
   // [CSL-TEST]: set timer for callback to add packet to queue for testing CSL TX (every 5 seconds)
  ieee154e_vars.cslTxTestTimer = opentimers_start(2000,
                                                  TIMER_PERIODIC,
                                                  TIME_MS,
                                                  isr_ieee154ecsl_addPacketToQueueForTestingCslTx_cb);
...
}
...
```

Ilustración 79: Temporizador para Generación de Tramas en Función *ieee154e_init* (**IEEE802154Ecsl.c**)

La *callback* **isr_ieee154ecsl_addPacketToQueueForTestingCslTx_cb** generará un nuevo paquete en la cola para su transmisión en el caso que no haya ya una trama pendiente asociada al mismo nodo remoto. En caso que dicho paquete de datos ya exista, no se realizará ninguna acción con el fin de no saturar la cola de paquetes.

```
...
// [CST-TEST]: test timer interrupt callback to put a new packet on queue for testing CSL TX mode.
void isr_ieee154ecsl_addPacketToQueueForTestingCslTx_cb () {
  OpenQueueEntry_t* pkt;
  open_addr_t neighbor;

  // Simulate a ficticious address for CSL testing.
  neighbor.addr_64b[0]=0x00;
  neighbor.addr_64b[1]=0x11;
  neighbor.addr_64b[2]=0x22;
  neighbor.addr_64b[3]=0x33;
```

```
neighbor.addr_64b[4]=0x44;
neighbor.addr_64b[5]=0x55;
neighbor.addr_64b[6]=0x66;
neighbor.addr_64b[7]=0x77;
neighbor.type=ADDR_64B;

// Switch off all the leds.
leds_all_off();

// CREATE DATA PACKET ONLY IF NOT ALREADY EXISTS ON QUEUE.
if (openqueue_macGetDataPacket(&neighbor) == NULL) {
        // get freebuffer.
        pkt = openqueue_getFreePacketBuffer(COMPONENT_IEEE802154E);
        if(pkt==NULL) {
                // registro del error & fin de operaciones.
                openserial_printError(COMPONENT_IEEE802154E,ERR_NO_FREE_PACKET_BUFFER,
                                        (errorparameter_t)0, (errorparameter_t)0);
                endOps();
                return;
        }

        // Declaración de propiedad sobre el paquete.
        pkt->creator = COMPONENT_SIXTOP_TO_IEEE802154E;
        pkt->owner   = COMPONENT_SIXTOP_TO_IEEE802154E;

        // Frame type (data frame).
        pkt->l2_frameType=IEEE154_TYPE_DATA;

        // Neighbor address.
        memcpy(&(pkt->l2_nextORpreviousHop),&neighbor,sizeof(open_addr_t));

        // TX retries
        pkt->l2_retriesLeft = TXRETRIES;

        // record this packet's dsn (for matching the ACK)
        pkt->l2_dsn = ieee154e_vars.cslDSN++;

        // this is a new packet which I never attempted to send
        pkt->l2_numTxAttempts = 0;

        // transmit with the default TX power
```

```
            pkt->l1_txPower = TX_POWER;

            // add a IEEE802.15.4 header
            ieee802154_prependHeader(pkt,
                                     pkt->l2_frameType,
                                     IEEE154_IELIST_NO,
                                     IEEE154_FRAMEVERSION,
                                     IEEE154_SEC_NO_SECURITY,
                                     pkt->l2_dsn,
                                     &(pkt->l2_nextORpreviousHop)
                                   );

            // reserve space for 2-byte CRC
            packetfunctions_reserveFooterSize(pkt,2);
  }
  return;
}
...
```

Ilustración 80: Función callback *isr_ieee154ecsl_addPacketToQueueForTestingCslTx_cb* (**IEEE802154Ecsl.c**)

Tal y como se puede observar en el método anterior**, se ha definido una dirección de envío directamente sobre el código para la realización de las pruebas**. El motivo de esta decisión se basa en dos razones principales: por un lado, la ausencia de un sistema externo (DAG) para la identificación y distribución topológica de la red (DAG) con sus miembros y direcciones MAC asociadas. Por otro lado, la simplicidad en la realización de las pruebas dado que la definición de esta dirección común simplifica y unifica la realización de las pruebas al verificar la transmisión y recepción de paquetes con dicha dirección en ambos procesos en lugar de con la dirección propia.

Así mismo, **se ha modificado la actividad de la FSM asociada al evento fin de trama de datos** (***activity_csl_data_ti5***) en el proceso de envío de la trama de datos con el fin de verificar que todo ha sido correcto y proceder al encendido del led naranja para notificar visualmente la correcta transmisión. Del mismo modo y debido a la imposibilidad de disponer de dos placas OpenBase para la depuración simultánea del transmisor y receptor, este método fuerza que no haya un reconocimiento (ACK) de la trama de datos (a pesar de que la máquina de estados desarrollada así lo contempla), dando por finalizado el proceso de transmisión.

A continuación se adjunta las partes del código fuente modificadas en la actividad en la cual se puede identificar de manera sencilla las modificaciones realizadas a través del identificador de comentarios de prueba (CSL-TEST):

```
...
/**
 \brief Activity for CSL TX stage [data ri5].
  This method is invoked from ISR-mode "ieee154ecsl_endOfFrame" function when a end of frame event fires
  while state = S_CSLTXDATA.
  The functionality is to change state, cancel #tt4, and notify upper layer and schedule about successful TX. In
  case ACK required, start process for receive ACK frame (arm #tt5).
*/
port_INLINE void activity_csl_data_ti5(PORT_RADIOTIMER_WIDTH capturedTime) {
   bool listenForAck;

   // change state
   changeState(S_CSLRXACKOFFSET);

   // cancel tt4
   radiotimer_cancel();


 // turn off the radio
   radio_rfOff();
   ieee154e_vars.radioOnTics+=(radio_getTimerValue()-ieee154e_vars.radioOnInit);

   // record the captured time
   //ieee154e_vars.lastCapturedTime = capturedTime;

   // decides whether to listen for an ACK
   if (packetfunctions_isBroadcastMulticast(&ieee154e_vars.dataToSend->l2_nextORpreviousHop)==TRUE) {
      listenForAck = FALSE;
   } else {
      listenForAck = TRUE;
   }

   // [CSL-TEST]: testing code for toggling orange led to indicate OK TX.
   if((ieee154e_vars.dataToSend->l2_frameType == IEEE154_TYPE_DATA) &&
               (ieee154e_vars.dataToSend->owner == COMPONENT_SIXTOP_TO_IEEE802154E)) {
              leds_sync_blink();
              listenForAck = FALSE;
```

```
   }
   // [CSL-TEST]: end test code

   if (listenForAck==TRUE) {
      // arm tt5
      radiotimer_schedule(DURATION_tt5);
   } else {
      // indicate succesful Tx to schedule to keep statistics
      schedule_indicateTx(&ieee154e_vars.asn,TRUE);

      // [CSL-TEST]: comment notification
      // indicate to upper later the packet was sent successfully
      //notif_sendDone(ieee154e_vars.dataToSend,E_SUCCESS);
      // reset local variable
      //ieee154e_vars.dataToSend = NULL;
      // [CSL-TEST]: end test code

      // abort
      endOps();
   }
}
...
```

Ilustración 81: Modificaciones para Pruebas sobre Actividad *activity_csl_data_ti5* (**IEEE802154Ecsl.c**)

Finalmente y como se ha indicado en el apartado **6.4.5.3 - Temporizador de Transmisión CSL**, el sistema de transmisión encargado de verificar periódicamente la presencia de datos pendiente de envío, hace uso del planificador para la vinculación entre direcciones de nodos remotos y datos asociados pendientes de envío, procediendo a aplicar un mecanismo *round-robin* en su procesamiento.

En este sentido y para la realización de las pruebas, se ha modificado el perfil inicial de planificación (*schedule*) definiendo una planificación de nueve slots donde solamente el primero de ellos es de tipo transmisión (CELLTYPE_TX) siendo el resto de tipo recepción (CELLTYPE_RX). Esto implicará que si la frecuencia actual de verificación de datos pendientes de envío es 60 ms y el planificador dispone de 9 slots, el tiempo máximo de envío de un paquete será aproximadamente de 540 ms.

A continuación se adjunta el código fuente de las modificaciones realizadas sobre el planificador para la realización de las pruebas (**schedule.c**).

```
/**
\brief Initialize this module.
\post Call this function before calling any other function in this module.
*/
void schedule_init() {
...
  // [CSL TEST]: Only first cell of nine cells in slot is CELLTYPE_TX
  memset(&temp_neighbor,0,sizeof(temp_neighbor));
  // Simulate a ficticious address for CSL testing.
  temp_neighbor.addr_64b[0]=0x00;
  temp_neighbor.addr_64b[1]=0x11;
  temp_neighbor.addr_64b[2]=0x22;
  temp_neighbor.addr_64b[3]=0x33;
  temp_neighbor.addr_64b[4]=0x44;
  temp_neighbor.addr_64b[5]=0x55;
  temp_neighbor.addr_64b[6]=0x66;
  temp_neighbor.addr_64b[7]=0x77;
  temp_neighbor.type=ADDR_64B;

  schedule_addActiveSlot(
      running_slotOffset,     // slot offset
      CELLTYPE_TX,            // type of slot
      FALSE,                  // shared?
      0,                      // channel offset
      &temp_neighbor          // neighbor
  );
  running_slotOffset++;

  // el resto de tipo RX
  memset(&temp_neighbor,0,sizeof(temp_neighbor));
  for (i=0;i<8;i++) {
    schedule_addActiveSlot(
      running_slotOffset,     // slot offset
      CELLTYPE_RX,            // type of slot
      FALSE,                  // shared?
      0,                      // channel offset
      &temp_neighbor          // neighbor
    );
    running_slotOffset++;
  }
```

```
...
    // [CSL-TEST]: end test code
}
...
```

Ilustración 82: Modificaciones para Pruebas sobre Función *schedule_init* (**SCHEDULE.c**)

7.4.2.2 Nodo Receptor (RX)

Las pruebas para la realización del nodo receptor se basan en la validación de la trama wake-up recibida en el muestreo periódico producido cada *macCSLPeriod* y permanecer en estado dormido durante el tiempo indicado por el atributo rendezvous, tras lo cual se iniciará de nuevo para proceder a la recepción de la trama de datos.

En este caso, las modificaciones realizadas para la validación de este comportamiento comprende la realización de dos tareas principales. Por un lado, la modificación de las actividades invocada por la recepción final de las tramas de *wake-up* (**activity_csl_wakeup_ri5**) y datos (**activity_csl_data_ri5**) y por otro lado, la alteración de la función de notificación (**notif_receive**) invocada tras la recepción para que proceda al encendido del led indicativo.

A continuación, en la siguiente ilustración se muestra la modificación realizada en la actividad asociada a la recepción completa de la trama de *wake-up* (***activity_csl_wakeup_ti5***) la cual comprende la validación de la dirección destino recibida contra la dirección establecida en el envío en lugar de contra la dirección propia por los motivos ya indicados en el punto anterior (**IEEE802154Ecsl.c**).

```
...
/**
 \brief Activity for CSL RX Sampling stage [wake-up ri5].
  This method is invoked from ISR-mode "ieee154ecsl_endOfFrame" function when a end of frame event fires
   while state = S_CSLRXWAKEUP.
  The functionality is to change state, cancel #rt4, validate wake-up frame and destination, and arm #rt4 (rztime).
*/
port_INLINE void activity_csl_wakeup_ri5(PORT_RADIOTIMER_WIDTH capturedTime) {
...
    // [CSL-TEST]: hard-code mac address to match dest address on schedule in tx activity.
   myID.addr_64b[0]=0x00;
   myID.addr_64b[1]=0x11;
   myID.addr_64b[2]=0x22;
```

```
   myID.addr_64b[3]=0x33;
   myID.addr_64b[4]=0x44;
   myID.addr_64b[5]=0x55;
   myID.addr_64b[6]=0x66;
   myID.addr_64b[7]=0x77;
   myID.type=ADDR_64B;
   // [CSL-TEST]: end test code
...
   // Verificamos que se trata de una trama WAKE-UP, perteneciente a la misma PAN ID, y dirigida a mi.
   if(ieee802514_header.frameType==IEEE154_TYPE_MULTIPURPOSE)  {
       // [CSL-TEST]: comment and substituted due to mac address is hard-coded in tx and rx for testing.
       // if(packetfunctions_sameAddress(&ieee802514_header.dest,idmanager_getMyID(ADDR_16B)) &&
       //   packetfunctions_sameAddress(&ieee802514_header.panid,idmanager_getMyID(ADDR_PANID))) {

       packetfunctions_mac64bToMac16b(&myID,&myID16b);
       if(packetfunctions_sameAddress(&ieee802514_header.dest, &myID16b) &&
          packetfunctions_sameAddress(&ieee802514_header.panid,idmanager_getMyID(ADDR_PANID))) {
       // [CSL-TEST]: end test code
...
```

Ilustración 83: Modificaciones para Pruebas sobre Actividad *activity_csl_wakeup_ti5* (**IEEE802154Ecsl.c**)

Del mismo modo, se muestra a continuación las modificaciones llevadas a cabo en la actividad a la recepción completa de la trama de datos (***activity_csl_data_ti5***) la cual comprende igualmente la validación de la dirección destino recibida, la modificación para forzar que no se proceda al envío de la trama de reconocimiento (ACK) por los motivos de depuración ya descritos, y la invocación al método *notif_receive* para el encendido del led de recepción correcta (**IEEE802154Ecsl.c**).

```
...
/**
 \brief Activity for CSL RX Sampling stage [data ri5].
  This method is invoked from ISR-mode "ieee154ecsl_endOfFrame" function when a end of frame event fires
   while state = S_CSLRXDATA.
  The functionality is to change state, cancel #rt4, validate data frame, and transmit ACK.
*/
port_INLINE void activity_csl_data_ri5(PORT_RADIOTIMER_WIDTH capturedTime) {
   ...
     // [CSL-TEST]: Force to not send ack packet
     ieee802514_header.ackRequested = 0;
     // [CSL-TEST]: end test code
```

```
...
    // check if ack requested
    if (ieee802514_header.ackRequested==1) {
      // arm rt5
      radiotimer_schedule(DURATION_rt5);
    } else {
      // [CSL-TEST]: comment synchronization
      // synchronize to the received packet if I'm not a DAGroot and this is my preferred parent
      //if (idmanager_getIsDAGroot()==FALSE &&
      //     neighbors_isPreferredParent(&(ieee154e_vars.dataReceived->l2_nextORpreviousHop))) {
      //   synchronizePacket(ieee154e_vars.syncCapturedTime);
      //}
      // [CSL-TEST]: end test code
      // indicate reception to upper layer (no ACK asked)
      notif_receive(ieee154e_vars.dataReceived, 1);
      // reset local variable
      ieee154e_vars.dataReceived = NULL;
      // abort
      endOps();
    }
...
```

Ilustración 84: Modificaciones para Pruebas sobre Actividad *activity_csl_data_ti5* (**IEEE802154Ecsl.c**)

Por último, se muestra a continuación las modificaciones realizadas sobre el método *notif_receive* empleada para la notificación a los niveles superiores de la recepción de una nueva trama.

En este caso, se ha modificado la firma del método para incluir un nuevo parámetro de entrada denominado *action* el cual determinará si la notificación es debida a una acción correcta o incorrecta en el proceso. Esto es debido a que dicha notificación puede producirse desde otros puntos del código fuente en los cuales no se representa una correcta recepción. De este modo y a través de este parámetro, es posible determinar si la operación de recepción ha sido completamente satisfactoria o no procediendo en este caso al encendido del led naranja (recepción correcta) o led rojo (recepción incorrecta).

```
...
// Modified notif_receive signature to add action for CSL testing purposes. Packet is removed on endOps in order
// to not fill all the slots on queue due to no sixtop action is defined for receive and process incoming packets.
// Then, we comment this actions here and add remove packet on endOps method.
void notif_receive(OpenQueueEntry_t* packetReceived, uint8_t action) {

  // [CSL-TEST]: comment and add led (orange) blink if OK or led (red) toggle if KO.

  // record the current ASN
  // memcpy(&packetReceived->l2_asn, &ieee154e_vars.asn, sizeof(asn_t));

  // indicate reception to the schedule, to keep statistics
  // schedule_indicateRx(&packetReceived->l2_asn);

  // associate this packet with the virtual component
  // COMPONENT_IEEE802154E_TO_SIXTOP so sixtop can knows it's for it
  // packetReceived->owner        = COMPONENT_IEEE802154E_TO_SIXTOP;

  // post RES's Receive task
  // scheduler_push_task(task_sixtopNotifReceive,TASKPRIO_SIXTOP_NOTIF_RX);

  if (action == 1)   leds_sync_blink();
  else               leds_error_toggle();

  // [CSL-TEST]: end test code

  // wake up the scheduler
  SCHEDULER_WAKEUP();
}
...
```

Ilustración 85: Modificaciones para Pruebas sobre Función *notif_receive* (**IEEE802154Ecsl.c**)

Finalmente y como se comentó en la descripción del escenario de pruebas, el firmware desplegado en la mota receptora ha sido ligeramente modificado con el fin de desactivar el comportamiento transmisor y garantizar que cualquier encendido de led está asociado al proceso de recepción y no al proceso de transmisión.

Para ello, los cambios aplicados implican el comentado de los temporizadores asociados a dicho proceso de transmisión, indicados a continuación en la siguiente ilustración.

