Demostrador de Internet of Things con la tecnología IEEE 802.15.4e utilizando la plataforma OpenMote y el sistema operativo OpenWSN y theThings.io

Francisco Mudarra Capdepont

Máster Universitario en Ingeniería de Telecomunicación
Área: Telemática

Jose López Vicario

Junio 2018

FICHA DEL TRABAJO FINAL

Título del trabajo:	*Demostrador de Internet of Things con la tecnología IEEE 802.15.4e utilizando la plataforma OpenMote y el sistema operativo OpenWSN y theThings.io.*
Nombre del autor:	*Francisco Mudarra Capdepont*
Nombre del consultor/a:	*Jose López Vicario*
Nombre del PRA:	*Xavier Vilajosana Guillen*
Fecha de entrega:	06/2018
Titulación:	*Máster Universitario en Ingeniería de Telecomunicación*
Área del Trabajo Final:	*Telemática*
Idioma del trabajo:	*Español*
Palabras clave	*Internet Of Things (IoT), OpenWSN, OpenMote*

Resumen del Trabajo

En los últimos años, Internet de las cosas (IoT) se ha convertido en una de las tendencias más relevantes dentro del mundo de las telecomunicaciones.

El principal objetivo de este TFM consiste en el desarrollo de un sistema que capaz de demostrar el uso del IoT utilizando para ello la tecnología IEEE 802.15.4e, combinado con el hardware de la plataforma OpenMote y el sistema operativo OpenWSN y almacenando los datos en la plataforma theThings.io.

Para ello se desarrollará una aplicación en Python que funcionará bajo el entorno de OpenWSN, que será capaz de recibir las lecturas de un sensor de temperatura y otro de luminosidad. Estos datos serán procesados y tratados en una Raspberry Pi y se presentará una comparativa entre los modelos de Fog computing y de Cloud computing.

Se presentarán los resultados obtenidos demostrando así la fiabilidad del sistema y la utilidad del mismo, quedando probada la demostración de un sistema IoT.

Abstract (in English, 250 words or less):

In recent years, the Internet of Things (IoT) has become one of the most significant trends in the world of telecommunications.

The main objective of this TFM (project) is the development of a system that can demonstrate the use of the IoT using the IEEE 802.15.4e technology, combined with the OpenMote platform hardware and the OpenWSN operating system, and storing the data in the theThings.io platform.

For this, an application in Python will be developed that will work under the OpenWSN environment, which will be able to receive the readings of both a temperature and a luminosity sensor. These data will be processed and treated in a Raspberry Pi and a comparison between the Fog computing and Cloud computing models will be presented.

The results obtained will be presented, demonstrating the reliability and usefulness of the system, resulting in the demonstration of an 'IoT system'.

Índice

Índice de figuras

Índice tablas

1. Introducción

1.1 Contexto y justificación del Trabajo

Este proyecto nace por el actual auge del Internet de las cosas (IoT – Internet of Things). En los últimos años, Internet de las cosas se ha convertido en una de las tendencias más relevantes dentro del mundo de las telecomunicaciones. Es por lo tanto, una de las motivaciones de este TFM, tanto la documentación de esta nueva tecnología, la investigación de todas las alternativas posibles para los sistemas IoT y el desarrollo de una plataforma que permita la demostración de un sistema funcional IoT.

Con una demanda cada vez mayor de conexiones a internet, con mayores velocidades y de más dispositivos conectados a red, donde prácticamente cualquier dispositivo electrónico se pueda conectar a internet, es el marco ideal para el IoT.

Las posibilidades del IoT van desde la automatización de procesos, domótica, eficiencia energética o gestión de la información. La eficiencia energética y el ahorro energético es una de las principales preocupaciones ambientales actuales y en las que se hace especial hincapié y uno de los pilares básicos de la domótica. En este proyecto se utilizarán nodos OpenMote sobre el sistema OpenWSN, se utilizarán los protocolos IEEE 802.15.4e, HTTP y plataformas como Raspberry Pi y THETHINGS.IO.

1.2 Objetivos del Trabajo

El objetivo principal de este trabajo fin de máster es el desarrollo de una red IoT utilizando el estándar IEEE 802.15.4e sobre un sistema OpenWSN y con la plataforma THETHINGS.IO. Y el desarrollo HW de la misma red en la plataforma OpenMote y con una Raspberry Pi.

Dentro del objetivo principal se puede desglosar los siguientes objetivos:

- Familiarizarse y aprender a utilizar la plataforma OpenWSN. Estudiar su funcionamiento y la puesta a punto del equipo para su correcta utilización.

- Estudio y análisis del estándar IEEE 802.15.4e. Así como de los estándares más relevantes para poder realizar una comparativa entre ellos.

