

UNIVERSIDAD POLITÉCNICA DE MADRID
ESCUELA TÉCNICA SUPERIOR DE INGENIEROS DE
TELECOMUNICACIÓN



TRABAJO FIN DE GRADO

Grado en Ingeniería de Tecnologías y Servicios de
Telecomunicación

IMPLEMENTACIÓN DE ESTRATEGIAS DE
OPTIMIZACIÓN EN UN BANCO DE PRUEBAS PARA
REDES INALÁMBRICAS DE SENSORES COGNITIVAS

Manuel Alarcón Granero

Julio 2015

TRABAJO FIN DE GRADO

Título: IMPLEMENTACIÓN DE ESTRATEGIAS DE OPTIMIZACIÓN
EN UN BANCO DE PRUEBAS PARA REDES INALÁMBRICAS
DE SENSORES COGNITIVAS

Autor: MANUEL ALARCÓN GRANERO

Tutor: ELENA ROMERO PERALES

Ponente: ALVARO ARAUJO PINTO

Departamento: DEPARTAMENTO DE INGENIERÍA ELECTRÓNICA

TRIBUNAL

Presidente: D. OCTAVIO NIETO-TALADRIZ GARCÍA

Vocal: D. ÁLVARO DE GUZMÁN FERNÁNDEZ GONZÁLEZ

Secretario: D. ALVARO ARAUJO PINTO

Suplente: D. MIGUEL ÁNGEL SÁNCHEZ GARCÍA

CALIFICACIÓN:

Madrid, a de de 2015.

UNIVERSIDAD POLITÉCNICA DE MADRID
ESCUELA TÉCNICA SUPERIOR DE INGENIEROS DE
TELECOMUNICACIÓN



TRABAJO FIN DE GRADO

Grado en Ingeniería de Tecnologías y Servicios de
Telecomunicación

IMPLEMENTACIÓN DE ESTRATEGIAS DE
OPTIMIZACIÓN EN UN BANCO DE PRUEBAS PARA
REDES INALÁMBRICAS DE SENSORES COGNITIVAS

Manuel Alarcón Granero

Julio 2015

Resumen

Una red de sensores es aquella formada por una serie de dispositivos con acceso a información del medio cuya misión es la de monitorizar diferentes parámetros del entorno. Estas redes, denominadas Wireless Sensor Networks (WSNs), están formadas por dispositivos con conectividad inalámbrica lo que les da mayor versatilidad y flexibilidad, siendo un reto la autonomía energética y la fiabilidad de las comunicaciones.

Para hacer frente a los retos de fiabilidad, autonomía y seguridad, una evolución de estas redes se ha centrado en proporcionar a los nodos de la red capacidad cognitiva para captar el estado del espectro y así cambiar de forma adaptativa los parámetros de las transmisiones. Esta evolución se denomina Cognitive Wireless Sensor Networks (CWSNs).

En el contexto de la Universidad Politécnica de Madrid, dentro del B105 - Electronic Systems Lab (B105 lab) del Departamento de Ingeniería Electrónica (DIE), estas redes son uno de los principales objetos de investigación. Uno de los proyectos dentro de éste ámbito es la puesta en marcha de un banco de pruebas para CWSNs usando el nodo cognitive New Generation Device (cNGD) desarrollado en el mismo laboratorio.

El trabajo desarrollado durante este Trabajo Fin de Grado (TFG) que se recoge en este documento se engloba dentro de este proyecto y ha consistido en la implementación de estrategias de optimización en las áreas de la seguridad y el consumo en el banco de pruebas.

El trabajo incluye la comprensión del hardware y software que se ha utilizado. Además, en esta etapa se incluye el estudio de las estrategias que se han implementado.

Para la implementación se han tenido que desarrollar las funciones necesarias dentro de la arquitectura cognitiva que implementa el cNGD. Esto engloba tanto la creación de nuevas funciones como la adaptación de otras ya implementadas en el nodo.

Para finalizar, se ha realizado una aplicación demostradora que muestra el funcionamiento del sistema mediante la modificación de ciertos parámetros de transmisión. Con esta aplicación se han realizado diferentes pruebas de funcionamiento en los nodos comprobando la validez de la implementación.

PALABRAS CLAVE:

Redes de Sensores Inalámbricas Cognitivas, cNGD, estrategias de optimización, banco de pruebas.

Abstract

A Wireless Sensor Network is one formed by a series of devices with access to information of the medium whose task is to sense various parameters of the environment. These networks, called Wireless Sensor Networks (WSNs) consist of devices with wireless connectivity, giving them more versatility and flexibility. The energy consumption and reliability of communications are still a challenge in these networks.

To address the challenges of reliability, autonomy and security, an evolution of these networks has focused on providing network nodes cognitive ability to sense the state of the spectrum to adaptively change the transmissions' parameters. This evolution is called Cognitive Wireless Sensor Network (CWSN).

In the context of the Universidad Politécnica de Madrid (UPM), within the B105 - Electronic Systems Lab (B105 lab) of the Departamento de Ingeniería Electrónica (DIE), these networks are one of the main purposes of research. One project in this area is the implementation of a testbed for CWSNs using the cognitive New Generation Device (cNGD) node developed in the same lab.

The work developed in this Trabajo Fin de Grado (TFG) as set out in this document is included within this project and consisted in the implementation of optimization strategies in the areas of security and energy consumption in the testbed.

The work includes the understanding of the hardware and software used. In addition, at this stage, the study of the strategies that have been implemented is included.

For implementation there have developed the necessary functions within the cognitive architecture that implements the cNGD. This covers both the creation of new functions such as adaptation of already implemented in the node.

Finally, there has been implemented a demonstrator application showing the operation of the system by changing certain parameters of transmission. With this application different functional tests have been executed in the nodes checking the validity of the implementation.

KEY WORDS:

Cognitive Wireless Sensor Network, cNGD, optimization strategies, testbed.

Índice

| | |
|---|------|
| Resumen..... | VII |
| Abstract | IX |
| Lista de figuras, ecuaciones y tablas | XIII |
| Capítulo 1. Introducción..... | 1 |
| 1.1. Objetivos | 3 |
| 1.2. Desarrollo del trabajo..... | 4 |
| 1.3. Estructura de la memoria..... | 4 |
| Capítulo 2. Estudio previo | 5 |
| 2.1. Hardware del cNGD..... | 5 |
| 2.2. Firmware del cNGD | 6 |
| 2.3. Arquitectura cognitiva..... | 8 |
| 2.4. Herramientas utilizadas | 9 |
| 2.4.1. MPLAB X | 9 |
| 2.4.2. Programador ICD 3 | 9 |
| 2.4.3. RS232SHIELD | 10 |
| Capítulo 3. Implementación del algoritmo de seguridad..... | 11 |
| 3.1. Funciones de la arquitectura cognitiva | 11 |
| 3.1.1. Estructura de datos | 12 |
| 3.1.2. Optimizer..... | 13 |
| 3.1.3. Repository | 16 |
| 3.1.4. Execution..... | 17 |
| 3.1.5. VCC | 17 |
| Capítulo 4. Implementación del algoritmo de reducción de consumo..... | 19 |
| 4.1. Funciones de la arquitectura cognitiva | 19 |
| 4.1.1. Estructura de datos | 20 |
| 4.1.2. Optimizer..... | 21 |
| 4.1.3. Repository | 25 |
| 4.1.4. VCC | 28 |
| 4.1.5. Execution..... | 28 |
| 4.1.6. Discovery | 28 |
| Capítulo 5. Aplicación de prueba de los algoritmos y resultados | 29 |
| 5.1. Diseño..... | 29 |
| 5.2. Implementación | 30 |
| 5.3. Resultados obtenidos..... | 31 |

| | |
|--|----|
| Capítulo 6. Conclusiones y líneas futuras..... | 33 |
| 6.1. Conclusiones..... | 33 |
| 6.2. Líneas futuras | 34 |
| Referencias..... | 36 |
| Lista de acrónimos | 38 |

Lista de figuras, ecuaciones y tablas

| | |
|---|----|
| Figura 1.1 Topologías de red de las WSNs, obtenida de [1]..... | 1 |
| Figura 1.2 Reparto del espectro radioeléctrico en EEUU, de [12] | 2 |
| Figura 1.3 Bandas ISM resaltadas en el rango de los 300 MHz a los 8 GHz, obtenida de [13] | 2 |
| Figura 1.4 Diagrama de funcionamiento de la radio cognitiva, obtenida de [14] | 3 |
| Figura 2.1 Vista detallada del cNGD, obtenida de [18] | 5 |
| Figura 2.2 Diagrama de bloques de la adaptación software de la pila de protocolos MiWi, obtenida de [20]..... | 7 |
| Figura 2.3 Arquitectura del firmware del cNGD, obtenida de [20]..... | 7 |
| Figura 2.4 CAgents del Connectivity Brokerage, obtenido de [23] | 8 |
| Figura 2.5 Arquitectura del CRModule, obtenida de [13] | 8 |
| Figura 2.6 Nodo con rs232SHIELD conectado, obtenida de [17] | 10 |
| Figura 3.1 Etapas del algoritmo de seguridad..... | 11 |
| Figura 3.2 Paso de coordenadas de Optimizer a Repository | 14 |
| Figura 3.3 Ejemplo de mapa de clusters, obtenido de [27] | 15 |
| Figura 3.4 Paso de mensajes cuando se detecta un nodo atacante | 18 |
| Figura 4.1 Diagrama de transiciones de la implementación propuesta | 20 |
| Figura 4.2 Diagrama con las acciones que realiza la función Cons y los parámetros Action que se pasan cuando inicia el algoritmo el propio nodo..... | 22 |
| Figura 4.3 Diagrama con las acciones que realiza la función Cons y los parámetros Action que se pasan cuando inicia el algoritmo otro nodo de la red | 23 |
| Figura 4.4 Diagrama de mensajes entre sub-módulos cuando se recibe una petición de cambio de canal | 24 |
| Figura 4.5 Diagrama de mensajes entre sub-módulos cuando se recibe una respuesta a una petición de cambio de canal | 24 |
| Figura 4.6 Diagrama de mensajes entre sub-módulos cuando se recibe una respuesta de cambio de canal con un canal diferente al propuesto inicialmente..... | 25 |
| Figura 4.7 Diagrama de peticiones a Optimizer cuando se recibe información de sensado de otro nodo y éste se encuentra en estado Clear..... | 27 |
| Figura 4.8 Diagrama de peticiones a Optimizer cuando se recibe información de sensado de otro nodo y éste se encuentra en estado esperando decisión resto de nodos..... | 27 |
| Figura 4.9 Diagrama de peticiones a Optimizer cuando se recibe información de sensado de otro nodo y éste se encuentra en estado esperando decisión final..... | 27 |
| Figura 5.1 Propuesta de escenario de CWSN en el B-105 lab | 29 |
| Figura 5.2 Diagrama de ejecución de la aplicación | 31 |
| Ecuación 3.1 Cálculo de nuevo radio y centro cuando se recibe un paquete que pertenece a un cluster..... | 15 |
| Ecuación 3.2 Cálculo de la distancia entre dos puntos | 15 |
| Tabla 3.1 Funciones implementadas para el algoritmo de seguridad | 13 |
| Tabla 3.2 Valores del parámetro Action de los mensajes dirigidos a Repository | 16 |
| Tabla 3.3 Valores del parámetro DataType de los mensajes dirigidos a Repository | 16 |
| Tabla 3.4 Ejemplo de tabla de atacantes inicializada con dos nodos en la red | 17 |
| Tabla 4.1 Funciones nuevas o modificadas para el algoritmo de reducción del consumo..... | 21 |
| Tabla 4.2 Valores del parámetro Action de los mensajes con destino Optimizer | 21 |

| | |
|--|----|
| Tabla 4.3 Valores del parámetro Param1 cuando se recibe un mensaje con Action igual a ActProcRq..... | 22 |
| Tabla 3.2 Valores del parámetro Action de los mensajes dirigidos a Repository | 25 |
| Tabla 3.3 Valores del parámetro DataType de los mensajes dirigidos a Repository | 26 |

Capítulo 1. Introducción

Una red de sensores inalámbrica o Wireless Sensor Network (WSN) consiste en una serie de dispositivos, llamados nodos, distribuidos a lo largo de un área geográfica sobre la que recogen datos. Estos nodos son capaces de conectarse de manera inalámbrica entre ellos y de procesar los datos que recogen para tomar decisiones.

Los nodos de estas redes suelen ser dispositivos sencillos. Normalmente constan de un microcontrolador para procesar y almacenar los datos que reciben, los sensores que recogen la información, un transceptor para comunicarse con el resto de nodos de la red y una fuente de energía, normalmente baterías.

El número de nodos que forman una red de este tipo puede variar desde unos pocos hasta varios cientos y pueden conectarse siguiendo diferentes topologías como podemos ver en la Figura 1.1.

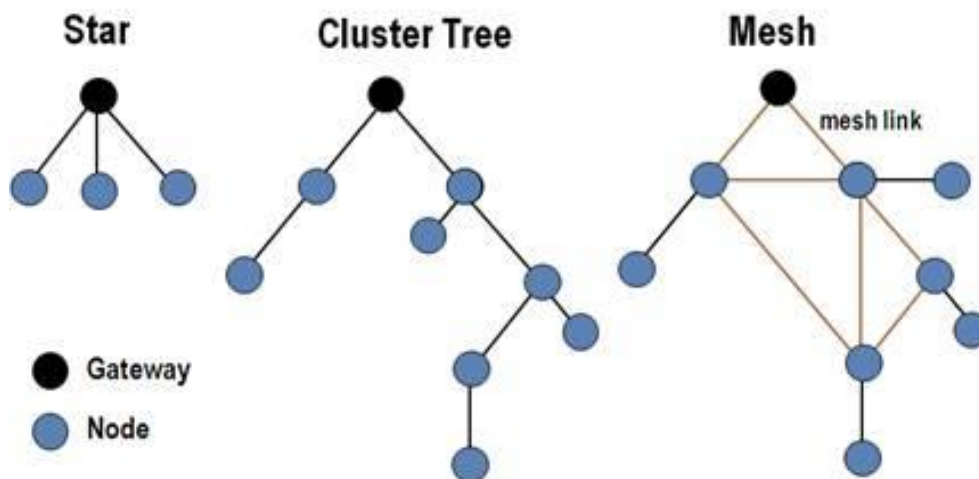
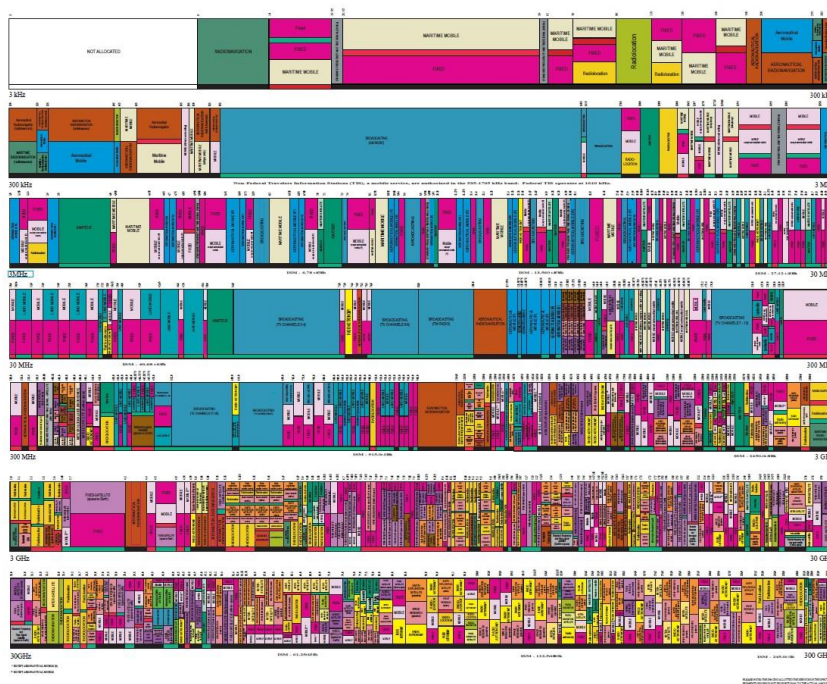


