UNIVERSIDAD DE MÁLAGA

ESCUELA TÉCNICA SUPERIOR DE
INGENIERÍA DE TELECOMUNICACIÓN

TRABAJO DE FIN DE GRADO

Plataforma inalámbrica para la web física

GRADO EN INGENIERÍA DE
TECNOLOGÍAS DE TELECOMUNICACIÓN

JOSÉ ANTONIO YÉBENES GÁLVEZ
MÁLAGA, 2018

Escuela Técnica Superior de Ingeniería de Telecomunicación Universidad de Málaga

Titulación: Grado en Ingeniería de Tecnologías de Telecomunicación

Reunido el tribunal examinador en el día de la fecha, constituido por:

D./D$^{\underline{a}}$. _____

D./D$^{\underline{a}}$. _____

D./D$^{\underline{a}}$. _____

para juzgar el Trabajo de Fin de Grado titulado:

Plataforma inalámbrica para la web física

del alumno *D./D$^{\underline{a}}$. José Antonio Yébenes Gálvez*
dirigido por *D./D$^{\underline{a}}$. Ignacio Herrero y José Manuel Cano*

ACORDÓ POR _____OTORGAR LA

CALIFICACIÓN DE_____

y, para que conste, se extiende firmada por los componentes del tribunal, la presente diligencia.

Málaga, a _____ de _____ de ____

El Presidente: El Vocal: El Secretario:

Fdo.:_____ Fdo.:_____ Fdo.:_____

Universidad de Málaga
Escuela Técnica Superior de Ingeniería de Telecomunicación

Plataforma inalámbrica para la web física

REALIZADO POR
José Antonio Yébenes Gálvez

DIRIGIDO POR
Ignacio Herrero y José Manuel Cano

Dpto. de: Tecnología Electrónica (DET)

Palabras clave: IOT, web, red, sensores, inalámbrica, física, objetos, internet

Titulación: Grado en Ingeniería de Tecnologías de Telecomunicación

Resumen:
Cuando la única forma de interactuar con algún dispositivo es a través de una aplicación especialidad el manejo de estos dispositivos se complica. De este problema nace la web física que se enfoca en eliminar las aplicaciones y volver a lo fundamental de la web: la URL.

Este trabajo de fin de grado ofrece una plataforma donde sea posible la comunicación desde un teléfono móvil a un dispositivo que únicamente te ha enviado una URL. Para ello se estable una red inalámbrica a 868 MHz usando el microcontrolador CC1350 donde hay conexión desde un servidor a los nodos.

Málaga, 10 de febrero de 2018

Universidad de Málaga
Escuela Técnica Superior de Ingeniería de Telecomunicación

Wireless framework for physical web

Author
José Antonio Yébenes Gálvez

Supervisors
Ignacio Herrero y José Manuel Cano

Department: Tecnología Electrónica (DET)

Degree: Grado en Ingeniería de Tecnologías de Telecomunicación

Keywords: IOT, web, network, sensors, wireless, physical web, internet

Abstract: When the only way to interact with a device is through a specialized application, the handling of these devices is complicated. The physical web focuses on this problem, removing applications and return to the fundamental of the web: the URL.

This end-of-degree project offers a platform where communication from a mobile phone to a device that only sent you a URL is possible. For this, an 868 MHz wireless network is established using the CC1350 microcontroller where there is connection from a server to the nodes.

Málaga, 10 de febrero de 2018

A mi familia,
por facilitarme cumplir mis objetivos.
A Lucía,
por corregir mi rumbo cuando lo pierdo.

Acrónimos

API	*Application Programming Interface*
APL	*Application*
BLE	*Bluetooth Low Energy*
CCS	*Code Composer Studio*
CM0	*Cortex M0*
CM3	*Cortex M3*
CSMA/CA	*Carrier Sense Multiple Access with Collision Avoidance*
DFE	*Directed Frame Exchange*
ETSIT	Escuela Técnica Superior de Ingeniería de Telecomunicación
H2M	*Human-to-Machine*
HP	*Hewlett-Packard*
ICall	*Indirect Call Framework*
IEEE	*Institute of Electrical and Electronics Engineers*
IOT	Internet de las cosas o *Internet Of Things*
ISM	*Industrial, Scientific and Medical*
LBT	*Listen Before Talk*
M2M	*Machine-to-Machine*
MAC	*Medium Access Control*
MPU	microprocesador

MCU	microcontrolador
MT	*Managment and Test*
MVC	Modelo Vista Controllador
NPI	interfaz de protocolo de red
NWK	*Network*
PAN	red de área personal o *personal area network*
PHY	*Physical*
QR	*Quick Response*
RF	Radiofrecuencia
RTOS	sistema operativo en tiempo real
TFG	Trabajo Fin de grado
UART	*Universal Asynchronous Receiver/Trasnsmitter*
uBle	*Micro BLE Stack*
UMA	Universidad de Málaga
URL	*Uniform Resource Locator*
Wavenis-OSA	Wavenis Open Standard Alliance
WSN	*Wireless Sensor Networks*

Índice

Acrónimos **XI**

I Introducción 1

1 Objetivos **3**

2 Estado del arte **5**
Web física 5
Redes inalámbricas e internet de las cosas 7

II Desarrollo del proyecto 11

3 Descripción general **13**
Esquema General 13
Nodo 14
Concentrador 14
Servidor 14

4 Red inalámbrica **15**
Elementos de la red 15
Nodo 15
TI-15.4 Stack 20
Concentrador 25

5 Servidor **31**
Introducción 31
Estructura del servidor 31
Back-end 32
Front-end 32

III Pruebas y funcionamiento **35**

6 Prueba en entorno real **37**
Localización 37

7 Prueba de consumo **39**
Introducción 39
Medida del consumo 39
Resultados 40

8 Funcionamiento **43**
Inicio de la red 43
Administración de los nodos 46
Mapa 48

IV Conclusiones y lineas futuras **51**

Conclusiones **53**

Líneas futuras **55**

V Apéndices **57**

A Estructuras de mensajes **59**
A.1 Mensajes OTA 59
A.2 Mensajes Protocol Buffers 72

Bibliografía **80**

Índice de figuras

1.1 Aparcamiento de bicicletas públicas de la ciudad de Málaga 4

2.1 Logo de la web física 5
2.2 Ejemplos de comunicación cercana 6

3.1 Esquema general del proyecto 13

4.1 Diagrama de bloques de Simplelink™CC13x0 16
4.2 Diagrama de bloques de la aplicación 17
4.3 Diagrama de estados de la función de inicio 18
4.4 Flujo de la aplicación 19
4.5 Configuración como dispostivo único y como coprocesador 21
4.6 Aplicación ICall - Abstracción del protocolo 23
4.7 Ejemplo de comunicación usando el módulo ICALL 24
4.8 Arquitectura de software a alto nivel de las aplicaciones TI 15.4-Stack 2.1.0 Linux® 26
4.9 Estructura del directorio TI 15.4-Stack 2.1.0 Linux® 27

5.1 Estructura del directorio del servidor 33

7.1 Programas de LabView 40
7.2 Consumo en mA del nodo a la máxima frecuencia de muestreo 40

8.1 Web de gestión de instancias en Amazon Web Services 44
8.3 Apariencia de la web cuando ya está conectado el concentrador 44
8.2 Apariencia de la web cuando no está conectado el concentrador . . . 45
8.4 Apariencia de la web cuando hay dos nodos conectados 46
8.5 Panel de administración de los nodos 46
8.6 Panel de administración de un nodo tipo genérico 47
8.7 Panel de administración de un tipo parking de bicicletas 48
8.8 Vista del mapa con las posiciones de los nodos 49

Índice de Tablas

4.1 Formato de la trama *Eddystone-URL* 20
4.2 Bandas permitidas en TI 15.4-Stack y las frecuencias de sus canales . 22

Parte I

Introducción

Capítulo 1

Objetivos

La web física es un término que describe la forma de comunicar cualquier objeto físico con la web. A partir este enfoque, es posible navegar y controlar objetos en el mundo a través de dispositivos móviles. Esto ofrece a los usuarios la forma de realizar sus tareas diarias utilizando los objetos de su entorno. Para utilizar este enfoque, lo primero es seleccionar la manera con la que el objeto se comunicará con el usuario, tal como códigos QR o etiquetas RFID. De los diferentes enfoques que tiene la web física, el que se va a tratar en este proyecto es el basado en proximidad inalámbrica.

El modelo que plantea la web física, los objetos pueden necesitan un canal por donde enviar y recibir datos y el problema se agrava cuando los objetos están distribuidos y no tienen cerca un punto de acceso a internet o tienen que funcionar con baterías. En estos casos, buscar una solución de comunicación inalámbrica de gran alcance y bajo consumo es prioritario.

Con estos conceptos en mente, el presente trabajo de fin de grado va a abordar el concepto de web física, implementando una red inalámbrica que permita comunicarse con los objetos con el fin de ser una red versátil.

Aplicación de ejemplo

Para comprobar el funcionamiento de la red, se realizará una aplicación de ejemplo que podría simular la aplicación a una ciudad inteligente donde la red tendría dos nodos diferentes: nodo genérico y nodo aparcamiento

Nodo genérico

El nodo genérico es el nodo más simple de la plataforma, este permite enviar mensajes de la web física que serán definidos desde la web, además este nodo enviará periódicamente mensajes de seguimiento al servidor con información del estado del dispositivo, como por ejemplo temperatura y batería.

Nodo aparcamiento

Este nodo hereda todas las funciones del nodo genérico pero además simulará un uso que podría tener la web física. En este caso simulará un aparcamiento de bicicletas públicas de los que se pueden encontrar por muchas ciudades (véase figura 1.1). El dispositivo simulará los estados de los candados que mantiene a cada bicicleta anclada con un array de unos y ceros, donde el uno significará abierto y el cero significará cerrado.

Figura 1.1: Aparcamiento de bicicletas públicas de la ciudad de Málaga

La idea de funcionamiento sería simple el usuario llegaría a las bicicletas, y le llegaría una notificación de la web física con un enlace donde poder elegir una bicicleta y al seleccionar una, el candado de esta se abriría. De este funcionamiento en este proyecto solamente se ha abordado la parte de poder enviar la información desde la web al nodo, dejando lo demás como línea futura de este Trabajo Fin de grado (TFG).

Capítulo 2

Estado del arte

Contenido

Web física **5**
Redes inalámbricas e internet de las cosas **7**

Web física

El primer concepto que hay que conocer para afrontar el proyecto es el de Web Física. La Web Física es un término que describe el proceso de presentar objetos cotidianos en internet[1]. Este enfoque ofrece a los usuarios móviles la posibilidad de gestionar sus tareas diarias en el uso de objetos cotidianos. Los objetos comienzan a ser inteligentes y remotamente controlables. Este modelo permite a los usuarios móviles navegar y controlar los objetos físicos que rodean al dispositivo móvil. Además, esto ayuda al desempeño de tareas diarias dependiendo de los objetos cercanos [2].

Figura 2.1: Logo de la web física

Podemos mencionar en este contexto a los conocidos códigos *Quick Response* (QR), los códigos QR son un código de barras en dos dimensiones, a menudo utilizados para mapear *Uniform Resource Locator* (URL) con objetos físicos [3].

Las etiquetas inalámbricas son uno de los enfoques más utilizados para el marcado de objetos físicos. Las etiquetas inalámbricas pueden soportar protocolos como

Bluetooth Low Energy (BLE) y *WiFi*. Los protocolos mencionados son soportados por la mayoría de los teléfonos móviles modernos.

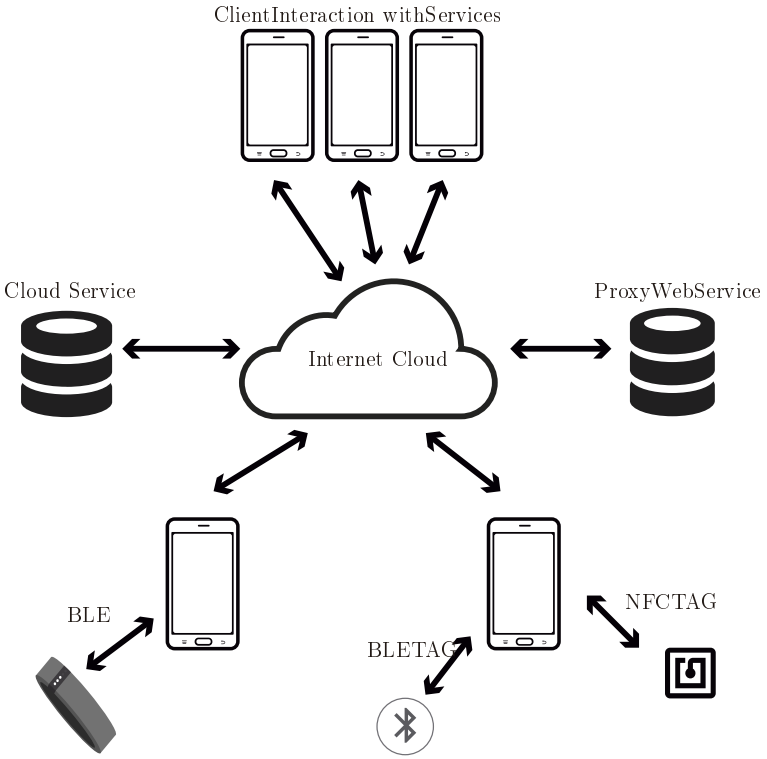


Figura 2.2: Ejemplos de comunicación cercana

En la Web Física, personas, lugares y objetos tienen páginas web que proveen información y mecanismos de interacción. Sin embargo, es la amplitud y la profundidad de la pila que rodea a la web, que hacen de esta una atractiva visión para la evolución del Internet de las cosas o *Internet Of Things* (IOT).[1]

Las páginas web son una fantástica tecnología para interacción *Human-to-Machine* (H2M), pero muchos de los casos de uso del IOT son interacciones *Machine-to-Machine* (M2M). Los formatos de datos usados por `Schema.org` y otros, permiten a los agentes de usuario y servicios en la nube analizar los datos para eventos, organizaciones, personas, lugares, productos y así sucesivamente, acutando sobre ellos de forma interactiva y proactiva. [1]

Uno de los primeros proyectos en fomentar esta idea fue *Hewlett-Packard* (HP) con *Cooltown*, que usaba balizas infrarrojas para transmitir URLs. Más recientemente, BLE proporciona una similar baliza de bajo consumo que puede emitir URL

en paquetes periódicos (`www.uribeacon.org`). [1]

Redes inalámbricas e internet de las cosas

Introducción

El futuro de internet tiene como meta integrar diferentes tecnologías de comunicación, cableadas e inalámbricas, con el objetivo de contribuir sustancialmente a mejorar el concepto de IOT [4]. Aunque hay muchas maneras de describir el IOT, podemos definirlo como una red con objetos interconectados con direcciones únicas, basadas en un protocolo estándar de comunicación.[5]

Los sensores de bajo costo han facilitado la proliferación de *Wireless Sensor Networks* (WSN) en muchos escenarios como monitorización medioambiental, agricultura, salud, y construcciones inteligentes. WSN están caracterizadas por una alta heterogeneidad porque están basadas en diferentes soluciones, propietarias y no propietarias. Este gran rango de soluciones está retrasando actualmente desarrollos a gran escala de estas tecnologías a fin de que se obtenga una gran red virtual de sensores que permita integrar todos las existentes redes de sensores.[6]

Las redes de sensores basadas en sistemas cerrados o propietarios son islas conectivamente hablando, con limitadas comunicaciones con el mundo exterior. Por lo general, es necesario usar *gateways* con conocimiento específico de la aplicación para exportar los datos de la WSN a otros dispositivos conectados a Internet. Además, no hay comuniación directa entre diferentes protocolos a menos que se implementen complejas conversiones específicas para la aplicación en los *gateways* o *proxies*.