```
/**
  \brief This function initializes this module.
  Call this function once before any other function in this module, possibly during boot-up.
*/
// [CSL]: Modificaciones sobre método inicial referidas a los temporizadores y al estado inicial.
void ieee154e_init() {
...
   // [CSL]: set timer for checking frames on local queue to transmit.
  //  ieee154e_vars.txTimer = opentimers_start(macCSLTxChkFreq, TIMER_PERIODIC,
  //                                           TIME_TICS, isr_ieee154ecsl_txtimer_cb);

   // [CSL-TEST]: set timer for callback to add packet to queue for testing CSL TX (every 5 seconds)
  // ieee154e_vars.cslTxTestTimer = opentimers_start(2000, TIMER_PERIODIC, TIME_MS,
  //                                                 isr_ieee154ecsl_addPacketToQueueForTestingCslTx_cb);
...
}
```

Ilustración 86: Desactivación del Proceso de Transmisión en Función *ieee154e_init* (**IEEE802154Ecsl.c**)

7.4.3. ALCANCE DE LAS PRUEBAS

El código desarrollado para la implementación del mecanismo CSL contempla la transmisión y recepción de tramas de datos así como sus respectivas tramas de reconocimiento (ACK), incluyendo la incorporación previa del mecanismo de notificación asíncrono basado en tramas *wake-up* (*wake-up sequence*) tal y como se ha descrito a lo largo de la presente memoria.

Sin embargo, la disponibilidad de una única placa OpenBase limita en gran medida las posibilidades de depuración en lo que respecta a las tramas de reconocimiento (ACK) dado que sería preciso validar simultáneamente el envío y recepción de la trama en ambos extremos.

En este sentido y ante esta situación, en el alcance de las pruebas se ha considerado en todo momento que **el nodo transmisor no requiere de trama de reconocimiento (ACK) a través del parámetro *ackRequested*** (valor falso) tal y como se encuentra contemplado dentro del estándar.

8. RESULTADOS Y VALORACIONES

La realización de la fase de pruebas sobre la implementación CSL llevada a cabo a partir del análisis y diseño establecido, ha permitido la detección y corrección de ciertos aspectos funcionales y de comportamiento no identificados inicialmente.

Este proceso, necesario en todo ciclo normal de desarrollo software, ha posibilitado por tanto el incremento en la calidad de los desarrollos así como la garantía en el comportamiento funcional de acuerdo a los objetivos y requerimientos iniciales.

En este sentido y utilizando el escenario de pruebas descrito como entorno de validación, se ha verificado la correcta generación de la secuencia de tramas *wake-up* así como la correspondiente actualización de los tiempos *rendezvous* y la posterior transmisión de la trama de datos, verificando además dicho comportamiento de manera visual a través del encendido de los leds correspondientes.

Respecto al nodo receptor, se ha verificado igualmente la correcta recepción de las tramas *wake-up* así como el análisis y validación de su estructura. Sin embargo, es importante indicar en este punto que la verificación del elemento receptor no ha podido ser completa, observando un comportamiento no esperado por medio del cual el tiempo *rendezvous* recibido no es actualizado en cada recepción de tramas *wake-up*. Como resultado de esta situación, se produce un problema de pérdida de sincronización entre transmisor y receptor que conlleva a una descoordinación entre el tiempo pendiente para iniciar la transmisión de la trama de datos en el nodo transmisor y el tiempo restante que debe esperar el nodo receptor para llevar a cabo la recepción de dicha trama.

Aún así, tanto el sistema de transmisión como el sistema de recepción se encuentran completamente desarrollados e implementados sobre la solución OpenWSN de acuerdo a la máquina de estados definida y las actividades asociadas a cada uno de los estados, relegando la resolución a dicha situación dentro de los procesos de evolución y desarrollo establecidos dentro de las líneas futuras identificadas.

9. CONCLUSIONES

El estándar IEEE 802.15.4, creado para cubrir la necesidad de estándares inalámbricos de baja tasa y bajo consumo para aplicaciones en redes de sensores, representa la base de desarrollo del emergente concepto IoT (*Internet of Things*) donde cualquier dispositivo podrá disponer de la capacidad de comunicarse con su entorno mediante la transmisión y recepción de datos, posibilitando el desarrollo de servicios adicionales de valor añadido sobre el conjunto de funcionalidades básicas.

Dentro de este marco, el Trabajo Final de Master ha tenido como objetivo principal el desarrollo, implementación y validación del mecanismo de bajo consumo CSL (*Coordinated Sampled Listening*) para el acceso al medio, definido y especificado en el estándar 802.15.4 y más concretamente en la enmienda 802.15.4e para la capa MAC, empleando para ello una de las implementaciones de referencia *open-source* del estándar IEEE802.15.4 más importantes como es OpenWSN.

En este sentido, el trabajo ha tenido una alta componente de investigación y desarrollo (I+D) sobre la cual se ha adecuado el problema de investigación, centrado en el modo de operación CSL, a los objetivos planteados, definiendo una marco de análisis e investigación relevante en relación al ámbito de estudio del estándar IEEE802.15.4 y su implementación dentro del proyecto de software libre OpenWSN.

Como resultado de esta realización, se ha obtenido un trabajo con una alta coherencia interna en lo referente al análisis y argumentación así como en el diseño y elaboración del marco teórico sobre el cual se han apoyado los posteriores desarrollos realizados mediante la programación en lenguaje C sobre OpenWSN.

A este respecto y en relación a la realización de la parte práctica, se considera, a nivel personal, que se ha realizado con éxito el análisis de un problema aparentemente sencillo pero en la práctica con gran cantidad de complejidades, extrayendo con ello las conclusiones de los conocimientos adquiridos, aportando como resultado soluciones de síntesis innovadoras fruto de dichas reflexiones, y obteniendo resultados coherentes con los procesos llevados a cabo, todo ello dentro del marco estructural y de buenas prácticas establecidas en el actual desarrollo de OpenWSN.

Respecto a los objetivos inicialmente establecidos para el desarrollo del proyecto, se ha alcanzado la totalidad de ellos salvo la operatividad completa del mecanismo de recepción desarrollado, detectando ciertas mejoras y aspectos complejos no depurados

que deberán ser tratados durante la realización de líneas futuras y que han quedado al margen del presente trabajo por cuestiones de plazo. Por su parte, los objetivos alcanzados por completo han incluido la adquisición de los conocimientos y experiencia en las metodologías de desarrollo, pruebas y depuración sobre sistemas empotrados reales, y más concretamente sobre la plataforma OpenMote-CC2538, así como el conocimiento de estado del arte en cuanto al desarrollo de estándares LR-WPAN y en particular la estructura, modo de funcionamiento y operatividad a nivel funcional y nivel radio de OpenWSN.

Del mismo modo, se ha llevado a cabo el estudio detallado y el análisis de la especificación para CSL así como la determinación de su modo de funcionamiento en transmisión y recepción, y el uso de técnicas *preamble sampling*, procediendo a su diseño, implementación y codificación en lenguaje C sobre OpenWSN, contribuyendo con ello a incrementar el grado de cumplimiento del estándar y posibilitar nuevas vías de estudio, evolución, desarrollo y aplicabilidad.

Respecto a la metodología propuesta para el desarrollo del trabajo, ha existido una correcta adecuación con el plan previsto, aplicando la metodología en cascada para el desarrollo global del proyecto y la metodología ágil (Scrum) dentro de la fase de desarrollo. En este sentido, la metodología ágil, inicialmente prevista con *sprints* de dos semanas, han tenido que verse modificada en una ocasión a un plazo superior (tres semanas) dada la complejidad de las tareas y la disponibilidad personal de esfuerzo estimado, obligando a redimensionar las estimaciones inicialmente consideradas para las tareas del sprint.

En consecuencia, esto ha obligado a realizar un redimensionamiento del primer sprint así como una asignación menor de tareas para los siguientes sprints (mantenidos en dos semanas) y una re-priorización de las tareas contempladas con complejidad media-alta.

En cuanto a la modalidad de realización del trabajo final de máster con estas características, cabe destacar la gran dedicación de esfuerzo dedicado, muy superior al inicialmente estimado, dado el desconocimiento inicial en cuanto a los procedimientos iniciales, necesarios para poder comenzar la realización de los trabajos objeto del proyecto así como el alto esfuerzo dedicado a la propia parte creativa y depurativa inherente en cualquier tipo de desarrollo software.

Estos trabajos, entre los cuales habría que destacar los procedimientos de compilación, depuración y despliegue, la configuración del entorno y las conectividades de elementos, han supuesto una importante carga de esfuerzo sobre el esfuerzo inicialmente estimado, lo cual ha provocado una desviación respecto a la planificación inicial que ha sido necesario ir compensando en el tiempo en las fases posteriores mediante el incremento de la dedicación estimada. En cualquier caso y a este respecto, cabe destacar que a consecuencia de esta dedicación adicional, se ha adquirido un mayor conocimiento respecto a todos estos elementos dado el nivel de profundidad y estudio que ha sido necesario dedicar para ello, lo cual considero un aspecto muy positivo dentro de la elaboración del trabajo respecto a la adquisición de conocimientos a pesar del sobresfuerzo exigido.

Finalmente pero no menos importante, un aspecto especialmente relevante a destacar dentro del desarrollo del proyecto es la contribución realizada al desarrollo del software libre. La implementación realizada dentro del presente trabajo es un reto tecnológico por su complejidad y su innovación dentro del marco de OpenWSN, por lo que su incorporación dentro de la implementación de OpenWSN permitirá aumentar la cobertura funcional del producto además de establecer nuevas líneas de evolución y trabajo futuro junto a la comunidad de desarrollo de OpenWSN (entre la cual ya me considero) las cuales permitirán optimizar y ampliar los desarrollos realizados y con ello la aplicabilidad futura de la solución.

10. LÍNEAS FUTURAS

El presente apartado tiene como objetivo describir las líneas de trabajo futuro que no han podido ser realizadas o explotadas en el trabajo y que han quedado pendientes.

En este sentido, los **aspectos relativos a la implementación del mecanismo CSL objeto del presente trabajo que han quedado fuera del alcance** y que son por tanto líneas de investigación y desarrollo futuras son las siguientes:

- **Transmisión Sincronizada** (*synchronized)* en la cual la capa MAC conoce la fase y periodo CSL del dispositivo destino. Esto implica que la longitud de la secuencia *wake-up* sea solamente el tiempo de guarda respecto al desplazamiento del reloj en base al tiempo obtenido en la última actualización de la fase y periodo CSL desde el dispositivo destino.
- **Elemento de Información LE CSL IE** por medio del cual el transmisor y receptor se intercambian el valor de su fase y periodo CSL con el fin de mantener la sincronización entre ellos y reducir las duraciones de las secuencias de tramas *wake-up* (*wake-up sequence*), reduciendo con ello el consumo derivado de la interfaz radio.
- **Activación y Desactivación del Modo de Operación y Soporte para Transmisión de Mensajes Urgentes**, por medio de la cual el modo CSL podrá ser activado o desactivado por el nivel superior (a través de la configuración de un valor de periodo CSL *macCSLPeriod* distinto o igual a cero), proporcionado el soporte para funcionar de distintos modos en la misma implementación bajo distintas condiciones de entorno y necesidades de las motas.

  Así mismo, se considerará el soporte necesario para la transmisión de mensajes urgentes los cuales tendrán un tratamiento personalizado frente al total de paquetes pendientes de envío para su entrega ágil y eficiente en el extremo remoto.
- **Soporte para el Envío y Recepción de Varias Tramas Simultáneas** (*frame pending bit*) por medio de la cual es posible secuenciar varias transmisiones o recepciones de tramas sin necesidad de iniciar nuevos ciclos de apagado y encendido de la interfaz radio, optimizando el tiempo global de transmisión y reduciendo los consumos derivados de encendido y apagado de la interfaz radio.
- **CSL sobre múltiples canales**, en cuyo caso el modo de funcionamiento CSL es extendido a varios canales definidos a través de la propiedad *macCSLChannelMask*.

Este caso implica que la operación de muestreo (*sampling*) sea realizada para cada canal, de menor a mayor (*round-robin*).

De esta manera, para transmisión no-sincronizada, la transmisión CSL enviará una secuencia de tramas *wake-up* de longitud igual al producto entre el número de canales y el valor del atributo *macCSLPeriod* antes del envío de la trama de datos o información (*payload frame*).

Para transmisión sincronizada, la transmisión CSL calculará el siguiente tiempo de muestreo de canal y el número de canal, y procederá al envío en el siguiente muestreo de canal sobre el canal correcto con una secuencia corta de tramas *wake-up*. En este caso, la fase CSL será la duración desde el instante actual hasta el siguiente muestreo del canal del primer canal seleccionado dentro de la máscara de canales definida por *macCSLChannelMask*.

11. GLOSARIO

A continuación se presenta el listado de términos y acrónimos más relevantes empleados dentro del desarrollo de la memoria.

6LoWPAN *IPv6 over Low power Wireless Personal Area Networks*

AR *Acknowledgment Request*

ARM *(Acorn/Advanced) RISC Machine*

ASH *Auxiliary Security Header*

ASN *Absolute Slot Number*

BLE *Bluetooth Low Energy*

BSDL *Boundary Scan Description Language*

BSP *Board Support Package*

CAP *Content Access Period*

CCA *Clear Channel Assessment*

CFP *Contention Free Period*

CoAP *Constrained Application Protocol*

CSL *Coordinated Sampled Listening*

CSMA-CA *Carrier Sense Multiple Access with Collision Avoidance*

DSN *Data Sequence Number*

EB *Enhanced Beacon*

EBR *Enhanced Beacon Request*

ED *Energy Detection*

FCS *Frame Control Sequence*

FFD	*Full Function Device*
FSM	*Finite-State Machine*
GTS	*Guaranteed Time Slot*
I2C Bus	*Inter-Integrated Circuit Bus*
IDE	*Integrated Development Environment*
IE	*Information Element*
IEEE	*Institute of Electrical and Electronics Engineers*
IETF	*Internet Engineering Task Force*
IoT	*Internet of Things*
JTAG	*Joint Test Action Group*