- Estudio de los diferentes trabajar relacionados y de los nuevos conceptos de computación: Cloud computing, edge computing y fog computing.

- Estudio y documentación de las plataformas OpenWSN, OpenMote y Raspberry.

- Desarrollar un software que permita controlar la red implementada, adquisición de datos, procesado de los datos, representación de los datos, almacenamientos de los datos y control de los mismos con los lenguajes Python y C.

- Desarrollo de un software que permita la transmisión de datos con la plataforma de THETHINGS.IO y la representación de los mismos dentro de la plataforma.

- Elaboración de una memoria que sirva como presentación del trabajo realizado.

1.3 Enfoque y método seguido

Cómo propuestas para el desarrollo de un demostrador de IoT, se han propuesto varios tipos de redes que permitan demostrar el uso del IoT y sus posibles beneficios. Como punto de partida, se proponen tres posibles soluciones:

a) Aplicaciones de domótica que sirvan para el control de la vivienda y para mejorar el uso energético de la misma. Se propone una red formada por un sensor de luminosidad y otro de temperatura que irán reportando constantemente la cantidad de luz detectada y la temperatura medida. A su vez, esto podrá servir para abrir o cerrar persianas o para activar o desactivar la climatización según se desee. También podrá ser algo que el usuario pueda controlar según lo necesite.

b) Un control de corriente que permita activar o desactivar a petición del usuario. Esta opción finalmente se descarta que no ofrece un análisis de datos demasiado sustancial.

c) Un control de consumo de gasolina de un automóvil, para que se pueda analizar si es rentable o no la compra de un nuevo vehículo. También se ha descartado esta opción ya que la instalación sería complicada y no se usaría el estándar IEEE 802.15.4e el cual es un requisito inicial de este TFM.

Finalmente se ha optado por modelar una red de domótica que incluirá un sensor de temperatura, un sensor de luminosidad y un actuador que permita el control del usuario de ciertos elementos. Todos los sensores se crearan de forma simulada dentro de la plataforma OpenWSN. Con esto conseguiremos, una red funcional, con suficientes datos como para hacer un procesado de la información antes de enviar los datos a la nube y de tener un sistema que sea capaz de recibir

información y de enviar información a los sensores para que estos puedan ser modificados.

Es también importante resaltar que el trabajo las dos partes del trabajo. La primera de ellas donde se ha invertido especial esfuerzo, es la parte de documentación, de análisis del problema, de la investigación de cada uno de los aspectos tecnológicos que engloba el sistema y de los diferentes estándares que son utilizado.

Y en la segunda parte, el desarrollo del sistema y la parte de innovación del mismo. Este trabajo sobre la aplicación de IoT en domótica no es único, y ya existen diversos sistemas de control de domótica en el mercado.

Aunque ya se comentará con más en detalle, las principales ventajas de este trabajo respectos a otros, es la mejora de la simplicidad al usar la plataforma OpenWSN, la reducción del consumo y puesto que es código abierto cualquiera podrá continuar con el trabajo. Además se incluye un procesado previo para reducir el número de retransmisiones que se envían a la nube reduciendo así el ancho de banda requerido. También se desarrollará un software modular y configurable por usuario para que no sea necesario modificar el código.

1.4 Planificación del Trabajo

Se presenta a continuación el diagrama de Gantt con la planificación temporal del TFM donde se recogen los hitos más importantes y las fechas clave de la programación.

La planificación inicial se ha visto afectada debido a la necesidad de familiarizarse con las múltiples disciplinas que implica el desarrollo del TFM. El estudio de los estándares en los que está basado el sistema, protocolos y plataformas han requerido un esfuerzo especial que ha provocado el retraso del proyecto.