Visión general de las soluciones existentes

En este apartado presentaremos una rápida visión general de las principales tecnologías usadas para WSN [7]. Analizaremos las soluciones que no están basadas en los protocolos de internet.

ZIGBEE

ZigBee es una tecnología de red inalámbrica desarrollada por la ZigBee Alliance para baja tasa de transmisión de datos y aplicaciones de corto alcance [8]. La pila de protocolos ZigBee está compuesta por 4 principales capas: la capa *Physical* (PHY), la capa *Medium Access Control* (MAC), la capa *Network* (NWK) y la capa

Application (APL). PHY y MAC de ZigBee están definidas por el estándar *Institute of Electrical and Electronics Engineers* (IEEE) 802.15.4, mientras el resto de la pila está definida por la especificación ZigBee.

Esta versión inicial del IEEE 802.15.4, en la que ZigBee está basado, funciona en la bandas de 868 MHz (Europa), 915 MHz (Norteamérica) y 2.4 GHz (global).

Una nueva especificación de ZigBee es RF4CE [9], que tiene una simplificada pila de red para topologías en estrellas solamente, ofreciendo una solución simple para el control remoto de electrónica de consumo.

Z-WAVE

Z-Wave es un protocolo inalámbrico desarrollado por ZenSys y promovida por la Z-Wave Alliance para automatizaciones residenciales y pequeños entornos comerciales. El principal objetivo de permitir transmisiones seguras de cortos mensajes desde una unidad de control hasta uno más nodos en la red [10]. Z-Wave esta organizado de acuerdo a un arquitectura cumpuesta por 5 capas principales: PHY, MAC, transferencia, enrutado y capas de aplicación.

Z-Wave opera principalmente en la banda de 900 MHz (868 MHz en Europa y 908 MHz en Estados Unidos) y 2.4 GHz. Z-Wave permite tasas de transmisión a 9.6 kb/s, 40 kb/s y 200 kb/s.

INSTEON

INSTEON [**?**] es una solución desarrollada por SmartLabs y promovida por la INSTEON Alliance. Una de las distintivas características de INSTEON es el hecho de como define una topología de red compuesta de Radiofrecuencia (RF) y *power line links*. Los dispositivos pueden ser RF, *power-line*, o pueden soportar ambos tipos de comunicación.

INSTEON opera a 904 MHz como frecuencia central, con una tasa de datos bruta de 38.4 kb/s.

Los dispositivos INSTEON son duales, lo que significa que cualquiera de ellos pueden tener el rol de emisor, receptor o repetidor. La comunicación entre ambos dispositivos que no estén en el mismo rango se logra mediante un enfoque «multisalto» que usa los repetidores en un esquema de sincronización temporal.

WAVENIS

Wavenis es un protocolo inalámbrico desarrollado por Coronis System para el control y monitorización de aplicaciones en entornos exigentes, incluida la domótica y la automatización de edificios. Wavenis actualmente está promovida y gestionada por la Wavenis Open Standard Alliance (Wavenis-OSA). Está definido por la funcionalidad de las capas física, de enlace y de red [11]. Los servicios de Wavenis pueden ser accedidos desde capas superiores mediante una *Application Programming Interface* (API).

Wavenis opera principalmente en las bandas de 433 MHz, 868 MHz y 915 MHz, que son bandas reservadas para *Industrial, Scientific and Medical* (ISM) en Asia, Europa y Estados Unidos. Algunos productos también operan en la banda de 2.4 GHz. Las tasas de transmisión mínimas y máximas dadas por Wavenis son 4.8 kb/s y 100 kb/s, respectivamente, con 19.2 kb/s como valor típico.

IEEE 802.15.4

Parte II

Desarrollo del proyecto

Capítulo 3

Descripción general

Contenido

Esquema General 13
Nodo 14
Concentrador 14
Servidor 14

Esquema General

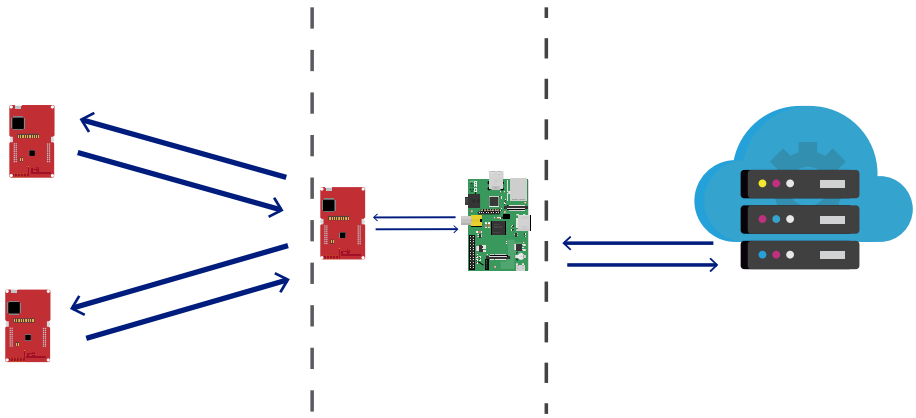


Figura 3.1: Esquema general del proyecto

El presente trabajo de fin de grado se puede representar como se observa en la figura 3.1, donde están representados los diferentes elementos que componen la

arquitectura.

Del diagrama se extrae que hay cuatro dispositivos distintos: nodo, concentrador, Raspberry Pi y servidor. Cada uno de estos elementos tiene su propia linea de ejecución diferente de los demás. También se observa como están interconectados los diferentes dispositivos usando protocolos diferentes según sea la comunicación. En las siguientes secciones se repasa la función de cada uno de los dispositivos y como se comunican con los demás.

Nodo

El nodo es el extremos de la red, y es el encargado de comunicarle al usuario la URL tal y como define la web física. Este está basado en el microcontrolador CC1350 SimpleLink™de Texas Instruments que nos permite comunicación en dos bandas de frecuencia diferentes, en nuestro caso 868MHz y 2.4Ghz.

Con el uso de estas dos bandas de frecuencia, el nodo se comunicará con el usuario usando la banda de 2.4GHz y el protocolo bluetooth. Y con el resto de la red usando la banda de 868MHz y el protocolo TI 15.4 Stack.

Concentrador

El concentrador es el nodo central de la red TI 15.4 Stack, este se encarga de comunicarse con los nodos a 868MHz. Este dispositivo ejecuta un código precompilado, que implementa una capa 802.15.4e/g MAC/PHY y proporciona una interfaz basada en el protocolo *Managment and Test* (MT) que conecta el dispositivo con el host linux, en nuestro caso una Raspberry Pi.

El concentrador está compuesto por dos dispositivos diferentes, un dispositivo CC1350 que actúa como coprocesador de una RaspberryPi.

Servidor

El servidor ofrece una interfaz donde es posible administrar la red y enviar comandos a los sensores. Está compuesto por un servidor NodeJS y una aplicación AngularJS que dan soporte al *Back-end* y *Fron-end*.

Capítulo 4

Red inalámbrica

Contenido

Elementos de la red **15**

Nodo **15**

TI-15.4 Stack **20**

Concentrador **25**

Elementos de la red

Nodo

Introducción

La aplicación implementa el dispositivo de la red, que le permite conectarse a la red creada por el concentrador. El sensor periódicamente envía reportes de datos en intervalos configurados por el concentrador y este responde con mensajes de rastreo.

Arquitectura Hardware

ARM Cortex M0 (Núcleo radio)

El núcleo *Cortex M0* (CM0) en el CC1350 es responsable de la interfaz audio, y traduce complejas instrucciones del núcleo *Cortex M3* (CM3) en bits que son enviados a través del enlace radio. Para el protocolo TI 15.4-Stack, el CM0 implementa la capa PHY de la pila de protocolos.

El *firmware* del núcleo de radio no está destinado a ser usado o modificado por la aplicación del desarrollador.

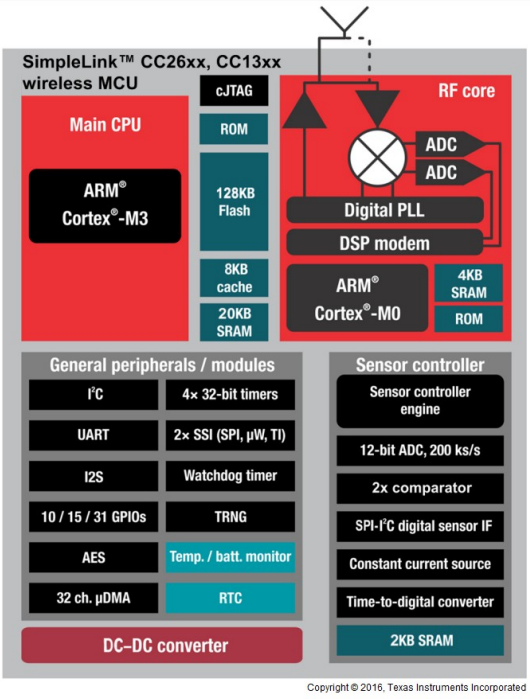


Figura 4.1: Diagrama de bloques de Simplelink™CC13x0

ARM Cortex M3 (Núcleo del sistema)

El núcleo CM3 está diseñado para ejecutar la pila del protocolo inalámbrico desde la capa de enlace hasta la capa de aplicación de usuario. La capa de enlace actúa como interfaz del núcleo de radio cómo un módulo software llamado *RF driver*.

Flash, RAM y periféricos

El CC1350 contiene en el sistema 128KB de memoria flash programable, 20KB de SRAM, y un amplio rango de periféricos. La memora flash se divide en partes que se pueden borrar de 4KB. El CC1350 también contiene 8kB de caché SRAM que puede ser utilizada para extender la capacidad de la RAM o puede funcionar como una caché normal para incrementar el rendimiento de la aplicación. Otros periféricos incluidos son UART, I2C, I2S, AES, TRNG, temperatura y monitor de la batería.

TI-RTOS

TI-RTOS es un entorno operativo para proyectos TI 15.4-Stack en dispositivos CC1350. El kernel TI-RTOS es una versión adaptada del kernel SYS/BIOS y funciona como un sistema operativo con controladores en tiempo real, con prioridades, multitarea y herramientas para la sincronización y planificación.

Arquitectura de la aplicación

En la figura 4.2 se muestra el diagrama de bloques de la aplicación del nodo.

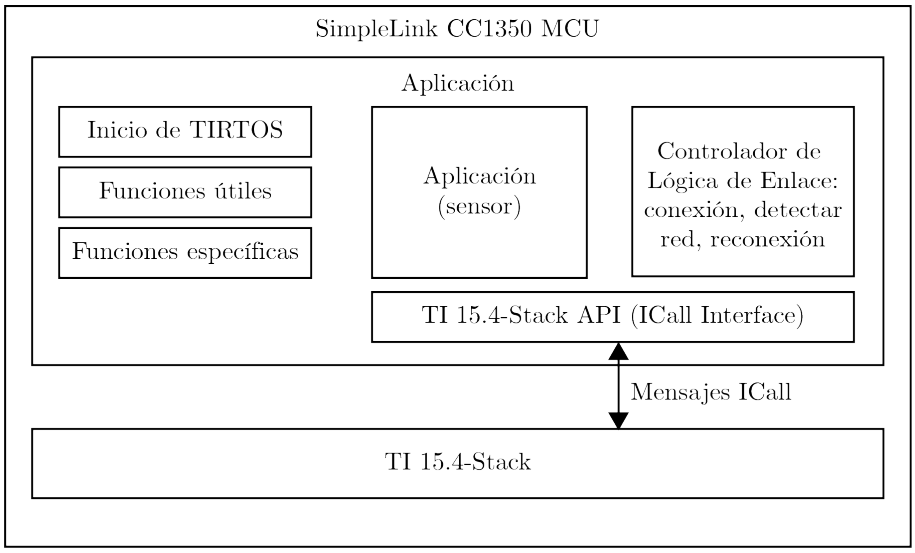


Figura 4.2: Diagrama de bloques de la aplicación

Aplicación: Este bloque contiene la lógica específica de la aplicación a desarrollar.