LBR	*Low-Power Border Router*
LE	*Low Energy*
LE CSL IE	*Low Energy Coordinated Sampled Listening Information Element*
LED	*Light Emitting Diode*
LLNs	*Low Power and Lossy Networks*
LQI	*Link Quality Indicator*
LR-WPAN	*Low-Rate Wireless Personal Area Network*
M2M	*Machine-to-Machine*
MAC	*Medium Access Control*
MCPS	*MAC Common Part Sublayer*
MCPS-SAP	*MAC Common Part Sublayer – Service Access Point*

MFR	*MAC Footer*
MHR	*MAC Header*
MLME	*MAC Sublayer Management Entity*
MLME-SAP	*MAC Sublayer Management Entity – Service Access Point*
MPDU	*MAC Protocol Data Unit*
MSDU	*MAC Service Data Unit*
MTU	*Maximum Transfer Unit*
PAN	*Personal Area Network*
PANC	*PAN Coordinator*
PD	*PHY Data*
PD-SAP	*PHY Data– Service Access Point*
PHY	*Physical Layer*
PIB	*PAN Information Base*
PLME	*PHY Management Entity*
PLME-SAP	*PHY Management Entity – Service Access Point*
PPDU	*PHY Protocol Data Unit*
PPM	*Parts Per Million*
RAM	*Random Access Memory*
REST	*Representational State Transfer*
RFD	*Reduced Function Device*
RIT	*Receiver Initiated Transmission*

RPL	*Routing Protocol for Low power and Lossy Networks*
RTC	*Real Time Clock*
RZ Time IE	*Rendezvous Time Information Element*
SAP	*Service Access Point*
SFD	*Start-of-Frame Delimiter*
SOC	*System-On-Chip*
SPI	*Serial Peripheral Interface*
TCP	*Transmission Control Protocolo*
TSCH	*Time-Slotted Channel Hopping*
UDP	*User Datagram Protocol*
USB	*Universal Serial Bus*
UWB	*Ultra-Wide Band*
WPAN	*Wireless Personal Area Network*

12. BIBLIOGRAFÍA

[1] Alcázar, Victor (2013). **Estudio de las Prestaciones de IEEE 802.15.4e**. Trabajo Final de Grado. Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels. Universidad Politécnica de Catalunya. [internet]. Disponible en:

http://upcommons.upc.edu/pfc/handle/2099.1/19708

[2] Anastasi, Giussepe (2014). **From IEEE 802.15.4 to IEEE 802.15.4e: Another Step towards the Internet of (Important) Things**. Pervasive Computing & Networking Lab (PerLab). Dept. of Information Engineering, University of Pisa [internet]. Disponible en:

http://www.iet.unipi.it/g.anastasi/talks/2014-Guangzhou.pdf

[3] Bachir Abdelmalik, Dohler Mischa, Watteyne Thomas, y Leung Kin K (2010). **MAC Essentials for Wireless Sensor Networks**. IEEE Communications Surveys & Tutorials, Vol.12, No 2, Second Quarter 2010 [internet]. Disponible en:

http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=5451759&url=http%3A%2F%2Fieeexplore.ieee.org%2Fxpls%2Fabs_all.jsp%3Farnumber%3D5451759

[4] Bononi, Luciano (2013). **IEEE 802.15.4, ZigBee & Open Source**. Dipartimento di Scienze dell'Informazione. Facoltà di Scienze Matematiche, Fisiche e Naturali [internet]. Disponible en:

http://www.cs.unibo.it/bononi/SRW2015/SRW2013_6.pdf

[5] Chunhui, Allan y Zhu, Youngsoo Kim (2012). **Review of IEEE 802.15.4 MAC Layer Power Save Mechanisms**. IEEE P802.15 Working Group for Wireless Personal Area Networks (WPANs). IEEE 802.15-12-0384-00-004q [internet]. Disponible en:

https://mentor.ieee.org/802.15/dcn/12/15-12-0384-01-004q-review-of-ieee-802-15-4-mac-layer-power-save-mechanisms.pptx

[6] Dunkels, Adam (2011). **The ContikiMAC Radio Duty Cycling Protocol**. SICS Technical Report T2011:13. ISSN 1100-3154 [internet]. Disponible en:

http://dunkels.com/adam/dunkels11contikimac.pdf

[7] Herrera, Laura (2009). **Diseño de un Sistema de Localización e Identificación Basado en Redes de Sensores Inalámbricos**. Proyecto Final de Carrera. Escola Tècnica Superior d'Enginyeria de Telecomunicació de Barcelona [internet]. Disponible en:

http://upcommons.upc.edu/pfc/bitstream/2099.1/7973/1/Pfc%20-%20Laura%20Herrera.pdf

[8] Hong, Wei (2009). **Low Energy SubGroup Report**. Subgroup update for 802.15.4e Arch Rock Corporation. IEEE 15-09-0523-00-004e [internet]. Disponible en:

https://mentor.ieee.org/802.15/dcn/09/15-09-0646-00-004e-low-energy-subgroup-report.ppt

[9] IEEE Computer Society – IEEE Standard for Local and Metropolitan Area Networks (2011). **Part 15.4: Low-Rate Wireless Personal Area Networks (LR-WPANs).** Revision de IEEE Standard 802.15.4-2006 [internet]. Disponible en:

http://standards.ieee.org/getieee802/download/802.15.4-2011.pdf

[10] IEEE Computer Society – IEEE Standard for Local and Metropolitan Area Networks (2012). **Part 15.4: Low-Rate Wireless Personal Area Networks (LR-WPANs). Amendment 1: MAC Sublayer** [internet]. Disponible en:

http://standards.ieee.org/getieee802/download/802.15.4e-2012.pdf

[11] Muñoz, Rodrigo (2011). **Interconexión de Redes de Sensores Inalámbricos 802.15.4 en Localizaciones Remotas**. Trabajo Final de Grado. Universidad Carlos III de Madrid [internet]. Disponible en:

http://e-archivo.uc3m.es/bitstream/handle/10016/11934/PFC_Rodrigo_Munoz_Castejon.pdf ?sequence=1

[12] **OpenMote [sitio web]** (2014). Wiki de la página web oficial de OpenWSN [internet]. Disponible en:

www.openmote.com

[13] **OpenWSN Confluence [sitio web]** (2014). Wiki de la página web oficial de OpenWSN [internet]. Disponible en:

https://openwsn.atlassian.net/wiki/download/attachments/1081520/openwsn_overview.ppt

[14] Vilajosana Xavier, Robles Inés et al. (2014). **LLN PLUGFEST IETF 90 Report**. Toronto, Canada [internet]. Disponible en:

https://openwsn.atlassian.net/wiki/download/attachments/35553282/ietf90_toronto_plugfest_slides.pdf

[15] Watteyne Thomas y Adjih Cedric (2014). **OpenWSN: Implementing the Internet Of Important Things**. Workshop Internet of Things / Equipex FIT IoT-LAB. Montbonnot, France [internet]. Disponible en:

https://www.iot-lab.info/wp-content/uploads/2014/11/141106_openwsn_iotlab_public.pdf

[16] Watteyne Thomas et al (2012). **OpenWSN: Open-Source Standards-Based Protocol Stacks for Wireless Mesh Networks**. Visión global de OpenWSN disponible en sitio web de OpenWSN [internet]. Disponible en:

https://openwsn.atlassian.net/wiki/download/attachments/1081520/openwsn_overview.ppt

[17] Watteyne Thomas, Vilajosana Xavier, Kerkez Branko, Chraim Fabien, Weekly Kevin, Qin Wang, Glaser Steven y Pister Kris (2012). **OpenWSN: a Standards-Based Low-Power Wireless Development Environment**. Transactions on Emerging Telecommunications Technologies 2012. 23 Pags 480–493 [internet]. Disponible en:

http://robotics.eecs.berkeley.edu/~pister/publications/2012/openwsnETT.pdf

13. ANEXO A – ESTRUCTURA DE PAQUETES DE TRABAJO

El presente anexo tiene como objetivo proporcionar una descripción detallada de cada uno de los paquetes de trabajo desarrollados en la elaboración del presente trabajo final de máster (TFM).

Cada paquete de trabajo (PT), detallará la siguiente información básica:

- **Objetivos** cubiertos dentro del paquete de trabajo.
- **Entradas de información** necesarias para la ejecución de las actividades comprendidas dentro del paquete de trabajo.
- **Salidas o entregables** que serán generados tras la realización del paquete de trabajo.
- **Detalle de las actividades** a ejecutar, junto con los responsables de su ejecución.
- **Posibles restricciones o condicionantes** para la ejecución de las actividades.

Los siguientes apartados describirán cada una de las fases o tareas de la metodología. Esta descripción se basa en la definición de paquetes de trabajo (PT).

La siguiente ilustración recoge la estructura general de paquetes de trabajo seguida durante el desarrollo de las tareas del Trabajo Final de Master.

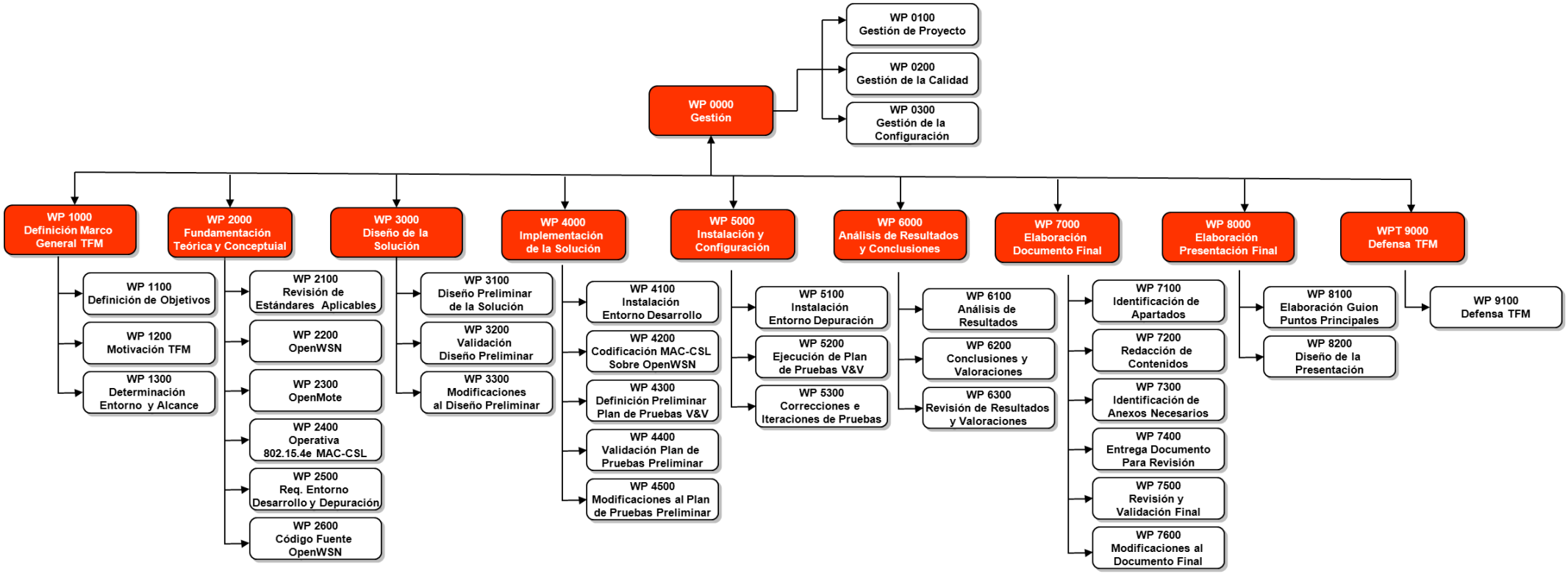


Ilustración 87: Estructura de Paquetes de Trabajo (WP)

13.1. WP-0000: GESTIÓN

<table>
<tr><td>PROYECTO:</td><td colspan="2">Implementación del mecanismo de acceso al medio (MAC) IEEE 802.15.4e CSL sobre OpenWSN y plataforma OpenMote.</td><td>HOJA 1 DE 1</td></tr>
<tr><td>CÓDIGO WP:</td><td colspan="2">WP-0000</td><td rowspan="5">MEDIOS MATERIALES
No Aplicable (N/A)</td></tr>
<tr><td>DENOMINACIÓN WP:</td><td colspan="2">Gestión</td></tr>
<tr><td>RESPONSABLE WP:</td><td colspan="2">Sergio Gonzalo / Pere Tuset</td></tr>
<tr><td>INICIO WP:</td><td colspan="2">Selección Tema e Inicio de TFM (Kick-Off)</td></tr>
<tr><td>FINALIZACIÓN WP:</td><td colspan="2">Defensa de TFM</td></tr>
<tr><td colspan="4">ENTRADAS:</td></tr>
<tr><td colspan="4">■ Acuerdo de Selección de Temática TFM
■ Documento Contractual de Cesión de Material OpenMote</td></tr>
<tr><td colspan="4">ACTIVIDADES:</td></tr>
<tr><td colspan="4">El presente paquete de trabajo se articula en las siguientes tareas o actividades:
■ WP-0100: Gestión de Proyecto. Seguimiento y control de la planificación, detección de desviaciones y aplicación de medidas correctivas.
■ WP-0200: Gestión de la Calidad. Seguimiento y control de la calidad del proyecto, asegurando la correcta aplicación de la metodología definida, verificando y validando los productos finales, así como el control documental del proyecto.
■ WP-0300: Gestión de Configuración. Control de los distintos elementos de configuración del proyecto.</td></tr>
<tr><td colspan="4">SALIDAS:</td></tr>
<tr><td colspan="4">■ Informes de Seguimiento</td></tr>
</table>

13.2. WP-1000: DEFINICIÓN MARCO GENERAL DEL TFM

<table>
<tr><td>PROYECTO:</td><td colspan="2">Implementación del mecanismo de acceso al medio (MAC) IEEE 802.15.4e CSL sobre OpenWSN y plataforma OpenMote.</td><td>HOJA 1 DE 1</td></tr>
<tr><td>CÓDIGO WP:</td><td colspan="2">WP-1000</td><td rowspan="5">MEDIOS MATERIALES
No Aplicable (N/A)</td></tr>
<tr><td>DENOMINACIÓN WP:</td><td colspan="2">Definición Marco General del TFM</td></tr>
<tr><td>RESPONSABLE WP:</td><td colspan="2">Sergio Gonzalo San José</td></tr>
<tr><td>INICIO WP:</td><td colspan="2">Selección Tema e Inicio de TFM (Kick-Off)</td></tr>
<tr><td>FINALIZACIÓN WP:</td><td colspan="2">Fundamentación Teórica y Conceptual del TFM</td></tr>
<tr><td colspan="4">ENTRADAS:</td></tr>
<tr><td colspan="4">■ Acuerdo de Selección de Temática TFM</td></tr>
<tr><td colspan="4">ACTIVIDADES:</td></tr>
<tr><td colspan="4">El presente paquete de trabajo se articula en las siguientes tareas o actividades:
■ WP-1100: Definición de Objetivos TFM. Determinación de objetivos principales en el desarrollo del TFM seleccionado.
■ WP-1200: Motivación TFM. Identificación de principales motivaciones en la selección y desarrollo del TFM.
■ WP-1300: Determinación de Entorno y Alcance. Establecimiento de condiciones de entorno en el desarrollo del TFM así como el alcance cubierto bajo el TFM.</td></tr>
<tr><td colspan="4">SALIDAS:</td></tr>
<tr><td colspan="4">■ PEC1 con definición del Marco General de desarrollo del TFM en el cual se incluyen los objetivos principales y la motivación del desarrollo.</td></tr>
</table>

13.3. WP-2000: FUNDAMENTACIÓN TEÓRICA Y CONCEPTUAL

<table>
<tr><td>PROYECTO:</td><td colspan="2">Implementación del mecanismo de acceso al medio (MAC) IEEE 802.15.4e CSL sobre OpenWSN y plataforma OpenMote.</td><td>HOJA 1 DE 1</td></tr>
<tr><td>CÓDIGO WP:</td><td colspan="2">WP-2000</td><td rowspan="5">MEDIOS MATERIALES
No Aplicable (N/A)</td></tr>
<tr><td>DENOMINACIÓN WP:</td><td colspan="2">Fundamentación Teórica y Conceptual</td></tr>
<tr><td>RESPONSABLE WP:</td><td colspan="2">Sergio Gonzalo San José</td></tr>
<tr><td>INICIO WP:</td><td colspan="2">Definición Marco General del TFM</td></tr>
<tr><td>FINALIZACIÓN WP:</td><td colspan="2">Diseño de la Solución</td></tr>
<tr><td colspan="4">ENTRADAS:</td></tr>
<tr><td colspan="4">■ Definición Marco General del TFM</td></tr>
<tr><td colspan="4">ACTIVIDADES:</td></tr>
<tr><td colspan="4">El presente paquete de trabajo se articula en las siguientes tareas o actividades:
■ WP-2100: Revisión de Estándares Aplicables. Revisión de los estándares aplicables en el desarrollo del TFM (IEEE 802.15.4 y 802.15.4e).
■ WP-2200: OpenWSN. Adquisición de conocimiento preliminar de la plataforma OpenWSN a través del análisis de la información presente en la sede web y el entorno wiki disponible.
■ WP-2300: OpenMote. Adquisición de conocimiento preliminar de la tecnología OpenMote a través del análisis de la información presente en la sede web.
■ WP-2400: Operativa IEEE 802.15.4e MAC-CSL. Análisis y revisión de documentación adicional disponible referente al modo MAC-CSL así como análisis pormenorizado del funcionamiento detallado a partir del estándar.
■ WP-2500: Requerimientos Entorno Desarrollo y Depuración. Revisión de los requerimientos base para los entornos de desarrollo y depuración sobre OpenWSN y OpenMote y configuración inicial de sistema operativo y paquetes requeridos.</td></tr>
</table>

■ **WP-2600: Código Fuente OpenWSN**. Estudio de la organización y estructura del código fuente de OpenWSN así como determinación preliminar del mecanismo de introducción de MAC-CSL en el producto.
SALIDAS:
■ Conocimiento preliminar de las condiciones iniciales requeridas para el proceso de diseño de la solución MAC-CSL en OpenWSN.

13.4. WP-3000: DISEÑO DE LA SOLUCIÓN

<table>
<tr><td>PROYECTO:</td><td colspan="2">Implementación del mecanismo de acceso al medio (MAC) IEEE 802.15.4e CSL sobre OpenWSN y plataforma OpenMote.</td><td>HOJA 1 DE 1</td></tr>
<tr><td>CÓDIGO WP:</td><td colspan="2">WP-3000</td><td rowspan="5">MEDIOS MATERIALES
No Aplicable (N/A)</td></tr>
<tr><td>DENOMINACIÓN WP:</td><td colspan="2">Diseño de la Solución</td></tr>
<tr><td>RESPONSABLE WP:</td><td colspan="2">Sergio Gonzalo San José</td></tr>
<tr><td>INICIO WP:</td><td colspan="2">Fundamentación Teórica y Conceptual</td></tr>
<tr><td>FINALIZACIÓN WP:</td><td colspan="2">Implementación de la Solución</td></tr>
<tr><td colspan="4">ENTRADAS:</td></tr>
<tr><td colspan="4">■ Fundamentación Teórica y Conceptual</td></tr>
<tr><td colspan="4">ACTIVIDADES:</td></tr>
<tr><td colspan="4">El presente paquete de trabajo se articula en las siguientes tareas o actividades:
■ WP-3100: Diseño Preliminar de la Solución. Con el conocimiento adquirido en el WP anterior, se elaborará un diseño preliminar de la solución de MAC-CSL a implementar.
■ WP-3200: Validación del Diseño Preliminar. Validación del diseño preliminar con el tutor para la aprobación y/o aplicación de correcciones y modificaciones no previstas o bien no óptimas.
■ WP-3300: Modificaciones y Correcciones al Diseño Preliminar (Diseño Definitivo). Aplicación de las modificaciones y correcciones determinadas en el WP anterior y elaboración del diseño definitivo de la solución.</td></tr>
<tr><td colspan="4">SALIDAS:</td></tr>
<tr><td colspan="4">■ Diseño Definitivo de la Solución MAC-CSL a implementar sobre OpenWSN.</td></tr>
</table>

13.5. WP-4000: IMPLEMENTACIÓN DE LA SOLUCIÓN

<table>
<tr><td>PROYECTO:</td><td colspan="2">Implementación del mecanismo de acceso al medio (MAC) IEEE 802.15.4e CSL sobre OpenWSN y plataforma OpenMote.</td><td>HOJA 1 DE 1</td></tr>
<tr><td>CÓDIGO WP:</td><td colspan="2">WP-4000</td><td rowspan="5">MEDIOS MATERIALES
No Aplicable (N/A)</td></tr>
<tr><td>DENOMINACIÓN WP:</td><td colspan="2">Implementación de la Solución</td></tr>