1	Kick-off TFM	0 días	mié 28/02/18	mié 28/02/18
2	Elección temática TFM	1 día	mié 28/02/18	mié 28/02/18
3	Análisis e investigación	7 días	mié 28/02/18	jue 08/03/18
4	Entender OpenWSN y puesta a punto del equipo	30 días	vie 09/03/18	jue 19/04/18
5	**PEC 1**	**6 días**	**mié 28/02/18**	**mié 07/03/18**
6	Evaluación del trabajo	6 días	**mié 28/02/18**	**mié 07/03/18**
7	Planificación	6 días	**mié 28/02/18**	**mié 07/03/18**
8	Entrega	0 días	**mié 07/03/18**	**mié 07/03/18**
9	**PEC 2**	**31 días**	**mié 07/03/18**	**mié 18/04/18**
10	Redacción 60% memoria	31 días	**mié 07/03/18**	**mié 18/04/18**
11	Configuración Windows	5 días	**mié 07/03/18**	**mar 13/03/18**
12	Instalación OpenVPN	2 días	**mié 14/03/18**	jue 15/03/18
13	OpenWSN	5 días	vie 16/03/18	jue 22/03/18
14	CoAP	2 días	vie 23/03/18	lun 26/03/18
15	THETHING.IO	5 días	mar 27/03/18	lun 02/04/18
16	Configuración Raspberry	5 días	mar 03/04/18	lun 09/04/18
17	Configuración firmware mote simulado	3 días	mar 10/04/18	jue 12/04/18
18	Primeras pruebas del sistema completo	3 días	vie 13/04/18	mar 17/04/18
19	Entrega	0 días	**mié 18/04/18**	**mié 18/04/18**
20	**PEC 3**	**25 días**	**jue 19/04/18**	**mié 23/05/18**
21	Redacción 100% memoria	25 días	**jue 19/04/18**	**mié 23/05/18**
22	Diseño y configuración almacenamiento de datosen local	7 días	**jue 19/04/18**	**vie 27/04/18**
23	Desarrollo de las aplicaciones y formato	10 días	**lun 30/04/18**	vie 11/05/18
24	Presentación de la información en THETHING.IO	5 días	lun 14/05/18	vie 18/05/18
25	Prerado para entregar	2 días	lun 21/05/18	mar 22/05/18
26	Entrega	0 días	**mié 23/05/18**	**mié 23/05/18**
27	**Preparación de la memoria**	**68 días**	**jue 08/03/18**	**dom 10/06/18**
28	Redacción memoría TFM	55 días	jue 08/03/18	**mié 23/05/18**
29	Revisión memoría TFM	13 días	**jue 24/05/18**	**dom 10/06/18**
30	Entrega memoría TFM	0 días	dom 10/06/18	**dom 10/06/18**
31	**Preparación de la presentación**	**11 días**	**vie 01/06/18**	**lun 18/06/18**
32	Preparación presentación	11 días	vie 01/06/18	vie 15/06/18
33	Entrega presentación	0 días	lun 18/06/18	lun 18/06/18
34	Proceso tribunal	6 días	lun 18/06/18	dom 24/06/18
35	**Resolución final**	**1 día**	dom 24/06/18	dom 24/06/18
36	Resolución final	**0 días**	**dom 24/06/18**	**dom 24/06/18**