Controlador de lógica de enlace: Implementa varias funciones específicas del IEEE 802.15.4 o Wi-SUN (para una configuración *frequency-hopping*) para formación, conexión y reconexión de la red.

Inicio de TI-RTOS: Inicializa la aplicación.

Funciones útiles Provee varias utilidades para usar LCD, temporizadores, botones y más.

Funciones específicas: Implementa funciones como guardado de datos, y provee una interfaz para gestionar pulsaciones botones o mostrar información esencial en un LCD.

TI 15.4-Stack API (API MAC Module): Este módulo proporciona una interfaz para gestión y los servicios de datos del 802.15.4 stack mediante el módulo *Indirect Call Framework* (ICall).

Función de inicio

La función *main()* dentro del archivo *main.c* es el punto de inicio de la ejecución de la aplicación. En este punto los componentes relaciones con la placa son inicializados. Las tareas se configuran en eta función, inicializando los parámetros necesarios como su prioridad y su tamaño en la pila. En el paso final, las interrupciones se habilitan y el planificador *SYS/BIOS* se inicia llamando a *BIOS_start()*.

Arquitectura general de la aplicación

Esta sección describe como la tarea de la aplicación esta estructura en más detalle.

Función de inicio de la aplicación

Después de que la tarea sea construida y el planificador *SYS/BIOS* se inicie, la función que se le pasa durante la construcción de la tarea es ejecutada cuando la tarea está lista.

Las funciones de gestión de la energía son inicializadas aquí y el módulo ICall se inicia con la función *ICall_init()*. La dirección IEEE address (programada por TI) es obtenida desde la memoria flash. La tarea de la aplicación (Aplicación Sensor) es inicializada y ejecutada.

Sensor_init() establece varios parámetros de configuración, así como:

- Inicializa las estructuras para los datos
- Inicializa el TI 15.4-Stack
- Configura la seguridad y el *Logical Link Controller*
- Registra las funciones de retorno MAC

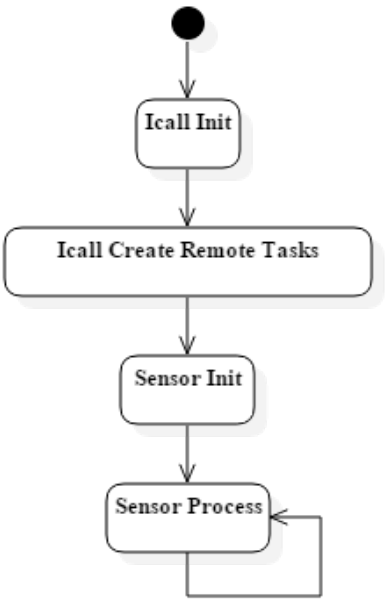


Figura 4.3: Diagrama de estados de la función de inicio

Tarea principal

Después de la función de inicialización, la tarea entra en un bucle infinito ejecutando siempre las mismas tareas, se puede ver en la figura

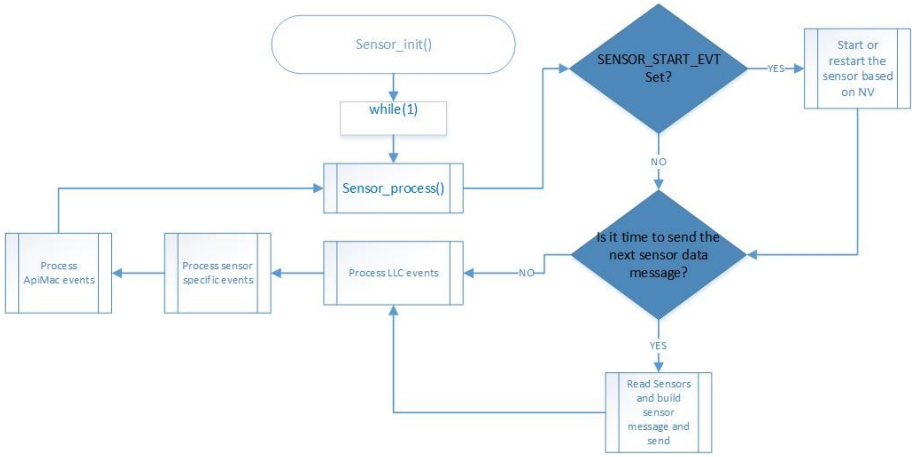


Figura 4.4: Flujo de la aplicación

Web física

La librería *Micro BLE Stack* (uBle) permite a la aplicación enviar un paquete en modo *broadcast* a todos los usuarios que estén en el radio de acción del dispositivo. Este paquete de datos es el que utilizamos para notificar a los usuarios con la información de la web física.

Los paquetes bluetooth usan la trama *Eddystone-URL*, estas forman parte del núcleo de la web física. Una vez el usuario la decodifica la trama podrá acceder a la URL si tiene conexión a internet.

Especificaciones de la trama

Potencia TX La potencia de transmisión es la potencia recibida a 0 metros, en dBm, en el rango de valores de -100 dBm a +20dBm con una resolución de 1 dBm.

Prefijo de la URL El prefijo de la url define la expansión utilizada por la url, por ejemplo “http://www.” o “https://” son codificadas por los bytes 0x00 o 0x03 respectivamente.

Byte offset	Campo	Descripción
0	Tipo de trama	valor = 0x10
1	Potencia TX	Potencia de TX calibrara a 0 m
2	Prefijo de la URL	Prefijo de la web
3+	URL codificada	Longitud de 1-17 bytes

Tabla 4.1: Formato de la trama *Eddystone-URL*

Sufijo de la URL El esquema de URL HTTP está definia por RFC 1738, por ejemplo "https://goo.gl/S6zT6P", y es usada para designar recursos accesibles usando HTTP.

Mensajes OTA

Los posibles mensajes entre el concentrador y el nodo, están definidos en el archivo *smsgs.h* (Apéndice A) .

Ambos, Nodo y Concentrador tienen que tener definidos las mismas estructuras de mensajes para una correcta comunicación.

TI-15.4 Stack

Introducción

El TI 15.4-Stack es una plataforma completa libre de derechos de autor para desarrollar aplicaciones que requieren una solución inalámbrica con topología en estrella, un extremado bajo consumo, largo alcance, fiable, robusto y seguro.

Este capítulo explica en detalle los diferentes modos de configuración de la red soportadas por el TI 15.4-Stack.

Elección de arquitectura

TI 15.4-Stack se puede utilizar con diferentes arquitecturas como las que se pueden observar en la figura 4.5: [12]

- Una configuración como dispositivo único la podemos observar en la figura 4.5 (izquierda). La aplicación y el protocolo son implementados en el CC13X0 como una solución *single-chip*. Esta configuración es la más simple y común

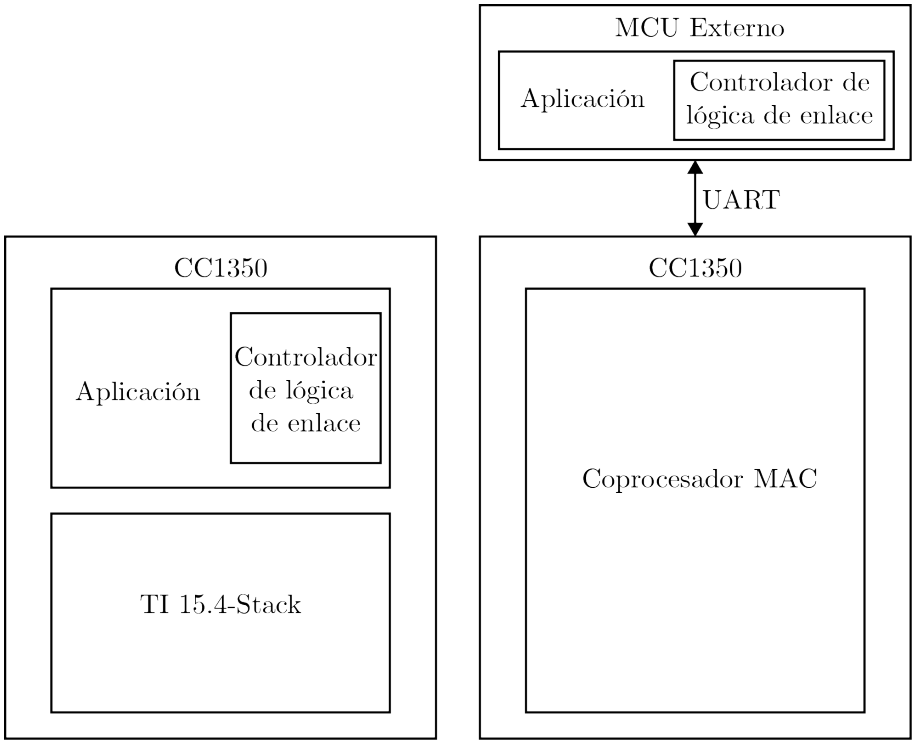


Figura 4.5: Configuración como dispostivo único y como coprocesador

cuando se usa el CC13X0 para nodos de la red. Esta configuración es la arquitectura más económica y de menor consumo.

- Una arquitectura como coprocesador se observa en la figura 4.5 (derecha). La pila de protocolos se ejecuta en el CC1350 mientras la aplicación es ejecuta en un microprocesador (MPU) o microcontrolador (MCU). La aplicación se comunica con el CC1350 usando el interfaz de protocolo de red (NPI) sobre una comunicación serie *Universal Asynchronous Receiver/Trasnsmitter* (UART). Esta configuración es usada en aplicaciones que añadan comunicación inalámbrica de rango alcance o en un ordenador sin los requerimientos para implementar los complejos protocolos asociados a una red inalámbrica.

Para el desarrollo de este TFG se utilizarán ambas arquitecturas, usando la arquitectura de dispositivo único para los nodos y la basada en coprocesador para el nodo central o concentrador.

Banda de frecuencias y tasa de transmisión

La elección de una banda y una tasa de transmisión puede elegirse configurando el apropiado atributo (PHY ID). Las opciones son explicadas en la tabla 4.2.

PHY ID	Tasa de datos	Frecuencia canal 0	Nº canales	Espacio canales
0	250 kbps	2405 MHz	16	5 MHz
1	50 kbps	902.2 MHz	129	200 kHz
30	50 kbps	863.125 MHz	34	200 kHz
128	50 kbps	403.3 MHz	7	200 kHz
129	5 kbps	902.2 MHz	129	200 kHz
130	5 kbps	403.3 MHz	7	200 kHz
131	5 kbps	863.125 MHz	34	200 kHz
132	200 kbps	902.4 MHz	64	400 kHz
133	200 kbps	863.225 MHz	17	400 kHz

Tabla 4.2: Bandas permitidas en TI 15.4-Stack y las frecuencias de sus canales

En el TFG se han utilizado las frecuencias de 2.4GHz y 868MHz por ser bandas de uso libre en España. Usando la banda de 868Mhz para la comunicación de largo alcanzo y la de 2.4GHz para enviar los mensajes de la web física.

Indirect Call Framework

ICall es un módulo que provee un mecanismo para que la aplicación se comunique con los servicios del TI 15.4-Stack, así como con ciertos servicios primitivos proporcionados por el sistema operativo en tiempo real (RTOS). ICall permite que la aplicación y la pila del protocolo operen eficientemente, comunicandose y compartiendo recursos en un entorno unificado RTOS.

El componente central de la arquitectura ICall es el *dispatcher*, que facilita la comunicación entre la aplicación y las tareas del TI 15.4-Stack.

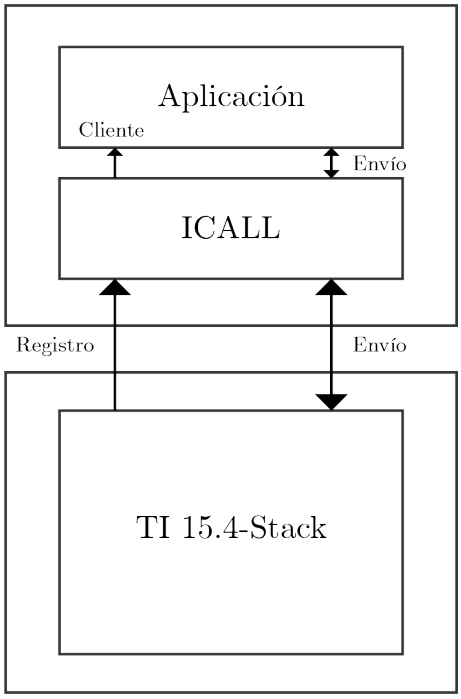


Figura 4.6: Aplicación ICall - Abstracción del protocolo

La figura 4.7 muestra un ejemplo de como un comando se envía desde la aplicación hasta el TI 15.4-Stack, con su correspondiente respuesta.

ICall_ init() inicializa la instancia del módulo ICall y la llamada *ICall_ createRemotetasks()* crea una tarea, con una función de entrada en una dirección conocida. Después de inicializar el ICall, la tarea de la aplicación se registra con el módulo ICall usando *ICall_ registerApp()*. Durante la ejecución de la tarea de la aplicación, esta envía un comando del protocolo como por ejemplo *ApiMac_ mlmeSetreqArray()*. El comando no se ejecuta en el hilo de la aplicación, en lugar, el comando es encapsulado en un mensaje ICall y es dirigido a la tarea del TI 15.4-Stack a través del módulo ICall. Mientras la aplicación espera al correspondiente mensaje. Cuando el TI 15.4-Stack finaliza la ejecución del comando, la respuesta es envíada a través del módulo ICall a la aplicación.