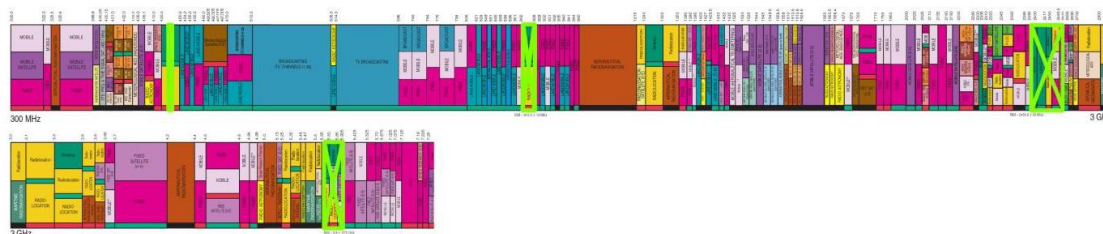
Figura 1.1 Topologías de red de las WSNs, obtenida de [1]

En cuanto a las diferentes tecnologías y protocolos de comunicación implementados en WSNs la mayoría están basados en el estándar 802.15.4 del *Institute of Electrical and Electronics Engineers* (IEEE) [2] para Wireless Personal Area Network (WPAN). El propósito de este estándar está centrado en la habilitación de comunicación entre dispositivos con bajo coste y velocidad. Se definen los niveles de enlace (MAC) y físico (PHY) del modelo Open Systems Interconnection (OSI). Algunos de los protocolos basados en este estándar son ZigBee [3], WirelessHART [4], ISA100.11 [5] y MiWi [6]. Otros estándares muy utilizados son el IEEE 802.11 [7] en el que está basado Wi-Fi [8] y el 802.15.1 [9] en el que está basado Bluetooth [10].

En los últimos años se ha producido un incremento en el uso de dispositivos con conectividad inalámbrica. En 2013, las ventas mundiales de móviles inteligentes (smartphones) alcanzaron 968 millones de unidades, lo que supone un aumento del 42,3% respecto al 2012 [11]. El uso de las bandas Industrial, Scientific and Medical (ISM) por parte de estos dispositivos ha generado saturación en el espectro, lo que ha repercutido en la calidad de servicio, Quality of Service (QoS), dificultando la conectividad.



Como podemos ver en la Figura 1.2, donde se muestra el reparto del espectro en EEUU en el año 2011, el grado de saturación y fragmentación del espectro es enorme, coincidiendo este problema también en Europa. El rango de frecuencias asignado a las bandas ISM es muy pequeño, lo que queda representado en la Figura 1.3. Esta circunstancia ha impulsado la búsqueda de soluciones para disminuir el grado de saturación en estas bandas tan utilizadas.



Una de las soluciones es la Radio Cognitiva, Cognitive Radio (CR), con dispositivos capaces de conocer el estado del espectro y colaborar con los diferentes elementos de la red. Esta colaboración consiste en el intercambio de información del espectro para mejorar la eficiencia espectral y optimizar parámetros globales como el consumo, la fiabilidad o la seguridad.

Un transceptor radio de un dispositivo con CR es capaz de conocer el estado del espectro electromagnético y de adaptarse a él, transmitiendo en frecuencias que no estén siendo utilizadas o las que tengan menor ruido. Esto lo vemos resumido en la Figura 1.4.

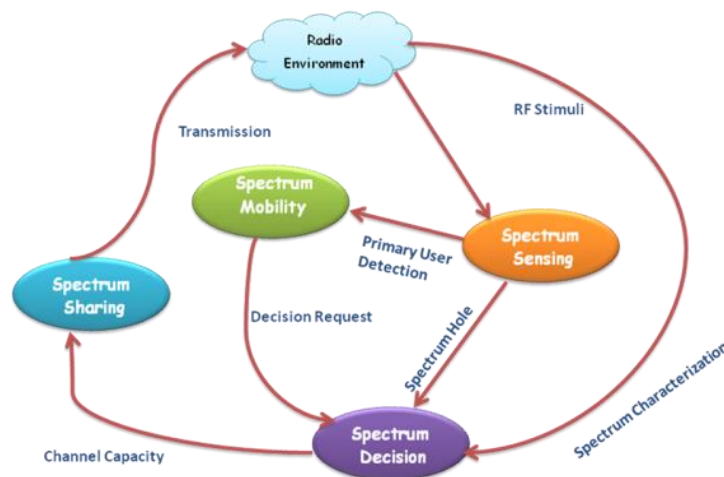


Figura 1.4 Diagrama de funcionamiento de la radio cognitiva, obtenida de [14]

Los dispositivos que incorporan la CR pueden transmitir y recibir en diferentes frecuencias, tanto en bandas libres como en bandas bajo licencia. Algunos de los estándares en los que se está trabajando con este concepto son el grupo 802.22 del IEEE para WRAN (Wireless Regional Area Network), que aprovecha las bandas de televisión no utilizadas [15], o el estándar del Grupo 2 del 802.15 del IEEE [16].

Estas técnicas no solo van a ayudar a optimizar las comunicaciones en las redes que lo implementen, sino que será beneficioso para el resto de redes ya que al ajustar los parámetros de transmisión restarán saturación en las frecuencias que utilicen el resto de redes.

De la aplicación de las técnicas anteriores sobre WSNs surge el concepto de redes de sensores inalámbricas cognitivas, Cognitive Wireless Sensor Networks (CWSNs). Una de las metas principales de este ámbito es el diseño de nodos con las prestaciones necesarias para incorporar el concepto de CR. Además, los nodos deben poder alimentarse con baterías y no ser necesaria una continua supervisión de estos.

Debido a la escasez de bancos de pruebas para CWSNs que existen en la actualidad, uno de los objetivos principales del B105 – Electronic Systems Lab (B105 lab) es el despliegue de un banco de pruebas para estas redes utilizando el nodo cognitive New Generation Device (cNGD) desarrollado en el mismo laboratorio.

1.1. Objetivos

El objetivo principal del trabajo es la implementación de estrategias de optimización en los ámbitos de la seguridad y el consumo en un banco de pruebas para CWSNs. Para la consecución de este objetivo se dividirá en varios objetivos secundarios que se puedan abordar de forma sencilla. Estos objetivos van a ser los siguientes:

- Familiarización con las herramientas. Consiste en la lectura de trabajos realizados anteriormente sobre los nodos y la ejecución de aplicaciones ya implementadas en estos.
- Estudio y comprensión de los algoritmos. Como el objetivo principal es la implementación de estrategias de optimización, otra tarea muy importante es la comprensión de dichas estrategias. La primera estrategia estará centrada en la seguridad de las CWSNs y la segunda en la reducción del consumo.

- Implementación. Se implementarán dos estrategias de optimización. Este objetivo incluye el desarrollo del código.
- Desarrollo de la aplicación demostradora. Se tendrá que implementar una aplicación demostradora que valide el código desarrollado.
- Pruebas. Por último se realizarán todas las pruebas que sean necesarias para comprobar todos los casos en los que se puedan encontrar los nodos con una red completa y que la ejecución del código sea correcta.

1.2. Desarrollo del trabajo

El trabajo se ha dividido en las siguientes etapas:

- Estudio previo. El primer paso tomado para la consecución de este trabajo ha sido la adquisición de conocimientos sobre las CWSNs y del entorno de trabajo.
 - Estudio de trabajos realizados anteriormente sobre CWSNs en el laboratorio y de la documentación facilitada sobre los algoritmos que se han implementado.
 - Familiarización con las herramientas de programación de Microchip, en concreto MPLAB X, y revisión de los conceptos de programación de microcontroladores en C.
- Desarrollo. Consiste en la implementación de los dos algoritmos sobre los nodos disponibles en el laboratorio. Tras esto, se ha desarrollado una aplicación que sirva de demostración del correcto funcionamiento del código implementado.
- Pruebas. Se han realizado las pruebas necesarias para comprobar el comportamiento de los algoritmos en el nodo.
- Documentación. Escritura de esta memoria.

1.3. Estructura de la memoria

Este documento va a seguir la siguiente estructura.

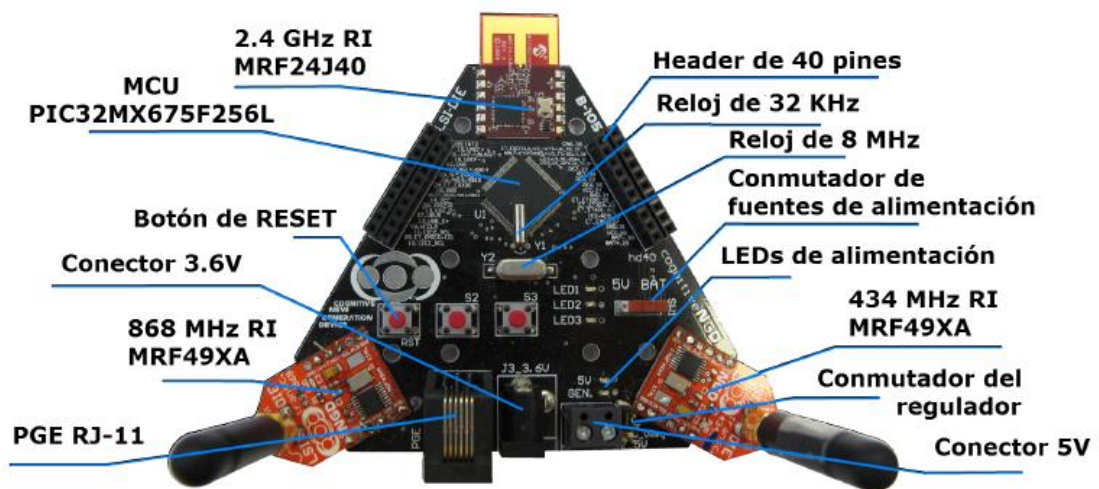
- En el capítulo 2 se caracteriza el *hardware* (HW) y el *software* (SW) sobre el que se ha realizado el trabajo y se detallan las herramientas que se han utilizado.
- En los capítulos 3 y 4 se detalla el proceso seguido para la implementación de ambos algoritmos, describiendo cada una de las funciones que se han realizado en cada uno de los sub-módulos de la arquitectura cognitiva.
- En el capítulo 5 se explican las decisiones para el diseño y la implementación de la aplicación de prueba de las estrategias.
- Por último, en el capítulo 6 se exponen las conclusiones y se plantean las líneas futuras de los trabajos relacionados con este.
- Para finalizar, se incluye una lista de referencias y otra de acrónimos utilizados.

Capítulo 2. Estudio previo

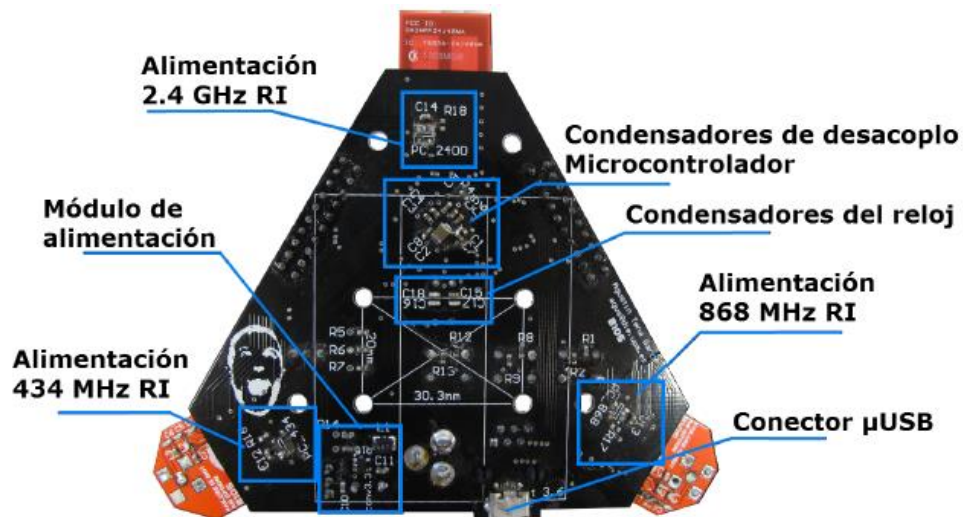
En este capítulo se va a presentar la plataforma sobre la que se ha realizado el trabajo. Se detallarán tanto el *hardware* como el *firmware* del que disponen los nodos y la arquitectura cognitiva sobre la que se ha desarrollado este trabajo. Por último, se presentarán las herramientas que se han utilizado durante el trabajo.

2.1. Hardware del cNGD

El hardware en el que se realiza este trabajo es el nodo cNGD desarrollado en el B105 lab y el cual viene detallado en [17]. Aquí expondremos brevemente algunas de sus características más importantes y que se han tenido en cuenta a la hora de realizar el trabajo ya que influyen a la hora de implementar código sobre el nodo.



(a) Vista superior



(b) Vista inferior

Figura 2.1 Vista detallada del cNGD, obtenida de [18]

El hardware del nodo cumple requisitos de bajo consumo, bajos recursos, bajo coste y disponibilidad de varias bandas de frecuencias para las comunicaciones. El requisito de bajos recursos tiene especial interés a la hora de desarrollar software por lo que pasamos a describir algunas de sus características principales:

- Microcontrolador. El MCU que incorpora el nodo es el PIC32MX675F256L [19], de 32 bits y fabricado por Microchip. Tiene 100 pines y sus características son:
 - o Memoria. 256 kB de memoria flash y 64 kB de memoria RAM.
 - o Reloj interno. Frecuencia máxima de funcionamiento de 80 MHz.
 - o Modos de funcionamiento. Varios modos para reducir el consumo.
 - o *Timers*. Cinco *timers* de 16 bits, pudiendo utilizar dos de ellos para hacer uno de 32 bits.
- Interfaces radio. El nodo dispone de tres interfaces radio, por lo que es capaz de transmitir y recibir a través de tres frecuencias diferentes. Éstas son 434 MHz, 868 MHz y 2,4 GHz. Con esto se cubren parte de las bandas ISM disponibles en Europa. Para poder reducir el consumo de los nodos sin tener que renunciar a tener las tres interfaces, éstas se pueden activar o desactivar cuando no se estén utilizando.
- Alimentación. El nodo tiene varias posibilidades de alimentación, siendo la principal las baterías. También se puede alimentar a través de USB, del conector RJ-11 o del conector de 3,6 V.
- Módulos de expansión. El nodo tiene la posibilidad de expandir su funcionalidad mediante módulos de expansión que se conectan a los *headers* disponibles. Uno de los módulos que se han utilizado en este trabajo es el que permite a los nodos comunicarse a través de línea serie RS232, dando la posibilidad de comprobar la funcionalidad del código mediante trazas.