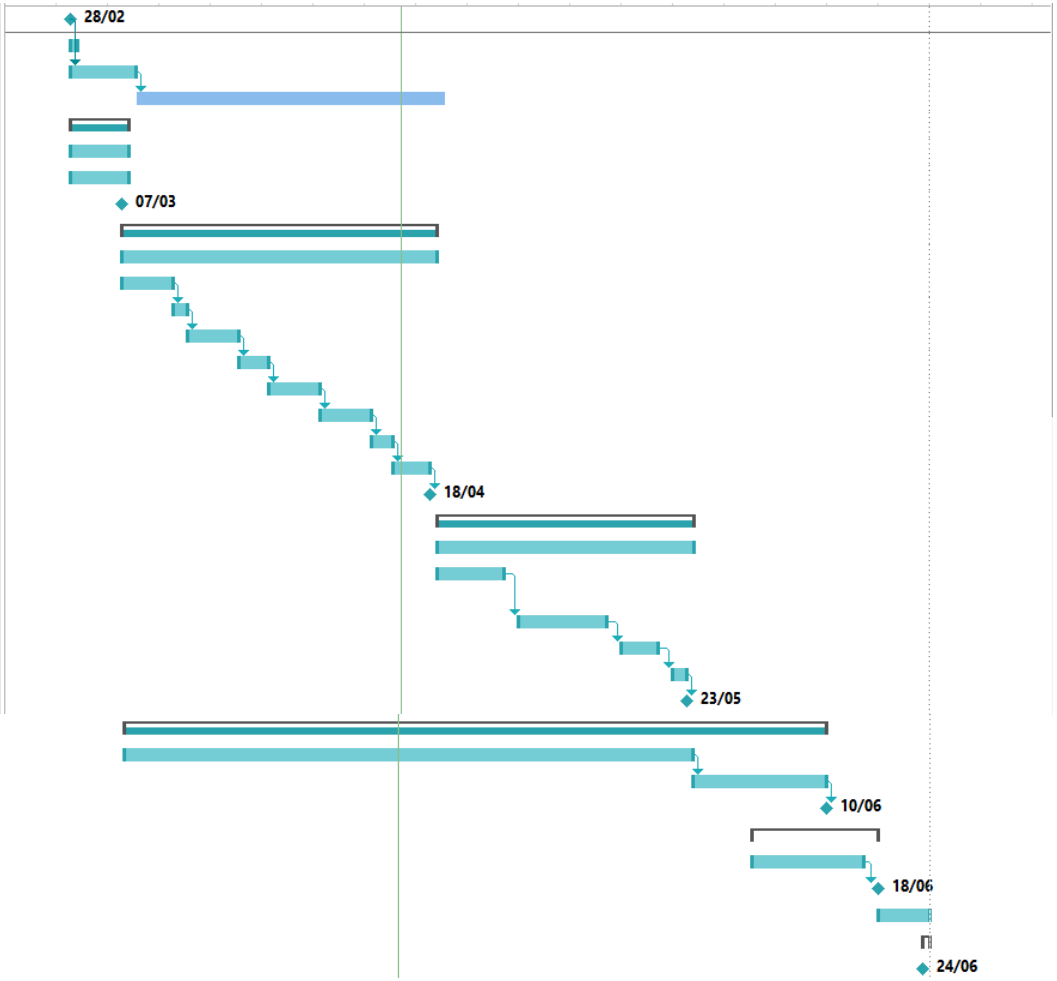


Ilustración 1: Diagrama de Gantt

1.5 Breve sumario de productos obtenidos

Los principales productos obtenidos en este TFM son:

- Modificación del firmware de OpenWSN para la emulación de sensores que simularán la emisión de datos. Para ello se enviarán datos de forma pseudoaleatoria, y en función de la hora actual, para simular un funcionamiento tipo. La función se desarrollará en C y se cargará como firmware dentro de OpenWSN.

- Se dota al firmware también de capacidad de adquirir datos desde el nodo principal lo que permitirá el control de los sensores pudiéndolos activar o desactivar.

- Una aplicación en Python, instalada en la Raspberry PI que se encargará de comunicarse con los diferentes sensores utilizando el protocolo de comunicación CoAP.

- Procesado de la información previa a la emisión de datos a la nube para reducir así el número de retransmisiones y del ancho de banda requerido.

- Módulo encargado de la comunicación la plataforma THETHING.IO y su posterior representación en la plataforma.

1.6 Breve descripción de los otros capítulos de la memoria

A continuación, se detalla un breve resumen de los capítulos de los que constará la memoria:

- **Capítulo 2:** Contexto del trabajo. Se detallarán los aspectos tecnológicos en los que se basa este trabajo. Introducción y detallado del Internet de las Cosas (IoT). Descripción del estándar IEEE 802.15.4e. Descripción de la plataforma OpenWSN, así como su configuración y software requerido. Descripción de los estándares de comunicación como HTTP.

- **Capítulo 3:** Descripción de sistema. Se detallarán como se ha desarrollado el sistema, que elemento lo forman, descripción del software desarrollado, diseño del hardware y sus características.

- **Capítulo 4:** Verificación del sistema. En este capítulo se detallarán las pruebas realizadas, comportamiento y la funcionalidad del mismo.

- **Capítulo 5:** Conclusiones y líneas futuras. Se resaltarán los aspectos más importantes del proyecto, que conclusiones pueden ser tomadas y el aporte del mismo y por último que líneas futuras puede tomar el proyecto (continuación del desarrollo, comercialización, etc).

2. Contexto de trabajo

2.1 Internet de las Cosas (IoT)

2.1.1 Introducción

El objetivo principal del IoT nace de la necesidad de conectar cualquier dispositivo a internet y con ello crear una nueva infinidad de aplicaciones que sin duda se ha convertido en uno de los temas más relevantes del sector y con más proyección de los últimos años.

Aunque no existe una definición universal para el Internet de las Cosas, es un concepto que se refiere a la interconexión digital de objetos cotidianos a internet [3].

El termino IoT, fue usado por primera vez por el profesor Kevin Ashton en 1999. Kevin Ashton creía que el RFID ("Radio Frequency Identificacion") era un prerrequisito para el Internet de las Cosas [4].

Es importante resaltar que la conexión de dispositivos electrónicos a internet no es concepto nuevo y que ya se utilizaba anteriormente. Ya en la década de los 80s y los 90s se utilizaban tecnologías como el M2M (Machine-to-machine) a través de las redes GSM. En 1980, en la universidad de Carnegie Melon ya se conectó una máquina de Coca-Cola a internet para su monitorización. Otros dispositivos como datafonos, son un ejemplo de dispositivos conectados a la red [5].