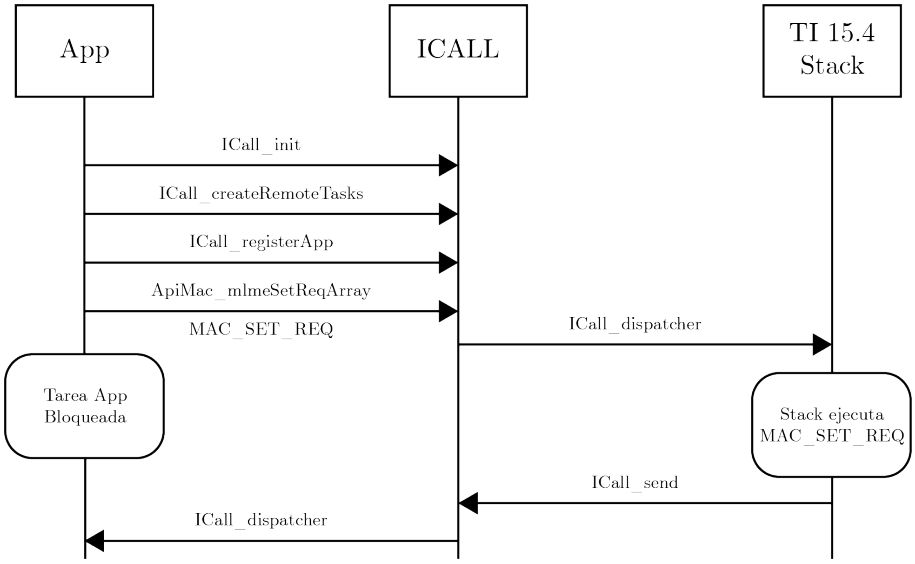


Figura 4.7: Ejemplo de comunicación usando el módulo ICALL

uBle

uBle es una variante del paquete BLE-Stack para los dispositivos con conectividad Sub1-GHz y 2.4GHz, como el CC1350. Este paquete permite a las aplicaciones ser encontradas, escanear o actuar como monitor de conexión. El paquete uBle utiliza el *MultiMode RF Driver*. El *MultiMode RF Driver* permite a las aplicaciones usar ambos modos donde otro protocolo de comunicación es integrado junto al uBle.

Restricciones y requisitos

El Micro BLE Stack tiene las siguientes restricciones y requisitos:

- Las opciones de diseño dependen de una parcial integración de ICall para guardar recursos del sistema. En el caso de uso del módulo ICall, se tiene que utilizar el sistema de gestión de pila del módulo ICall.
- No puede haber interacción humana-computador porque no existe separación entre el controlador y el host.
- Las opciones de privacidad no son soportadas.
- Para minimizar el consumo de memoria y eliminar redundantes cambios de contexto el uBle no utiliza diferentes tareas en TI-RTOS.
- Solo configuraciones utilizando el *MultiMode RF Driver* pueden ser usada con otros protocolos RF.

Modos de operación

Modo Beacon

Las especificaciones IEEE 802.15.4 definen un modo de operación *beacon-enabled* donde el dispositivo coordinador de la red de área personal o *personal area network* (PAN) transmite *beacons* para indicar su presencia y permite que otros dispositivos encuentren la PAN y se sincronicen. Los *beacons* proporcionan información sobre las especificaciones de la super-trama, que ayuda a los dispositivos con la intención de unirse a la red a sincronizarse y conocer los parámetros de la red antes de comenzar el proceso de unión. La super-trama está dividida en periodos activo e inactivos. Durante el perido activo, los dispositivos se comunican usando el procedimiento *Carrier Sense Multiple Access with Collision Avoidance* (CSMA/CA) excepto en la banda de 863MHz donde se usa el procedimiento *Listen Before Talk* (LBT) para el acceso al canal. Los periodo inactivos permiten a los dispositivos en la red conservar energía.

Modo NonBeacon

Las especificaciones IEEE 802.15.4 definen un modo de operación *non-beacon* donde el coordinador de la red no envía *beacons* periódicos. El modo *non-beacon* es un modo de operación asíncrono donde los dispositivos se comunican usando el mecanismo CSMA/CA.

Modo FrequencyHopping

Las aplicaciones que son desarrolladas usando el TI 15.4-Stack pueden ser configuradas para operar en redes con saltos de frecuencia donde los dispositivos de la red cambian de frecuencia. Este modo de funcionamiento está basado en el modo *Directed Frame Exchange* (DFE) de las especificaciones de Wi-SUN FAN.

Concentrador

Introducción

En este capítulo se describe la arquitectura y funcionamiento del concentrador (nodo central). Para este desarrollo se ha utilizado el proyecto de ejemplo que proporciona el fabricante llamado *TI 15.4-Stack Linux Gateway.*

La aplicación del concentrador en Linux proporciona la funcionalidad de concentrador de la red, añadiendo una interfaz como servidor socket para comunicarse con

la aplicación *Gateway*. Las aplicaciones del Concentrador y *Gateway* establecen un puente entre el protocolo IEEE 802.15.4 con el protocolo IP siendo una gran punto de comienzo para el IOT.

Diagrama de bloques y modelo de la interfaz

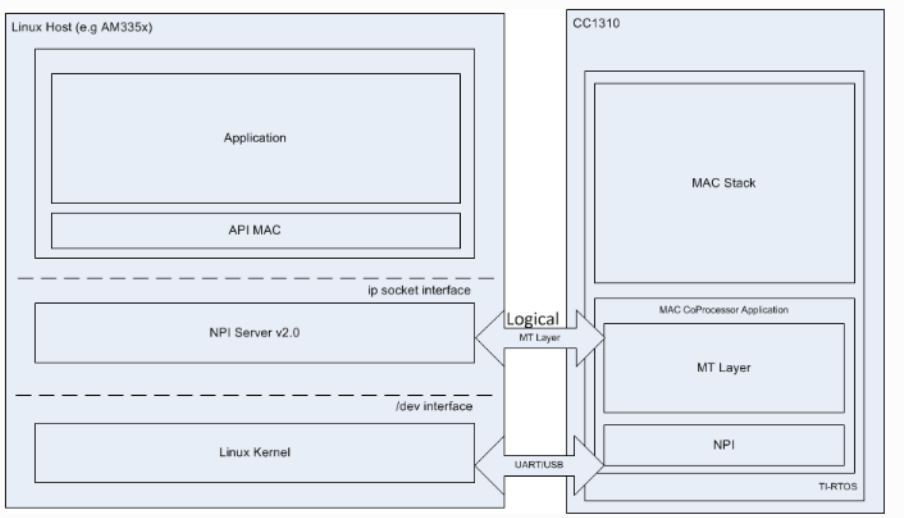


Figura 4.8: Arquitectura de software a alto nivel de las aplicaciones TI 15.4-Stack 2.1.0 Linux®

Esta sección describe la arquitectura de alto nivel basada en coprocesador, los componentes software, y la arquitectura general del sistema (veáse figura 4.8). El coprocesador es una entidad que implementa el estándar MAC IEEE 802.15.4e/g en un chip dedicado y provee una interfaz serie por la que un procesador externo controla y procesa las operaciones del coprocesador.

El concentrador se centra en una arquitectura escalable con una división perfecta donde el procesador ejecuta las capas sobre el IEEE 802.15.4e/g MAC/PHY.

En esta aplicación, el programa se ejecuta en una plataforma basada en Linux. Aunque los componentes de alto nivel , pueden ser conceptualmente aplicados a otras plataformas no basadas en Linux. Los componentes desarrollados serán descritos más adelante.

La interfaz entre el procesador y el coprocesador están definidas como capas lógicas que están separadas en esta arquitectura: una capa física (por ejemplo, USB o

UART), una capa lógica de enlace, y la capa de presentación.

Componentes software:

Aplicación del coprocesador: Es el programa ejecutandose en el dispositivo CC1350. Esta aplicación implementa una capa 802.15.4e/g MAC/PHY y proporciona una comunicación serie.

Kernel Linux : El kernel Linux provee los controladores para la interfaz serie que está disponible en un puerto físico (por ejemplo, USB).

Aplicación TI 15.4-Stack: Este módulo implementa la aplicación usando el protocolo 802.15.4e/g y la estructura del modelo MT.

Descripción del SDK

Figura 4.9: Estructura del directorio TI 15.4-Stack 2.1.0 Linux®

La figura 4.9 muestra la estructura del directorio de instalación del TI 15.4-Stack 2.1.0. A continuación se explica una descripción de alto nivel de cada carpeta:

components: Contiene las siguientes librerías:

- **common:** Rutinas para características del sistema operativo, como lectura y escritura de ficheros.
- **nv:** Simula una memoria no volátil, como la usada en sistemas empotrados.
- **api:** Interfaz de mensajes API MAC y MT

docs: Documentos como la guía de desarrollo y la guía de comandos MAC para el coprocesador.

example: Aplicación de ejemplo

- **cc13xx-sbl:** Herramientas para la actualización para los dispositivos CC13x0.
- **collector:** Aplicación de ejemplo que demuestra como iniciar una red, permitir la conexión de dispositivos y recoger datos desde dispositivos remotos.
- **gateway:** Una aplicación basada en Node.js™que crea un servidor local y muestra la información de la red y los datos de los nodos.
- **npi_server2:** Interfaz socket para comunicarse con el coprocesador.
- **google:** Contiene un makefile para descargar e instalar el compilador de *Google protocol buffer*.

firmware Precompilados ficheros .hex para el coprocesador.

prebuilt Compilación para ejecutar la aplicación de ejemplo en una BeagleBone Black.

scripts Contiene fragmentos de ficheros makefile usados para compilar la aplicación de ejemplo.

Funcionamiento

El proyecto comienza en la función *main()* en el fichero *linux_main.c*, donde se inicializan las diferentes interfaces, se lee el fichero de configuración y ejecuta la función *App_main()* del fichero *appsrv.c*.

La función *App_main()* se encarga de inicializar los dos hilos de ejecución principales que tiene el programa, *client-thread* y *collector-thread*. La tarea del cliente se encarga de conectarse al servidor y procesar la transmisión y recepción de datos por ese canal. Por otro lado la tarea del concentrador, se encarga de generar la red TI 15.4-Stack y procesar los mensajes enviados por este protocolo.

Hilo del cliente web

Este hilo mantiene la comunicación con el servidor, y espera recibir un mensaje de este. Cuando un mensaje es recibido la función *appsrv_handle_appClient_request()* es la encargada de procesar el mensaje y notificar a la red TI 15.4-Stack a través del hilo del concentrador.

Hilo del concentrador

La función *Collector_process()* del fichero *collector.c* contiene la lógica de esta tarea, que se describe en la figura ??.

Protocol Buffers

Para facilitar el envío de datos entre el concentrador y el servidor, se ha utilizado *Protocol Buffers*.

Protocol Buffers es un mecanismo flexible, eficiente y automatizado para estructurar datos estructurados. Solo es necesario indicar como se estructuran los datos y al compilarse generan la implementación en multiples lenguajes de programación de los mecanismo para codificar y descodificar datos.

Funcionamiento

La estructura de los datos a codificar se definen en archivos .proto. Cada mensaje es una pequeña estructura que contiene una serie de parejas clave-valor. (Apéndice A)

Como se observa en el listado 4.1, el formato de los mensajes es simple y similar a la definición de variables en código C. Una vez definidos los mensajes, se ejecuta el compilador de *Protocol Buffers* para el lenguaje de tu aplicación, en nuestro caso C.

Listado 4.1: Ejemplo de estructura con Protocol Buffers

```
/*!
  Configuration Request message: sent from controller to
     the sensor.
*/
message Smsgs_configReqMsg
{
    /*! Command ID - 1 byte */
```

```
    required Smsgs_cmdIds cmdId = 1;
    /*! Reporting Interval */
    required uint32 reportingInterval = 2;
    /*! Polling Interval */
    required uint32 pollingInterval = 3;
    /*! Time now */
    required uint32 timeNow=4;
    /*! URL ble */
    required string url = 5;
}
```

Las estructuras de datos para nuestra aplicación se pueden observar en el Apéndice A. En este podemos observa que hemos creado un fichero con las mismas estructuras a smsgs.h para poder convertir los mensajes que nos llegan de los nodos en mensajes *Protocol Buffers* y así poder enviarlos al servidor.

Capítulo 5

Servidor

Contenido

Introducción **31**
Estructura del servidor **31**
Back-end **32**
Front-end **32**

Introducción

El servidor se puede dividir en dos partes *Front-end* y *Back-End* que son términos que se refieren a la separación entre una capa de presentación y una capa de acceso a datos respectivamente.

Estructura del servidor

El servidor está compuesto por el directorio que se observa en la figura 5.1 y a continuación se describe la función de cada directorio:

api: en este directorio se encuentran las definiciones de llamadas REST al servidor.

appClient: Esta carpeta contiene el *Back-end* de la web:

- **devices:** se definen las funciones relacionadas con la gestión de los nodos.
- **models:** aquí se definen los modelos para guardarlos en la base de datos.
- **nwkinfo:** se definen las funciones relacionadas con la gestión del concentrador.

protofiles: aquí se almacenan los ficheros .proto de *Protocol Buffers*.

appClient.js: en este fichero se inicia el servidor que se comunica con el concentrador y se procesan los mensajes.

node_modules: librerías utilizadas en el proyecto.

webserver: en este directorio está contenida la lógica del *Front-end*:

public: interfaz del cliente en AngularJS.

routes: Definición de rutas.

views: archivos html de las vistas.

webserver.js: se inicia el cliente y se gestiona las peticiones a la web por parte del usuario.

gateway.js: inicia el *back-end* y el *front-end* y la comunicación entre ambas por *web-sockets*.

run_gateway.sh: *script* para iniciar el servidor.

Back-end

El *Back-end* es el área que se dedica a la lógica, en este encontramos una interfaz que se encarga de comunicarse con el concentrador y otra que se encarga de comunicarse con el *Front-end* usando *Web-Sockets*.

Al inicio del archivo *appClient.js* se abre el puerto 3000 y se mantiene a la escucha hasta que el concentrador se conecta. Cuando el concentrador se conecta, este el envía la información de configuración.

Toda la información de la red se almacena en una base de datos MongoDB®, también se almacena los datos enviados por los nodos así como sus parámetros de configuración.

Una vez la conexión entre el concentrador y el *back-end* se ha establecido, ambos se quedan a la espera de que el usuario desde el *front-end* permita la conexión de los nodos a la red.