Como vemos, las características del cNGD satisfacen las especificaciones necesarias de un nodo para CWSN, ya que es capaz de trabajar en diferentes bandas de frecuencia, en este caso tres bandas ISM de Europa. Tiene la posibilidad de trabajar en modos de bajo consumo tanto con los diferentes modos de funcionamiento del microcontrolador como apagando las interfaces radio que no utilice. Además, permite el desarrollo de nuevas funcionalidades mediante los módulos de expansión.

2.2. Firmware del cNGD

El firmware implementado en el nodo y que fue desarrollado en [20] tiene la función de optimizar, adaptar e integrar las pilas de protocolos de cada uno de los transceptores en una única pila, y de proporcionar una interfaz que simplifica el trabajo del programador mediante una serie de funciones que son las que acceden al *hardware* y que denominamos Hardware Abstraction Layer (HAL).

La pila de protocolos de los transceptores se resume en la Figura 2.2. Esta pila es la misma para los protocolos P2P [21] y MiWi [22] que son los que están implementados en el nodo.

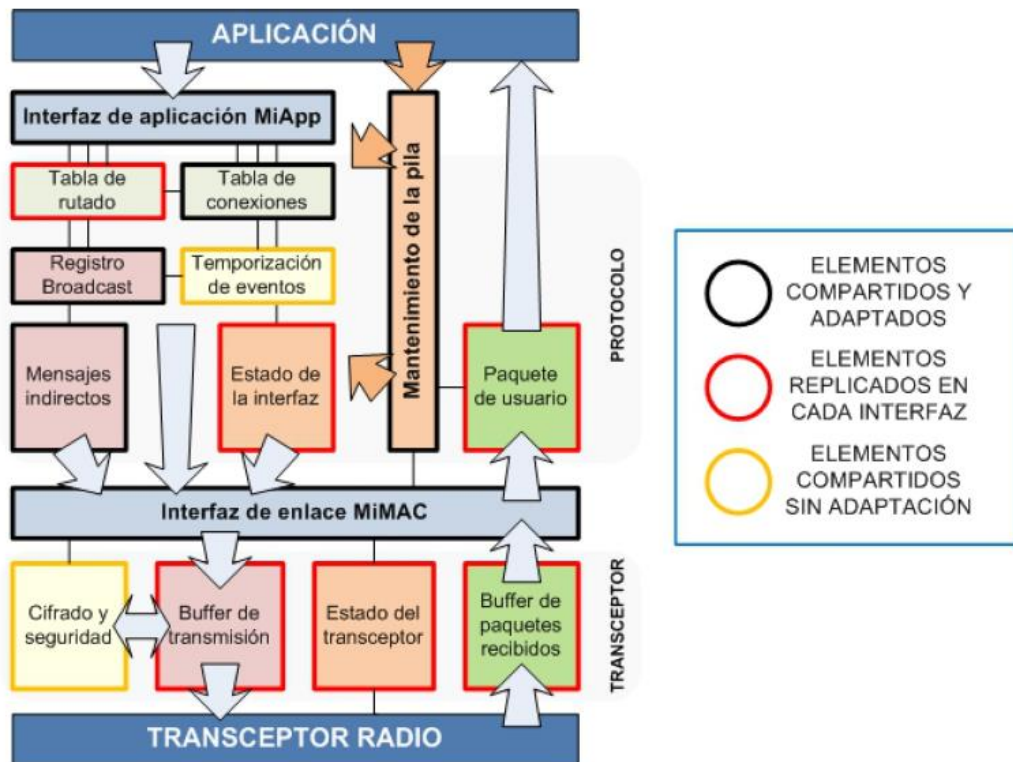


Figura 2.2 Diagrama de bloques de la adaptación software de la pila de protocolos MiWi, obtenida de [20]

Con el desarrollo de la HAL realizada en [20], se consigue simplificar la labor del programador a la hora de utilizar la pila de protocolos de las interfaces, teniendo que hacer llamadas a funciones para acceder a los recursos. Además, la HAL da flexibilidad para cambiar los transceptores, añadir nuevos o implementar nuevas funcionalidades. La arquitectura del firmware y la función de la HAL quedan resumidas en la Figura 2.3.

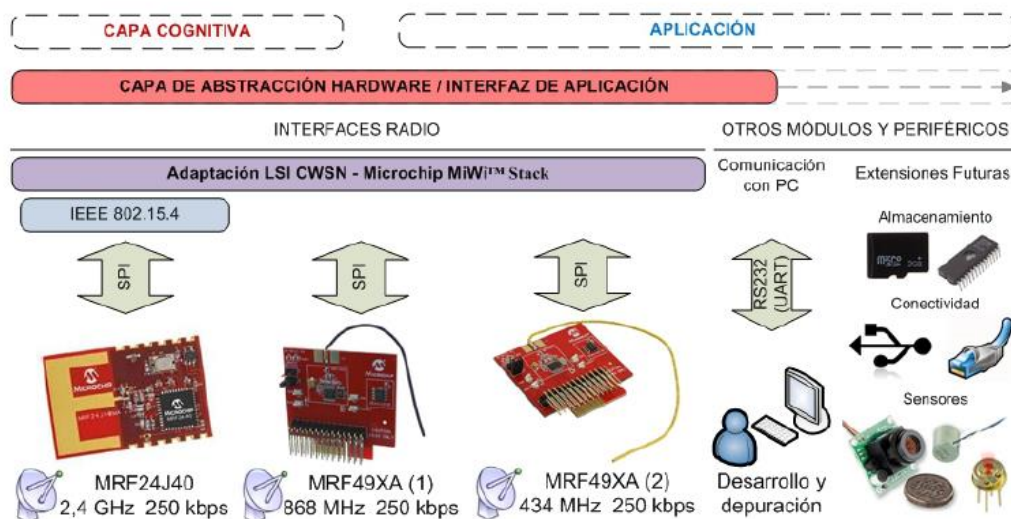


Figura 2.3 Arquitectura del firmware del cNGD, obtenida de [20]

Las funciones implementadas actualmente en la HAL van desde la inicialización del nodo hasta la gestión de las comunicaciones. Algunas de las funciones que interesan en este trabajo son conocer el canal activo en una interfaz, enviar y recibir paquetes o comprobar la tabla de conexiones.

2.3. Arquitectura cognitiva

La implementación de la arquitectura cognitiva realizada en [13] es la encargada de dar soporte a la implementación de estrategias cognitivas en el nodo. Esta arquitectura se basó en la arquitectura presentada en el *Connectivity Brokerage* [23] que se realizó con la colaboración de las universidades de UC Berkeley, TU Berlin, Universidad de Ozyegin, Universidad Politécnica de Madrid e IMEC. El modelo de esta arquitectura lo podemos ver en la Figura 2.4.

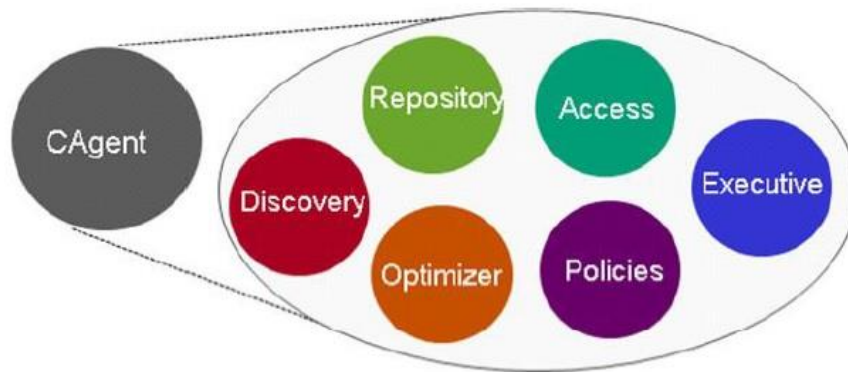


Figura 2.4 CAgents del Connectivity Brokerage, obtenido de [23]

El esquema general de la arquitectura implementada en [13] se puede ver en la Figura 2.5.

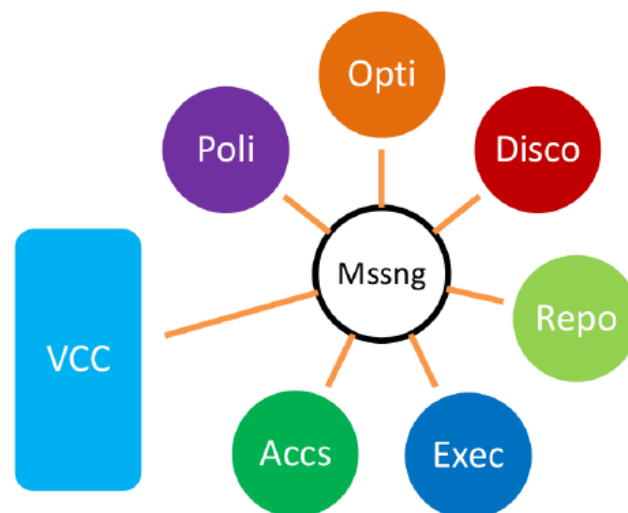


Figura 2.5 Arquitectura del CRModule, obtenida de [13]

La función de cada sub-módulo de la arquitectura es:

- Repository. Sub-módulo encargado de almacenar la información necesaria para la estrategia cognitiva. Aquí se recibirá la información y se almacenará para cuando otro sub-módulo de la arquitectura lo requiera.
- Discovery. Es el encargado de caracterizar diferentes parámetros del entorno. Este sub-módulo será el encargado de obtener el nivel de ruido en los canales.

- Optimization. En este sub-módulo se implementarán las rutinas de las estrategias cognitivas. Mientras este sub-módulo realiza el proceso cognitivo podrá realizar peticiones a otros sub-módulos del CRModule o incluso a otros sub-módulos en otros nodos.
- Execution. Los resultados del proceso de optimización tendrán que ser ejecutados. Este sub-módulo es el encargado de ejecutar las decisiones tomadas por el sub-módulo Optimization.
- Access Control. Debido a la naturaleza cooperativa de las estrategias cognitivas, son necesarios mecanismos de seguridad y control que sepan qué nodos tienen permisos para hacer acciones sobre el resto de nodos de la red. Éste sub-módulo se encarga de manejar la información de los permisos que tienen los nodos conocidos para realizar acciones en los sub-módulos del CRModule al que pertenece.
- Policy Support. Las estrategias cognitivas tienen unos valores que determinan las decisiones que se toman. Este sub-módulo tiene la información sobre esos valores y es consultado por el resto de sub-módulos para tomar las decisiones oportunas.
- Messenger. Se encarga de conectar el resto de sub-módulos entre ellos. Maneja los mensajes que se mandan el resto de sub-módulos y comprueba, si el mensaje proviene de otro nodo, si tiene permisos o no mediante petición al sub-módulo Access Control.
- VCC. El último de estos sub-módulos es el encargado de proporcionar una capa de abstracción para la comunicación de los mensajes de control entre nodos, de forma que la estrategia se pueda llevar a cabo de forma colaborativa.

La implementación realizada en [13] fue desarrollada para funcionar directamente sobre la pila de protocolos de Microchip y para funcionar sobre una plataforma hardware distinta. Por ello, posteriormente, en [24], se realizó una adaptación de esta arquitectura para funcionar con la pila de protocolos y el firmware mencionados en el apartado anterior.

2.4. Herramientas utilizadas

Una vez presentado todo el HW y SW del que se dispone, en este apartado se van a presentar las herramientas que se han utilizado para la realización del trabajo.

2.4.1. MPLAB X

Esta herramienta es la Integrated Development Environment (IDE) que proporciona Microchip para la programación de sus microcontroladores. Está basado en NetBeans IDE, open-source de Oracle. El compilador utilizado ha sido el XC32 que proporciona Microchip en la última versión disponible. Los compiladores son compatibles hacia atrás, lo que quiere decir que se puede seguir compilando el código con compiladores más recientes. Algunas de las opciones que ofrece y que se han utilizado para este trabajo son las siguientes:

- Herramientas de depuración.
- *Parsing* y control de sintaxis en tiempo real.
- Hipervínculos que permiten una navegación rápida para acceder a las declaraciones.

Para más información acerca de esta herramienta se puede consultar el manual disponible en [25].

2.4.2. Programador ICD 3

Este dispositivo es el que permite programar el microcontrolador. El dispositivo que vamos a usar es el In-Circuit Debugger (ICD) 3 [26], desarrollado por Microchip para programar sus productos. La característica principal de este dispositivo es que, en combinación con MPLAB X,

ofrece la posibilidad de depurar en tiempo real el *software* que se ejecute en el MCU. Permite un máximo de 6 puntos de parada.

2.4.3. RS232SHIELD

Es una placa de expansión para el cNGD que fue desarrollada en [17]. Esta placa permite la conexión del nodo con el ordenador mediante puerto serie haciendo posible la depuración y las pruebas de la ejecución del código mediante el envío de trazas por este puerto. La configuración necesaria del puerto serie para la comunicación con el nodo es la siguiente:

- *Bit rate*: 115200 bits/s
- Bits por trama: 8 bits
- Paridad: No
- Bits de parada: 1 bit

Esta placa ha sido utilizada para la depuración mediante trazas y la realización de pruebas de la funcionalidad del código implementado. Una imagen del nodo con la placa conectada se puede ver en la Figura 2.6.



Figura 2.6 Nodo con rs232SHIELD conectado, obtenida de [17]

Una vez vistas todas las herramientas que se han utilizado, se pasa a detallar la implementación de los dos algoritmos.

Capítulo 3. Implementación del algoritmo de seguridad

Este algoritmo, desarrollado en [27], está basado en el comportamiento invariante de los nodos durante su ejecución. Se centra en la necesidad de aportar seguridad a las CWSNs. A continuación se describe el objetivo principal de la estrategia y las diferentes etapas de las que consta.

La meta principal de esta estrategia de optimización es la de detectar ataques a la red de nodos que se quieran hacer pasar por usuarios primarios de la red. Un usuario primario de la red es aquel que tiene prioridad en el uso del espectro sobre los usuarios secundarios, por tanto ocupará más tiempo el canal. Los usuarios secundarios son aquellos que utilizan un canal cuando ven un hueco en el espectro donde no se está transmitiendo y sólo lo deben ocupar por un breve período de tiempo para no afectar a las comunicaciones de los usuarios primarios.

Para la detección de los nodos intrusos, o que tienen un funcionamiento anómalo, el algoritmo requiere de dos fases:

- Fase de aprendizaje. El aprendizaje se basa en que los nodos de la red tienen el mismo comportamiento durante toda su ejecución. En esta fase se van a situar los nodos que están en nuestra red midiendo la potencia recibida cuando nos comunicamos con ellos y el tiempo que transcurre entre los paquetes que envían. Esto se hace procesando los paquetes que se reciben, guardando el valor del *Received Signal Strength Indicator* (RSSI) y el tiempo que ha transcurrido entre dos paquetes.