<tr><td>RESPONSABLE WP:</td><td colspan="2">Sergio Gonzalo San José</td></tr>
<tr><td>INICIO WP:</td><td colspan="2">Diseño de la Solución</td></tr>
<tr><td>FINALIZACIÓN WP:</td><td colspan="2">Validación y Pruebas</td></tr>
<tr><td colspan="4">ENTRADAS:</td></tr>
<tr><td colspan="4">■ Diseño Definitivo de la Solución MAC-CSL a implementar sobre OpenWSN.</td></tr>
<tr><td colspan="4">ACTIVIDADES:</td></tr>
<tr><td colspan="4">El presente paquete de trabajo se articula en las siguientes tareas o actividades:
■ WP-4100: Instalación y Configuración Entorno Desarrollo. Instalación y configuración del entorno necesario para el desarrollo y despliegue del nuevo desarrollo para MAC-CSL.
■ WP-4200: Codificación MAC-CSL sobre OpenWSN. Codificación en la plataforma OpenWSN de la solución MAC-CSL e integración de ambos elementos.
■ WP-4300: Definición Preliminar del Plan de Pruebas de Verificación y Validación. Elaboración de un plan de pruebas preliminar para la verificación y validación funcional de la implementación de MAC-CSL realizada.
■ WP-4400: Validación del Plan de Pruebas Preliminar. Validación del plan de pruebas preliminar con el tutor para la aprobación y/o aplicación de correcciones y modificaciones.
■ WP-4500: Modificaciones y Correcciones al Plan de Pruebas Preliminar (Plan Definitivo). Aplicación de las modificaciones y correcciones determinadas en el WP</td></tr>
</table>

anterior y elaboración del plan de pruebas definitivo.
SALIDAS:
■ Implementación de MAC-CSL sobre OpenWSN. ■ Plan de Pruebas Definitivo para la Verificación y Validación Funcional.

13.6. WP-5000: VALIDACIÓN Y PRUEBAS

<table>
<tr><td>PROYECTO:</td><td colspan="2">Implementación del mecanismo de acceso al medio (MAC) IEEE 802.15.4e CSL sobre OpenWSN y plataforma OpenMote.</td><td>HOJA 1 DE 1</td></tr>
<tr><td>CÓDIGO WP:</td><td colspan="2">WP-5000</td><td rowspan="5">MEDIOS MATERIALES
No Aplicable (N/A)</td></tr>
<tr><td>DENOMINACIÓN WP:</td><td colspan="2">Validación y Pruebas</td></tr>
<tr><td>RESPONSABLE WP:</td><td colspan="2">Sergio Gonzalo San José</td></tr>
<tr><td>INICIO WP:</td><td colspan="2">Implementación de la Solución</td></tr>
<tr><td>FINALIZACIÓN WP:</td><td colspan="2">Análisis de Resultados y Conclusiones</td></tr>
<tr><td colspan="4">ENTRADAS:</td></tr>
<tr><td colspan="4">■ Implementación de MAC-CSL sobre OpenWSN.
■ Plan de Pruebas Definitivo para la Verificación y Validación Funcional.</td></tr>
<tr><td colspan="4">ACTIVIDADES:</td></tr>
<tr><td colspan="4">El presente paquete de trabajo se articula en las siguientes tareas o actividades:
■ WP-5100: Instalación y Configuración Entorno Depuración. Instalación y configuración del entorno necesario para la depuración y análisis del código desarrollado durante la fase de pruebas.
■ WP-5200: Ejecución del Plan de Pruebas de Verificación y Validación. Realización y seguimiento de los casos de prueba del plan de pruebas sobre la funcionalidad y comportamiento del código desarrollado.
■ WP-5300: Correcciones e Iteraciones de Pruebas. Identificación de errores y deficiencias, aplicación de correctivos y repetición de los casos de pruebas relacionados hasta obtener los resultados esperados.</td></tr>
<tr><td colspan="4">SALIDAS:</td></tr>
<tr><td colspan="4">■ Implementación definitiva validada y verificada de MAC-CSL sobre OpenWSN.</td></tr>
</table>

13.7. WP-6000: ANÁLISIS DE RESULTADOS Y CONCLUSIONES

<table>
<tr><td>PROYECTO:</td><td colspan="2">Implementación del mecanismo de acceso al medio (MAC) IEEE 802.15.4e CSL sobre OpenWSN y plataforma OpenMote.</td><td>HOJA 1 DE 1</td></tr>
<tr><td>CÓDIGO WP:</td><td colspan="2">WP-6000</td><td rowspan="5">MEDIOS MATERIALES
No Aplicable (N/A)</td></tr>
<tr><td>DENOMINACIÓN WP:</td><td colspan="2">Análisis de Resultados y Conclusiones</td></tr>
<tr><td>RESPONSABLE WP:</td><td colspan="2">Sergio Gonzalo San José</td></tr>
<tr><td>INICIO WP:</td><td colspan="2">Validación y Pruebas</td></tr>
<tr><td>FINALIZACIÓN WP:</td><td colspan="2">Elaboración Documento Final del TFM (Memoria)</td></tr>
<tr><td colspan="4">ENTRADAS:</td></tr>
<tr><td colspan="4">■ Implementación definitiva validada y verificada de MAC-CSL sobre OpenWSN.</td></tr>
<tr><td colspan="4">ACTIVIDADES:</td></tr>
<tr><td colspan="4">El presente paquete de trabajo se articula en las siguientes tareas o actividades:
■ WP-6100: Análisis de Resultados. Consolidación y correlación de resultados y métricas obtenidas. Análisis de los resultados.
■ WP-6200: Conclusiones y Valoraciones. Valoración de los resultados obtenidos y comparativa respecto a los resultados esperados. Conclusiones finales sobre los resultados y determinación de acciones futuras.
■ WP-6300: Revisión de Resultados y Valoraciones. Revisión junto al tutor de los resultados y conclusiones obtenidas y validación de éstos.</td></tr>
<tr><td colspan="4">SALIDAS:</td></tr>
<tr><td colspan="4">■ Análisis de Resultados y Conclusiones Finales.</td></tr>
</table>

13.8. WP-7000: ELABORACIÓN DOCUMENTO FINAL TFM (MEMORIA)

<table>
<tr><td>PROYECTO:</td><td colspan="2">Implementación del mecanismo de acceso al medio (MAC) IEEE 802.15.4e CSL sobre OpenWSN y plataforma OpenMote.</td><td>HOJA 1 DE 1</td></tr>
<tr><td>CÓDIGO WP:</td><td colspan="2">WP-7000</td><td rowspan="5">MEDIOS MATERIALES
No Aplicable (N/A)</td></tr>
<tr><td>DENOMINACIÓN WP:</td><td colspan="2">Elaboración Documento Final TFM (Memoria)</td></tr>
<tr><td>RESPONSABLE WP:</td><td colspan="2">Sergio Gonzalo San José</td></tr>
<tr><td>INICIO WP:</td><td colspan="2">Análisis de Resultados y Conclusiones</td></tr>
<tr><td>FINALIZACIÓN WP:</td><td colspan="2">Elaboración Presentación Final del TFM (Defensa)</td></tr>
<tr><td colspan="4">ENTRADAS:</td></tr>
<tr><td colspan="4">▪ Análisis de Resultados y Conclusiones Finales.</td></tr>
<tr><td colspan="4">ACTIVIDADES:</td></tr>
<tr><td colspan="4">El presente paquete de trabajo se articula en las siguientes tareas o actividades:
▪ WP-7100: Identificación de Apartados del Documento Final. Identificación de los contenidos, estructura e índice que conformará el documento de memoria final.
▪ WP-7200: Redacción de Contenidos de cada Apartado. Redacción y elaboración de los contenidos de cada uno de los apartados de la memoria final.
▪ WP-7300: Identificación de Anexos Necesarios al Documento Principal. Identificación de los anexos que son preciso incluir en el documento de memoria final.
▪ WP-7400: Entrega de Documento para Revisión Final. Entrega del documento de memoria para su revisión por parte del tutor..
▪ WP-7500: Revisión y Validación Final. Revisión y validación del contenido y estructura del documento de memoria final por parte del tutor.
▪ WP-7600: Modificaciones y Correcciones al Documento y Elaboración de Memoria Final. Identificación de errores y deficiencias en los contenidos de la memoria final y aplicación de modificaciones y correcciones. Cierre del documento para la disponibilidad del documento de memoria final del TFM.</td></tr>
<tr><td colspan="4">SALIDAS:</td></tr>
<tr><td colspan="4">▪ Documento Final del TFM (Memoria)</td></tr>
</table>

13.9. WP-8000: ELABORACIÓN PRESENTACIÓN FINAL TFM

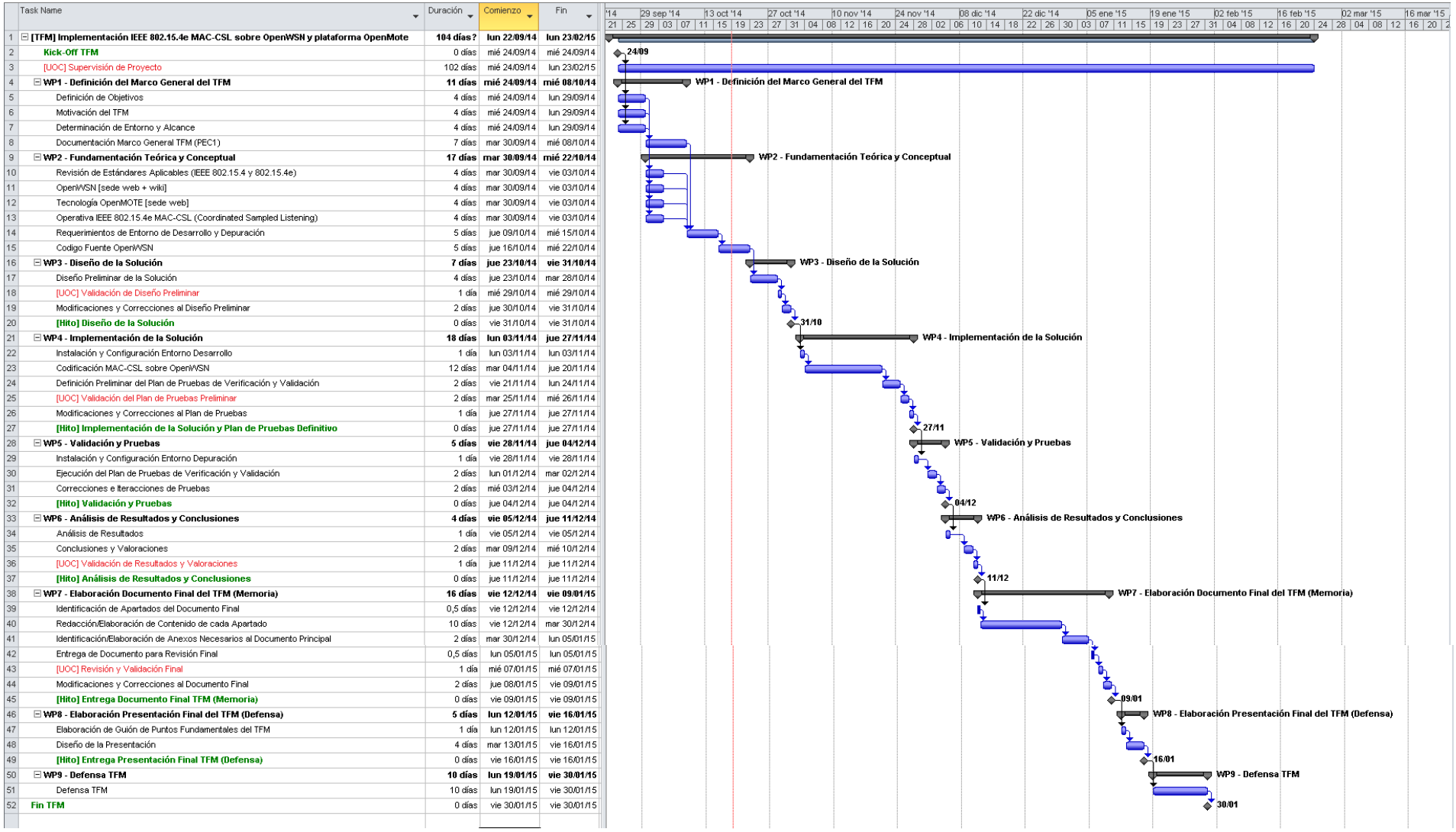
<table>
<tr><td>PROYECTO:</td><td colspan="2">Implementación del mecanismo de acceso al medio (MAC) IEEE 802.15.4e CSL sobre OpenWSN y plataforma OpenMote.</td><td>HOJA 1 DE 1</td></tr>
<tr><td>CÓDIGO WP:</td><td colspan="2">WP-8000</td><td rowspan="5">MEDIOS MATERIALES
No Aplicable (N/A)</td></tr>
<tr><td>DENOMINACIÓN WP:</td><td colspan="2">Elaboración Presentación Final TFM (Defensa)</td></tr>
<tr><td>RESPONSABLE WP:</td><td colspan="2">Sergio Gonzalo San José</td></tr>
<tr><td>INICIO WP:</td><td colspan="2">Elaboración Documento Final TFM (Memoria)</td></tr>
<tr><td>FINALIZACIÓN WP:</td><td colspan="2">Defensa TFM</td></tr>
<tr><td colspan="4">ENTRADAS:</td></tr>
<tr><td colspan="4">■ Documento Final del TFM (Memoria).
■ Presentación Final del TFM (Defensa).</td></tr>
<tr><td colspan="4">ACTIVIDADES:</td></tr>
<tr><td colspan="4">El presente paquete de trabajo se articula en las siguientes tareas o actividades:
■ WP-8100: Elaboración Guion de Puntos Fundamentales del TFM. Determinación de aspectos y puntos principales del contenido de la presentación de defensa del TFM.
■ WP-8200: Diseño de la Presentación. Organización y secuenciación de contenidos de la presentación. Diseño gráfico y aspectos fundamentales del contenido en formato presentación de acuerdo al guion de puntos determinado anteriormente.</td></tr>
<tr><td colspan="4">SALIDAS:</td></tr>
<tr><td colspan="4">■ Presentación Final del TFM (Defensa)</td></tr>
</table>

13.10. WP-9000: DEFENSA TFM

<table>
<tr><td>PROYECTO:</td><td colspan="2">Implementación del mecanismo de acceso al medio (MAC) IEEE 802.15.4e CSL sobre OpenWSN y plataforma OpenMote.</td><td>HOJA 1 DE 1</td></tr>
<tr><td>CÓDIGO WP:</td><td colspan="2">WP-9000</td><td rowspan="5">MEDIOS MATERIALES
No Aplicable (N/A)</td></tr>
<tr><td>DENOMINACIÓN WP:</td><td colspan="2">Defensa TFM</td></tr>
<tr><td>RESPONSABLE WP:</td><td colspan="2">Sergio Gonzalo San José</td></tr>
<tr><td>INICIO WP:</td><td colspan="2">Elaboración Presentación Final del TFM (Defensa)</td></tr>
<tr><td>FINALIZACIÓN WP:</td><td colspan="2">Defensa TFM</td></tr>
<tr><td colspan="4">ENTRADAS:</td></tr>
<tr><td colspan="4">■ Documento Final del TFM (Memoria).
■ Presentación Final del TFM (Defensa).</td></tr>
<tr><td colspan="4">ACTIVIDADES:</td></tr>
<tr><td colspan="4">El presente paquete de trabajo se articula en las siguientes tareas o actividades:
■ WP-9100: Defensa TFM. Presentación del TFM y defensa de su contenido frente al tribunal calificador.</td></tr>
<tr><td colspan="4">SALIDAS:</td></tr>
<tr><td colspan="4">■ TFM Finalizado.</td></tr>
</table>

14. ANEXO B – PLANIFICACIÓN DETALLADA

A continuación se presenta la planificación detallada empleada para el desarrollo y seguimiento del Trabajo Final de Master, descrita en términos de los paquetes de trabajo definidos en el trabajo, descritos en el **Anexo A – Estructura de Paquetes de Trabajo**.



	Task Name	Duración	Comienzo	Fin
1	**[TFM] Implementación IEEE 802.15.4e MAC-CSL sobre OpenWSN y plataforma OpenMote**	**104 días?**	**lun 22/09/14**	**lun 23/02/15**
2	**Kick-Off TFM**	0 días	mié 24/09/14	mié 24/09/14
3	[UOC] Supervisión de Proyecto	102 días	mié 24/09/14	lun 23/02/15
4	**WP1 - Definición del Marco General del TFM**	**11 días**	**mié 24/09/14**	**mié 08/10/14**
5	Definición de Objetivos	4 días	mié 24/09/14	lun 29/09/14
6	Motivación del TFM	4 días	mié 24/09/14	lun 29/09/14
7	Determinación de Entorno y Alcance	4 días	mié 24/09/14	lun 29/09/14
8	Documentación Marco General TFM (PEC1)	7 días	mar 30/09/14	mié 08/10/14
9	**WP2 - Fundamentación Teórica y Conceptual**	**17 días**	**mar 30/09/14**	**mié 22/10/14**
10	Revisión de Estándares Aplicables (IEEE 802.15.4 y 802.15.4e)	4 días	mar 30/09/14	vie 03/10/14
11	OpenWSN [sede web + wiki]	4 días	mar 30/09/14	vie 03/10/14
12	Tecnología OpenMOTE [sede web]	4 días	mar 30/09/14	vie 03/10/14
13	Operativa IEEE 802.15.4e MAC-CSL (Coordinated Sampled Listening)	4 días	mar 30/09/14	vie 03/10/14
14	Requerimientos de Entorno de Desarrollo y Depuración	5 días	jue 09/10/14	mié 15/10/14
15	Codigo Fuente OpenWSN	5 días	jue 16/10/14	mié 22/10/14
16	**WP3 - Diseño de la Solución**	**7 días**	**jue 23/10/14**	**vie 31/10/14**
17	Diseño Preliminar de la Solución	4 días	jue 23/10/14	mar 28/10/14
18	[UOC] Validación de Diseño Preliminar	1 día	mié 29/10/14	mié 29/10/14
19	Modificaciones y Correcciones al Diseño Preliminar	2 días	jue 30/10/14	vie 31/10/14
20	**[Hito] Diseño de la Solución**	0 días	vie 31/10/14	vie 31/10/14
21	**WP4 - Implementación de la Solución**	**18 días**	**lun 03/11/14**	**jue 27/11/14**
22	Instalación y Configuración Entorno Desarrollo	1 día	lun 03/11/14	lun 03/11/14
23	Codificación MAC-CSL sobre OpenWSN	12 días	mar 04/11/14	jue 20/11/14
24	Definición Preliminar del Plan de Pruebas de Verificación y Validación	2 días	vie 21/11/14	lun 24/11/14
25	[UOC] Validación del Plan de Pruebas Preliminar	2 días	mar 25/11/14	mié 26/11/14
26	Modificaciones y Correcciones al Plan de Pruebas	1 día	jue 27/11/14	jue 27/11/14
27	**[Hito] Implementación de la Solución y Plan de Pruebas Definitivo**	0 días	jue 27/11/14	jue 27/11/14
28	**WP5 - Validación y Pruebas**	**5 días**	**vie 28/11/14**	**jue 04/12/14**
29	Instalación y Configuración Entorno Depuración	1 día	vie 28/11/14	vie 28/11/14
30	Ejecución del Plan de Pruebas de Verificación y Validación	2 días	lun 01/12/14	mar 02/12/14
31	Correcciones e Iteracciones de Pruebas	2 días	mié 03/12/14	jue 04/12/14
32	**[Hito] Validación y Pruebas**	0 días	jue 04/12/14	jue 04/12/14
33	**WP6 - Análisis de Resultados y Conclusiones**	**4 días**	**vie 05/12/14**	**jue 11/12/14**
34	Análisis de Resultados	1 día	vie 05/12/14	vie 05/12/14
35	Conclusiones y Valoraciones	2 días	mar 09/12/14	mié 10/12/14
36	[UOC] Validación de Resultados y Valoraciones	1 día	jue 11/12/14	jue 11/12/14
37	**[Hito] Análisis de Resultados y Conclusiones**	0 días	jue 11/12/14	jue 11/12/14
38	**WP7 - Elaboración Documento Final del TFM (Memoria)**	**16 días**	**vie 12/12/14**	**vie 09/01/15**
39	Identificación de Apartados del Documento Final	0,5 días	vie 12/12/14	vie 12/12/14
40	Redacción/Elaboración de Contenido de cada Apartado	10 días	vie 12/12/14	mar 30/12/14
41	Identificación/Elaboración de Anexos Necesarios al Documento Principal	2 días	mar 30/12/14	lun 05/01/15
42	Entrega de Documento para Revisión Final	0,5 días	lun 05/01/15	lun 05/01/15
43	[UOC] Revisión y Validación Final	1 día	mié 07/01/15	mié 07/01/15
44	Modificaciones y Correcciones al Documento Final	2 días	jue 08/01/15	vie 09/01/15
45	**[Hito] Entrega Documento Final TFM (Memoria)**	0 días	vie 09/01/15	vie 09/01/15
46	**WP8 - Elaboración Presentación Final del TFM (Defensa)**	**5 días**	**lun 12/01/15**	**vie 16/01/15**
47	Elaboración de Guión de Puntos Fundamentales del TFM	1 día	lun 12/01/15	lun 12/01/15
48	Diseño de la Presentación	4 días	mar 13/01/15	vie 16/01/15
49	**[Hito] Entrega Presentación Final TFM (Defensa)**	0 días	vie 16/01/15	vie 16/01/15
50	**WP9 - Defensa TFM**	**10 días**	**lun 19/01/15**	**vie 30/01/15**
51	Defensa TFM	10 días	lun 19/01/15	vie 30/01/15
52	**Fin TFM**	0 días	vie 30/01/15	vie 30/01/15

Ilustración 88: Planificación del Trabajo Final de Master (TFM)

15. ANEXO C – CÓDIGO FUENTE DE ACTIVIDADES CSL FSM

El presente anexo tiene como objetivo proporcionar el detalle acerca del código fuente asociado a cada una de las actividades relacionadas con los estados de la máquina de estados de transmisión y recepción, excluyendo en todos los casos el código fuente añadido o modificado para la realización de la fase de pruebas.

15.1. MODO RECEPCIÓN

15.1.1. ACTIVIDADES

15.1.1.1 activity_csl_wakeup_ri1