Front-end

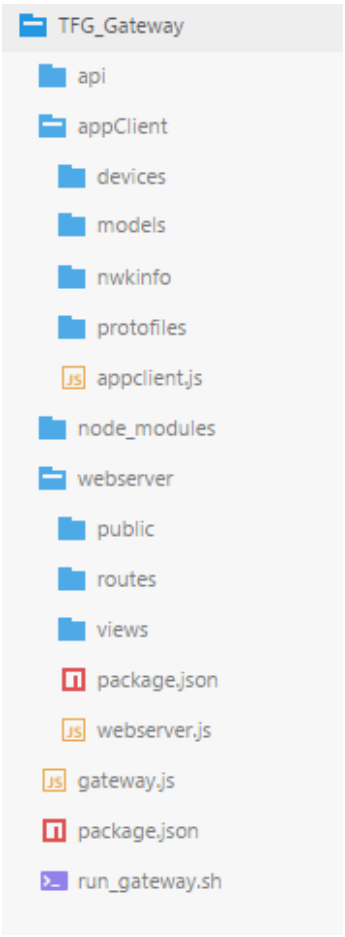


Figura 5.1: Estructura del directorio del servidor

El *Front-end* es la interfaz del servidor con el usuario. Para facilitar el control de las vistas se ha utilizado el *framework AngularJS*.

Angular es un *framework* Modelo Vista Controllador (MVC) para la construcción de aplicaciones de una única página del lado del cliente en HTML y JavaScript.

Como se ha mencionado anteriormente, el patrón que se usa en Angular es el conocido como Modelo, Vista y Controlador:

Vistas: Será el código HTML y todo lo que presente los datos o información.

Controladores: Se encarga de toda la lógica de la aplicación.

Modelo: El modelo es la estructura que define un tipo de dato, y permite acceder a él en la base de datos.

Parte III

Pruebas y funcionamiento

Capítulo 6

Prueba en entorno real

Contenido

Localización **37**

Localización

Capítulo 7

Prueba de consumo

Contenido

Introducción **39**
Medida del consumo **39**
Resultados **40**

Introducción

Para una exitosa implementación del IOT uno de los requisitos más importantes es un consumo eficiente del sistema.

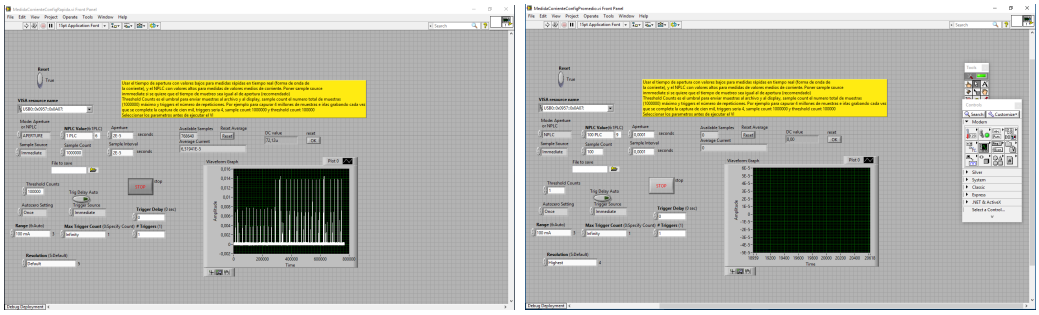
La comunicación inalámbrica entre los nodos comienza a ser un problema cuando la fuente de alimentación es limitada. Un sistema que pueda funcionar con una sola pila AAA durante años es lo ideal en el IOT [13]. Por ello, durante el desarrollo de este proyecto se ha utilizado como protocolo inalámbrico el IEEE 802.15.4, que destaca su bajo consumo.

Medida del consumo

Para realizar la medida de consumo se ha utilizado el multímetro digital Keysight 34411A, que proporciona 6 dígitos y medio de resolución y una velocidad de muestreo de 50000 muestras/s. [14]

Para el control del multímetro se ha utilizado dos programas de LabView, uno de ellos permite obtener datos a la máxima velocidad de muestreo aunque eso li-

mite la resolución a 4 dígitos (figura 7.1a) y el otro programa permite tomar datos promediados en intervalos de tiempo (figura 7.1b).



(a) Captura rápida (b) Captura promedio

Figura 7.1: Programas de LabView

Resultados

Aunque el objetivo de esta prueba es conocer el consumo medio de corriente del nodo. Antes de nada se necesita conocer que fondo de escala utilizar en el multímetro para la medida en promedio, para ello se hace uso del programa de captura a máxima frecuencia de muestreo, donde se obtiene la corriente instantánea máxima que consume el nodo, unos 15 mA (figura 7.2).

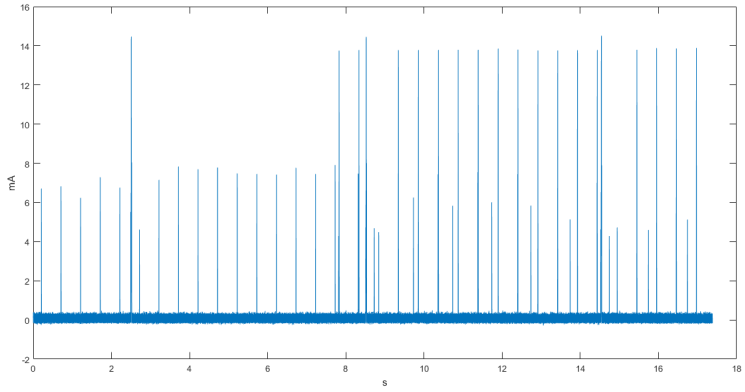


Figura 7.2: Consumo en mA del nodo a la máxima frecuencia de muestreo

Finalmente, utilizando el programa de medida promediada de LabView con un fondo de escala de 100mA, realizamos una medida de la corriente promedio durante 50 ciclos de subidas al servidor. Con esta configuración se obtiene que el consumo

medio de un nodo es de 169.6 μA, lo que equivale a más de 5 meses de funcionamiento utilizando una pila de botón CR 2477.

Capítulo 8

Funcionamiento

Contenido

Inicio de la red **43**
Administración de los nodos **46**
Mapa **48**

Inicio de la red

1. Iniciar el servidor

El primer paso es iniciar el servidor, ya que toda la red depende de este. Como se ha utilizado Amazon Web Services, desde su web iniciamos la instancia que corre en el inicio el servidor de la aplicación (véase figura 8.1.

Después de iniciar el servidor, es posible acceder a la web de gestión desde la dirección dns de la instancia del servidor en el puerto 5000. Inicialmente el concentrador no está conectado por lo que nos aparece un mensaje indicándolo, figura 8.2.

2. Encender concentrador

Antes de enceder la RaspberryPi, conectamos el coprocesador por USB y damos conexión a Internet a la RaspberryPi. La RaspberryPi está configurada para ejecutar el programa del concentrador y conectarse al servidor, por lo que no es necesaria más interacción con el concentrador. Después de encenderse la RaspberryPi, la web habrá cambiado y ahora ya nos indicará algunos parámetros de la web, figura 8.3.

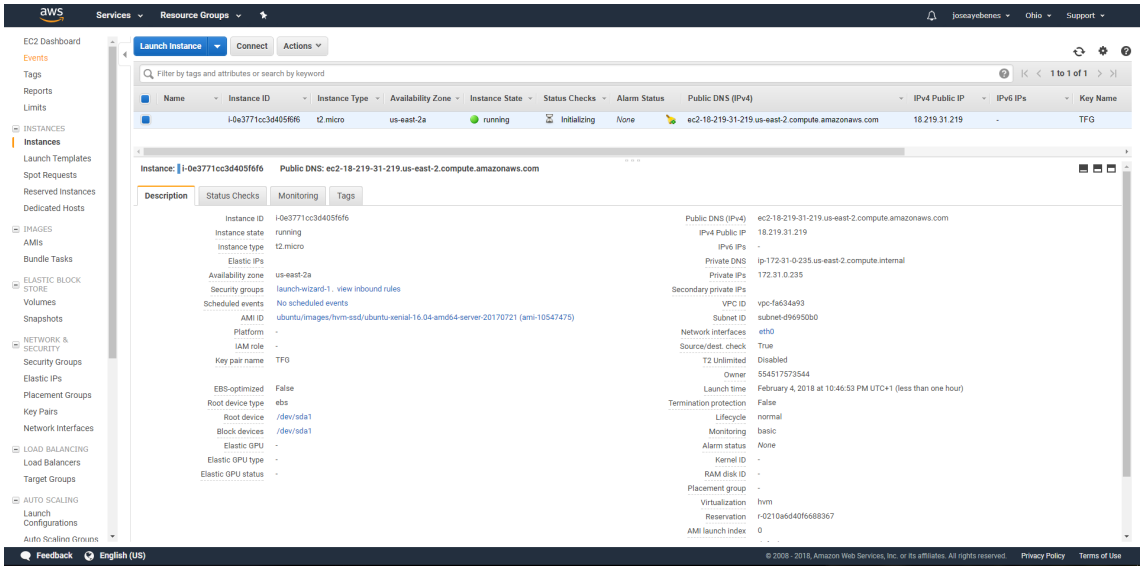


Figura 8.1: Web de gestión de instancias en Amazon Web Services

En la parte superior de la web se observan los parámetros de la red como son su dirección PAN, el estado que indica si la red permite la conexión de los nodos, el tipo de comunicación y si está activada la seguridad.

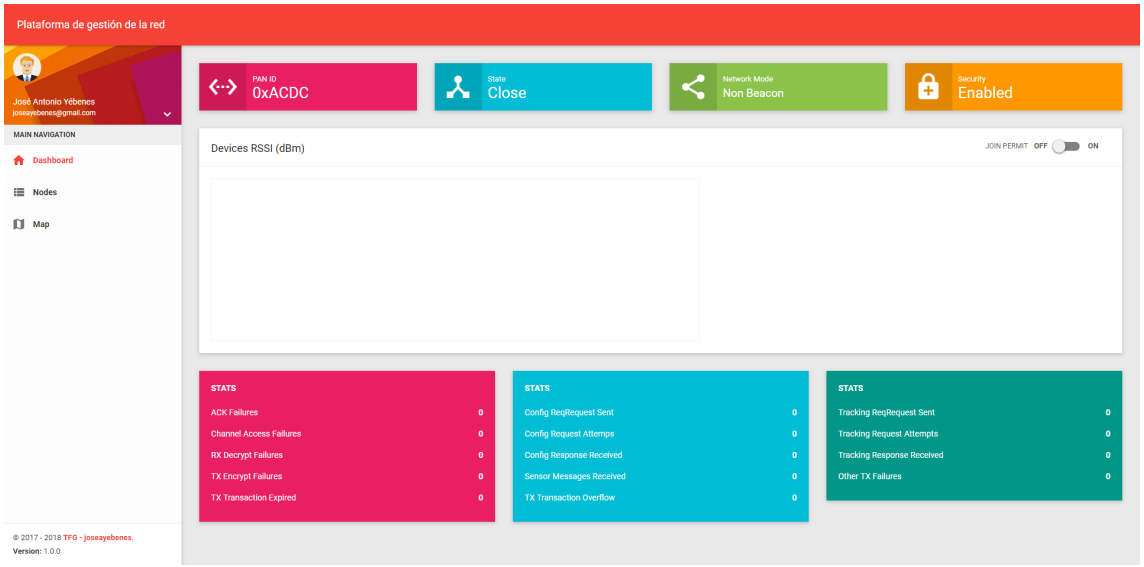


Figura 8.3: Apariencia de la web cuando ya está conectado el concentrador

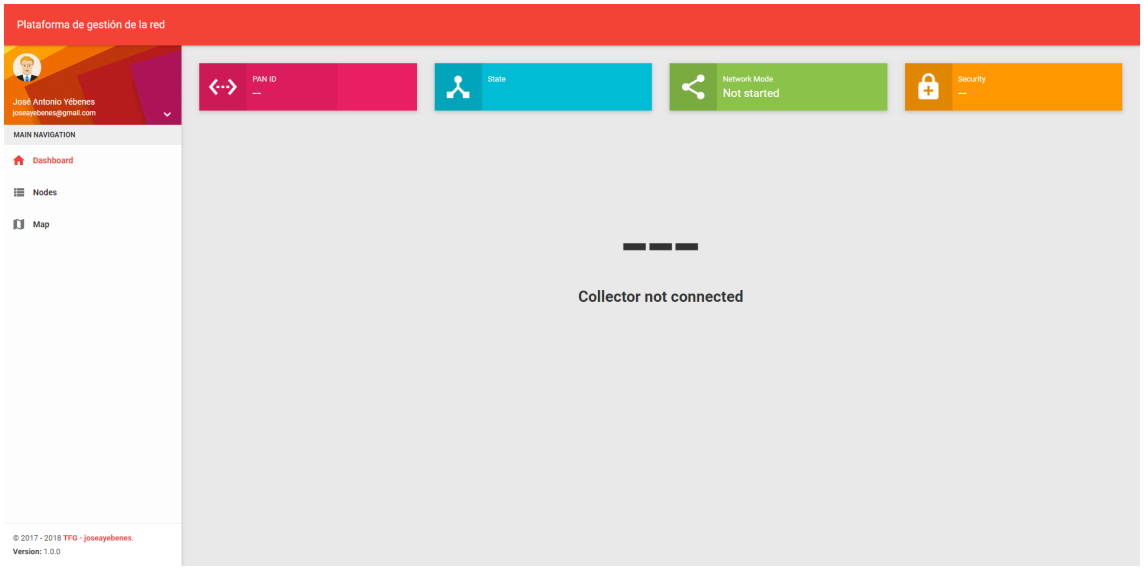


Figura 8.2: Apariencia de la web cuando no está conectado el concentrador

2. Encender nodos

Encendemos todos los nodos necesarios, en las pruebas realizadas hemos utilizado dos.

3. Permitir conexiones de los nodos

Para permitir la conexión de los nodos a la red, es necesario poner el botón "*JOIN PERMIT*" a "ON". Después de eso, los nodos que ya haya encendidos, se conectarán a la red y aparecerán en la web.