Cuando transcurre el tiempo suficiente como para tener los datos necesarios para situar todos los nodos de la red, se procesan todos los paquetes que se han recibido para realizar un mapa. En este mapa se van a marcar las zonas donde se encuentran los paquetes de aplicación. Estas zonas denominadas clusters van a tener un centro (valor medio del RSSI y tiempo entre paquetes de todos los paquetes que contiene el cluster) y un radio, que determinará si un paquete pertenece a un cluster o no.

Para generar la lista de clusters primero se normalizan con respecto a los valores máximos de las coordenadas los valores almacenados y luego se va procesando cada par de coordenadas incluyéndolas en un cluster.

- Fase de detección. En esta fase el algoritmo se encargará de procesar el valor del RSSI y tiempo entre paquetes de los paquetes que va recibiendo y comprobando que están contenidos en un cluster de los anteriores. Si un paquete no está contenido en un cluster, se marca el nodo del que procedía como atacante ya que tiene un comportamiento anómalo y se informa al resto de nodos de la red. Si un número determinado de nodos detectan a un mismo nodo como atacante desconectan ese nodo de la red.

3.1. Funciones de la arquitectura cognitiva

Las etapas que sigue el algoritmo quedan resumidas en la Figura 3.1.

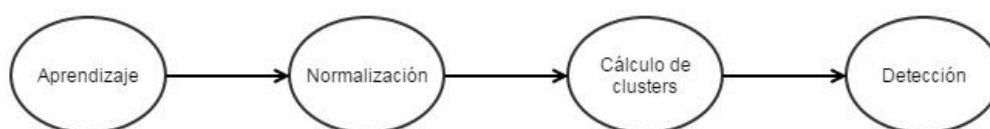


Figura 3.1 Etapas del algoritmo de seguridad

En cada paso de una etapa a otra se activa un *flag* que hace que no se vuelva a entrar a esa etapa ya que no es necesario volver nunca a una etapa anterior.

A continuación se van a describir las estructuras que se han definido para almacenar los datos necesarios para la implementación del algoritmo y las funciones que se usan durante la ejecución del mismo.

3.1.1. Estructura de datos

Se han definido tres nuevas estructuras de datos:

- **Coordenadas.** En esta estructura se almacena la potencia de cada paquete recibido y el tiempo transcurrido entre éste y el anterior. Ambos parámetros se almacenan en una variable tipo *double* ya que serán necesarios decimales cuando se normalice. Para definir este tipo de estructura utilizaremos “coord”.

```
typedef struct coordenadas {
    double RSSI;
    double tiempo;
} coord;
```

- **Clusters.** En esta estructura se guardan los clusters generados durante la etapa de cálculo de clusters. Cada cluster está definido por un centro que es una coordenada definida anteriormente, un radio y el número de paquetes con el que se ha formado el cluster que será necesario para hacer los cálculos del centro y el radio del cluster. Para definir este tipo de estructura utilizaremos “cl”.

```
typedef struct cluster {
    coord centro;
    double radio;
    int nMuestras;
} cl;
```

- **Atacantes.** En las variables de este tipo se guardará la información relativa a la detección de un nodo atacante. Los parámetros almacenados son la dirección del nodo que se ha detectado como atacante, la dirección del nodo que lo ha detectado y un byte que indica si ha sido detectado como atacante o no. Éste último byte será el que se compruebe para saber cuántas veces se ha detectado el mismo nodo como atacante. Para definir variables de este tipo se utiliza “at”.

```
typedef struct atacantes {
    BYTE direccionAtacante[MY_ADDRESS_LENGTH];
    BYTE direccionDetector[MY_ADDRESS_LENGTH];
    BYTE esAtacante;
} at;
```

Para almacenar todos los datos se han definido tres listas:

- `coord Lista_Paq_Rec_Aprendizaje[MAX_PAQ_APRENDIZAJE]`
En esta lista se almacenará la información de cada paquete que se reciba durante la etapa de aprendizaje. El tamaño es fijo y definido por `MAX_PAQ_APRENDIZAJE`.
- `cl Lista_Clusters[MAX_CLUSTERS]`
En esta lista se guardarán los clusters que se generen a partir de los paquetes de la lista anterior. En este caso también tendrá un tamaño fijo y definido por `MAX_CLUSTERS`.

- `at Tabla_Atacantes[(CONNECTION_SIZE+1)*(CONNECTION_SIZE+1)]`
Esta matriz guardará los datos de los atacantes detectados por el resto de nodos de la red y por el propio nodo que almacena la lista. En este caso el tamaño dependerá del número de nodos que tenga en la tabla de conexiones y variará cada vez que se conecte un nodo. Inicialmente la tabla contiene las direcciones de todos los nodos tanto en el campo de `direccionAtacante` como en el de `direccionDetector` y el campo `esAtacante` a cero. El valor `CONNECTION_SIZE` es el número de nodos con los que está conectado un nodo.

Las funciones que se han desarrollado para la implementación de este algoritmo y el sub-módulo del CRMódulo en el que están contenidas se detallan en la siguiente tabla:

| Sub-módulo | Funciones |
|------------|--|
| Optimizer | <code>BOOL CRM_Optm_Calculo_Coordenadas()</code> <code>void CRM_Optm_Normalizar_Coordenadas(coord *listaPaquetes)</code> <code>void CRM_Optm_Calculo_Clusters()</code> <code>double CRM_Optm_Calculo_Distancia(coord ptol, coord pto2)</code> <code>BOOL CRM_Optm_Detectar_Atacante()</code> |
| Repository | <code>void inicializarTablaAtacantes()</code> <code>BOOL CRM Repo_Proc_Mens_Att(REPO MSSG_RCVD *Petición)</code> |
| Execution | <code>BOOL CRM_Exec_DisconNode(EXEC_MSSG_RCVD *Petición)</code> |

Tabla 3.1 Funciones implementadas para el algoritmo de seguridad

A continuación se detallan todas las funciones de la tabla anterior:

3.1.2. Optimizer

Este sub-módulo es el encargado de las funciones de procesamiento de los paquetes que se van recibiendo. También se encarga de temporizar la ejecución de las etapas de aprendizaje, normalización y cálculo de clusters cambiando el *flag* que indica que se ha terminado cada etapa.

Para la ejecución de las etapas descritas en la Figura 3.1 se ha utilizado la rutina de atención a la interrupción del *timer* 4. Cada vez que se produce esta interrupción el algoritmo comprueba si hay datos en el *buffer* de recepción de la interfaz que se está usando en ese momento. Dependiendo de la etapa en la que se encuentre el algoritmo se va a ejecutar una de las funciones que se describen a continuación.

Durante la etapa de aprendizaje la función que se va a ejecutar es:

```
BOOL CRM_Optm_Calculo_Coordenadas()
```

Esta función se encarga de obtener la potencia y el tiempo transcurrido entre dos paquetes y mandárselos a Repository para almacenarlos en la lista de paquetes recibidos. Con cada potencia y tiempo obtenidos, se comprueba si es mayor que el máximo de entre los que se han procesado y si lo es lo guarda para después normalizar.

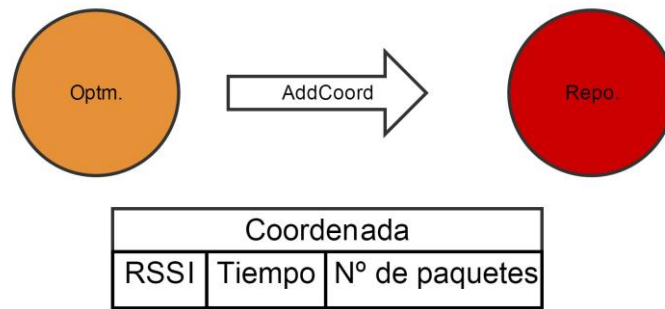


Figura 3.2 Paso de coordenadas de Optimizer a Repository

Una vez transcurrido el tiempo de aprendizaje, se activa el *flag* que indica que se ha realizado esta etapa y se pasa a la siguiente. En la siguiente interrupción del *timer 4* se van a normalizar con respecto a las mayores coordenadas los paquetes que se han recibido en la etapa anterior. La normalización de las coordenadas de los paquetes se realiza mediante la ejecución de la siguiente función:

```
void CRM_Optm_Normalizar_Coordenadas(coord *listaPaquetes)
```

Esta función necesita el puntero a la lista de paquetes recibidos. Para normalizar las coordenadas obtenidas se recorre la lista dividiendo cada valor de potencia y tiempo entre el máximo obtenido durante la etapa de aprendizaje. Los valores resultantes se vuelven a almacenar en la lista de paquetes recibidos.

Una vez finalizada la ejecución de esta función se activa el *flag* que indica que se ha terminado de normalizar las coordenadas de todos los paquetes recibidos. Al volver a producirse una interrupción del *timer 4* se ejecutará la etapa de cálculo de clusters. Esta etapa consiste en incluir cada paquete recibido en una estructura que denominamos cluster y que consta de un centro, que va a ser una coordenada como la de los paquetes y un radio. A un mismo cluster pueden pertenecer varios paquetes. La función para calcular los clusters es la siguiente:

```
void CRM_Optm_Calculo_Clusters()
```

Esta función es la encargada de crear los clusters que se usarán posteriormente para detectar si un nodo es atacante o no. Para ello, se recorre la lista de paquetes recibidos, que se le pide a Repository, comprobando si pertenece a un cluster ya creado o no. Para saber si un paquete pertenece a un cluster se calcula la distancia entre el centro de cada uno de ellos y el paquete y si es menor que el radio se dice que pertenece al cluster. Si un paquete no pertenece a ninguno, se crea uno nuevo con centro las coordenadas de ese paquete y un radio inicial configurable. Cuando un paquete pertenece a alguno ya creado, se añade al cluster calculando el nuevo centro y radio usando las siguientes fórmulas:

$$radio = radio\ anterior + distancia$$

$$RSSI_{nuevo} = \frac{(RSSI_{cluster} * n) + RSSI_{paquete}}{n + 1}$$

$$tiempo_{nuevo} = \frac{(tiempo_{cluster} * n) + tiempo_{paquete}}{n + 1}$$

Donde “distancia” es el dato que se calcula en la función CRM_Optm_Calculo_Distancia descrita a continuación y “n” el número de paquetes contenidos en el cluster.

Ecuación 3.1 Cálculo de nuevo radio y centro cuando se recibe un paquete que pertenece a un cluster

Cuando termina de procesar todos los paquetes que se han recibido quedaría un mapa de clusters como el de la Figura 3.3.

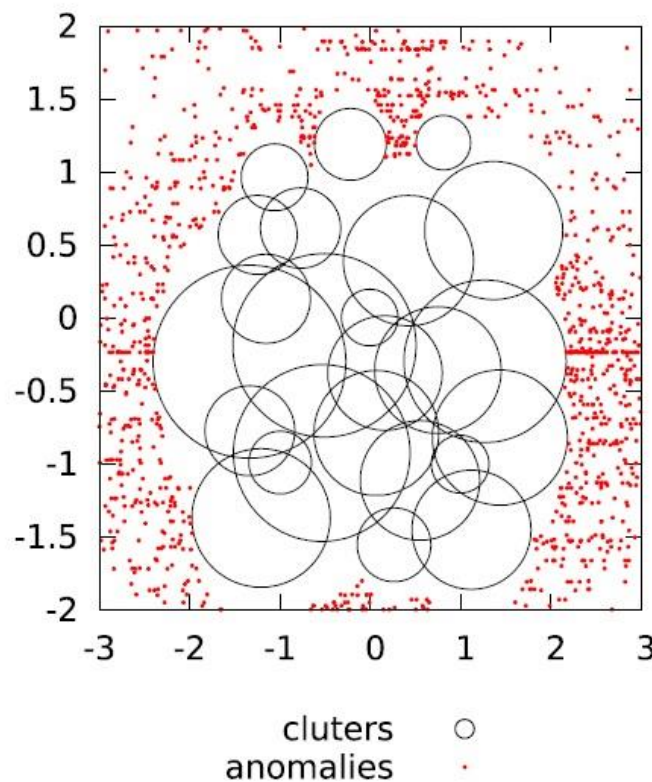


Figura 3.3 Ejemplo de mapa de clusters, obtenido de [27]

Para calcular la distancia entre dos coordenadas se ha definido la siguiente función:

```
double CRM_Optm_Calculo_Distancia(coord ptol, coord pto2)
```

Esta es una función auxiliar para calcular la distancia de dos puntos con dos dimensiones, en nuestro caso las coordenadas de los paquetes y de los centros de los clusters. Se llama a esta función cada vez que se tiene que comprobar que un paquete recibido pertenece a un cluster. Se utiliza tanto para la fase de cálculo de clusters como para la de detección. La ecuación para calcular la distancia es la siguiente:

$$distancia = \sqrt{(RSSI_{cluster} - RSSI_{paquete})^2 + (tiempo_{cluster} - tiempo_{paquete})^2}$$

Ecuación 3.2 Cálculo de la distancia entre dos puntos

Por último, en la etapa final de detección se ejecuta la siguiente función:

```
BOOL CRM_Optm_Detectar_Atacante()
```

Esta función es la encargada de procesar los mensajes de aplicación que llegan durante la fase de detección del algoritmo. Con cada paquete comprueba que pertenezca a un cluster y si no pertenece a ninguno añade al nodo que lo envió a la lista de atacantes. Tras añadirlo a la lista se manda un mensaje a través de VCC al resto de nodos notificando que se ha detectado un nodo como atacante. Por último, se comprueba si el nodo que se ha detectado ha sido detectado por más nodos de la red (configurable dependiendo del número de nodos que haya en la red) y si ha ocurrido manda un mensaje a Execution para que lo desconecte de la red.

Otra tarea que realiza este sub-módulo es la de reiniciar la tabla donde se guardan las detecciones de atacantes que se han detectado. Esto se realiza para que cada cierto tiempo se tenga que volver a detectar a un atacante, asegurándose así que las detecciones son correctas y evitando que un nodo que haya sido detectado incorrectamente como atacante no tenga la posibilidad de volver a conectarse a la red.

3.1.3. Repository

La estructura de los mensajes que tiene que recibir este sub-módulo es la siguiente:

```
typedef struct _REPO_MSSG_RCVD
{
    INPUT BYTE OrgModule;
    REPACTION Action;
    INPUT BYTE *EUI_Nodo;
    INPUT REPODATATYPE DataType;
    INPUT radioInterface Transceiver;
    IOPUT void *Param1;
    IOPUT void *Param2;
    IOPUT void *Param3;
    IOPUT void *Param4;
} REPO_MSSG_RCVD;
```

Los valores que puede tomar el parámetro Action de estos mensajes para este algoritmo se resumen en la siguiente tabla:

| REPACTION | Acción que se realiza |
|-----------|-----------------------|
| ActStr | Almacenar dato |
| ActSndDta | Enviar dato |

Tabla 3.2 Valores del parámetro Action de los mensajes dirigidos a Repository

Los valores que puede tomar el parámetro DataType son los siguientes:

| REPODATATYPE | Acción que se realiza u objeto que se devuelve |
|--------------|--|
| AddCoord | Almacenar coordenadas |
| DetAtt | Almacenar detección de otro nodo |
| InitTAtt | Inicializar tabla de atacantes |
| EnvListaPaq | Devuelve puntero a la lista de paquetes |
| EnvListaCl | Devuelve puntero a la lista de clusters |
| EnvTablaAt | Devuelve puntero a la tabla de atacantes |

Tabla 3.3 Valores del parámetro DataType de los mensajes dirigidos a Repository

Las funciones que se han implementado en este sub-módulo se detallan a continuación:

- `void inicializarTablaAtacantes()`

Es la función encargada de inicializar la tabla de atacantes al iniciar el nodo y cuando se pasa el tiempo preestablecido para reiniciarla.