```
...
/**
 \brief Activity for CSL RX Sampling stage [wake-up ri1].
 This method is invoked in ISR-mode from "isr_ieee154ecsl_newChannelSample" function when a new CSL Channel Sample is fired (every "macCSLPeriod" time).
 \note The FSM state should be S_SLEEP (initial state), raising an error in otherwise.
*/
port_INLINE void activity_csl_wakeup_ri1() {              // Activity for stage [ri1] on CSL RX Sampling.
  // if the previous sample took too long or the state is incorrect, we will not be in the right state.
  // register error indicating CSL sample number where the problem have been detected since sampling startup.
  if (ieee154e_vars.state!=S_SLEEP) {
    // log the error
    openserial_printError(COMPONENT_IEEE802154E,ERR_WRONG_STATE_IN_START_CSL_SAMPLING,
                          (errorparameter_t)ieee154e_vars.state,  (errorparameter_t)ieee154e_dbg.num_cslSamples);
    // Abort in case of error.
    endOps();
    return;
  }
  openserial_stop();    // stop using serial

   // change state --> Next State will be S_CSLRXWAKEUPOFFSET in order to prepare for CSL Rx wake-up.
  changeState(S_CSLRXWAKEUPOFFSET);

  radiotimer_schedule(DURATION_rt1);  // arm rt1
}
...
```

Ilustración 89: Función *activity_csl_wakeup_ri1* (**IEEE802154Ecsl.c**)

15.1.1.2 activity_csl_wakeup_ri2

```
...
/**
 \brief Activity for CSL RX Sampling stage [wake-up ri2].
  This method is invoked from ISR-mode "isr_ieee154ecsl_timer" function when FSM timer fires
  (expiring "duration_rt1") while state = S_CSLRXWAKEUPOFFSET.  The functionality is to prepare the radio
  for receiving packets.
*/
port_INLINE void activity_csl_wakeup_ri2() {            / Activity for stage [ri2] on CSL RX Sampling.
   // change state
   changeState(S_CSLRXWAKEUPPREPARE);

   // We use always the same frequency so it is not necessary to change it on radio.
   // Enable the radio in Rx mode. The radio does not actively listen yet.
   radio_rxEnable();
   ieee154e_vars.radioOnInit=radio_getTimerValue();
   ieee154e_vars.radioOnThisSample=TRUE;

   // arm rt2
   radiotimer_schedule(DURATION_rt2);

   // Change state.
   // If this action is not performed before rt2 has expired, we need to manage the error in activity_csl_rie1.
  // This means that radi prepare has exceeded the maximum allowed time (duration_rt2).
   changeState(S_CSLRXWAKEUPREADY);
}
...
```

Ilustración 90: Función *activity_csl_wakeup_ri2* (**IEEE802154Ecsl.c**)

15.1.1.3 activity_csl_wakeup_ri3

```
...
/**
 \brief Activity for CSL RX Sampling stage [wake-up ri3].
  This method is invoked from ISR-mode "isr_ieee154ecsl_timer" function when FSM timer fires (expiring
"duration_rt2") while state = S_CSLRXWAKEUPREADY.  The radio is configured and this interrupt gives it the
"go" signal to start listening.
*/
port_INLINE void activity_csl_wakeup_ri3() {            // Activity for stage [ri3] on CSL RX Sampling.
```

```
  // change state
  changeState(S_CSLRXWAKEUPLISTEN);

  // give the 'go' to receive
  radio_rxNow();

  // arm rt3
  radiotimer_schedule(DURATION_rt3);
}
...
```

Ilustración 91: Función *activity_csl_wakeup_ri3* (**IEEE802154Ecsl.c**)

15.1.1.4 activity_csl_wakeup_ri4

```
...
/**
 \brief Activity for CSL RX Sampling stage [wake-up ri4].
  This method is invoked from ISR-mode "ieee154ecsl_startOfFrame" function when a start of frame event fires while state = S_CSLRXWAKEUPLISTEN. The functionality is to capture the time, cancel #rt3 and arm #rt4 (max time to receive all the packet).
*/
port_INLINE void activity_csl_wakeup_ri4(PORT_RADIOTIMER_WIDTH capturedTime) {
  // change state
  changeState(S_CSLRXWAKEUP);

  // cancel rt3
  radiotimer_cancel();

  // record the captured time
  ieee154e_vars.lastCapturedTime = capturedTime;

  // record the captured time to sync
  ieee154e_vars.syncCapturedTime = capturedTime;

  radiotimer_schedule(DURATION_rt4);
}
...
```

Ilustración 92: Función *activity_csl_wakeup_ri4* (**IEEE802154Ecsl.c**)

15.1.1.5 activity_csl_wakeup_ri5

```
...
/**
 \brief Activity for CSL RX Sampling stage [wake-up ri5].
  This method is invoked from ISR-mode "ieee154ecsl_endOfFrame" function when a end of frame event fires while state = S_CSLRXWAKEUP. The functionality is to change state, cancel #rt4, validate wake-up frame and destination, and arm #rt4 (rztime).
*/
port_INLINE void activity_csl_wakeup_ri5(PORT_RADIOTIMER_WIDTH capturedTime) {
 // En este punto ya hemos recibido la trama por lo que hay que hacer las siguientes validaciones:
 //   1.- Cambiar el estado a S_CSLRXWAKEUPVALIDATE y cancelar el temporizador rt4.
 //   2.- Apagar la radio.
 //   3.- Analizar el mensaje recibido y parsear su cabecera.
 //   4.- Verificar que se trata de una trama de tipo wake-up.
//   5.- Comprobar que el destinatario soy yo mismo.
//
// En caso afirmativo, usamos el rz-time recibido para saber cuánto tiempo dormir hasta recibir la trama de datos.
// En caso negativo, desactivamos la radio e iniciamos de nuevo el proceso de channel sampling.

  ieee802154_header_iht ieee802514_header;
  uint16_t rztime;

  // actualizamos el estado.
  changeState(S_CSLRXWAKEUPVALIDATE);

  // cancelamos el timer #rt4
  radiotimer_cancel();

  // apagamos la radio
  radio_rfOff();

  // obtenemos un buffer en el cual poder guardar los datos recibidos.
  ieee154e_vars.wakeupReceived = openqueue_getFreePacketBuffer(COMPONENT_IEEE802154E);
  if (ieee154e_vars.wakeupReceived==NULL) {
     // registro del error & fin de operaciones.
     openserial_printError(COMPONENT_IEEE802154E,ERR_NO_FREE_PACKET_BUFFER,
                           (errorparameter_t)0, (errorparameter_t)0);
     endOps();
     return;
  }
```

```
// Declaración de propiedad sobre el paquete.
ieee154e_vars.wakeupReceived->creator = COMPONENT_IEEE802154E;
ieee154e_vars.wakeupReceived->owner   = COMPONENT_IEEE802154E;

/*
The do-while loop that follows is a little parsing trick.  Because it contains a while(0) condition, it gets executed only once. The behavior is:
- if a break occurs inside the do{} body, the error code below the loop gets executed. This indicates something is wrong with the packet being parsed.
- if a return occurs inside the do{} body, the error code below the loop does not get executed. This indicates the received packet is correct.
*/
do {   // Este "loop" es ejecutado sólo en una ocasión.
   // Obtenemos la trama con los datos recibidos desde el buffer de recepción de la radio.
   ieee154e_vars.wakeupReceived->payload =
                                        &(ieee154e_vars.wakeupReceived->packet[FIRST_FRAME_BYTE]);
   radio_getReceivedFrame(   ieee154e_vars.wakeupReceived->payload,
                             &ieee154e_vars.wakeupReceived->length,
                             sizeof(ieee154e_vars.wakeupReceived->packet),
                             &ieee154e_vars.wakeupReceived->l1_rssi,
                             &ieee154e_vars.wakeupReceived->l1_lqi,
                             &ieee154e_vars.wakeupReceived->l1_crc);

   // Finalizamos si la longitud no es correcta.
   if (ieee154e_vars.wakeupReceived->length<LENGTH_CRC ||
      ieee154e_vars. wakeupReceived->length>LENGTH_IEEE154_MAX ) {
     openserial_printError(COMPONENT_IEEE802154E,ERR_INVALIDPACKETFROMRADIO,
                           (errorparameter_t)2, ieee154e_vars. wakeupReceived->length);
     break;
   }

   // toss CRC (2 últimos bytes)
   packetfunctions_tossFooter(ieee154e_vars.wakeupReceived, LENGTH_CRC);

   // si CRC no es válido, finalizamos el proceso.
   if (ieee154e_vars.wakeupReceived->l1_crc==FALSE) { break; }

   // parseamos la trama IEEE802.15.4 WAKE-UP y su cabecera.
   ieee802154_retrieveWakeUpHeader(ieee154e_vars.dataReceived,&ieee802514_header,&rztime);
```

```
// En el caso de que no sea una cabecera IEEE802.15.4 válida, finalizamos el proceso.
if (ieee802514_header.valid==FALSE) { break; }

// Verificamos que se trata de una trama WAKE-UP, perteneciente a la misma PAN ID, y dirigida a mi.
if(ieee802514_header.frameType==IEEE154_TYPE_MULTIPURPOSE) {
        if(packetfunctions_sameAddress(&ieee802514_header.dest,idmanager_getMyID(ADDR_16B)) &&
          packetfunctions_sameAddress(&ieee802514_header.panid,idmanager_getMyID(ADDR_PANID))) {
             // En este caso, dormimos un tiempo RZ Time y actualitzar el estado a S_CSLRXDATAOFFSET
             changeState(S_CSLRXDATAOFFSET);
             // registro del tiempo de captura
            ieee154e_vars.lastCapturedTime = capturedTime;

            // Tratamos el caso en el cual rztime_ie.time sea cero, es decir, sea la ultima trama wake-up antes
            // del envío de la trama de datos.
            if (rztime == 0) {
              // Establecemos el timer rt1 (consideramos el mismo tiempo de offset para la recepción de la
              // trama wake-up que para la de datos).
              radiotimer_schedule(DURATION_rt1);
            } else {
              // Establecemos el timer al valor rt1 pero desplazado el tiempo indicado desde el rendezvous time.
             radiotimer_schedule(DURATION_rt1 + rztime);
            }
          // Descartamos el paquete una vez recibido y tratado.
          openqueue_freePacketBuffer(ieee154e_vars.wakeupReceived);
          // clear local variable
          ieee154e_vars.wakeupReceived = NULL;
          // si hemos llegado aquí, retornamos para no ejecutar el codigo inferior.
         return;
 }
// En caso que sea una trama de mi PANID pero no dirigida a mi, entonces dormimos un tiempo igual a:
//   - RZ time + Maximum length payload frame + secure ack frame (consideramos un tiempo igual a
//      RZ time + TsSlotDuration)  dado que TsSlotDuration es el tiempo utilizado en OpenWSN-TSCH
//       para enviar y recibir una trama de datos + ack.
else if( ! packetfunctions_sameAddress(&ieee802514_header.dest,idmanager_getMyID(ADDR_16B)) &&
           packetfunctions_sameAddress(&ieee802514_header.panid,idmanager_getMyID(ADDR_PANID))) {
            // La limpieza del paquete y los datos recibidos será realizada en el metodo activity_csl_wakeup_rie4
            radiotimer_schedule(rztime + TsSlotDuration);

            // Descartamos el paquete una vez recibido y tratado.
            openqueue_freePacketBuffer(ieee154e_vars.wakeupReceived);
```

```
                // clear local variable
                ieee154e_vars.wakeupReceived = NULL;
                return;
            }
        }
    } while(0);

    // En cualquier otro caso, descartamos el paquete e iniciamos de nuevo el proceso de channel sampling.
    openqueue_freePacketBuffer(ieee154e_vars.dataReceived);

    // clear local variable
    ieee154e_vars.dataReceived = NULL;

    // abort
    endOps();
}
...
```

Ilustración 93: Función *activity_csl_wakeup_ri5* (**IEEE802154Ecsl.c**)

15.1.1.6 activity_csl_data_ri2

```
...
/**
 \brief Activity for CSL RX Sampling stage [data ri2].
  This method is invoked from ISR-mode "isr_ieee154ecsl_timer" function when FSM timer fires (expiring "duration_rt1") while state = S_CSLRXDATAOFFSET.
  The functionality is to prepare the radio for receiving packets.
*/
port_INLINE void activity_csl_data_ri2() {
    // change state
    changeState(S_CSLRXDATAPREPARE);

    // We use always the same frequency so it is not necessary to change it on radio.
    // Enable the radio in Rx mode. The radio does not actively listen yet.
    radio_rxEnable();
    ieee154e_vars.radioOnInit=radio_getTimerValue();
    ieee154e_vars.radioOnThisSample=TRUE;

    // arm rt2
    radiotimer_schedule(DURATION_rt2);
```

```
   // change state
   // If this action is not performed before rt2 has expired, we need to manage the error in activity_csl_data_rie1.
  //  This means that radio  prepare has exceeded the maximum allowed time (duration_rt2).
   changeState(S_CSLRXDATAREADY);
}
...
```

Ilustración 94: Función *activity_csl_data_ri2* (**IEEE802154Ecsl.c**)

15.1.1.7 activity_csl_data_ri3

```
...
/**
 \brief Activity for CSL RX Sampling stage [data ri3].
  This method is invoked from ISR-mode "isr_ieee154ecsl_timer" function when FSM timer fires (expiring "duration_rt2") while state = S_CSLRXDATAREADY.  The radio is configured and this interrupt gives it the "go" signal to start listening.
*/
port_INLINE void activity_csl_data_ri3() {     // Activity for stage [data ri3] on CSL RX Sampling.
   // change state
   changeState(S_CSLRXDATALISTEN);

   // give the 'go' to receive
   radio_rxNow();

   // arm rt3
   radiotimer_schedule(DURATION_rt3);
}
...
```

Ilustración 95: Función *activity_csl_data_ri3* (**IEEE802154Ecsl.c**)

15.1.1.8 activity_csl_data_ri4