En la figura 8.4 se observa la apariencia de la web cuando dos dispositivos está conectados. En la parte central se observa una gráfica con potencia de la señal de cada uno de los nodos en dBi, y en la parte inferior estadísticas generales de la red.

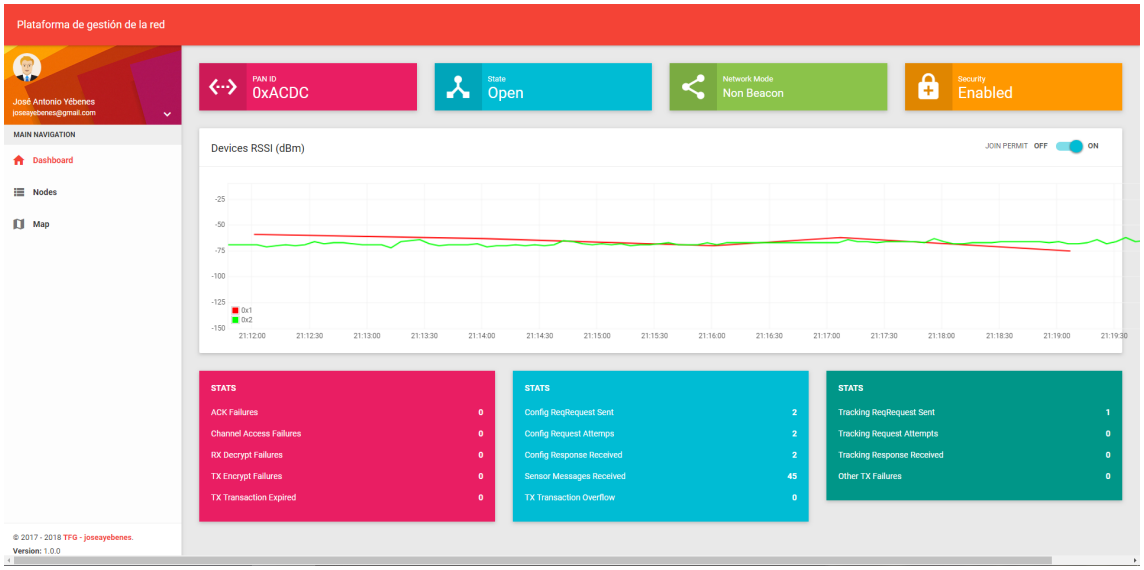


Figura 8.4: Apariencia de la web cuando hay dos nodos conectados

Administración de los nodos

Si se hace click en la barra de navegación derecha en el botón "*Nodes*", se accede al panel de administración de los nodos, en el aparecen la lista de nodos conectados y la información relevante de cada uno, como se observa en la figura 8.5.

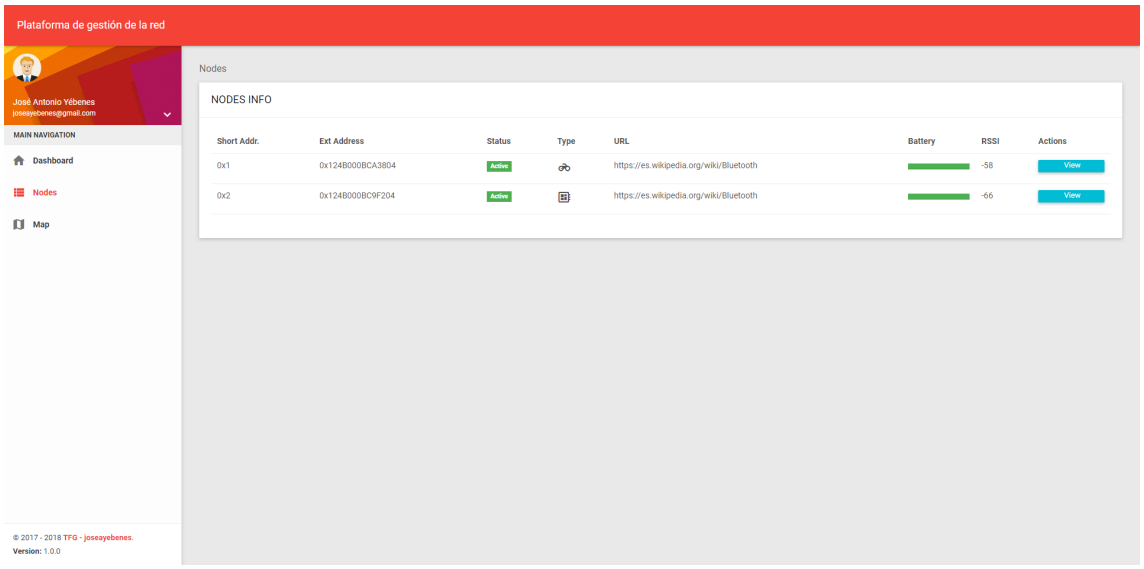


Figura 8.5: Panel de administración de los nodos

Haciendo click en alguno de los botones "*View*", se accede a la administración del nodo correspondiente, como se aprecia en la tabla hay dos tipos de nodos, uno representado con una bicicleta y el otro con el símbolo de un microcontrolador, que representan un nodo destinado al control de un parking de bicicletas y un nodo genérico de información respectivamente. Cada tipo de nodo tiene una interfaz de administración diferente.

Administración de nodo genérico

En la figura 8.6, se puede observar toda la información relevante sobre el nodo seleccionado.

En el cuadro de "*Physical Web - URL*" es posible modificar la URL que envía dicho nodo. Justo debajo del título aparece la URL que actualmente está enviando. En el cuadro se puede introducir cualquier URL de tipo HTTPs y al hacer click en el botón se enviará el cambio al nodo.

También es posible indicar cuales son las coordenadas del nodo desde el cuadro "*Position*".

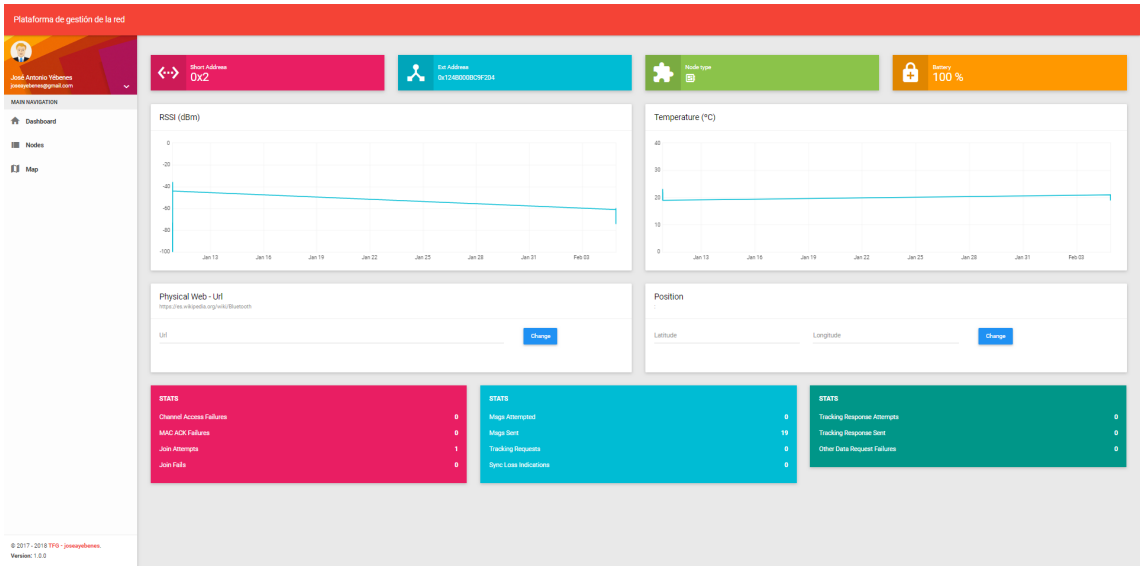


Figura 8.6: Panel de administración de un nodo tipo genérico

Administración de un tipo parking de bicicletas

El panel de administración de este tipo de nodo se observa en la figura 8.7. Esta web permite lo mismo que la web de un nodo genérico, pero además hay un cuadro llamado "*Parking*", en este cuadro se observaría el estado de cada anclaje del parkin, y si se hace click en cualquiera de ellos el estado del anclaje cambiaría de abierto a cerrado o vicerversa. Esta función está simulada en el nodo como un array donde se guardan los estados de los candados.

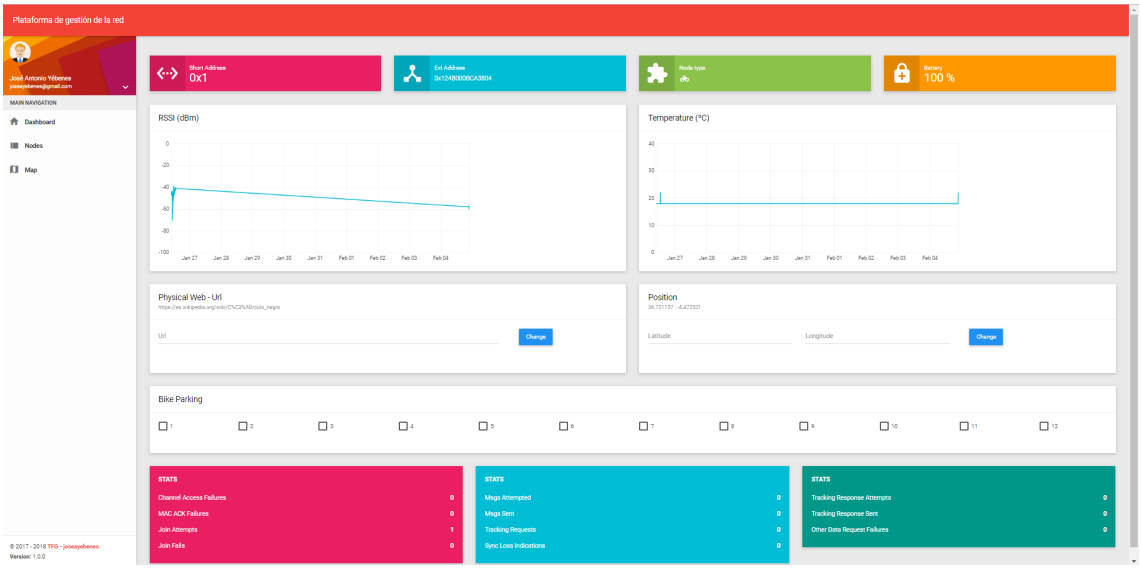


Figura 8.7: Panel de administración de un tipo parking de bicicletas

Mapa

Desde la web también es posible observar donde está cada nodo en la red, para ello accediendo desde la barra lateral a la opción "*Map*" se observa un mapa como el que se puede observar en la figura 8.8. En este mapa podemos observar las posiciones de los nodos que se han indicado en sus respectivos paneles de administración y además desde el formulario inferior es posible cambiar la posición del concentrador y su radio de alcance. Con esto es posible tener una idea de si alguno de los nodos se ha colocado demasiado cerca del borde del alcance del concentrador.

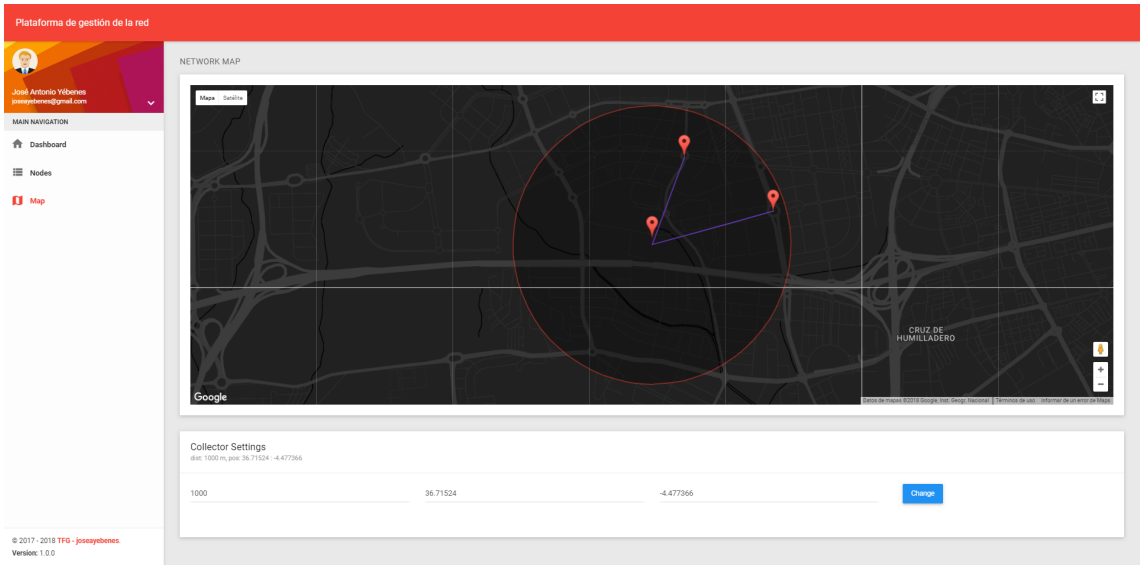


Figura 8.8: Vista del mapa con las posiciones de los nodos

Parte IV

Conclusiones y lineas futuras

Conclusiones

Después del desarrollo del proyecto, es pertinente hacer una valoración final del mismo, respecto a los resultados obtenidos, las expectativas o el resultado de la experiencia acumulada.

Este TFG nace con el objetivo de ser un trabajo multidisciplinar que abarque gran parte de los conocimientos adquiridos durante una titulación generalista, como son las comunicaciones, telemática y electrónica. Desde este punto se hizo un análisis de diferentes tecnologías recientes y se seleccionaron dos como principales: el concepto de "Web física" y el microntrolador "CC1350". De esta forma surge el objetivo principal de este TFG que es la integración de estas dos tecnologías en una plataforma que pudiera utilizarse como una solución al problema de la interacción humano-máquina.