Un ejemplo en el que un nodo sólo estuviese conectado a otro la tabla de conexiones quedaría de la siguiente forma:

| | | direccionDetector | |
|-------------------|-------------|-------------------|-------------|
| | | Dir. nodo 1 | Dir. nodo 2 |
| direccionAtacante | Dir. nodo 1 | 0 | 0 |
| | Dir. nodo 2 | 0 | 0 |

Tabla 3.4 Ejemplo de tabla de atacantes inicializada con dos nodos en la red

- `BOOL CRM_Repo_Proc_Mens_Att(REPO_MSSG_RCVD *Peticion)`

Función encargada de almacenar los mensajes con datos de atacantes procedentes de otros nodos. Cuando se ha incluido el nodo atacante en la tabla correspondiente se manda la información al sub-módulo Optimizer para que compruebe si hay que desconectarlo.

3.1.4. Execution

Este sub-módulo se va a encargar de desconectar de la red a los nodos que sean detectados como atacantes por varios nodos. La estructura de los mensajes con destino este sub-módulo es la siguiente:

```
typedef struct _EXEC_MSSG_RCVD
{
    INPUT BYTE OrgModule;
    INPUT EXEC ACTION Action;
    INPUT radioInterface Transceiver;
    INPUT BYTE Action2;
    INPUT BYTE Param1;
    INPUT void *Param2;
    INPUT void *Param3;
    OUTPUT void *Param4;
} EXEC_MSSG_RCVD;
```

El parámetro Action que se va a utilizar para desconectar los atacantes va a ser ActDisconn. Al recibir un mensaje con este parámetro se ejecutará la función:

```
BOOL CRM_Exec_DisconNode(EXEC_MSSG_RCVD *Peticion)
```

A la que hay que pasarle el mensaje que se ha recibido en el sub-módulo con la posición de la dirección del nodo atacante en la tabla de conexiones en el parámetro Param1.

3.1.5. VCC

Este sub-módulo no ha sido necesario modificarlo para la implementación de esta estrategia. Es el encargado de enviar los mensajes de control entre los diferentes nodos. Se ha reservado la interfaz de 434 MHz para la transmisión de estos paquetes.

El parámetro Action de los mensajes dirigidos a este sub-módulo para mandar mensajes de control a otros nodos debe ser ActSend.

El paso de mensajes entre sub-módulos que se realiza en esta función si se detecta un atacante se detalla en la Figura 3.4.

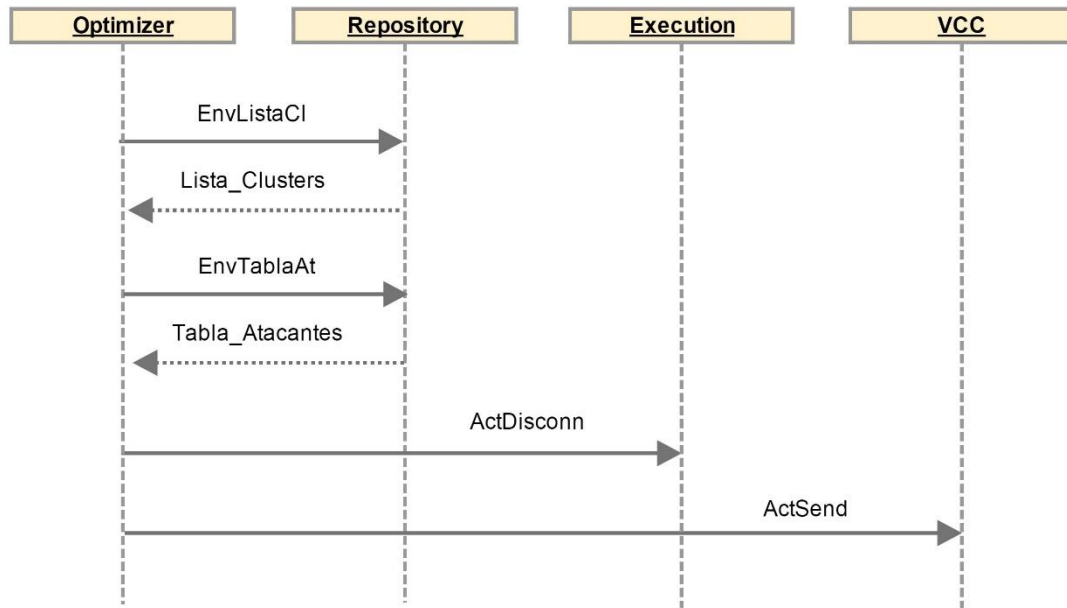


Figura 3.4 Paso de mensajes cuando se detecta un nodo atacante

Capítulo 4. Implementación del algoritmo de reducción de consumo

Este algoritmo trata de reducir el consumo de los nodos de la red mediante teoría de juegos.

Debido al constante cambio en la calidad del enlace de los canales de transmisión, en los momentos en que la calidad es mala los nodos para CWSN tienen un consumo más alto debido a las retransmisiones, la disminución de la QoS, pérdida de paquetes, etc. Este algoritmo hace frente a esos problemas mediante la toma de decisiones y la colaboración entre nodos de la red.

La ejecución de la estrategia sigue el siguiente esquema:

- Cada vez que se recibe un paquete de aplicación y se supera un umbral de retransmisiones se calculan los costes de transmitir en el canal actual y los de usar la estrategia para cambiar de canal. Estos costes se calculan de la siguiente forma:

$$C_{TXcanalocupado} = C_{tx} * n_{rtx}$$
$$C_{cambiocanal} = C_{sensado} + (C_{tx} + C_{rx}) * n_{msg}$$

Donde “ n_{rtx} ” es el número de retransmisiones en el canal actual y “ n_{msg} ” el número de mensajes necesarios para efectuar un cambio de canal.

- Si el coste de cambiar de canal es menor que el coste de transmitir en el canal actual se inicia el algoritmo. Primero el nodo que decide cambiar de canal envía la información del espectro al resto de nodos conectados a él.
- El resto de nodos evalúan si aceptan o no cambiar a ese canal y contestan si quieren cambia. Si no aceptan cambiar al canal propuesto inicialmente proponen otro.
- Cuando todos los nodos que quieren cambiar de canal se ponen de acuerdo a qué canal cambiar, se cambia de canal. Si no se ponen de acuerdo cada uno cambia al canal que propuso inicialmente.

A continuación se pasa a describir la implementación propuesta para esta estrategia:

4.1. Funciones de la arquitectura cognitiva

La implementación de este algoritmo se ha realizado siguiendo un esquema de máquinas de estados finitos tal y como se presenta en la Figura 4.1.

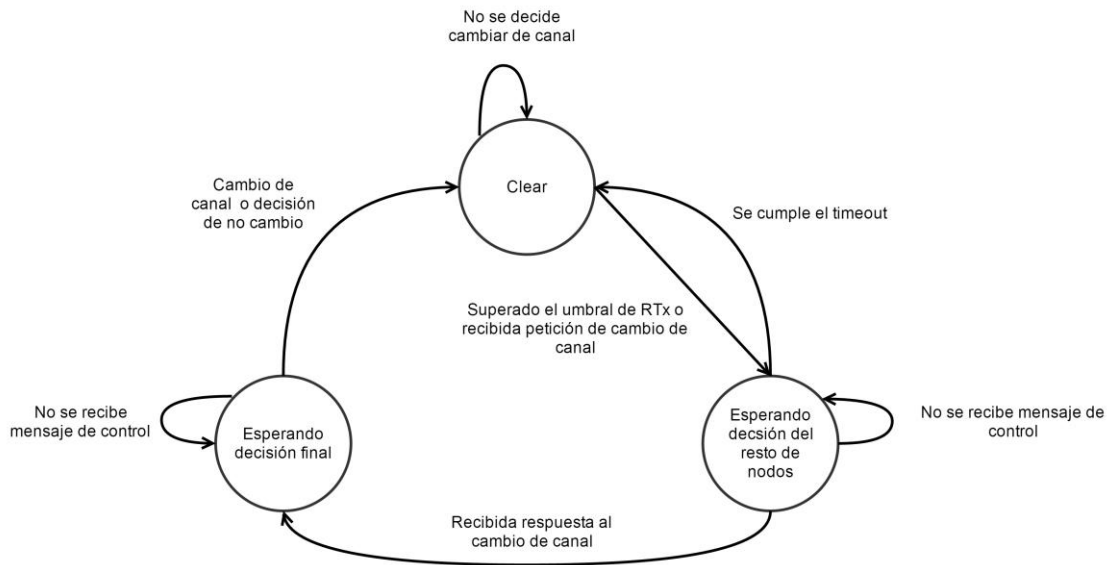


Figura 4.1 Diagrama de transiciones de la implementación propuesta

Las transiciones entre estados de la Figura 4.1 dependen de las respuestas de los otros nodos a las peticiones de cambio de canal que se les hace.

El paso de mensajes de petición de cambio de canal y las respuestas se producen a través de VCC, habiendo reservado la interfaz de 434 MHz disponible en los nodos para tal efecto.

4.1.1. Estructura de datos

Para la implementación de este algoritmo ha sido necesaria la definición de listas para almacenar los datos necesarios para la implementación del algoritmo. Se han definido las siguientes listas:

- `BYTE MIWIXXXX_rtx[MIWIXXXXNumChannels]`
Listas de las retransmisiones que se producen en los canales de la interfaz de XXXX MHz. El valor de XXXX puede ser 0868 o 2400. Cada vez que se envía un mensaje de aplicación se va guardando el número de retransmisiones que se han producido.
- `BYTE MIWIXXXX_RSSI_optimo_ext[CONNECTION_SIZE]`
Lista que almacena el RSSI óptimo que obtiene cada nodo de la red para la interfaz de XXXX MHz. Los valores que puede tomar XXXX son 0868 o 2400. Esta lista nos va a servir para elegir el canal al que se va a cambiar.
- `BYTE MIWIXXXX_canal_optimo_ext[CONNECTION_SIZE]`
Lista que almacena el canal óptimo de cada nodo de la red para la interfaz de XXXX MHz. Los valores que puede tomar XXXX son 0868 o 2400.
- `BYTE CanalOptimoExt[CONNECTION_SIZE]`
En esta lista se almacenarán los canales óptimos de los nodos de la red.

- `radioInterface riCanalOptimoExt[CONNECTION_SIZE]`
Se utiliza esta lista para almacenar la interfaz a la que pertenecen los canales que se han almacenado en la lista anterior.
- `UINT16 NumMssgIntercambiados[CONNECTION_SIZE]`
En esta lista se almacena el número de mensajes intercambiados con cada nodo de la red.

En la siguiente tabla se pueden ver las funciones que se han implementado para este algoritmo y en qué sub-módulo del CRModule se han incluido:

| Sub-módulo | Funciones nuevas o modificadas |
|------------|---|
| Optimizer | <code>BOOL CRM_Optm_Cons(OPTM_MSSG_RCVD *Peticion)</code> <code>BOOL CRM_Optm_Calcular_Costes(BYTE n_rtx)</code> <code>BOOL CRM_Optm_Processor(OPTM_MSSG_RCVD *Peticion)</code> |
| Repository | <code>BOOL CRM_Repo_SendDat(REPO_MSSG_RCVD *Peticion)</code> <code>void CRM_Repo_NodoPropio(REPO_MSSG_RCVD *Peticion)</code> <code>BOOL CRM_Repo_NodosRed(REPO_MSSG_RCVD *Peticion)</code> <code>void CRM_Repo_NRTx(BYTE n_rtx, BYTE canal, radioInterface ri)</code> <code>void CRM_Repo_Mensajes_Intercambiados(BYTE *Address)</code> <code>void CRM_Repo_Str_RSSI(radioInterface ri)</code> <code>BOOL CRM_Repo_Reiniciar_RTx(void)</code> |
| Execution | <code>BOOL CRM_Exec_ChonHoping(EXEC_MSSG_RCVD *Peticion)</code> |
| Discovery | <code>BYTE CRM_Disc_SignalDetection(DISC_MSSG_RCVD *Peticion)</code> |

Tabla 4.1 Funciones nuevas o modificadas para el algoritmo de reducción del consumo

A continuación se pasa a describir las funciones de cada uno de los sub-módulos del CRModule para procesar las peticiones de cambio y obtener el resultado final.

4.1.2. Optimizer

La estructura de los mensajes dirigidos a este sub-módulo y que se usará posteriormente para enviarle información desde otros sub-módulos es la siguiente:

```
typedef struct _OPTM_MSSG_RCVD
{
    INPUT BYTE OrgModule;
    INPUT OPTACTION Action;
    IOPUT void *Param1;
    IOPUT void *Param2;
    INPUT radioInterface Transceiver;
    INPUT BYTE *EUINodo;
} OPTM_MSSG_RCVD;
```

El sub-módulo Optimizer es el encargado de la ejecución del algoritmo. Decidirá el inicio del cambio de canal, procesará las respuestas de los otros nodos de la red y pedirá al resto de sub-módulos la ejecución de determinadas acciones o la información que necesite durante el proceso. El parámetro Action de dicha estructura puede tomar los valores que se indican en la siguiente tabla:

| OPTACTION | Acción que se realiza |
|------------------|---|
| ActCons | Iniciar algoritmo reducción de consumo |
| ActProcRq | Procesar mensaje recibido por VCC |
| SubActCambio | Sensar el espectro y decidir canal optimo |
| SubActRespCambio | Decidir canal con los datos de otro nodo |

Tabla 4.2 Valores del parámetro Action de los mensajes con destino Optimizer

En el caso de recibir un mensaje con parámetro Action igual a ActProcRq el parámetro Param1 tendrá que tener uno de los siguientes valores:

| OPTPROCACTION | Acción que se realiza |
|-----------------|---|
| ProcCambioCanal | Procesar el mensaje de cambio de canal recibido |
| ProcRespCambio | Procesar la respuesta a la petición de cambio |
| ProcDecFinal | Procesar decisión final de cambio de canal |

Tabla 4.3 Valores del parámetro Param1 cuando se recibe un mensaje con Action igual a ActProcRq

Las funciones que se han implementado o modificado en este sub-módulo se exponen a continuación:

- `BOOL CRM_Optm_Cons(OPTM_MSSG_RCVD *Peticion)`

Es la función encargada de pedir al sub-módulo Discovery la información del ruido en los canales de las interfaces de 868 MHz y 2,4 GHz, obteniéndose así un canal óptimo por cada interfaz y el valor del RSSI en ese canal. De esos dos canales se elegirá el óptimo y se guardará la interfaz a la que pertenece.