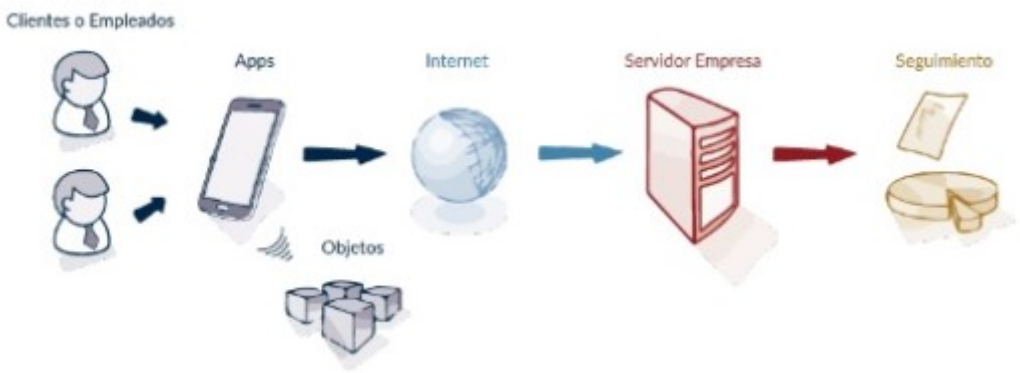


Ilustración 2: Definición IoT [2]

2.1.2 Arquitectura de los sistemas IoT

La arquitectura de los sistemas IoT se pueden agrupar en 4 capas bien diferenciadas. La capa de sensor, capa de redes y "Gateways", capa de gestión de sensores y capa de aplicaciones.

- **Capa de sensores**

La capa más baja está compuesta por objetos inteligentes integrados con sensores. Los sensores permiten la interconexión del mundo físico y digital, permitiendo que la información en tiempo real sea recolectada y procesada. La reducción del hardware ha permitido la producción de potentes sensores en formas mucho más pequeñas que se integran en objetos en el mundo físico.

Hay varios tipos de sensores para diferentes propósitos. Los sensores tienen la capacidad de tomar medidas tales como temperatura, calidad del aire, movimiento y electricidad. Un sensor puede medir la propiedad física y convertirla en señal que pueda ser entendida por un instrumento.

La mayoría de los sensores requieren conectividad a los agregadores de sensores (gateways). Esto puede ser en forma de red de área local (LAN) como conexiones Ethernet y Wi-Fi o red de área personal (PAN) como ZigBee, Bluetooth y Ultra-Wideband (UWB). Para los sensores que no requieren conectividad con los agregadores de sensores, su conectividad con los servidores / aplicaciones de back-end puede proporcionarse mediante Wide Area Network (WAN), como GSM, GPRS y LTE. Los sensores que usan conectividad de baja potencia y baja velocidad de datos, generalmente forman redes conocidas comúnmente como redes inalámbricas de sensores (WSN). Las WSN están ganando popularidad, ya que pueden acomodar muchos más nodos de sensores a la vez que conservan la duración adecuada de la batería y cubren áreas grandes [6].

- **Capa de redes y gateway**

Estos diminutos sensores producirán un volumen masivo de datos y esto requiere una infraestructura de red cableada o inalámbrica robusta y de alto rendimiento como medio de transporte. Las redes actuales, a menudo vinculadas con protocolos muy diferentes, se han utilizado para admitir redes de máquina a máquina (M2M) y sus aplicaciones.

Con la demanda necesaria para atender una gama más amplia de servicios IOT y aplicaciones tales como servicios transaccionales de alta velocidad, aplicaciones sensibles al contexto, etc., se necesitan múltiples redes con diversas tecnologías y protocolos de acceso para trabajar entre sí en una configuración heterogénea. Estas redes pueden ser en forma de modelos privados, públicos o híbridos y están diseñadas para cumplir los requisitos de comunicación de latencia, ancho de banda o seguridad.

Una posible implementación podría consistir en una infraestructura de red convergente que resuelva la fragmentación mediante la integración de redes dispares en una única plataforma de red. La abstracción de capa de red convergente permite que varias organizaciones compartan y usen

la misma red de forma independiente para que su información se enrute sin comprometer sus requisitos de privacidad, seguridad y rendimiento [6].

- **Capa de gestión de servicio**

La gestión de servicio posibilita el procesamiento de la información a través de análisis, controles de seguridad, modelado de procesos y administración de dispositivos.

El análisis también se puede llevar a cabo en otras capas dentro de la arquitectura IOT.

La administración de datos es la capacidad de administrar el flujo de información de datos siempre de vital importancia la seguridad de los mismos en todo el sistema de IoT [6].

- **Capa de aplicación**

Existen diversas aplicaciones de sectores industriales que pueden aprovechar IoT. Las aplicaciones pueden ser específicas de un sector en particular, y otras aplicaciones pueden ser atractivas para múltiples sectores de la industria [6].