Durante el desarrollo de este TFG surgieron problemas debidos principalmente al pequeño tiempo que llevaba el microcontrolador en el mercado que dificultaba la búsqueda de información. Con el tiempo fue surgiendo más documentación que hizo cambiar la linea de diseño del proyecto hasta llegar a la que plasma este TFG.

Finalmente, se han visto concluidos satisfactoriamente los principales objetivos de este proyecto, que han permitido un mejor conocimiento de las tecnologías utilizadas y de la metodología de trabajo para llevar un proyecto desde concepto hasta prototipo, pasando por las fases de investigación, de un producto como ha sido este trabajo de fin de grado.

Líneas futuras

Durante el desarrollo del TFG se ha observado un crecimiento en el uso de las tecnologías que implementa este proyecto, lo que vaticina un futuro donde el concepto de web física sea familiar a la población. De este crecimiento se extraen lineas futuras de desarrollo en este ámbito como son:

- Mejora del alcance analizando las diferentes alternativas que ofrece TI 15.4-Stack.
- Crear un nodo que emule una máquina expendedora, donde el usuario pueda realizar su compra a través de la web física.
- Mejora de la duración de la batería optimizando el código y añadiendo tecnología de recolección de energía.
- Creación de placas de circuito impreso para nodo y concentrador que permita reducir el tamaño.

José Antonio Yébenes Gálvez
10 de febrero de 2018

Parte V

Apéndices

Apéndice A

Estructuras de mensajes

A.1. Mensajes OTA

Listado A.1: Archivo smsgs.h

```
#ifndef SMGSS_H
#define SMGSS_H

/*********
 Includes
 *********/

#include <stdbool.h>
#include <stdint.h>

#ifdef __cplusplus
extern "C"
{
#endif

/*!
 \defgroup overAir Over-the-air Messages
 <BR>
 This header file defines the over-the-air messages
    between the collector
 and the sensor applications.
 <BR>
 Each field in these message are formatted low byte first
```

```
    , so a 16 bit field
 with a value of 0x1516 will be sent as 0x16, 0x15.
 <BR>
 The first byte of each message (data portion) contains
    the command ID (@ref
 Smsgs_cmdIds).
 <BR>
  @{
 */

/**********
 Constants and definitions
 **********/

/*! Sensor Message Extended Address Length */
#define SMGS_SENSOR_EXTADDR_LEN 8

/*! Config Request message length (over-the-air length)
    */
#define SMSGS_CONFIG_REQUEST_MSG_LENGTH 23
/*! Config Response message length (over-the-air length)
    */
#define SMSGS_CONFIG_RESPONSE_MSG_LENGTH 28
/*! Length of the bikeParkingState portion of the
    Smsgs_configRspMsg_t   */
#define SMSGS_CONFIG_BIKEPARKING_LEN 4

/*! Tracking Request message length (over-the-air length)
     */
#define SMSGS_TRACKING_REQUEST_MSG_LENGTH 1
/*! Tracking Response message length (over-the-air length
    ) */
#define SMSGS_TRACKING_RESPONSE_MSG_LENGTH 1

/*! Length of a sensor data message with no configured
    data fields */
#define SMSGS_BASIC_SENSOR_LEN (7 +
    SMGS_SENSOR_EXTADDR_LEN)
/*! Length of the tempSensor portion of the sensor data
    message */
#define SMSGS_SENSOR_TEMP_LEN 4
```

```
/*! Length of the lightSensor portion of the sensor data
    message */
#define SMSGS_SENSOR_LIGHT_LEN 2
/*! Length of the humiditySensor portion of the sensor
    data message */
#define SMSGS_SENSOR_HUMIDITY_LEN 4
/*! Length of the messageStatistics portion of the sensor
     data message */
#define SMSGS_SENSOR_MSG_STATS_LEN 40
/*! Length of the configSettings portion of the sensor
    data message */
#define SMSGS_SENSOR_CONFIG_SETTINGS_LEN 8

/*! Toggle Led Request message length (over-the-air
    length) */
#define SMSGS_TOGGLE_LED_REQUEST_MSG_LEN 1
/*! Toggle Led Request message length (over-the-air
    length) */
#define SMSGS_TOGGLE_LED_RESPONSE_MSG_LEN 2

/*! Toggle Ble Request message length (over-the-air
    length) */
#define SMSGS_TOGGLE_BLE_REQUEST_MSG_LEN 1
/*! Toggle Ble Request message length (over-the-air
    length) */
#define SMSGS_TOGGLE_BLE_RESPONSE_MSG_LEN 2

/*! Change Ble URL Request message length (over-the-air
    length) */
#define SMSGS_CHANGE_BLE_URL_REQUEST_MSG_LEN 11
/*! Change Ble URL Response message length (over-the-air
    length) */
#define SMSGS_CHANGE_BLE_URL_RESPONSE_MSG_LEN 11

/*! Parking State message length (over-the-air length) */
#define SMSGS_PARKING_STATE_MSG_LEN 1
/*! Bike Select message length (over-the-air length) */
#define SMSGS_BIKE_SELECT_MSG_LEN 2
/*! Parking State message length (over-the-air length) */
#define SMSGS_PARKING_STATE_RSP_MSG_LEN 5
```

```
/*!
 Message IDs for Sensor data messages.  When sent over-
    the-air in a message,
 this field is one byte.
 */
 typedef enum
 {
    /*! Configuration message, sent from the collector to
        the sensor */
    Smsgs_cmdIds_configReq = 1,
    /*! Configuration Response message, sent from the
       sensor to the collector */
    Smsgs_cmdIds_configRsp = 2,

    /*! Tracking request message, sent from the the
       collector to the sensor */
    Smsgs_cmdIds_trackingReq = 3,
     /*! Tracking response message, sent from the sensor
        to the collector */
    Smsgs_cmdIds_trackingRsp = 4,

    /*! Sensor data message, sent from the sensor to the
       collector */
    Smsgs_cmdIds_sensorData = 5,

    /*! Toggle LED message, sent from the collector to
       the sensor */
    Smsgs_cmdIds_toggleLedReq = 6,
    /*! Toggle LED response msg, sent from the sensor to
       the collector */
    Smsgs_cmdIds_toggleLedRsp = 7,

    /*! Change URL request message, sent from the
       collector to the sensor */
    Smsgs_cmIds_changeBleUrl = 8,
    /*! Change URL response message, sent from the sensor
        to the collector */
    Smsgs_cmIds_changeBleUrlRsp = 9,
```

```
    /*! Bike parking state message, sent from the sensor
       to the collector*/
    Smsgs_cmIds_bikeParkingState = 10,
    /*! Bike Select message, sent from the sensor to the
       collector*/
    Smsgs_cmIds_bikeSelect = 11,
    /*! Bike parking state, sent from the sensor to the
       collector*/
    Smsgs_cmIds_bikeParkingStateRsp = 12,

 } Smsgs_cmdIds_t;

/*!
 Frame Control field states what data fields are included
     in reported
 sensor data, each value is a bit mask value so that they
     can be combined
 (OR'd together) in a control field.
 When sent over-the-air in a message this field is 2
    bytes.
 */
typedef enum
{
    /*! Temperature Sensor */
    Smsgs_dataFields_tempSensor = 0x0001,
    /*! Message Statistics */
    Smsgs_dataFields_msgStats = 0x0002,
    /*! Config Settings */
    Smsgs_dataFields_configSettings = 0x0004,

} Smsgs_dataFields_t;

/*!
 Status values for the over-the-air messages
 */
typedef enum
{
    /*! Success */
    Smsgs_statusValues_success = 0,
    /*! Message was invalid and ignored */
```

```
    Smsgs_statusValues_invalid = 1,
    /*!
     Config message was received but only some frame
        control fields
     can be sent or the reportingInterval or
        pollingInterval fail
     range checks.
     */
    Smsgs_statusValues_partialSuccess = 2,
} Smsgs_statusValues_t;

/*!
 Type of nodes
 */
typedef enum
{
    Smsgs_typeNode_simpleNode = 0x01,
    Smsgs_typeNode_parkingBike = 0x02,
    //add new type
} Smsgs_typeNode_t;




/**********
 Structures - Building blocks for the over-the-air sensor
     messages
 *********/

/*!
 Configuration Request message: sent from controller to
    the sensor.
 */
typedef struct _Smsgs_configreqmsg_t
{
    /*! Command ID - 1 byte */
    Smsgs_cmdIds_t cmdId;
    /*! Reporting Interval */
    uint32_t reportingInterval;
    /*! Polling Interval */
    uint32_t pollingInterval;
```

```
    /*! Time now */
    uint32_t timeNow;
    /*! URL BLE  */
    uint8_t url[10];
} Smsgs_configReqMsg_t;

/*!
 Configuration Response message: sent from the sensor to
    the collector
 in response to the Configuration Request message.
 */
typedef struct _Smsgs_configrspmsg_t
{
    /*! Command ID - 1 byte */
    Smsgs_cmdIds_t cmdId;
    /*! Response Status - 2 bytes */
    Smsgs_statusValues_t status;
    /*! Type Node - 1 byte*/
    Smsgs_typeNode_t type;
    /*! Frame Control field - 2 bytes - bit mask of
       Smsgs_dataFields */
    uint16_t frameControl;
    /*! Reporting Interval - 4 bytes */
    uint32_t reportingInterval;
    /*! Polling Interval - 4 bytes */
    uint32_t pollingInterval;
    /*! Time now */
    uint32_t timeNow;
    /*! URL BLE  */
    uint8_t url[10];
    /*! BikeParking State - only if type is
       Smsgs_typeNode_parkingBike*/
    uint32_t bikeParkingState;
} Smsgs_configRspMsg_t;

/*!
 Tracking Request message: sent from controller to the
    sensor.
 */
typedef struct _Smsgs_trackingreqmsg_t
{
```

```
    /*! Command ID - 1 byte */
    Smsgs_cmdIds_t cmdId;
} Smsgs_trackingReqMsg_t;

/*!
 Tracking Response message: sent from the sensor to the
    collector
 in response to the Tracking Request message.
 */
typedef struct _Smsgs_trackingrspmsg_t
{
    /*! Command ID - 1 byte */
    Smsgs_cmdIds_t cmdId;
} Smsgs_trackingRspMsg_t;

/*!
 Toggle LED Request message: sent from controller to the
    sensor.
 */
typedef struct _Smsgs_toggleledreqmsg_t
{
    /*! Command ID - 1 byte */
    Smsgs_cmdIds_t cmdId;
} Smsgs_toggleLedReqMsg_t;

/*!
 Toggle LED Response message: sent from the sensor to the
     collector
 in response to the Toggle LED Request message.
 */
typedef struct _Smsgs_toggleledrspmsg_t
{
    /*! Command ID - 1 byte */
    Smsgs_cmdIds_t cmdId;
    /*! LED State - 0 is off, 1 is on - 1 byte */
    uint8_t ledState;
} Smsgs_toggleLedRspMsg_t;


/*!
 Temp Sensor Field
```

```
 */
typedef struct _Smsgs_tempsensorfield_t
{
    /*!
     Ambience Chip Temperature - each value represents a
        0.01 C
     degree, so a value of 2475 represents 24.75 C.
     */
    int16_t ambienceTemp;
    /*!
     Object Temperature - each value represents a 0.01 C
     degree, so a value of 2475 represents 24.75 C.
     */
    int16_t objectTemp;
} Smsgs_tempSensorField_t;

/*!
 Message Statistics Field
 */
typedef struct _Smsgs_msgstatsfield_t
{
    /*! total number of join attempts (associate request
       sent) */
    uint16_t joinAttempts;
    /*! total number of join attempts failed */
    uint16_t joinFails;
    /*! total number of sensor data messages attempted.
       */
    uint16_t msgsAttempted;
    /*! total number of sensor data messages sent. */
    uint16_t msgsSent;
    /*! total number of tracking requests received */
    uint16_t trackingRequests;
    /*! total number of tracking response attempted */
    uint16_t trackingResponseAttempts;
    /*! total number of tracking response success */
    uint16_t trackingResponseSent;
    /*! total number of config requests received */
    uint16_t configRequests;
    /*! total number of config response attempted */
    uint16_t configResponseAttempts;
```

```
    /*! total number of config response success */
    uint16_t configResponseSent;
    /*!
     Total number of Channel Access Failures.  These are
        indicated in MAC data
     confirms for MAC data requests.
     */
    uint16_t channelAccessFailures;
    /*!
     Total number of MAC ACK failures. These are
        indicated in MAC data
     confirms for MAC data requests.
     */
    uint16_t macAckFailures;
    /*!
     Total number of MAC data request failures, other
        than channel access
     failure or MAC ACK failures.
     */
    uint16_t otherDataRequestFailures;
    /*! Total number of sync loss failures received for
       sleepy devices. */
    uint16_t syncLossIndications;
    /*! Total number of RX Decrypt failures. */
    uint16_t rxDecryptFailures;
    /*! Total number of TX Encrypt failures. */
    uint16_t txEncryptFailures;
    /*! Total number of resets. */
    uint16_t resetCount;
    /*!
     Assert reason for the last reset -  0 - no reason, 2
         - HAL/ICALL,
     3 - MAC, 4 - TIRTOS
     */
    uint16_t lastResetReason;
    /*! Amount of time taken for node to join */
    uint16_t joinTime;
    /*! Delay between sending a packet and receiving an
       ack */
    uint16_t interimDelay;
} Smsgs_msgStatsField_t;
```

```
/*!
 Message Statistics Field
 */
typedef struct _Smsgs_configsettingsfield_t
{
    /*!
     Reporting Interval - in millseconds, how often to
        report, 0
     means reporting is off
     */
    uint32_t reportingInterval;
    /*!
     Polling Interval - in millseconds (32 bits) - If the
         sensor device is
     a sleep device, this states how often the device
        polls its parent for
     data. This field is 0 if the device doesn't sleep.
     */
    uint32_t pollingInterval;
} Smsgs_configSettingsField_t;

/*!
 Sensor Data message: sent from the sensor to the
    collector
 */
typedef struct _Smsgs_sensormsg_t
{
    /*! Command ID */
    Smsgs_cmdIds_t cmdId;
    /*! Extended Address */
    uint8_t extAddress[SMGS_SENSOR_EXTADDR_LEN];
    /*! Frame Control field - bit mask of
       Smsgs_dataFields */
    uint16_t frameControl;
    /*! Battery field */
    uint32_t battery;
    /*!
     Temp Sensor field - valid only if
        Smsgs_dataFields_tempSensor
     is set in frameControl.
```

```
     */
    Smsgs_tempSensorField_t tempSensor;
    /*!
     Message Statistics field - valid only if
        Smsgs_dataFields_msgStats
     is set in frameControl.
     */
    Smsgs_msgStatsField_t msgStats;
    /*!
     Configuration Settings field - valid only if
     Smsgs_dataFields_configSettings is set in
        frameControl.
     */
    Smsgs_configSettingsField_t configSettings;

} Smsgs_sensorMsg_t;



/**********
 Structures - Building blocks for the change url(over-the
    -air sensor messages)
 *********/

/*!
 Change URL message: sent from the collector to the
    sensor
 */
typedef struct _Smsgs_changebleurl_t
{
    /*! Command ID - 1 byte */
    Smsgs_cmdIds_t cmdId;
    /*! New URL - 10 bytes */
    uint8_t url[10];
} Smsgs_changeBleUrl_t;

/*!
 Change URL Response message: sent from the collector to
    the sensor
 */
typedef struct _Smsgs_changebleurlrsp_t
```

```
{
    /*! Command ID - 1 byte */
    Smsgs_cmdIds_t cmdId;
    /*! New URL - 10 bytes */
    uint8_t url[10];
} Smsgs_changeBleUrlRsp_t;



/*********
 Structures - Building blocks for the bikeParking control
     (over-the-air sensor messages)
 *********/


/*!
  Bike Parking State request message: sent from the
     collector to the sensor
 */
typedef struct _Smsgs_bikeparkingstatereq_t
{
    /*! Command ID - 1 byte */
    Smsgs_cmdIds_t cmdId;
} Smsgs_bikeParkingStateReqMsg_t;

/*!
 Bike Select request message: sent from the collector to
    the sensor
 */
typedef struct _Smsgs_bikeselectedreq_t
{
    /*! Command ID - 1 byte */
    Smsgs_cmdIds_t cmdId;
    /*! Bike Number - 1 byte*/
    uint8_t bikeNumber;
} Smsgs_bikeSelectReqMsg_t;

/*!
 Bike parking State or Bike select response message: sent
     from the sensor to the collector
 */
```

```
typedef struct _Smsgs_parkingstatersp_t
{
    /*! Command ID - 1 byte */
    Smsgs_cmdIds_t cmdId;
    /*! Bike Parking State - each bit represents a bike -
        4 bytes*/
    uint32_t parkingState;
} Smsgs_parkingStateRspMsg_t;

/*
 * @}
*/

#ifdef __cplusplus
}
#endif

#endif /* SMGSS_H */
```