Una vez hecho esto se distingue entre si el algoritmo ha sido iniciado en el propio nodo u otro nodo dependiendo del parámetro Action:

- Si el parámetro Action del mensaje que se ha recibido era SubActCambio se envía un mensaje por VCC dirigido al sub-módulo Repository del resto de nodos con la información sensada.
- Por el contrario, si el parámetro Action del mensaje que recibe es SubActRespCambio, que indica que el algoritmo ha sido iniciado por otro nodo, se procesa la información que se ha recibido, detectando si el canal que ha recibido del otro nodo estaba entre los mejores y acepta el canal propuesto. Si no está entre sus mejores, manda su mejor canal al resto de nodos.

Todo esto queda resumido en las Figura 4.2 y 4.3.

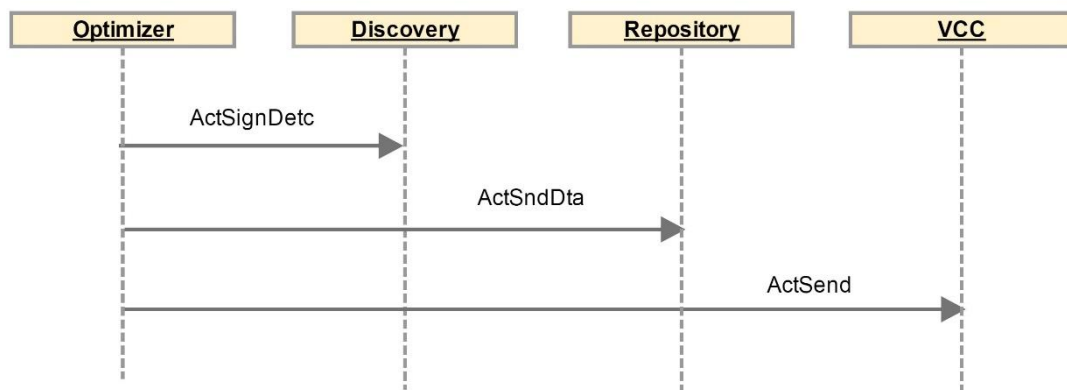


Figura 4.2 Diagrama con las acciones que realiza la función Cons y los parámetros Action que se pasan cuando inicia el algoritmo el propio nodo

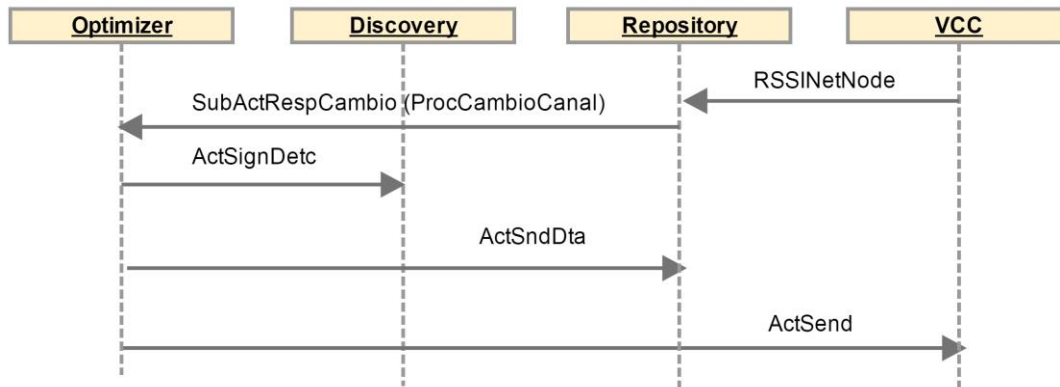


Figura 4.3 Diagrama con las acciones que realiza la función Cons y los parámetros Action que se pasan cuando inicia el algoritmo otro nodo de la red

Los parámetros que se pasan a los sub-módulos se encuentran en el apartado correspondiente a ese sub-módulo.

- `BOOL CRM_Optm_Calcular_Costes (BYTE n_rtx)`

Esta función se encarga de calcular los costes asociados a la transmisión en un canal ruidoso y los costes de cambiar de canal para decidir si iniciar o no el proceso para cambiar de canal. Se pide de argumento el número de retransmisiones del último mensaje de aplicación que se ha intentado enviar y el valor de salida es *TRUE* o *FALSE* dependiendo si se decide iniciar el algoritmo o no.

- `BOOL CRM_Optm_Processor (OPTM_MSSG_RCVD *Petición)`

Esta función es la que se encarga de procesar los mensajes provenientes de otros nodos. Existen tres tipos de mensajes de respuesta, una petición de cambio de canal, una respuesta a la petición de cambio o una decisión final.

Dependiendo del estado en el que se encuentre el nodo receptor y del mensaje que reciba se realizará una acción u otra. Las acciones realizadas se detallan a continuación:

- Estado "Clear". Cuando se esté en este estado sólo se procesarán los mensajes de petición de cambio de canal. El parámetro Param1 del mensaje dirigido al sub-módulo Optimizer tiene que ser ProcCambioCanal.

Cuando llegue un mensaje con ese parámetro se calculan los costes de cambio de canal y de transmisión en el canal actual y se manda respuesta al resto de nodos. Esto queda resumido en la Figura 4.4. Si la respuesta es positiva se llama a la función `CRM_Optm_Cons` para que haga una petición a Discovery para que realice el sensado del medio y finalmente se decide el canal óptimo.

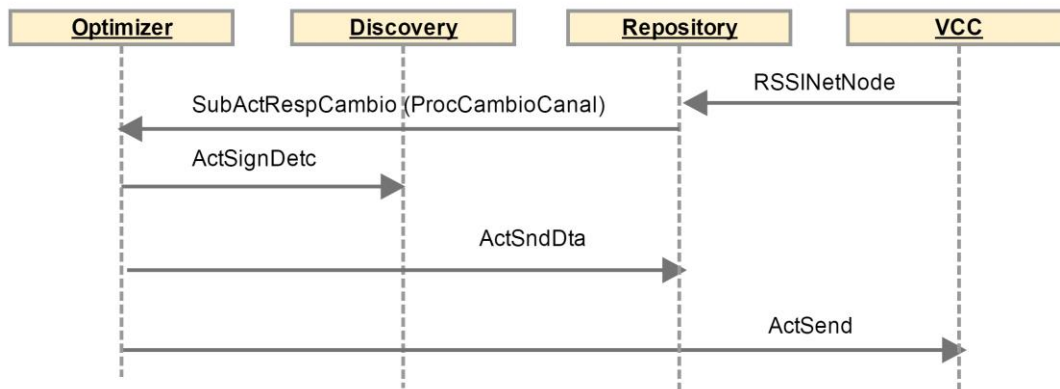


Figura 4.4 Diagrama de mensajes entre sub-módulos cuando se recibe una petición de cambio de canal

- Estado “EsperandoDecisionRestoNodos”. En este estado se esperan las respuestas al cambio de canal de todos los nodos que estén conectados. Si se recibe una respuesta negativa se evalúa el porcentaje de mensajes de aplicación que se han intercambiado y si hay uno con el que ha intercambiado un porcentaje alto de mensajes se rechaza el cambio de canal; si no ha intercambiado un porcentaje alto sigue con el proceso de cambio. El parámetro Param1 del mensaje dirigido al sub-módulo Optimizer tiene que ser ProcRespCambio.

Los pasos de mensajes entre los sub-módulos cuando se recibe una respuesta a un mensaje de cambio de canal queda resumido en la Figura 4.5.

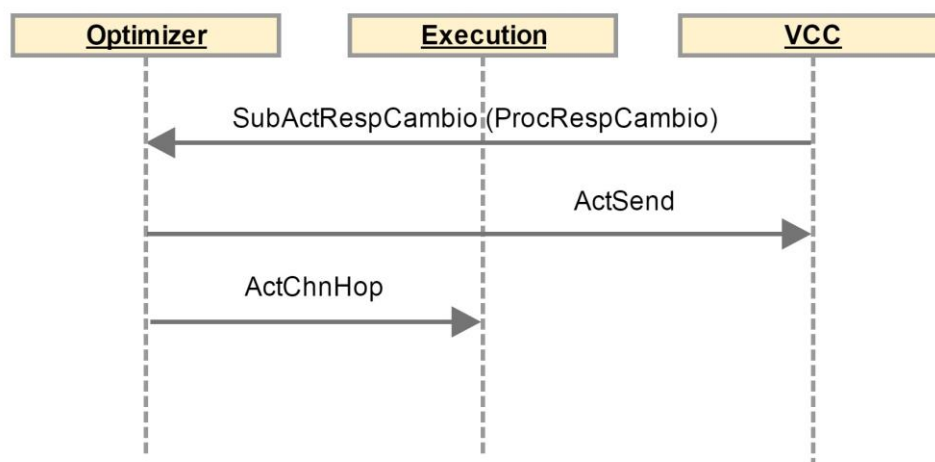


Figura 4.5 Diagrama de mensajes entre sub-módulos cuando se recibe una respuesta a una petición de cambio de canal

- Estado “EsperandoDecFinal”. En este estado todos los nodos en la red deciden a qué canal cambiarse. Todos los nodos de la red reciben la información del espectro del nodo que inició la estrategia y comprueban si el canal elegido por ese está entre sus mejores. Si lo está manda un mensaje al resto de nodos confirmando que se va a cambiar a ese canal y se cambia de canal. Si no está entre sus mejores rechaza el cambio a ese canal y se cambia al que decidió inicialmente. El resultado de este estado siempre va a ser un cambio de canal.

El parámetro Action del mensaje dirigido al sub-módulo Optimizer tiene que ser ActProcRq con Param1 ProcRespCambio si es un mensaje de respuesta afirmativa o negativa a una petición, o SubActRespCambio con Param1 ProcDecFinal si ya se ha recibido el canal óptimo de ese nodo y se ha decidido no cambiar a ese. Los pasos de mensajes entre sub-módulos cuando se recibe una respuesta con un canal distinto al que se propuso inicialmente viene resumido en la Figura 4.6.

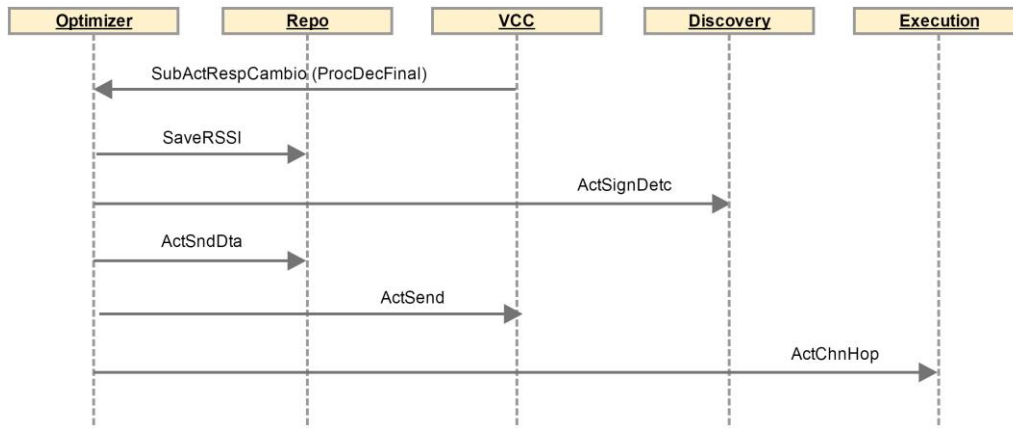


Figura 4.6 Diagrama de mensajes entre sub-módulos cuando se recibe una respuesta de cambio de canal con un canal diferente al propuesto inicialmente

4.1.3. Repository

Este sub-módulo será el encargado de almacenar la información que se reciba del resto de nodos de la red y enviarla al resto de sub-módulos si se la piden. La estructura de los mensajes con destino este sub-módulo es la siguiente:

```

typedef struct _REPO_MSSG_RCVD
{
    INPUT BYTE OrgModule;
    REPACTION Action;
    INPUT BYTE *EUI_Nodo;
    INPUT REPODATATYPE DataType;
    INPUT radioInterface Transceiver;
    IOPUT void *Param1;
    IOPUT void *Param2;
    IOPUT void *Param3;
    IOPUT void *Param4;
} REPO_MSSG_RCVD;
  
```

Los valores que puede tomar el parámetro Action de estos mensajes para este algoritmo se resumen en la siguiente tabla:

| REPACTION | Acción que se realiza |
|-----------|-----------------------|
| ActStr | Almacenar dato |
| ActSndDta | Enviar dato |

Tabla 4.4 Valores del parámetro Action de los mensajes dirigidos a Repository

Los valores que puede tomar el parámetro `DataType` son los siguientes:

| REPODATATYPE | Acción que se realiza u objeto que se devuelve |
|--------------|---|
| AddMsg | Actualizar número de mensajes enviados a un nodo |
| RstRTx | Reiniciar número de retransmisiones |
| EnvRTx | Devuelve el número de retransmisiones en un canal |
| EnvNMsg | Devuelve el número de mensajes intercambiados con un nodo |
| EnvRSSI | Devuelve canal y RSSI de una posición de la lista de canales ordenada por valor de RSSI |
| EnvRSSICh | Devuelve RSSI del canal que se le pide |

Tabla 4.5 Valores del parámetro `DataType` de los mensajes dirigidos a Repository

Cuando le lleguen mensajes de VCC los datos se almacenarán en `Param2`, como se explicará en la sección dedicada a ese sub-módulo, y cuando le lleguen mensajes de Optimizer devolverá la información en diferentes parámetros dependiendo del dato pedido. La información que se puede pedir y almacenar en Repository y las funciones implementadas se detallan a continuación:

- `BOOL CRM_Repo_SendDat(REPO_MSSG_RCVD *Petition)`
Esta función es la encargada de enviar la información que se le pide al sub-módulo Repository desde otros sub-módulos. Para la implementación de este algoritmo se ha añadido la posibilidad de devolver el número de retransmisiones en un canal, el canal y el RSSI en una posición de la lista de canales ordenada y el número de mensajes intercambiados con otro nodo.
- `void CRM_Repo_NodoPropio(REPO_MSSG_RCVD *Petition)`
En esta función se ha añadido la opción de almacenar el número de retransmisiones de un paquete de aplicación mediante el envío de un mensaje a Repository con el campo `Param1` del mensaje con el valor `EnvRTx`.
- `BOOL CRM_Repo_NodosRed(REPO_MSSG_RCVD *Petition)`
Esta función se encarga de recibir los mensajes de otros nodos con información de sensado. Almacena el valor del canal óptimo y de su potencia de ruido para las interfaces de 868 MHz y 2,4 GHz además del mejor canal de entre esos dos y la interfaz a la que pertenece. Dependiendo del estado en el que se encuentre el nodo en el momento en que recibe el mensaje, manda una petición al método `CRM_Optm_Processor` con un `Param1` que depende del estado en el que se encuentre el nodo para que el sub-módulo Optimizer procese la información del mensaje.
El paso de mensajes entre Repository y Optimizer cuando recibe información del espectro de otros nodos y el campo `Param1` para cada estado queda detallado en las Figuras 4.7, 4.8 y 4.9.