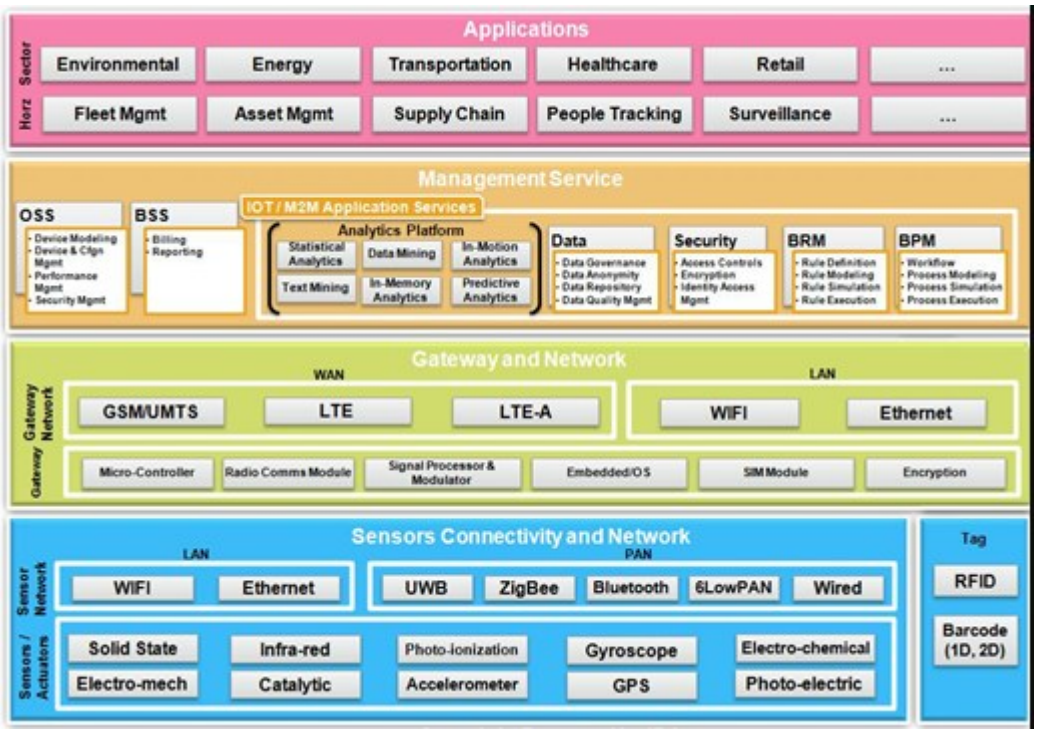


Ilustración 3: Arquitectura sistemas IoT [6]

2.1.3 Protocolos de los sistemas IoT

2.1.3.1 Introducción

Al igual que en todas las tecnologías emergentes, en sus comienzos se produce una especie de guerra entre diferentes fabricantes y entidades reguladoras. En el mundo del IoT existen diversos protocolos que luchan por convertirse en el estándar para IoT. A continuación se presenta un esquema con los protocolos más relevantes distribuidos según la capa OSI [8]:

- Infraestructura: (ex: 6LowPAN, IPv4/IPv6, RPL)
- Identificación: (ex: EPC, uCode, IPv6, URIs)
- Transporte: (ex: Wifi, Bluetooth, LPWAN)
- Descubrimiento: (ex: Physical Web, mDNS, DNS-SD)
- Protocolos de datos: (ex: MQTT, CoAP, AMQP, Websocket, Node, XMPP, RESP API)
- Gestión de dispositivos: (ex: TR-069, OMA-DM)
- Semántica: (ex: JSON-LD, Web Thing Model)
- Multi-capa de estructuras: (ex: Alljoyn, IoTivity, Weave, Homekit)