```
...
/**
 \brief Activity for CSL RX Sampling stage [data ri4].
  This method is invoked from ISR-mode "ieee154ecsl_startOfFrame" function when a start of frame event fires while state = S_CSLRXDATALISTEN.  The functionality is to capture the time, cancel #rt3 and arm #rt4 (max time to receive all the packet).
*/
port_INLINE void activity_csl_data_ri4(PORT_RADIOTIMER_WIDTH capturedTime) {  // Activity for stage [data ri4] on CSL RX Sampling.
   // change state
   changeState(S_CSLRXDATA);

   // cancel rt3
   radiotimer_cancel();

   // record the captured time
   ieee154e_vars.lastCapturedTime = capturedTime;

   // record the captured time to sync
   ieee154e_vars.syncCapturedTime = capturedTime;

   radiotimer_schedule(DURATION_rt4);
}
...
```

Ilustración 96: Función *activity_csl_data_ri4* (**IEEE802154Ecsl.c**)

15.1.1.9 activity_csl_data_ri5

```
...
/**
 \brief Activity for CSL RX Sampling stage [data ri5].
  This method is invoked from ISR-mode "ieee154ecsl_endOfFrame" function when a end of frame event fires while state = S_CSLRXDATA. The functionality is to change state, cancel #rt4, validate data frame, and transmit ACK.
*/
port_INLINE void activity_csl_data_ri5(PORT_RADIOTIMER_WIDTH capturedTime) {
   ieee802154_header_iht ieee802514_header;
   uint16_t lenIE=0;

   // change state
```

```
changeState(S_CSLTXACKOFFSET);

// cancel rt4
radiotimer_cancel();

// turn off the radio
radio_rfOff();
ieee154e_vars.radioOnTics+=radio_getTimerValue()-ieee154e_vars.radioOnInit;
// get a buffer to put the (received) data in
ieee154e_vars.dataReceived = openqueue_getFreePacketBuffer(COMPONENT_IEEE802154E);
if (ieee154e_vars.dataReceived==NULL) {
  // log the error
  openserial_printError(COMPONENT_IEEE802154E,ERR_NO_FREE_PACKET_BUFFER,
                        (errorparameter_t)0,  (errorparameter_t)0);
  // abort
  endOps();
  return;
}

// declare ownership over that packet
ieee154e_vars.dataReceived->creator = COMPONENT_IEEE802154E;
ieee154e_vars.dataReceived->owner   = COMPONENT_IEEE802154E;

/*
The do-while loop that follows is a little parsing trick.  Because it contains a while(0) condition, it gets
executed only once. The behavior is:
- if a break occurs inside the do{} body, the error code below the loop gets executed. This indicates something
  is wrong with the packet being parsed.
- if a return occurs inside the do{} body, the error code below the loop  does not get executed. This indicates
  the received packet is correct.
*/

do {     // this "loop" is only executed once
  // retrieve the received data frame from the radio's Rx buffer
  ieee154e_vars.dataReceived->payload = &(ieee154e_vars.dataReceived->packet[FIRST_FRAME_BYTE]);
  radio_getReceivedFrame(  ieee154e_vars.dataReceived->payload,   &ieee154e_vars.dataReceived->length,
                           sizeof(ieee154e_vars.dataReceived->packet),  &ieee154e_vars.dataReceived->l1_rssi,
                          &ieee154e_vars.dataReceived->l1_lqi, &ieee154e_vars.dataReceived->l1_crc);

  // break if wrong length
```

```
if (ieee154e_vars.dataReceived->length<LENGTH_CRC ||
   ieee154e_vars.dataReceived->length>LENGTH_IEEE154_MAX ) {
   // jump to the error code below this do-while loop
  openserial_printError(COMPONENT_IEEE802154E,ERR_INVALIDPACKETFROMRADIO,
                        (errorparameter_t)2, ieee154e_vars.dataReceived->length);
   break;
}

// toss CRC (2 last bytes)
packetfunctions_tossFooter(   ieee154e_vars.dataReceived, LENGTH_CRC);

// if CRC doesn't check, stop
if (ieee154e_vars.dataReceived->l1_crc==FALSE) {
   // jump to the error code below this do-while loop
   break;
}

// parse the IEEE802.15.4 header (RX DATA)
ieee802154_retrieveHeader(ieee154e_vars.dataReceived,&ieee802514_header);

// break if invalid IEEE802.15.4 header
if (ieee802514_header.valid==FALSE) {
   // break from the do-while loop and execute the clean-up code below
   break;
}

// store header details in packet buffer
ieee154e_vars.dataReceived->l2_frameType      = ieee802514_header.frameType;
ieee154e_vars.dataReceived->l2_dsn            = ieee802514_header.dsn;
ieee154e_vars.dataReceived->l2_IEListPresent  = ieee802514_header.ieListPresent;
memcpy(&(ieee154e_vars.dataReceived->l2_nextORpreviousHop),
            &(ieee802514_header.src),sizeof(open_addr_t));

// toss the IEEE802.15.4 header
packetfunctions_tossHeader(ieee154e_vars.dataReceived,ieee802514_header.headerLength);

// handle IEs xv poipoi
// reset join priority
// retrieve IE in sixtop
if ((ieee802514_header.valid==TRUE &&
```

```
      ieee802514_header.ieListPresent==TRUE &&
      // if it is not a beacon and have ie, the ie will be processed in sixtop
      ieee802514_header.frameType==IEEE154_TYPE_BEACON &&
      packetfunctions_sameAddress(&ieee802514_header.panid,idmanager_getMyID(ADDR_PANID)) &&
      ieee154e_processIEs(ieee154e_vars.dataReceived,&lenIE))==FALSE) {
      //log  that the packet is not carrying IEs
   }

   // toss the IEs including Synch
   packetfunctions_tossHeader(ieee154e_vars.dataReceived,lenIE);

   // record the captured time
   ieee154e_vars.lastCapturedTime = capturedTime;

   // if I just received an invalid frame, stop
   if (isValidRxFrame(&ieee802514_header)==FALSE) {
      // jump to the error code below this do-while loop
      break;
   }

   // check if ack requested
   if (ieee802514_header.ackRequested==1) {
      // arm rt5
      radiotimer_schedule(DURATION_rt5);
   } else {
       notif_receive(ieee154e_vars.dataReceived);
      // reset local variable
      ieee154e_vars.dataReceived = NULL;
      // abort
      endOps();
   }
   // everything went well, return here not to execute the error code below
   return;
} while(0);

// free the (invalid) received data so RAM memory can be recycled
openqueue_freePacketBuffer(ieee154e_vars.dataReceived);

// clear local variable
ieee154e_vars.dataReceived = NULL;
```

```
  // abort
  endOps();
}
...
```

Ilustración 97: Función *activity_csl_data_ri5* (**IEEE802154Ecsl.c**)

15.1.1.10 activity_csl_data_ri6

```
...
/**
 \brief Activity for CSL RX Sampling stage [data ri6].
  This method is invoked from ISR-mode "isr_ieee154ecsl_timer" function when FSM timer fires (expiring
"duration_rt5") while state = S_CSLTXACKOFFSET. The functionality is to prepare the radio for ACK sending.
*/
port_INLINE void activity_csl_data_ri6() {
  PORT_SIGNED_INT_WIDTH timeCorrection;
  header_IE_ht header_desc;

  // change state
  changeState(S_CSLTXACKPREPARE);

  // get a buffer to put the ack to send in
  ieee154e_vars.ackToSend = openqueue_getFreePacketBuffer(COMPONENT_IEEE802154E);
  if (ieee154e_vars.ackToSend==NULL) {
    // log the error
    openserial_printError(COMPONENT_IEEE802154E,ERR_NO_FREE_PACKET_BUFFER,
                          (errorparameter_t)0, (errorparameter_t)0);
    // indicate we received a packet anyway (we don't want to loose any)
    notif_receive(ieee154e_vars.dataReceived);
    // free local variable
    ieee154e_vars.dataReceived = NULL;
    // abort
    endOps();
    return;
  }
  // declare ownership over that packet
  ieee154e_vars.ackToSend->creator = COMPONENT_IEEE802154E;
  ieee154e_vars.ackToSend->owner   = COMPONENT_IEEE802154E;

  // calculate the time timeCorrection (this is the time when the packet arrive w.r.t the time it should be.
```

```
   timeCorrection =
(PORT_SIGNED_INT_WIDTH)((PORT_SIGNED_INT_WIDTH)ieee154e_vars.syncCapturedTime-
(PORT_SIGNED_INT_WIDTH)TsTxOffset);

   // add the payload to the ACK (i.e. the timeCorrection)
   packetfunctions_reserveHeaderSize(ieee154e_vars.ackToSend,sizeof(timecorrection_IE_ht));
   timeCorrection  = -timeCorrection;
   timeCorrection *= US_PER_TICK;
   ieee154e_vars.ackToSend->payload[0] = (uint8_t)((((uint16_t)timeCorrection)   ) & 0xff);
   ieee154e_vars.ackToSend->payload[1] = (uint8_t)((((uint16_t)timeCorrection)>>8) & 0xff);

   // add header IE header -- xv poipoi -- pkt is filled in reverse order..
   packetfunctions_reserveHeaderSize(ieee154e_vars.ackToSend,sizeof(header_IE_ht));
   //create the header for ack IE
   header_desc.length_elementid_type=(sizeof(timecorrection_IE_ht)<<
IEEE802154E_DESC_LEN_HEADER_IE_SHIFT)|
                        (IEEE802154E_ACK_NACK_TIMECORRECTION_ELEMENTID <<
IEEE802154E_DESC_ELEMENTID_HEADER_IE_SHIFT)|
                        IEEE802154E_DESC_TYPE_SHORT;
   memcpy(ieee154e_vars.ackToSend->payload,&header_desc,sizeof(header_IE_ht));

   // prepend the IEEE802.15.4 header to the ACK
   ieee154e_vars.ackToSend->l2_frameType = IEEE154_TYPE_ACK;
   ieee154e_vars.ackToSend->l2_dsn       = ieee154e_vars.dataReceived->l2_dsn;
   ieee802154_prependHeader(ieee154e_vars.ackToSend,
                        ieee154e_vars.ackToSend->l2_frameType,
                        IEEE154_IELIST_YES,//ie in ack
                        IEEE154_FRAMEVERSION,//enhanced ack
                        IEEE154_SEC_NO_SECURITY,
                        ieee154e_vars.dataReceived->l2_dsn,
                        &(ieee154e_vars.dataReceived->l2_nextORpreviousHop)
                        );

   // space for 2-byte CRC
   packetfunctions_reserveFooterSize(ieee154e_vars.ackToSend,2);

   // load the packet in the radio's Tx buffer
   radio_loadPacket(ieee154e_vars.ackToSend->payload,
                    ieee154e_vars.ackToSend->length);

   // enable the radio in Tx mode. This does not send that packet.
```

```
   radio_txEnable();
   ieee154e_vars.radioOnInit=radio_getTimerValue();
   ieee154e_vars.radioOnThisSample=TRUE;
   // arm rt6
   radiotimer_schedule(DURATION_rt6);

   // change state
   changeState(S_CSLTXACKREADY);
}
...
```

Ilustración 98: Función *activity_csl_data_ri6* (**IEEE802154Ecsl.c**)

15.1.1.11 activity_csl_data_ri7

```
...
/**
 \brief Activity for CSL RX Sampling stage [data ri7].
  This method is invoked from ISR-mode "isr_ieee154ecsl_timer" function when FSM timer fires (expiring "duration_rt2") while state = S_CSLRXDATAREADY. The radio is configured and this interrupt gives it the "go" signal to start listening.
*/
port_INLINE void activity_csl_data_ri7() {
   // change state
   changeState(S_CSLTXACKDELAY);

   // arm rt7
   radiotimer_schedule(DURATION_rt7);

   // give the 'go' to transmit
   radio_txNow();
}
...
```

Ilustración 99: Función *activity_csl_data_ri7* (**IEEE802154Ecsl.c**)

15.1.1.12 activity_csl_data_ri8

```
...
/**
 \brief Activity for CSL RX Sampling stage [data ri8].
  This method is invoked from ISR-mode "ieee154ecsl_startOfFrame" function when a SFD event fires while State
   is  S_CSLTXACKDELAY.  The functionality is to change the state to set TX, cancel #rt7 and arm #rt8.
*/
port_INLINE void activity_csl_data_ri8(PORT_RADIOTIMER_WIDTH capturedTime) {
   // change state
   changeState(S_CSLTXACK);

   // cancel rt7
   radiotimer_cancel();

   // record the captured time
   ieee154e_vars.lastCapturedTime = capturedTime;

   // arm rt8
   radiotimer_schedule(DURATION_rt8);
}
...
```

Ilustración 100: Función *activity_csl_data_ri8* (**IEEE802154Ecsl.c**)

15.1.1.13 activity_csl_data_ri9

```
...
/**
 \brief Activity for CSL RX Sampling stage [data ri9].
  This method is invoked from "ieee154ecsl_endOfFrame" function when a EOF event fires while  state  Is
  S_CSLTXACK. The functionality  is to change the state, cancel #rt8, free ack packet and notify upper layer.
*/
port_INLINE void activity_csl_data_ri9(PORT_RADIOTIMER_WIDTH capturedTime) {
  // change state
   changeState(S_CSLRXPROC);

   // cancel rt8
   radiotimer_cancel();

   // record the captured time
   ieee154e_vars.lastCapturedTime = capturedTime;
```

```
   // free the ack we just sent so corresponding RAM memory can be recycled
   openqueue_freePacketBuffer(ieee154e_vars.ackToSend);

   // clear local variable
   ieee154e_vars.ackToSend = NULL;

   // inform upper layer of reception (after ACK sent)
   notif_receive(ieee154e_vars.dataReceived);

   // clear local variable
   ieee154e_vars.dataReceived = NULL;

   // official end of Rx slot
   endOps();
}
...
```

Ilustración 101: Función *activity_csl_data_ri9* (**IEEE802154Ecsl.c**)

15.1.2. ACTIVIDADES DE EXCEPCIÓN

15.1.2.1 activity_csl_wakeup_rie1

```
...
/**
 \brief Activity for CSL RX Sampling error [wake-up rie1].
  This is triggered by #rt2 expiring, i.e. timer fires while state = S_CSLRXWAKEUPPREPARE. This is really an error state which indicates that #ri2  did not have enough time to execute. Chances are to set maxRxDataPrepare too small. The implemented behaviour is to log the error and move on the
  next CSL sample.
*/
port_INLINE void activity_csl_wakeup_rie1() {  // Activity for error event [rie1] on CSL RX Sampling.
   // log the error
   openserial_printError(COMPONENT_IEEE802154E,ERR_MAXRXWAKEUPPREPARE_OVERFLOWS,
                         (errorparameter_t)ieee154e_vars.state,
                         (errorparameter_t)ieee154e_dbg.num_cslSamples);
   // abort
   endOps());
}
...
```

Ilustración 102: Función *activity_csl_wakeup_rie1* (**IEEE802154Ecsl.c**)

15.1.2.2 activity_csl_wakeup_rie2

```
...
/**
 \brief Activity for CSL RX Sampling error [wake-up rie2].
  This is triggered by #rt3 expiring, i.e. timer fires while state = S_CSLRXWAKEUPLISTEN. If no packet is
received by the time, this timer expires, then none will be received ever and it is safe to switch of the radio. This
timer is set such that the radio will be on during two consecutive windows of  duration (#TsLongGT).
*/
port_INLINE void activity_csl_wakeup_rie2() { // Activity for error event [rie2] on CSL RX Sampling.

  // abort
  endOps();
}
...
```

Ilustración 103: Función *activity_csl_wakeup_rie2* (**IEEE802154Ecsl.c**)

15.1.2.3 activity_csl_wakeup_rie3

```
...
/**
 \brief Activity for CSL RX Sampling error [wake-up rie3].
  This is triggered by #rt4 expiring, i.e. timer fires while state = S_CSLRXWAKEUP. This is an error state which
indicates the radio took too long to transmit the data packet. The implemented behaviour is to log the error and
move on the next CSL sample.
*/
port_INLINE void activity_csl_wakeup_rie3() {
  // log the error
  openserial_printError(COMPONENT_IEEE802154E,ERR_WDWAKEUPDURATION_OVERFLOWS,
                        (errorparameter_t)ieee154e_vars.state,
                        (errorparameter_t)ieee154e_dbg.num_cslSamples);
  // abort
  endOps();
}
...
```

Ilustración 104: Función *activity_csl_wakeup_rie3* (**IEEE802154Ecsl.c**)

15.1.2.4 activity_csl_wakeup_rie4

```
...
/**
 \brief Activity for CSL RX Sampling error [wake-up rie4].
 This is triggered by rztime+TsSlotDuration expiring, i.e. timer fires while state = S_CSLRXWAKEUPVALIDATE because the wake-up frame is not for me. The behaviour is to free mem used and restart CSL sampling.
*/
port_INLINE void activity_csl_wakeup_rie4() {
   // abort
   endOps();
}
...
```

Ilustración 105: Función *activity_csl_wakeup_rie4* (**IEEE802154Ecsl.c**)

15.1.2.5 activity_csl_data_rie1

```
...
/**
 \brief Activity for CSL RX Sampling error [data_rie1].

  This is triggered by #rt2 expiring, i.e. timer fires while state = S_CSLRXDATAPREPARE. This is really an error state which indicates that #ri2  did not have enough time to execute. Chances are to set maxRxDataPrepare too small. The implemented behaviour is to log the error and move on the  next CSL sample.
*/
port_INLINE void activity_csl_data_rie1() {  // Activity for error event [rie1] on CSL RX Sampling.
   // log the error
   openserial_printError(COMPONENT_IEEE802154E,ERR_MAXRXDATAPREPARE_OVERFLOWS,
                         (errorparameter_t)ieee154e_vars.state,
                         (errorparameter_t)ieee154e_dbg.num_cslSamples);
   // abort
   endOps();
}
...
```

Ilustración 106: Función *activity_csl_data_rie1* (**IEEE802154Ecsl.c**)

15.1.2.6 activity_csl_data_rie2

```
...
/**
 \brief Activity for CSL RX Sampling error [data rie2].
  This is triggered by #rt3 expiring, i.e. timer fires while state = S_CSLRXDATALISTEN. If no packet is received by the time, this timer expires, then none will be received ever and it is safe to switch of the radio. This timer is set such that the radio will be on during two consecutive windows of duration (#TsLongGT).
*/
port_INLINE void activity_csl_data_rie2() { // Activity for error event [data rie2] on CSL RX Sampling.
   // abort
   endOps();
}
...
```

Ilustración 107: Función *activity_csl_data_rie2* (**IEEE802154Ecsl.c**)

15.1.2.7 activity_csl_data_rie3

```
...
/**
 \brief Activity for CSL RX Sampling error [data rie3].