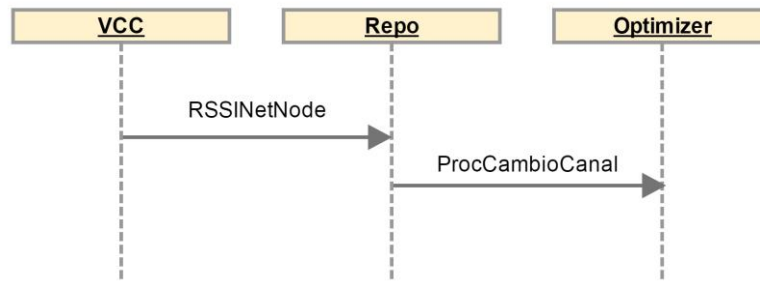


Figura 4.7 Diagrama de peticiones a Optimizer cuando se recibe información de sensado de otro nodo y éste se encuentra en estado Clear

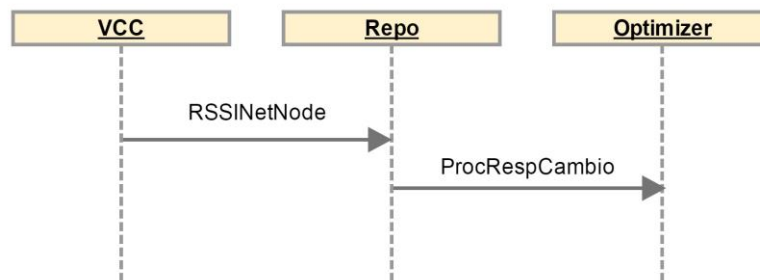


Figura 4.8 Diagrama de peticiones a Optimizer cuando se recibe información de sensado de otro nodo y éste se encuentra en estado esperando decisión resto de nodos.

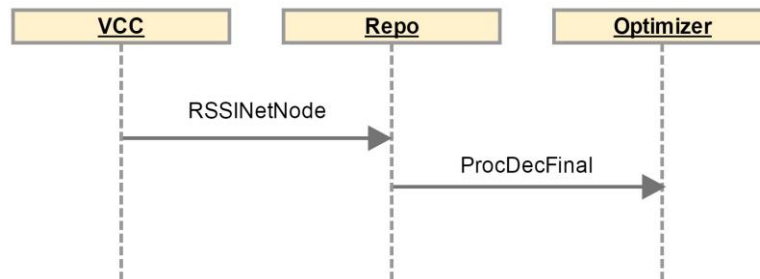


Figura 4.9 Diagrama de peticiones a Optimizer cuando se recibe información de sensado de otro nodo y éste se encuentra en estado esperando decisión final

- `void CRM_Repo_NRTx(BYTE n_rtx, BYTE canal, radioInterface ri)`
Es la función encargada de almacenar el número de retransmisiones que se producen al enviar un mensaje de aplicación. Los parámetros de entrada son la interfaz radio con la que se está transmitiendo, el canal en el que se han producido las retransmisiones y el número total de retransmisiones que se han producido.
- `void CRM_Repo_Mensajes_Intercambiados(BYTE *Address)`
Esta función es la encargada de actualizar el número de mensajes que se han recibido de un mismo nodo durante la ejecución de la aplicación. La llamada a esta función se realiza mediante un mensaje a Repository con el campo DataType con el valor AddMsg.

- `void CRM_Repo_Str_RSSI(radioInterface ri)`

Esta función se encarga de guardar el valor de RSSI obtenido del sensado del espectro realizado anteriormente por el sub-módulo Discovery.

- `BOOL CRM_Repo_Reiniciar_RTx(void)`

Es una función auxiliar. Se llama a esta función cada vez que se decide no cambiar de canal porque un nodo con el que se ha intercambiado un porcentaje elevado de mensajes ha decidido no cambiar de canal. Poniendo a cero el número de retransmisiones producidas se evita que se inicie el algoritmo dos veces consecutivas sin que se intente enviar un mensaje de aplicación nuevo.

4.1.4. VCC

En este sub-módulo se ha tenido que modificar el modo en el que se pasaban los mensajes recibidos con destino Repository. Al tener que enviar un número de datos superior al número de campos en el mensaje con posibilidad de incluir datos personalizados, y al disponer de punteros con la capacidad de incluir cualquier tipo de dato, se ha procesado la información útil de cada mensaje, incluyéndolo en un vector y pasando ese vector a través del puntero (que es el parámetro del mensaje Param2) al sub-módulo destino.

El parámetro Action para enviar mensajes de control a otros nodos tiene que ser ActSend.

Además, la funcionalidad de enviar mensajes *broadcast* mediante el sub-módulo VCC no estaba implementada en la HAL ni en el sub-módulo por lo que se ha modificado la estructura de los mensajes que se envían a VCC añadiendo el parámetro AddrMode. Además se ha modificado la función `Send_Buffer` en la HAL añadiendo un parámetro de entrada con el mismo nombre. Con esto, pasando el parámetro BROADCAST_ADDRMODE a los mensajes dirigidos a VCC se consigue la nueva funcionalidad.

4.1.5. Execution

La funcionalidad de este sub-módulo no ha sido modificada. Las peticiones que se le hacen desde Optimizer tienen la finalidad de cambiar el canal en el que se está transmitiendo. La función que se utiliza es:

```
BOOL CRM_Exec_ChannHopping(EXEC_MSSG_RCVD *Peticion)
```

El cometido de esta función es cambiar al canal que se le pasa en el parámetro Param1 del mensaje con destino Execution.

Para que se realice el cambio el parámetro Action del mensaje recibido deberá ser ActChnHop.

4.1.6. Discovery

La funcionalidad de este sub-módulo tampoco ha sido modificada.

Los mensajes que le envía Optimizer son para pedir la información del canal óptimo de cada interfaz y el valor del RSSI de ruido térmico que reciben. La función que se utiliza es:

```
BYTE CRM_Disc_SignalDetection(DISC_MSSG_RCVD *Peticion)
```

El cometido de esta función es la de escanear los canales de la interfaz radio que se le pase en el parámetro Transceiver de la petición y devolver el canal óptimo y el valor del RSSI de ruido térmico de ese canal. El canal lo devuelve como salida de la función y el valor del RSSI a través del parámetro Param4 de la petición. Por último, puntualizar que el valor necesario del parámetro Action para que se realice esta acción es ActSignDetect.

Capítulo 5. Aplicación de prueba de los algoritmos y resultados

En este capítulo se expone el proceso seguido para el diseño e implementación de la aplicación utilizada para comprobar el correcto funcionamiento de los dos algoritmos. El objetivo principal es la simulación del comportamiento de una aplicación de paso de mensajes entre diferentes nodos de la red.

5.1. Diseño

Como se ha mencionado anteriormente, la finalidad de la aplicación demostradora es la de simular el comportamiento de una aplicación que pueda tener una CWSN. El escenario que se va a utilizar para la aplicación va a tener un tiempo entre paquetes de 1 segundo y se transmitirá inicialmente a la máxima potencia de transmisión que permiten las interfaces que es 0 dBm.

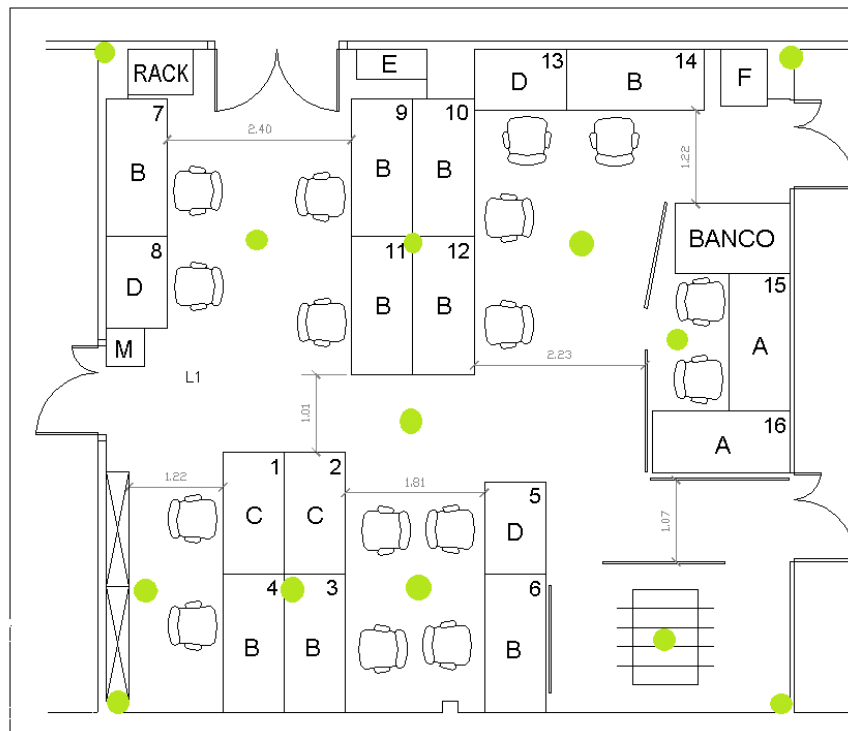


Figura 5.1 Propuesta de escenario de CWSN en el B-105 lab

Para nuestra aplicación se va a hacer variar aleatoriamente la transmisión de los paquetes de aplicación con un retardo de hasta cien milisegundos para comprobar la creación de los clusters del algoritmo de seguridad. Se usarán dos nodos ya que son suficientes para validar la correcta implementación del código. Para el algoritmo de seguridad se van a implementar los siguientes cambios en tiempo de ejecución:

- Un nodo cambia su potencia transmitida en cierto momento, comprobando que el otro lo detecta como atacante.
- En otro momento el segundo nodo cambia el tiempo entre paquetes detectándolo el nodo anterior como atacante.

Para la prueba del funcionamiento del algoritmo de reducción de consumo es necesario que haya retransmisiones en el canal por el que se transmiten los mensajes de aplicación. Para hacerlo se cambia la potencia de transmisión de la interfaz por la que se transmite y se quita la

antena de uno de los nodos provocando que el nivel de señal recibido en las interfaces no sea suficiente para la comunicación.

Debido a que van a existir mensajes de aplicación y de control es necesario definir una prioridad para el procesamiento de estos. Se han tomado las siguientes decisiones en cuanto al envío de mensajes de aplicación durante la ejecución de las estrategias:

- Seguridad: Al ser necesario un tiempo de procesamiento muy corto para la toma de decisiones, los mensajes de aplicación se siguen enviando normalmente. Cuando un mensaje de control llega, se procesa antes que todos los de aplicación que estén en el *buffer*.
- Reducción de consumo: Debido a que la decisión de cambiar de canal afecta al envío de mensajes de aplicación y que la decisión se toma teniendo en cuenta la opinión del resto de nodos de la red, se decide parar el envío de mensajes de aplicación durante la ejecución de la estrategia. El procesamiento de mensajes de aplicación sí sigue vigente durante la ejecución. La decisión de parar la transmisión de mensajes no va a afectar demasiado a la aplicación ya que la ejecución de la estrategia es de unos pocos segundos debido al tiempo de ejecución del sub-módulo Discovery para sensar el espectro.

Tras describir el funcionamiento que va a tener la aplicación demostradora se pasa a describir su implementación.

5.2. Implementación

La implementación de la aplicación tiene tres funciones que realizan las tareas de envío y recepción de los mensajes recibidos tanto de aplicación como de control. A continuación se detallan las funciones de la aplicación demostradora:

- `void Enviar_Paquete_Datos_App(radioInterface ri, BYTE modo, BYTE *addr)`

Esta función envía los mensajes de la aplicación. Para parar la transmisión de mensajes cuando se esté ejecutando el algoritmo de reducción de consumo se comprueba un *flag* siempre que se vaya a enviar un mensaje. También se encarga de contar el número de retransmisiones, guardarlo y de descartar los mensajes si se ha alcanzado el máximo de retransmisiones posibles. En los parámetros de entrada se le puede indicar la interfaz por la que se quiere transmitir, el modo de transmisión (*broadcast* o a un solo nodo) y en caso de que sea a un solo nodo, la dirección a la que se quiere enviar el mensaje.

- `BOOL Proc_Buff(RECEIVED_MESSAGE *Buffer)`

Es la función que procesa los mensajes recibidos. Principalmente guarda la dirección del nodo que envió el paquete y los datos que contiene en la memoria del MCU. También se encarga de dar prioridad a los mensajes de control haciendo que se procesen antes que los de aplicación si hay de los dos pendientes de procesar. Esto es sencillo ya que se ha reservado la interfaz de 434 MHz para enviar los mensajes de control lo que permite comprobar simplemente si hay datos en el *buffer* de recepción de esta interfaz o en la que está transmitiendo la aplicación.

- `void Recibir_info(void)`

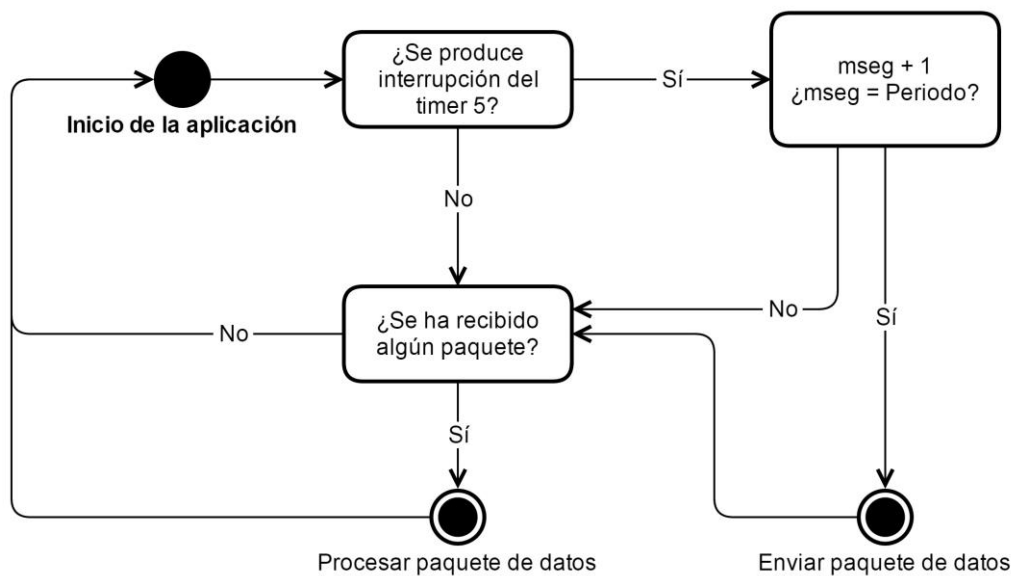
Esta función recibe todos los mensajes y comprueba si son de control o de aplicación, almacenándolos o realizando las tareas necesarias para la aplicación. Los datos se obtienen llamando a la función anterior que guarda los *buffer* de recepción en memoria. En este caso se ha decidido no mostrar los mensajes de aplicación ya que si se decidiese

enviar por la UART los mensajes de aplicación recibidos se perderían las trazas más importantes de la ejecución de las estrategias.

Las decisiones tomadas en cuanto a la ejecución de las funciones de la aplicación para que se cumpla la temporización, en el caso del envío de mensajes, y no perder ningún paquete, en el caso de la recepción, han sido la de utilizar el *timer 5* que no estaba siendo utilizado para ejecutar la función de envío de mensajes. El *timer* ha sido configurado con un período de un milisegundo y prioridad inferior al resto de *timers*.