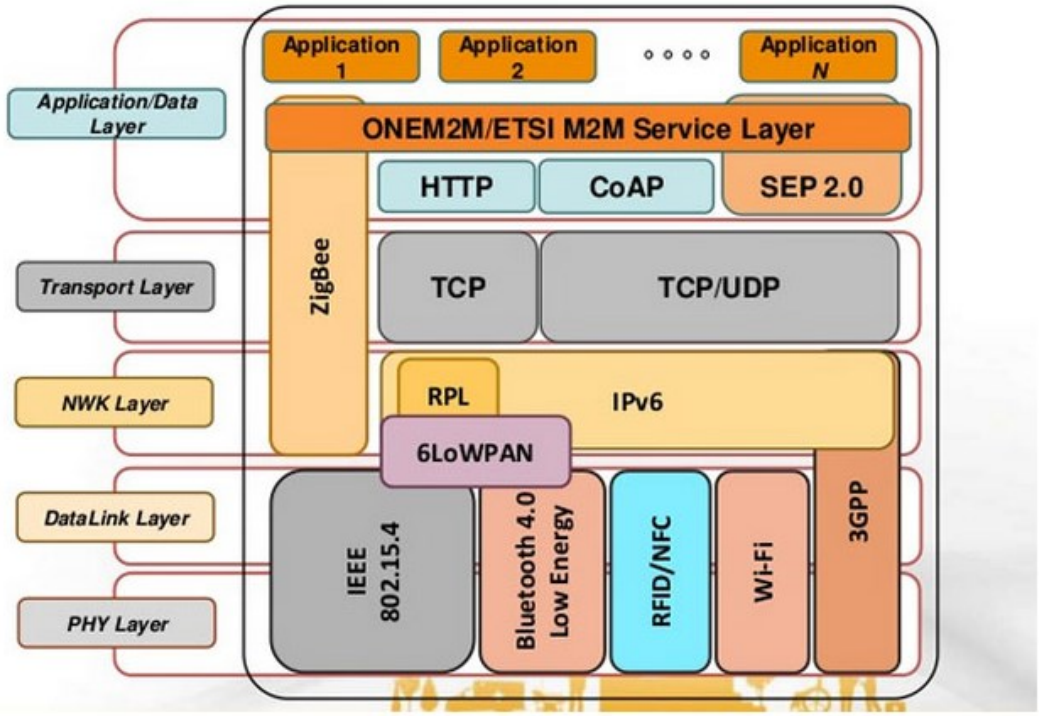


Ilustración 4: Protocolos IoT [8]

2.1.3.2 Protocolos de datos

Se detallaran los protocolos más relevantes del momento (a fecha de redacción del actual trabajo).

- MQTT
- SoAP
- CoAP
- XMPP
- REST

- MQTT

Se utiliza para dispositivos de campo con conexión celular o por satélite: Cada Kb de tráfico es valioso. Se utilizan comunicaciones bidireccionales en redes no confiables y para dispositivos alimentados por batería con bajo consumo de energía.
Los dispositivos pueden dormir, pero no el 95% del tiempo. Para ellos se utilizaría MQTT-S o CoAP [7]. No es un protocolo muy recomendado para el tipo de red que se propone en este TFM.

- MQTT-S

Muy parecido a MQTT pero para dispositivos que necesiten pasar más tiempo en estado de espera. Es posible escalar hasta diez veces más dispositivos que con MQTT. El cruce de direcciones NAT podría ser un problema en MQTT-S en comparación con MQTT, por lo que deben de ser direccionadas durante el proceso de diseño [7]. No es un protocolo muy recomendado para el tipo de red que se propone en este TFM.

- SoAP

SoAP (Simple Object Access Protocol), es un protocolo de intercambio de información basado en XML (lenguaje de marcos extensible utilizado para almacenar datos de forma legible) diseñado para Internet, se usa para codificar información de los requerimientos de los Web Services y responder a los mensajes antes de enviarlos a la red.

SOAP utiliza WSDL (Lenguaje de descripción Web Service) que es un formato XML que describe los servicios de red como un conjunto de puntos finales que operan los mensajes de petición / respuesta y UDDI (Integración universal de descripción y descubrimiento) que siendo una plataforma independiente, es una extensión del lenguaje XML que almacena y localiza aplicaciones Web Service. [21].

- REST

(Representational State Transfer), se basa en HTTP para intercambiar información y no necesita encapsulado extra para ello. Es más ligero y más sencillo de utilizar pero a la vez tiene algunas limitaciones. En vez de hacer peticiones encapsuladas en un “sobre SOAP” para solicitar un servicio para lo que es necesario el wsdl, en REST las peticiones se hacen mediante el protocolo HTTP con métodos GET, POST... sin necesidad de encapsularlo. Utiliza un lo camino de comunicación entre el dispositivo y la nube y prioriza el cruce de direcciones NAT [21].

- XMPP

Escalabilidad masiva en tiempo real para aproximadamente 100.000 nodos. Se usa también cuando los mensajes del tráfico son grandes y potencialmente complicados para cada dispositivo. También se usa cuando es necesaria una seguridad extra [7]. No es un protocolo muy recomendado para el tipo de red que se propone en este TFM.

- CoAP

Muy similar a MQTT-S pero con algunas mejoras respecto a ese protocolo. Con una arquitectura orientada a servicios web en vez de estar orientado a mensajes como MQTT. Es una plataforma de desarrollo libre a la que se puede acceder desde internet [7].

CoAP es, principalmente, un protocolo de “one-to-one” para transferir información de estado entre el cliente y el servidor. Si bien tiene soporte para la observación de recursos, CoAP es más adecuado para un modelo de transferencia de estado, no puramente basado en eventos. CoAP además proporciona soporte para la integración y el descubrimiento de contenido. CoAP envía y recibe paquetes UDP y está diseñado para solicitar y recibir información vía HTTP con el uso de clientes como GET, PUT, POST and DELETE [7]. Se adapta también al formato nodo-sensor con controladores de 8 bits y permite la utilización de redes 6loWPAN que fragmentan los paquetes IPv6 en pequeñas tramas de capa 2.