  This is triggered by #rt4 expiring, i.e. timer fires while state = S_CSLRXDATA. This is an error state which indicates the radio took too long to  transmit the data packet. The implemented behaviour is to log the error and move on the next CSL sample.
*/
port_INLINE void activity_csl_data_rie3() {
   // log the error
   openserial_printError(COMPONENT_IEEE802154E,ERR_WDDATADURATION_OVERFLOWS,
                         (errorparameter_t)ieee154e_vars.state,
                         (errorparameter_t)ieee154e_dbg.num_cslSamples);
   // abort
   endOps();
}
...
```

Ilustración 108: Función *activity_csl_data_rie3* (**IEEE802154Ecsl.c**)

15.1.2.8 activity_csl_data_rie4

```
...
/**
 \brief Activity for CSL RX Sampling stage [data rie4].

  This method is invoked from ISR-mode "isr_ieee154ecsl_timer" function when FSM timer fires (expiring "duration_rt5") while state = S_CSLTXACKOFFSET. The functionality is to prepare the radio for ACK sending.
*/
port_INLINE void activity_csl_data_rie4() {
  // log the error
  openserial_printError(COMPONENT_IEEE802154E,ERR_MAXTXACKPREPARE_OVERFLOWS,
                        (errorparameter_t)ieee154e_vars.state, (errorparameter_t)ieee154e_dbg.num_cslSamples);
  // abort
  endOps();
}
...
```

Ilustración 109: Función *activity_csl_data_rie4* (**IEEE802154Ecsl.c**)

15.1.2.9 activity_csl_data_rie5

```
...
/**
 \brief Activity for CSL RX Sampling stage [data rie5].
  This method is invoked from ISR-mode "isr_ieee154ecsl_timer" function when FSM timer fires (expiring "duration_rt7") while state = S_CSLTXACKDELAY. The functionality is to prepare the radio for ACK sending.
*/
port_INLINE void activity_csl_data_rie5() {
  // log the error
  openserial_printError(COMPONENT_IEEE802154E,ERR_WDRADIOTX_OVERFLOWS,
                        (errorparameter_t)ieee154e_vars.state,  errorparameter_t)ieee154e_dbg.num_cslSamples);
  // abort
  endOps();
}
...
```

Ilustración 110: Función *activity_csl_data_rie5* (**IEEE802154Ecsl.c**)

15.1.2.10 activity_csl_data_rie6

```
...
/**
 \brief Activity for CSL RX Sampling stage [data rie6].
  This method is invoked from ISR-mode "isr_ieee154ecsl_timer" function when FSM timer fires (expiring  rt8)
   while state = S_CSLTXACK. The functionality is to register an error when timer for sent ACK has expired.
*/
port_INLINE void activity_csl_data_rie6() {
   // log the error
   openserial_printError(COMPONENT_IEEE802154E,ERR_WDACKDURATION_OVERFLOWS,
                         (errorparameter_t)ieee154e_vars.state,  (errorparameter_t)ieee154e_dbg.num_cslSamples);
   // abort
   endOps();
}
...
```

Ilustración 111: Función *activity_csl_data_rie6* (**IEEE802154Ecsl.c**)

15.2. MODO TRANSMISIÓN

15.2.1. ACTIVIDADES

15.2.1.1 activity_csl_wakeup_ti1

```
...
/**
 \brief Activity for CSL TX stage [wake-up ti1].
 This method is invoked in ISR-mode from "isr_ieee154ecsl_txtimer_cb" function when TX Timer is fired (every
"macCSLTxChkFreq" time).
*/
port_INLINE void activity_csl_wakeup_ti1() {
    cellType_t  cellType;
    open_addr_t neighbor;

    // update CSL Mode to TX in order to avoid problems in FSM.
    ieee154e_vars.cslMode = CSL_TX_MODE;

    // Las acciones que realizaremos son las siguientes:
   //  - Avanzar el schedule para posicionar en la siguiente posición, el cual marcará la dirección a quien enviar.
   //  - Verificar el estado actual dentro de la FSM para comprobar que nos encontramos en un estado SLEEP.
   //  - Si no hay problema en ambos, verificar el tipo de slot en el schedule y preparar un paquete para enviar al
```

```
//    destino asociado a dicho slot.

// increment ASN (do this first so debug pins are in sync)
incrementAsnOffset();

// wiggle debug pins
debugpins_slot_toggle();
if (ieee154e_vars.slotOffset==0) {
   debugpins_frame_toggle();
}

// if the previous slot took too long, we will not be in the right state
if (ieee154e_vars.state!=S_SLEEP) {
   // log the error
   openserial_printError(COMPONENT_IEEE802154E,ERR_WRONG_STATE_IN_STARTSLOT,
                         (errorparameter_t)ieee154e_vars.state, (errorparameter_t)ieee154e_dbg.num_cslSamples);
   // abort
   endOps();
   return;
}

// advance the schedule
schedule_advanceSlot();

// stop using serial
openserial_stop();

// check the schedule to see what type of slot this is
cellType = schedule_getType();

// check whether we can send
if (schedule_getOkToSend()) {
    schedule_getNeighbor(&neighbor);
    ieee154e_vars.dataToSend = openqueue_macGetDataPacket(&neighbor);
} else {
    ieee154e_vars.dataToSend = NULL;
}
if ((ieee154e_vars.dataToSend!=NULL) && (cellType == CELLTYPE_TX {   // I have a packet to send
    // change state to start sending CSL preamble before send the data packet.
    changeState(S_CSLTXWAKEUPOFFSET);
```

```
      // record that I will attempt to transmit this packet
      ieee154e_vars.dataToSend->l2_numTxAttempts++;
      // arm tt1
      radiotimer_schedule(DURATION_tt1);
   } else {
      // abort because slot is not TX or there is no data to send, so we reset TX Mode state
      ieee154e_vars.cslMode = CSL_SLEEP_MODE;
   }
}
...
```

Ilustración 112: Función *activity_csl_wakeup_ti1* (**IEEE802154Ecsl.c**)

15.2.1.2 activity_csl_wakeup_ti2

```
...
/**
 \brief Activity for CSL TX stage [wake-up ti2].
  This method is invoked from ISR-mode "isr_ieee154ecsl_timer" function when FSM timer fires (expiring
  "duration_tt1") while state = S_CSLTXWAKEUPOFFSET.   The functionality is to prepare the radio for sending
   packets.
*/
port_INLINE void activity_csl_wakeup_ti2() {

   open_addr_t neighbor;
   uint16_t remaining_rztime;

   // El tiempo de rendezvous que será necesario esperar será macCSLMaxPeriod al tratarse de
   // una comunicación en modo TX no sincronizado.
   // Calculamos el valor del rz-time en cada trama wake-up de la secuencia dado que debe ir reduciéndose para
   // notificar al extemo remoto el tiempo restante hasta el envío de la trama de datos.

   // El tiempo lastCapturedTime se ve incrementado tras el envio de cada trama wake-up por lo que cada
   // vez tendrá un valor más proximo al macCSLMaxPeriod.

   ieee154e_vars.remainingRzTime = macCSLMaxPeriod - ieee154e_vars.lastCapturedTime;

   // Si da tiempo a enviar una nueva trama de wake-up antes de la finalización del ciclo...
   if (MaxWakeUpTxTime < ieee154e_vars.remainingRzTime) {
```

```
   // change state
   changeState(S_CSLTXWAKEUPPREPARE);

   //
   // Construimos el paquete Wake-Up con su valor RZTime asociado.
  //

  // obtenemos un buffer en el cual poder guardar los datos recibidos.

  ieee154e_vars.wakeupToSend = openqueue_getFreePacketBuffer(COMPONENT_IEEE802154E);
  if (ieee154e_vars.wakeupToSend==NULL) {
     // registro del error & fin de operaciones.
    openserial_printError(COMPONENT_IEEE802154E,ERR_NO_FREE_PACKET_BUFFER,
                          (errorparameter_t)0, (errorparameter_t)0);
    endOps();
    return;
  }

  // Declaración de propiedad sobre el paquete.
  ieee154e_vars.wakeupToSend->creator = COMPONENT_IEEE802154E;
  ieee154e_vars.wakeupToSend->owner   = COMPONENT_IEEE802154E;

  // El tipo de trama es Multipurpose.
  ieee154e_vars.wakeupToSend->l2_frameType = IEEE154_TYPE_MULTIPURPOSE;

  // El dsn del paquete se obtiene del actual DSN incrementado en cada paquete de la secuencia wake-up.
 ieee154e_vars.wakeupToSend->l2_dsn = ieee154e_vars.cslDSN++;

  // El destinatario es el mismo que el destinatario del mensaje de datos indicado en el schedule.
  schedule_getNeighbor(&neighbor);

 // Verificamos que se trata de una short address. En caso contrario generamos un mensaje de advertencia
 if(neighbor.type != ADDR_16B) {
     openserial_printError(COMPONENT_IEEE802154,ERR_IEEE154_UNSUPPORTED,
                           (errorparameter_t)1, (errorparameter_t)(neighbor.type));
 }
```

```
    // create frame header.
    ieee802154_createWakeUpFrame(ieee154e_vars.wakeupToSend,
                                 ieee154e_vars.wakeupToSend->l2_dsn,
                                 &neighbor,
                                 ieee154e_vars.remainingRzTime);

    // space for 2-byte CRC
    packetfunctions_reserveFooterSize(ieee154e_vars.wakeupToSend,2);
  }
  else {
    // update rztime in header.
    // Se posiciona 8 posiciones antes ya que escribimos el paquete de atras hacia delante y antes tenemos:
    // - 1 byte FCF
    // - 1 byte SEQ
    // - 2 bytes PANID
    // - 2 bytes DEST ADDR
    // - 2 bytes RZTIME IE HEADER
    // - 2 bytes RZTIME (el cual se sobrescribirá con el nuevo valor).
    *((uint8_t*)(ieee154e_vars.wakeupToSend->payload+8)) = (uint8_t)(ieee154e_vars.remainingRzTime &0xFF);
    *((uint8_t*)(ieee154e_vars.wakeupToSend->payload+9)) = (uint8_t)((ieee154e_vars.remainingRzTime >> 8) &
0xFF);
  }
   // load the packet in the radio's Tx buffer
   radio_loadPacket(ieee154e_vars.wakeupToSend->payload, ieee154e_vars.wakeupToSend->length);

   // enable the radio in Tx mode. This does not send the packet.
   radio_txEnable();
   ieee154e_vars.radioOnInit=radio_getTimerValue();
   ieee154e_vars.radioOnThisSlot=TRUE;

   // arm tt2
   radiotimer_schedule(DURATION_tt2);

   // change state
   changeState(S_CSLTXWAKEUPREADY);
  }
 else {
    // Si no da tiempo a enviar una nueva trama, esperamos un tiempo igual a remainingRzTime que será
    // el tiempo que queda pendiente hasta el envío de la trama de datos y actualizamos el estado para
```

```
    // pasar a la transmisión de éstos, a través del estado CSLTXDATAPREOFFSET.

    // Descartamos el paquete una vez finalizada la transmisión.
    openqueue_freePacketBuffer(ieee154e_vars.wakeupToSend);

    // clear local variable
    ieee154e_vars.wakeupToSend = NULL;

    // change state
    changeState(S_CSLTXDATAPREOFFSET);

    // arm remaining rendezvous time.
    radiotimer_schedule(ieee154e_vars.remainingRzTime);
   }
}
...
```

Ilustración 113: Función *activity_csl_wakeup_ti2* (**IEEE802154Ecsl.c**)

15.2.1.3 activity_csl_wakeup_ti3

```
...
/**
 \brief Activity for CSL TX stage [wake-up ti3].
  This method is invoked from ISR-mode "isr_ieee154ecsl_timer" function when FSM timer fires (expiring
  "duration_tt2") while state = S_CSLTXWAKEUPREADY.   The radio is configured and this interrupt gives it
   the "go" signal to start listening.
*/
port_INLINE void activity_csl_wakeup_ti3() {
   // change state
   changeState(S_CSLTXWAKEUPDELAY);

   // arm tt3
   radiotimer_schedule(DURATION_tt3);

   // give the 'go' to transmit
   radio_txNow();
}
...
```

Ilustración 114: Función *activity_csl_wakeup_ti3* (**IEEE802154Ecsl.c**)

15.2.1.4 activity_csl_wakeup_ti4

```
...
/**
 \brief Activity for CSL TX stage [wake-up ti4].

  This method is invoked from ISR-mode "ieee154ecsl_startOfFrame" function when a start of frame event fires
   while state = S_CSLRXWAKEUPDELAY. The functionality is to capture the time, cancel #tt3 and arm #tt4 (max
   time to send all the packet).
*/
port_INLINE void activity_csl_wakeup_ti4(PORT_RADIOTIMER_WIDTH capturedTime) {
   // cancel tt3
   radiotimer_cancel();

   // record the captured time
   ieee154e_vars.lastCapturedTime = capturedTime;

   // change state
   changeState(S_CSLTXWAKEUP);

   // arm tt4
   radiotimer_schedule(DURATION_tt4);
}
...
```

Ilustración 115: Función *activity_csl_wakeup_ti4* (**IEEE802154Ecsl.c**)

15.2.1.5 activity_csl_wakeup_ti5

```
...
/**
 \brief Activity for CSL RX Sampling stage [wake-up ri5].
  This method is invoked from ISR-mode "ieee154ecsl_endOfFrame" function when a end of frame event fires
   while state = S_CSLTXWAKEUP.
  The functionality is to change state, cancel #rt4, and start sending data frame and wait for ACK.
*/
port_INLINE void activity_csl_wakeup_ti5(PORT_RADIOTIMER_WIDTH capturedTime) {

  // record the captured time
  ieee154e_vars.lastCapturedTime = capturedTime;

   // change state
```

```
  changeState(S_CSLTXWAKEUPOFFSET);

  // arm tt1 (enviemos de nuevo una trama de wake-up o de datos, el tiempo será tt1).
  radiotimer_schedule(DURATION_tt1);
}
...
```

Ilustración 116: Función *activity_csl_wakeup_ti5* (**IEEE802154Ecsl.c**)

15.2.1.6 activity_csl_data_ti1

```
...
/**
 \brief Activity for CSL TX stage [data ti1].
  This method is invoked from ISR-mode "isr_ieee154ecsl_timer" function when FSM timer fires after expire
  remainingRzTime while state = S_CSLTXWAKEUPVALIDATE.
  The functionality is to prepare the data to be sent.
*/
port_INLINE void activity_csl_data_ti1() {
   // change state
  changeState(S_CSLTXDATAOFFSET);

   // arm tt1 (enviemos de nuevo una trama de wake-up o de datos, el tiempo será tt1).
   radiotimer_schedule(DURATION_tt1);
}
...
```

Ilustración 117: Función *activity_csl_data_ti1* (**IEEE802154Ecsl.c**)

15.2.1.7 activity_csl_data_ti2

```
...
/**
 \brief Activity for CSL TX stage [data ti2].
  This method is invoked from ISR-mode "isr_ieee154ecsl_timer" function when FSM timer fires (expiring
  "duration_tt1") while state = S_CSLTXDATAOFFSET. The functionality is to prepare the radio for sending
  packets.
*/
port_INLINE void activity_csl_data_ti2() {
  // change state
  changeState(S_CSLTXDATAPREPARE);

  // load the packet in the radio's Tx buffer
```

```
   radio_loadPacket(ieee154e_vars.dataToSend->payload,  ieee154e_vars.dataToSend->length);

   // enable the radio in Tx mode. This does not send the packet.
   radio_txEnable();
   ieee154e_vars.radioOnInit=radio_getTimerValue();
   ieee154e_vars.radioOnThisSlot=TRUE;

   // arm tt2
   radiotimer_schedule(DURATION_tt2);

   // change state
   changeState(S_CSLTXDATAREADY);
}
...
```

Ilustración 118: Función *activity_csl_data_ti2* (**IEEE802154Ecsl.c**)

15.2.1.8 activity_csl_data_ti3

```
...
/**
 \brief Activity for CSL TX stage [data ti3].
  This method is invoked from ISR-mode "isr_ieee154ecsl_timer" function when FSM timer fires (expiring
  “duration_tt2") while state = S_CSLTXDATAREADY. The radio is configured and this interrupt gives it the "go"
signal to start listening.
*/
port_INLINE void activity_csl_data_ti3() {
   // change state
   changeState(S_CSLTXDATADELAY);

   // arm tt3
   radiotimer_schedule(DURATION_tt3);

   // give the 'go' to transmit
   radio_txNow();
}
...
```

Ilustración 119: Función *activity_csl_data_ti3* (**IEEE802154Ecsl.c**)

15.2.1.9 activity_csl_data_ti4

```
...
/**
 \brief Activity for CSL TX stage [data ti4].
  This method is invoked from ISR-mode "ieee154ecsl_startOfFrame" function when a start of frame event fires
  while state = S_CSLTXDATADELAY. The functionality is to change state, cancel #tt3 and arm #tt4 (max time to
  sent all the packet).
*/
port_INLINE void activity_csl_data_ti4(PORT_RADIOTIMER_WIDTH capturedTime) {
   // change state
   changeState(S_CSLTXDATA);