A.2. Mensajes Protocol Buffers

Listado A.2: Archivo appsrv.h

```
syntax = "proto2";

import "smsgs.proto";
import "api_mac.proto";
import "llc.proto";
import "cllc.proto";
import "csf.proto";

/* MT System ID for TIMAC APP Server Interface Protobuf
   sub-system ID */
enum timacAppSrvSysId
{
  RPC_SYS_PB_TIMAC_APPSRV = 10;
}

/* Command IDs - each of these are associated with
   Request,
 * Responses, Indications, and Confirm messages
```

```
 */
enum appsrv_CmdId
{
  APPSRV_DEVICE_JOINED_IND = 1;
  APPSRV_DEVICE_LEFT_IND = 2;
  APPSRV_NWK_INFO_IND = 3;
  APPSRV_GET_NWK_INFO_REQ = 4;
  APPSRV_GET_NWK_INFO_RSP = 5;
  APPSRV_GET_NWK_INFO_CNF = 6;
  APPSRV_GET_DEVICE_ARRAY_REQ = 7;
  APPSRV_GET_DEVICE_ARRAY_CNF = 8;
  APPSRV_DEVICE_NOTACTIVE_UPDATE_IND = 9;
  APPSRV_DEVICE_DATA_RX_IND = 10;
  APPSRV_COLLECTOR_STATE_CNG_IND = 11;
  APPSRV_SET_JOIN_PERMIT_REQ = 12;
  APPSRV_SET_JOIN_PERMIT_CNF = 13;
  APPSRV_TX_DATA_REQ = 14;
  APPSRV_TX_DATA_CNF = 15;
  APPSRV_GET_COLLECTOR_STATS_REQ = 16;
  APPSRV_GET_COLLECTOR_STATS_RSP = 17;
}

enum nwkMode
{
    BEACON_ENABLED = 1;
    NON_BEACON = 2;
    FREQUENCY_HOPPING = 3;
}

// APPSRV_TX_DATA_REQ
message appsrv_txDataReq
{
    required appsrv_CmdId cmdId = 1 [default =
       APPSRV_TX_DATA_REQ];  // don't change this field
    required Smsgs_cmdIds msgId = 2;
    required ApiMac_deviceDescriptor devDescriptor = 3;
    optional Smsgs_configReqMsg configReqMsg = 4;
    optional Smsgs_toggleLedReqMsg toggleLedReq = 5;
    optional Smsgs_changeUrlReqMsg changeUrlReq = 6;
    optional Smsgs_bikeSelectReqMsg bikeSelectReq = 7;
    optional Smsgs_bikeParkingStateReqMsg parkingStateReq
```

```
        = 8;
}


//APPSRV_TX_DATA_CNF
message appsrv_txDataCnf
{
  required appsrv_CmdId cmdId = 1 [default =
     APPSRV_TX_DATA_CNF]; // don't change this field
  required int32 status = 2;
}

// APPSRV_SET_JOIN_PERMIT_REQ
message appsrv_setJoinPermitReq
{
    required appsrv_CmdId cmdId = 1 [default =
       APPSRV_SET_JOIN_PERMIT_REQ];  // don't change this
        field
 //    duration - duration for join permit to be turned on
     in
 //               milliseconds.
 //               0 sets it Off, 0xFFFFFFFF sets it ON
    indefinitely
 //               Any other non zero value sets it on for
    that duration */
    required int32 duration = 2;

}
// APPSRV_SET_JOIN_PERMIT_CNF
message appsrv_setJoinPermitCnf
{
  // provides the result of processing of the
  // APPSRV_SET_JOIN_PERMIT_REQ message
  required int32 status = 1;
}

// APPSRV_DEVICE_JOINED_IND
message appsrv_deviceUpdateInd
{
    required ApiMac_deviceDescriptor devDescriptor = 1;
    required ApiMac_capabilityInfo devCapInfo = 2;
```

```
}

// APPSRV_DEVICE_NOTACTIVE_UPDATE_IND
message appsrv_deviceNotActiveUpdateInd
{
    required ApiMac_deviceDescriptor devDescriptor = 1;
    required uint32 timeout = 2;
}


// APPSRV_DEVICE_DATA_RX_IND
message appsrv_deviceDataRxInd
{
    required ApiMac_sAddr srcAddr = 1;
    required sint32 rssi = 2;
    optional Smsgs_sensorMsg sDataMsg = 4;
    optional Smsgs_configRspMsg sConfigMsg = 5;
    optional Smsgs_changeUrlRspMsg sUrlMsg = 6;
    optional Smsgs_parkingStateRspMsg sParkingStateMsg =
      7;
}

message appsrv_collectorStateCngUpdateInd
{
    required uint32 state = 1;
}

// sub-message for the network parameters
// used by appsrv_nwkInfoUpdateInd
// and appsrv_getNwkInfoCnf
message appsrv_nwkParam
{
    required Llc_netInfo nwkInfo = 1;
    required int32 securityEnabled = 2;
    required nwkMode networkMode = 3;
    required Cllc_states state = 4;
}

message appsrv_nwkInfoUpdateInd
{
   required appsrv_nwkParam nwkinfo = 1;
```

```
}

// APPSRV_GET_NWK_INFO_REQ
message appsrv_getNwkInfoReq
{
  // APPSRV_GET_NWK_INFO_REQ - command ID used to
     identify this message
  // Get network Information Request - This API is called
      by the appClient application to retrieve
  // the network information
  // Returns appsrv_getNwkInfoRsp_t

  required appsrv_CmdId cmdId = 1 [default =
     APPSRV_GET_NWK_INFO_REQ];  // don't change this
     field
}

// APPSRV_GET_NWK_INFO_CNF
message appsrv_getNwkInfoCnf
{
  // APPSRV_GET_NWK_INFO_REQ - command ID used to
     identify this message
  // Get network Information Request - This API is called
      by the appClient application to retrieve
  // the network information

    required appsrv_CmdId cmdId = 1 [default =
       APPSRV_GET_NWK_INFO_CNF];  // don't change this
       field
    required uint32 status = 2;
    optional appsrv_nwkParam nwkinfo = 3;

}

// APPSRV_GET_DEVICE_ARRAY_REQ
message appsrv_getDeviceArrayReq
{
  // APPSRV_GET_DEVICE_ARRAY_REQ - command ID used to
     identify this message
  // Get list of the connected devices Request
  // This API is called by the appClient application to
```

```
      retrieve
  // the list of currently connected devices
  // Returns appsrv_getNwkInfoCnf

  required appsrv_CmdId cmdId = 1 [default =
     APPSRV_GET_DEVICE_ARRAY_REQ];  // don't change this
     field
}

// APPSRV_GET_DEVICE_ARRAY_CNF
message appsrv_getDeviceArrayCnf
{
    required appsrv_CmdId cmdId = 1 [default =
       APPSRV_GET_DEVICE_ARRAY_CNF];  // don't change
       this field
    required uint32 status = 2;
    repeated Csf_deviceInformation devInfo = 3;
}

message appsrv_getCollectorStatsReq
{
  required appsrv_CmdId cmdId = 1 [default =
     APPSRV_GET_COLLECTOR_STATS_REQ];
}

message appsrv_getCollectorStatsRsp
{
  required appsrv_CmdId cmdId = 1 [default =
     APPSRV_GET_COLLECTOR_STATS_RSP];
  /*!
  Total number of tracking request messages attempted
  */
  required uint32 trackingRequestAttempts = 2;
  /*!
  Total number of tracking request messages sent
  */
  required uint32 trackingReqRequestSent= 3;
  /*!
  Total number of tracking response messages received
  */
  required uint32 trackingResponseReceived= 4;
```

```
  /*!
  Total number of config request messages attempted
  */
  required uint32 configRequestAttempts= 5;
  /*!
  Total number of config request messages sent
  */
  required uint32 configReqRequestSent= 6;
  /*!
  Total number of config response messages received
  */
  required uint32 configResponseReceived= 7;
  /*!
  Total number of sensor messages received
  */
  required uint32 sensorMessagesReceived= 8;
  /*!
  Total number of failed messages because of channel
     access failure
  */
  required uint32 channelAccessFailures= 9;
  /*!
  Total number of failed messages because of ACKs not
     received
  */
  required uint32 ackFailures= 10;
  /*!
  Total number of failed transmit messages that are not
     channel access
  failure and not ACK failures
  */
  required uint32 otherTxFailures= 11;
  /*! Total number of RX Decrypt failures. */
  required uint32 rxDecryptFailures= 12;
  /*! Total number of TX Encrypt failures. */
  required uint32 txEncryptFailures= 13;
  /* Total Transaction Expired Count */
  required uint32 txTransactionExpired= 14;
  /* Total transaction Overflow error */
  required uint32 txTransactionOverflow= 15;
}
```

Bibliografía

[1] Roy Want, Bill N. Schilit, and Scott Jenson. Enabling the internet of things. *Computer*, 2015.

[2] Dmitry Namiot and Manfred Sneps-Sneppe. The physical web in smart cities. *Advances in Wireless and Optical Communications*, 2015.

[3] Yue Liu, Ju Yang, and Mingjun Liu. Recognition of QR code with mobile phones. In *Control and Decision Conference*.

[4] Gerd Kortuem, Fahim Kawsar, and Vasughi Sundramoorthy. Smart objects as building blocks for the internet of things. *IEEE Internet Computing*, 2009.

[5] Luca Mainetti, Luigi Patrono, and Antonio Vilei. Evolution of wireless sensor networks towards the internet of things: A survey. *Telecommunications and Computer Networks*, 2011.

[6] Michele Zorzi, Alexander Gluhak, and Sebastian Lange. From today's intranet of things to a future internet of things: a wireless and mobility-related view. *IEEE Wireless Communicationss*, 2010.

[7] Carles Gomez and Josep Paradells. Wireless home automation networks: A survey of architectures and technologies. *IEEE Communications Magazine*, 2010.

[8] Zigbee Alliance. Zigbee home automation public apllication profile. Technical report, Zigbee Alliance, 2007.

[9] Zigbee Alliance. The zigbee rf4ce standard. Technical report, Zigbee Alliance, 2009.

[10] Z-Wave. Z-wave protocol overview. Technical report, Z-Wave, 2007.

[11] Ana Belén García Hernando, José Fernán Martínez Ortega, Juan Manuel López Navarro, Aggeliki Prayati, and Luis Redondo López. *Problem Solving for Wireless Sensor Networks*. Springer, 2008.

[12] Texas instruments. Ti 15.4-stack - simplelink? cc13x0 ti 15.4-stack user's guide 2.04.00.00 documentation, jan 2018.

[13] Mahmoud Shuker Mahmoud and Auday AH Mohamad. A study of efficient power consumption wireless communication techniques/modules for internet of things (iot) applications. *Advances in Internet of Things*, 6(02):19, 2016.

[14] Keysight. *Keysight Technologies 34410A and 34411A Multimeters. Data sheet.*