Para la recepción de mensajes se comprueba que haya datos en alguno de los buffer de recepción. Es importante procesar los datos cuando llegan para que no se llegue a llenar el *buffer* y se pierdan paquetes. Por esto se comprueba con la interrupción del *timer 4* si se han recibido datos y si es así se procesan.

A continuación se presenta un diagrama con los pasos seguidos por la aplicación y las condiciones que se tienen que cumplir para que se ejecuten las funciones.



Donde *mseg* es la variable utilizada para contar los milisegundos transcurridos y *Período* el tiempo deseado entre paquetes.

Figura 5.2 Diagrama de ejecución de la aplicación

5.3. Resultados obtenidos

La finalidad de la aplicación demostradora era la de comprobar el correcto funcionamiento de la implementación propuesta. A continuación se detallan los resultados obtenidos de las pruebas realizadas para cada uno de los algoritmos:

- Algoritmo de seguridad. Como se ha explicado anteriormente durante la ejecución del algoritmo de seguridad se ha variado la potencia de transmisión y el tiempo entre paquetes de uno de los nodos. Se han realizado pruebas comprobando que el nodo que ha cambiado los parámetros de transmisión es detectado como atacante y expulsado de la red. Además, transcurrido un tiempo se comprueba que se reinicia la tabla de atacantes y se vuelve a conectar el nodo que ha sido detectado como atacante, volviendo a ser detectado y expulsado de la red.

- Algoritmo de reducción de consumo. Para las pruebas de este algoritmo era necesario conseguir retransmisiones en el canal por el que se estaban transmitiendo los paquetes de aplicación. Para esto se ha reducido la potencia de transmisión de los nodos al mínimo (-10 dBm) y se ha quitado la antena a uno de los nodos. Con esto se ha comprobado que se producían retransmisiones y que se iniciaba el algoritmo. A continuación se ha seguido el proceso que seguían los nodos para efectuar el cambio de canal mediante las trazas implementadas en el código, comprobando que se elegía finalmente un canal en el que transmitir.

Los resultados en cuanto al tiempo de ejecución de los algoritmos han sido:

- Algoritmo de seguridad. El tiempo de ejecución en caso peor es menor que 1 ms.
- Algoritmo de reducción de consumo. El tiempo de ejecución depende de las decisiones tomadas. Se han medido los siguientes tiempos de ejecución en caso peor:
 - Tiempo de ejecución cuando se decide no cambiar: 36 ms
 - Tiempo de ejecución cuando se decide cambiar al mismo canal: 3179 ms.
 - Tiempo de ejecución cuando no cambian al mismo canal: 3179 msEstos dos últimos tiempos son debidos principalmente al tiempo de sensado del espectro.
- Tiempo de ejecución cuando dos nodos deciden cambiar de canal al mismo tiempo y no se cambia: 164 ms

Capítulo 6. Conclusiones y líneas futuras

En este último capítulo se expondrán las conclusiones obtenidas tras el trabajo mediante la evaluación de los objetivos logrados y se detallarán los pasos a seguir en trabajos posteriores a este.

6.1. Conclusiones

El objetivo principal de este trabajo era la implementación de dos estrategias cognitivas en el *testbed* del laboratorio. La implementación tenía condiciones en cuanto al uso de recursos en los nodos ya que la memoria y la capacidad de procesamiento de éstos es limitada. Además las dos estrategias se tenían que diseñar de acuerdo con la arquitectura cognitiva que existe en el nodo.

Adicionalmente, se tenía que desarrollar una aplicación que consiguiese comprobar la funcionalidad de las estrategias y fuese útil para depurar el código.

Para lograr esto, en la sección 1.1 se propusieron objetivos. Las tareas que se han realizado para lograr esos objetivos se detallan a continuación:

- Familiarización con las herramientas. Las tareas que se han realizado para la consecución de este objetivo se detallan a continuación:
 - Instalación de MPLAB X, compilador, drivers de ICD 3, etc.
 - Ejecución de diferentes aplicaciones de prueba en los nodos, usando la herramienta de depuración de MPLAB, ejecutando paso a paso e incluyendo puntos de parada.
 - Lectura y comprensión de trabajos anteriores relacionados con el cNGD, su firmware y la arquitectura cognitiva.
 - Ejecución y comprensión de aplicaciones de prueba realizadas anteriormente en los nodos comprobando el envío de mensajes de aplicación y por VCC.
 - Configuración y prueba de programa para ver las trazas generadas en los nodos en el ordenador.
- Estudio y comprensión de los algoritmos. Este objetivo era uno de los pasos principales para lograr el objetivo final de la implementación de los algoritmos. Se estudiaron los dos artículos donde se presentaban los algoritmos que se iban a implementar en detalle consultando algunos detalles con los autores de dichos artículos.
- Implementación de las estrategias. Este era el objetivo principal del trabajo. Se ha desarrollado mediante la implementación de las funciones necesarias para cubrir la ejecución de las estrategias. Cada una de las funciones ha sido implementada en el submódulo correspondiente del CRModule. Para comprobar la ejecución del código se han incluido trazas que se envían por la UART. Cuando la depuración mediante trazas no era suficiente se ha tenido que ejecutar el programa en modo depuración añadiendo puntos de parada en las zonas del código que se querían probar y comprobando que el contenido de la memoria era el correcto.
Los resultados de este objetivo se desarrollan en detalle en los capítulos 3 y 4.
- Desarrollo de la aplicación. Este objetivo consistía en la elaboración de una aplicación demostradora que imitase el funcionamiento de una aplicación típica de CWSN. Para desarrollarlo se han utilizado las funciones de la HAL que mandan mensajes de datos. También se ha tenido que crear una función que procesa los mensajes que se reciben y los almacena en la memoria del MCU para liberar el buffer de recepción de la interfaz.

El resultado de este objetivo se ve en el capítulo 5.

- Pruebas. Para las pruebas se han utilizado dos nodos. Se han realizado diferentes pruebas para cada estrategia. Se detallan a continuación:
 - Algoritmo de seguridad. Se ha utilizado una función que se reduce la potencia de transmisión de uno de los nodos para que los paquetes que reciben los otros nodos se detecten como anómalos. Además, para probar todos los casos, se ha hecho que un nodo cambie la frecuencia de envío de mensajes de aplicación en un momento dado. Con esas dos pruebas, si el resto de nodos detectan al que cambia los parámetros como atacante se comprueba la funcionalidad del código.
 - Algoritmo de reducción del consumo. Para probar la funcionalidad en este caso ha sido necesario hacer que se produzcan retransmisiones en el canal por el que se están transmitiendo los mensajes de aplicación. Para ello se ha reducido la potencia de transmisión de los nodos al mínimo y ha sido necesario quitar la antena de uno de ellos.

6.2. Líneas futuras

Los aspectos en los que se puede seguir avanzando en cuanto al trabajo realizado en este documento y, en general, en la implementación de estrategias de optimización para CWSN van desde diferentes pruebas que se pueden realizar para probar la finalidad y la validez de los algoritmos que se han implementado hasta diferentes mejoras que se podrían realizar para agilizar el trabajo de programación y pruebas en los nodos. Se propone lo siguiente:

- En cuanto a pruebas que se pueden realizar sobre el trabajo realizado:
 - Prueba de funcionalidad en una red con más nodos. Las pruebas que se han realizado en este trabajo incluyen una red compuesta por dos nodos, lo que nos sirve para validar que la implementación ha sido correcta. Ejecutando los algoritmos en una red real con unos 50 nodos se podría comprobar mejor la validez de las simulaciones realizadas.
 - Prueba en diferentes topologías de red. En este trabajo se ha utilizado el protocolo P2P que es uno de los que están implementados en el nodo. Sería interesante probar cómo se comporta cada estrategia en diferentes topologías ya que, por ejemplo, el algoritmo de reducción de consumo podría tener diferente rendimiento para cada topología.
- En cuanto a posibles mejoras que se podrían llevar a cabo sobre el nodo, tanto HW como SW:
 - Mejora de algunos módulos de CRModule. En este trabajo se han realizado algunas modificaciones a funciones de paso de mensajes de la arquitectura cognitiva. Se puede mejorar este aspecto completando algunos de los sub-módulos, por ejemplo:
 - Haciendo que los mensajes de control que se envían los nodos sean procesados de una manera más eficiente.
 - Implementar las funciones que puede llevar a cabo el sub-módulo Discovery y Execution como la de conectar a uno o varios nodos de la red o realizar un escaneo de canales.
 - Mejorar la forma en que se comprueban los permisos de acceso añadiendo la comprobación de permisos cuando se reciben mensajes de otros sub-módulos del propio nodo.

- Actualizaciones OTAP (Over The Air Programming). Esta mejora consiste en dotar a los nodos de la posibilidad de ser programados de forma inalámbrica, permitiendo a los desarrolladores de código agilizar el proceso de carga y pruebas de los programas. Además, sería más sencillo reprogramar los nodos una vez que se haya desplegado el banco de pruebas.
- Nodo pasarela. Puede ser importante tener nodos que actúen como pasarelas para la red hacia el exterior. Para aplicar esta mejora se podría usar el propio cNGD con módulos de expansión que lo doten de conectividad con redes exteriores como puede ser Wi-Fi o Ethernet.
- Estado de la batería. Ya que los nodos están pensados para que se alimenten con baterías sería interesante poder contar con mecanismos de aviso de que un nodo se está quedando sin batería.
- Despliegue del banco de pruebas. Un aspecto muy importante en la prueba de estrategias de optimización para CWSN es contar con un escenario lo más ajustado a la realidad como sea posible.

Referencias

- [1] “¿Qué es una Red de Sensores Inalámbricos?,” 2009. [Online]. Available: <http://www.ni.com/white-paper/7142/es/>.
- [2] “IEEE 802.15 WPAN™ Task Group 4 (TG4).” [Online]. Available: <http://grouper.ieee.org/groups/802/15/pub/TG4.html>.
- [3] “Zigbee Alliance.” [Online]. Available: www.zigbee.org.
- [4] “HART Communication Foundation, WirelessHART Technology.” .
- [5] “ISA, Wireless Compliance Institute. ISA100 Wireless Systems for Automation.” [Online]. Available: <http://www.isa100wci.org>.
- [6] Microchip Technology Inc., “AN1066 Microchip MiWi™ Wireless Networking Protocol Stack,” 2007.
- [7] “IEEE 802.11™ Wireless Local Area Networks.” [Online]. Available: <http://www.ieee802.org/11/>.
- [8] “Wi-Fi Alliance.” [Online]. Available: <http://www.wi-fi.org/>.
- [9] “IEEE 802.15 Working Group for WPAN.” [Online]. Available: <http://www.ieee802.org/15/>.
- [10] “Bluetooth Alliance.” [Online]. Available: <https://www.bluetooth.org/en-us>.
- [11] “Informe ditrendia: Mobile en España y en el Mundo.” [Online]. Available: <http://www.ditrendia.es/wp-content/uploads/2014/07/Ditrendia-Informe-Mobile-en-España-y-en-el-Mundo.pdf>.
- [12] “United States Frequency Allocations.” [Online]. Available: http://www.ntia.doc.gov/files/ntia/publications/spectrum_wall_chart_aug2011.pdf.
- [13] G. Jara, “Diseño e implementación de una arquitectura para la gestión de comunicaciones de una red de sensores inalámbricas cognitiva,” PFC, ETSIT-UPM, 2013.
- [14] “Introduction to Cognitive Radio (CR).” [Online]. Available: <http://zoyok.com/blog/introduction-cognitive-radio-cr>.
- [15] C. Cordeiro, K. Challapali, D. Birru, and S. Shankar, “IEEE 802.22: The first worldwide wireless standard based on cognitive radios,” in *2005 1st IEEE International Symposium on New Frontiers in Dynamic Spectrum Access Networks, DySPAN 2005*, 2005, pp. 328–337.
- [16] “IEEE 802.15 WPAN™ Task Group 2 (TG2).” [Online]. Available: <http://www.ieee802.org/15/pub/TG2.html>.

- [17] A. Tena García, "Development of a multiple RF interfaced platform for Cognitive Wireless Sensor Networks," Tesis, ETSIT-UPM, 2013.
- [18] E. J. Cobo, "Diseño, implementación y prueba de una placa de expansión con sensores y actuadores para nodos inalámbricos con capacidades cognitivas," TFG, ETSIT-UPM, 2014.
- [19] Microchip Technology Inc., "PIC32MX5XX/6XX/7XX Family Data Sheet," 2009. [Online]. Available: ww1.microchip.com/downloads/en/DeviceDoc/61156H.pdf.
- [20] J. Domingo, "Diseño, optimización y prueba un nodo para una red de sensores inalámbrica con capacidades cognitivas," PFC, ETSIT-UPM, 2013.
- [21] "Peer-to-Peer Protocol (P2PP)." [Online]. Available: <http://www.p2psip.org/drafts/draft-baset-p2psip-p2pcommon-01.html>.
- [22] "Microchip MiWi™ Wireless Networking Protocol Stack." [Online]. Available: <http://ww1.microchip.com/downloads/en/AppNotes/AN1066-MiWiAppNote.pdf>.
- [23] J. Rabaey, A. Wolisz, A. O. Ercan, A. Araujo, F. Burghardt, S. Mustafa, A. Parsa, S. Pollin, I.-H. Wang, and P. Malagon, "Connectivity Brokerage — Enabling Seamless Cooperation in Wireless Networks," *White*, pp. 1–41, 2010.
- [24] J. M. Bermudo, "Adaptación y reestructuración de la implementación de una arquitectura cognitiva para redes de sensores inalámbricas," TFG, ETSIT-UPM, 2014.
- [25] Microchip Technology Inc., *MPLAB® X IDE User's Guide*. 2012.
- [26] Microchip Technology Inc., *In-Circuit Debugger User's Guide For MPLAB X IDE*. 2012.
- [27] J. Blesa, E. Romero, A. Rozas, and A. Araujo, "PUE attack detection in CWSNs using anomaly detection techniques," *EURASIP J. Wirel. Commun. Netw.*, vol. 2013, no. 1, p. 215, 2013.

Lista de acrónimos

| | |
|----------|---|
| WSN | Wireless Sensor Network |
| IEEE | Institute of Electrical and Electronics Engineers |
| WPAN | Wireless Personal Area Network |
| MAC | Medium Access Control |
| PHY | Physical Layer |
| OSI | Open Systems Interconnection |
| Wi-Fi | Wireless Fidelity |
| ISM | Industrial, Scientific and Medical |
| QoS | Quality of Service |
| CR | Cognitive Radio |
| WRAN | Wireless Regional Area Network |
| CWSN | Cognitive Wireless Sensor Network |
| B105 lab | B105 - Electronic Systems Lab |
| cNGD | Cognitive New Generation Device |
| HW | Hardware |
| SW | Software |
| μUSB | micro Universal Serie Bus |
| RI | Radio Interface |
| MCU | MicroController Unit |
| CRModule | Cognitive Radio Module |
| HAL | Hardware Abstraction Layer |
| P2P | Peer to Peer |
| IDE | Integrated Development Environment |
| ICD | In-Circuit Debugger |
| RSSI | Received Signal Strength Indicator |
| VCC | Virtual Channel Connection |
| OTAP | Over The Air Programming |