   // cancel tt3
   radiotimer_cancel();

   // record the captured time
   ieee154e_vars.lastCapturedTime = capturedTime;

   // arm tt4
   radiotimer_schedule(DURATION_tt4);
}
...
```

Ilustración 120: Función *activity_csl_data_ti4* (**IEEE802154Ecsl.c**)

15.2.1.10 activity_csl_data_ti5

```
...
/**
 \brief Activity for CSL TX stage [data ri5].
  This method is invoked from ISR-mode "ieee154ecsl_endOfFrame" function when a end of frame event fires
  while state = S_CSLTXDATA. The functionality is to change state, cancel #tt4, and notify upper layer and
  schedule about successful TX. In case ACK required,  start process for receive ACK frame (arm #tt5).
*/
port_INLINE void activity_csl_data_ti5(PORT_RADIOTIMER_WIDTH capturedTime) {
   bool listenForAck;

   // change state
   changeState(S_CSLRXACKOFFSET);
```

```
 // cancel tt4
  radiotimer_cancel();

  // turn off the radio
  radio_rfOff();
  ieee154e_vars.radioOnTics+=(radio_getTimerValue()-ieee154e_vars.radioOnInit);

  // record the captured time
  ieee154e_vars.lastCapturedTime = capturedTime;

  // decides whether to listen for an ACK
  if (packetfunctions_isBroadcastMulticast(&ieee154e_vars.dataToSend->l2_nextORpreviousHop)==TRUE) {
     listenForAck = FALSE;
  } else {
     listenForAck = TRUE;
  }

  if (listenForAck==TRUE) {
     // arm tt5
     radiotimer_schedule(DURATION_tt5);
  } else {
     // indicate succesful Tx to schedule to keep statistics
     schedule_indicateTx(&ieee154e_vars.asn,TRUE);
     // indicate to upper later the packet was sent successfully
     notif_sendDone(ieee154e_vars.dataToSend,E_SUCCESS);
     // reset local variable
     ieee154e_vars.dataToSend = NULL;
     // abort
     endOps();
  }
}
...
```

Ilustración 121: Función *activity_csl_data_ti5* (**IEEE802154Ecsl.c**)

15.2.1.11 activity_csl_data_ti6

```
...
/**
 \brief Activity for CSL TX stage [data ti6].
  This method is invoked from ISR-mode "isr_ieee154ecsl_timer" function when FSM timer fires (expiring
  "duration_tt5") while state = S_CSLRXACKOFFSET. The functionality is to prepare the radio for ACK reception.
*/
port_INLINE void activity_csl_data_ti6() {
   // change state
   changeState(S_CSLRXACKPREPARE);

   // enable the radio in Rx mode. The radio is not actively listening yet.
   radio_rxEnable();

   //caputre init of radio for duty cycle calculation
   ieee154e_vars.radioOnInit=radio_getTimerValue();
   ieee154e_vars.radioOnThisSlot=TRUE;

   // arm tt6
   radiotimer_schedule(DURATION_tt6);

   // change state
   changeState(S_CSLRXACKREADY);
}
...
```

Ilustración 122: Función *activity_csl_data_ti6* (**IEEE802154Ecsl.c**)

15.2.1.12 activity_csl_data_ti7

```
...
/**
 \brief Activity for CSL TX stage [data ti7].
  This method is invoked from ISR-mode "isr_ieee154ecsl_timer" function when FSM timer fires (expiring
  "duration_tt6") while state = S_CSLRXACKREADY. The radio is configured and this interrupt gives it the "go"
   signal to start listening.
*/
port_INLINE void activity_csl_data_ti7() {
   // change state
   changeState(S_CSLRXACKLISTEN);
```

```
   // start listening
   radio_rxNow();

   // arm tt7
   radiotimer_schedule(DURATION_tt7);
}
...
```

Ilustración 123: Función *activity_csl_data_ti7* (**IEEE802154Ecsl.c**)

15.2.1.13 activity_csl_data_ti8

```
...
/**
 \brief Activity for CSL TX stage [data ri8].
  This method is invoked from ISR-mode "ieee154ecsl_startOfFrame" function when a start of frame event fires
   while state = S_CSLRXACKLISTEN.The functionality is to change the state, cancel #tt7 and arm #tt8 (max time
   to receive the ack packet).
*/
port_INLINE void activity_csl_data_ti8(PORT_RADIOTIMER_WIDTH capturedTime) {
   // change state
   changeState(S_CSLRXACK);

   // cancel tt7
   radiotimer_cancel();

   // record the captured time
   ieee154e_vars.lastCapturedTime = capturedTime;

   // arm tt8
   radiotimer_schedule(DURATION_tt8);
}
...
```

Ilustración 124: Función *activity_csl_data_ti8* (**IEEE802154Ecsl.c**)

15.2.1.14 activity_csl_data_ti9

```
...
/**
 \brief Activity for CSL TX stage [data ti9].
  This method is invoked from ISR-mode "ieee154ecsl_endOfFrame" function when a end of frame event fires
  while state = S_CSLRXACK. The functionality is to change the state, cancel #tt8, analyze ack packet and notify
  upper layer.
*/
port_INLINE void activity_csl_data_ti9(PORT_RADIOTIMER_WIDTH capturedTime) {
   ieee802154_header_iht     ieee802514_header;
   uint16_t                  lenIE;

   // change state
   changeState(S_CSLTXPROC);

   // cancel tt8
   radiotimer_cancel();

   // turn off the radio
   radio_rfOff();
   //compute tics radio on.
   ieee154e_vars.radioOnTics+=(radio_getTimerValue()-ieee154e_vars.radioOnInit);

   // record the captured time
   ieee154e_vars.lastCapturedTime = capturedTime;

   // get a buffer to put the (received) ACK in
   ieee154e_vars.ackReceived = openqueue_getFreePacketBuffer(COMPONENT_IEEE802154E);
   if (ieee154e_vars.ackReceived==NULL) {
      // log the error
      openserial_printError(COMPONENT_IEEE802154E,ERR_NO_FREE_PACKET_BUFFER,
                            (errorparameter_t)0,
                            (errorparameter_t)0);
      // abort
      endOps();
      return;
   }
```

```
// declare ownership over that packet
ieee154e_vars.ackReceived->creator = COMPONENT_IEEE802154E;
ieee154e_vars.ackReceived->owner   = COMPONENT_IEEE802154E;

/*
The do-while loop that follows is a little parsing trick.  Because it contains a while(0) condition, it gets executed
only once. Below the do-while loop is some code to cleans up the ack variable. Anywhere in the do-while loop,
a break statement can be called to jump to the clean up code early. If the loop ends without a break, the
received  packet was correct. If it got aborted early (through a break), the packet was faulty.
*/
do { // this "loop" is only executed once
  // retrieve the received ack frame from the radio's Rx buffer
  ieee154e_vars.ackReceived->payload = &(ieee154e_vars.ackReceived->packet[FIRST_FRAME_BYTE]);
  radio_getReceivedFrame(   ieee154e_vars.ackReceived->payload,
                            &ieee154e_vars.ackReceived->length,
                            sizeof(ieee154e_vars.ackReceived->packet),
                            &ieee154e_vars.ackReceived->l1_rssi,
                            &ieee154e_vars.ackReceived->l1_lqi,
                            &ieee154e_vars.ackReceived->l1_crc);

  // break if wrong length
  if (ieee154e_vars.ackReceived->length<LENGTH_CRC ||
     ieee154e_vars.ackReceived->length>LENGTH_IEEE154_MAX) {
     // break from the do-while loop and execute the clean-up code below
    openserial_printError(COMPONENT_IEEE802154E,ERR_INVALIDPACKETFROMRADIO,
                  (errorparameter_t)1, ieee154e_vars.ackReceived->length);
     break;
  }

  // toss CRC (2 last bytes)
  packetfunctions_tossFooter(   ieee154e_vars.ackReceived, LENGTH_CRC);

  // break if invalid CRC
  if (ieee154e_vars.ackReceived->l1_crc==FALSE) {
     // break from the do-while loop and execute the clean-up code below
     break;
  }
```

```
// parse the IEEE802.15.4 header (RX ACK)
ieee802154_retrieveHeader(ieee154e_vars.ackReceived,&ieee802514_header);

// break if invalid IEEE802.15.4 header
if (ieee802514_header.valid==FALSE) {
   // break from the do-while loop and execute the clean-up code below
   break;
}

// store header details in packet buffer
ieee154e_vars.ackReceived->l2_frameType  = ieee802514_header.frameType;
ieee154e_vars.ackReceived->l2_dsn        = ieee802514_header.dsn;
memcpy(&(ieee154e_vars.ackReceived->l2_nextORpreviousHop),
           &(ieee802514_header.src),sizeof(open_addr_t));

// toss the IEEE802.15.4 header
packetfunctions_tossHeader(ieee154e_vars.ackReceived,ieee802514_header.headerLength);

// break if invalid ACK
if (isValidAck(&ieee802514_header,ieee154e_vars.dataToSend)==FALSE) {
   // break from the do-while loop and execute the clean-up code below
   break;
}
//hanlde IEs --xv poipoi
if (ieee802514_header.ieListPresent==FALSE){
   break; //ack should contain IEs.
}

if (ieee154e_processIEs(ieee154e_vars.ackReceived,&lenIE)==FALSE){
  // invalid IEs in ACK
  break;
}

// toss the IEs
packetfunctions_tossHeader(ieee154e_vars.ackReceived,lenIE);

// inform schedule of successful transmission
schedule_indicateTx(&ieee154e_vars.asn,TRUE);
```

```
      // inform upper layer
      notif_sendDone(ieee154e_vars.dataToSend,E_SUCCESS);
      ieee154e_vars.dataToSend = NULL;

      // in any case, execute the clean-up code below (processing of ACK done)
   } while (0);

   // free the received ack so corresponding RAM memory can be recycled
   openqueue_freePacketBuffer(ieee154e_vars.ackReceived);

   // clear local variable
   ieee154e_vars.ackReceived = NULL;

   // official end of Tx
   endOps();
}
...
```

Ilustración 125: Función *activity_csl_data_ti9* (**IEEE802154Ecsl.c**)

15.2.2. ACTIVIDADES DE EXCEPCIÓN

15.2.2.1 activity_csl_wakeup_tie1

```
...
/**
 \brief Activity for CSL TX stage [wake-up tie1].
  This is triggered by #tt2 expiring, i.e. timer fires while state = S_CSLTXWAKEUPPREPARE. This is really an
   error state which indicates that #ti2 did not have enough time to execute. Chances are to set
   maxTxDataPrepare too small. The implemented behaviour is to log the error and finish.
*/
port_INLINE void activity_csl_wakeup_tie1() {
   // log the error
   openserial_printError(COMPONENT_IEEE802154E,ERR_MAXTXDATAPREPARE_OVERFLOW,
                         (errorparameter_t)ieee154e_vars.state, (errorparameter_t)ieee154e_vars.slotOffset);
   // abort
   endOps();
}
...
```

Ilustración 126: Función *activity_csl_wakeup_tie1* (**IEEE802154Ecsl.c**)

15.2.2.2 activity_csl_wakeup_tie2

```
...
/**
 \brief Activity for CSL TX stage [wakeup tie2].
  This is triggered by #tt3 expiring, i.e. timer fires while state = S_CSLTXWAKEUPLISTEN. If no packet is
  received by the time, this timer expires, then  none will be received ever and it is safe to switch of the radio. This
  timer is set such that the radio will be on during two consecutive windows of duration (#TsLongGT).
*/
port_INLINE void activity_csl_wakeup_tie2() {
   // log the error
   openserial_printError(COMPONENT_IEEE802154E,ERR_WDRADIO_OVERFLOWS,
                         (errorparameter_t)ieee154e_vars.state, (errorparameter_t)ieee154e_vars.slotOffset);
   // abort
   endOps();
}
...
```

Ilustración 127: Función *activity_csl_wakeup_tie2* (**IEEE802154Ecsl.c**)

15.2.2.3 activity_csl_wakeup_tie3

```
...
/**
 \brief Activity for CSL TX stage [wake-up tie3].

  This is triggered by #tt4 expiring, i.e. timer fires while state = S_CSLTXWAKEUP. This is an error state which
  indicates the radio took too long to transmit the data packet. The implemented behaviour is to log the error and
  finish.
*/
port_INLINE void activity_csl_wakeup_tie3() {
   // log the error
   openserial_printError(COMPONENT_IEEE802154E,ERR_WDDATADURATION_OVERFLOWS,
                         (errorparameter_t)ieee154e_vars.state, (errorparameter_t)ieee154e_vars.slotOffset);
   // abort
   endOps();
}
...
```

Ilustración 128: Función *activity_csl_wakeup_tie3* (**IEEE802154Ecsl.c**)

15.2.2.4 activity_csl_data_tie1

```
...
/**
 \brief Activity for CSL TX stage [data tie1].
  This is triggered by #tt2 expiring, i.e. timer fires while state = S_CSLTXDATAPREPARE. This is really an error
  state which indicates that #ti2 did not have enough time to execute. Chances are to set maxTxDataPrepare too
  small. The implemented behaviour is to log the error and finish.
*/
port_INLINE void activity_csl_data_tie1() {
   // log the error
   openserial_printError(COMPONENT_IEEE802154E,ERR_MAXTXDATAPREPARE_OVERFLOW,
                         (errorparameter_t)ieee154e_vars.state, (errorparameter_t)ieee154e_vars.slotOffset);
   // abort
   endOps();
}
...
```

Ilustración 129: Función *activity_csl_data_tie1* (**IEEE802154Ecsl.c**)

15.2.2.5 activity_csl_data_tie2

```
...
/**
 \brief Activity for CSL TX Sampling error [data tie2].
  This is triggered by #tt3 expiring, i.e. timer fires while state = S_CSLTXDATADELAY. If no packet is received by
  the time, this timer expires, then none will be sent and it is safe to switch off the radio. This timer is set such that
  the radio will be on during two consecutive windows of duration (#TsLongGT).
*/
port_INLINE void activity_csl_data_tie2() {
   // log the error
   openserial_printError(COMPONENT_IEEE802154E,ERR_WDRADIO_OVERFLOWS,
                         (errorparameter_t)ieee154e_vars.state, (errorparameter_t)ieee154e_vars.slotOffset);
   // abort
   endOps();
}
...
```

Ilustración 130: Función *activity_csl_data_tie2* (**IEEE802154Ecsl.c**)

15.2.2.6 activity_csl_data_tie3

```
...
/**
 \brief Activity for CSL TX stage [data tie3].
  This is triggered by #tt4 expiring, i.e. timer fires while state = S_CSLTXDATA. This is an error state which
  indicates the radio took too long  to transmit the data packet. The implemented behaviour is to log the error and
  finish.
*/
port_INLINE void activity_csl_data_tie3() {
   // log the error
   openserial_printError(COMPONENT_IEEE802154E,ERR_WDDATADURATION_OVERFLOWS,
                         (errorparameter_t)ieee154e_vars.state, (errorparameter_t)ieee154e_vars.slotOffset);
   // abort
   endOps();
}
...
```

Ilustración 131: Función *activity_csl_data_tie3* (**IEEE802154Ecsl.c**)

15.2.2.7 activity_csl_data _tie4

```
...
/**
 \brief Activity for CSL TX stage [data tie4].
  This method is invoked from ISR-mode "isr_ieee154ecsl_timer" function when FSM timer fires (expiring
  "duration_tt6") while state = S_CSLRXACKPREPARE.  The functionality is to prepare the radio for ACK sending.
*/
port_INLINE void activity_csl_data_tie4() {
   // log the error
   openserial_printError(COMPONENT_IEEE802154E,ERR_MAXRXACKPREPARE_OVERFLOWS,
                         (errorparameter_t)ieee154e_vars.state, (errorparameter_t)ieee154e_vars.slotOffset);
   // abort
   endOps();
}
...
```

Ilustración 132: Función *activity_csl_data_tie4* (**IEEE802154Ecsl.c**)

15.2.2.8 activity_csl_data _tie5

```
...
/**
 \brief Activity for CSL TX stage [data tie5].
  This method is invoked from ISR-mode "isr_ieee154ecsl_timer" function when FSM timer fires (expiring
  "duration_tt7") while state = S_CSLRXACKLISTEN. The functionality is to notify upper level, updating the
  number of remaining retries of the packet.
*/
port_INLINE void activity_csl_data_tie5() {
   // indicate transmit failed to schedule to keep stats
   schedule_indicateTx(&ieee154e_vars.asn,FALSE);

   // decrement transmits left counter
   ieee154e_vars.dataToSend->l2_retriesLeft--;

   if (ieee154e_vars.dataToSend->l2_retriesLeft==0) {
      // indicate tx fail if no more retries left
      notif_sendDone(ieee154e_vars.dataToSend,E_FAIL);
   } else {
      // return packet to the virtual COMPONENT_SIXTOP_TO_IEEE802154E component
      ieee154e_vars.dataToSend->owner = COMPONENT_SIXTOP_TO_IEEE802154E;
   }

   // reset local variable
   ieee154e_vars.dataToSend = NULL;

   // abort
   endOps();
}
...
```

Ilustración 133: Función *activity_csl_data_tie5* (**IEEE802154Ecsl.c**)

15.2.2.9 activity_csl_data _tie6

```
...
/**
 \brief Activity for CSL TX stage [data tie6].
  This method is invoked from ISR-mode "isr_ieee154ecsl_timer" function when FSM timer fires (expiring
  "duration_rt8") while state = S_CSLRXACK. The functionality is to finish and clean registers.
*/
port_INLINE void activity_csl_data_tie6() {
   // abort
   endOps();
}
...
```

Ilustración 134: Función *activity_csl_data_tie6* (**IEEE802154Ecsl.c**)