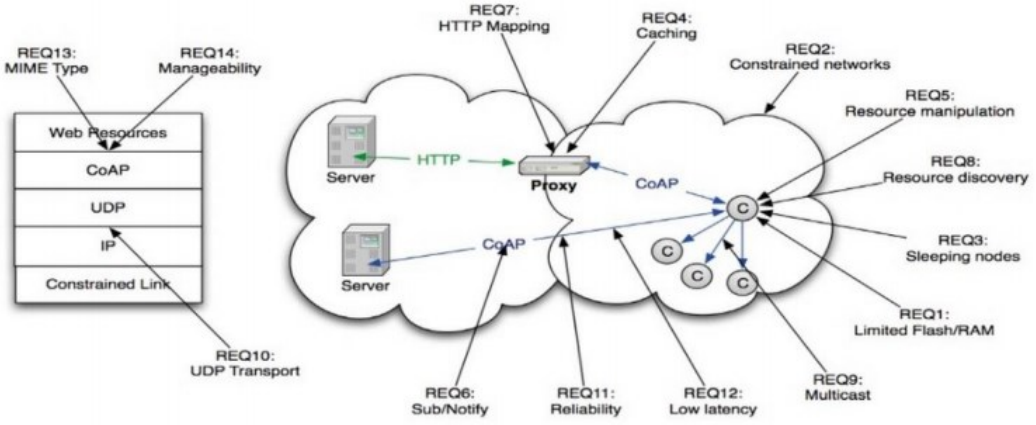


Ilustración 5: Capa y diseño CoAP [21]

CoAP es el protocolo de datos que se utilizará para la adquisición de datos entre los sensores. El uso del protocolo de transporte UDP reduce la sobrecarga provocadas por TCP. Es un protocolo sencillo y fácil de utilizar, donde la integración con HTTP hace que sea más intuitivo de utilizar. Es aceptado por OpenWSN algo que es fundamental para el sistema y dispone de una buena cantidad de librerías de acceso libre.

Se presenta a continuación, una tabla comparativa con las características más importantes de cada protocolo datos en IoT:

Protocolo	CoAP	XMPP	REST	MQTT
Transporte	UDP	TCP	TCP	TCP
Mensajería	Petición/Respuesta	Publicar/Suscribir Petición/Respuesta	Petición/Respuesta	Publicar/Suscribir Petición/Respuesta
2G, 3G, 4G Conveniencia (1000s nodos)	Excelente	Excelente	Excelente	Excelente
Conveniencia LLN (1000s nodos)	Excelente	Justa	Justa	Justa
Recursos de computación	10Ks RAM/Flash	10Ks RAM/Flash	10Ks RAM/Flash	10Ks RAM/Flash
Adecuado	Útil en Field Area Networks	Control remoto del cliente sobre electrodomésticos	Perfiles de bajo consumo	Extender la mensajería empresarial a las aplicaciones de IoT

Tabla 1: Comparativa protocolo de datos IoT [20]

2.1.3.3 Protocolos de comunicación

Algunos de los protocolos de comunicación más usados en el ámbito del IoT son los siguientes:

- IEEE 802.15.4 y IEEE 802.15.4e: Son un estándar que definen el acceso a nivel físico y de control de acceso para redes inalámbricas de área personal (PAN). Se usa principalmente para redes con tasas de transmisión bajas y de bajo consumo lo que lo hace ideal para el tipo de red que se pretende implementar en este TFM. Es el protocolo que utiliza OpenWSN, se comentará más en detalla en el próximo apartado.

- ZigBee: Es un protocolo de alto nivel de comunicación inalámbrica para su utilización con radiodifusión digital de bajo consumo, basada en el estándar IEEE 802.15.4 de redes

inalámbricas de área personal (wireless personal area network, WPAN). Su objetivo son las aplicaciones que requieren comunicaciones seguras con baja tasa de envío de datos y maximización de la vida útil de sus baterías. Opera en los 2.4Ghz y su ancho de banda es de hasta 250kbps.

- LoRaWan: Es una especificación para redes de baja potencia y área amplia, LPWAN (Low Power Wide Area Network), diseñada específicamente para dispositivos de bajo consumo. LoRaWAN se caracteriza por su uso en el Internet de las Cosas, y para conexiones: bidireccionales seguras, bajo consumo de energía, largo alcance de comunicación, bajas velocidades de datos, baja frecuencia de transmisión, movilidad y servicios de localización.

- SigFox: Al igual que los protocolos comentados anteriormente SigFox también es un alternativa como protocolo para IoT, donde se previa el bajo consumo. Se utilizan mensajes de 12 bytes yes valido para redes de hasta 50km. Una de sus principales ventajas es que persigue la compatibilidad con los principales fabricantes del mercado.

- Redes móviles: Un utilizar la propia red de telefonía móvil existente ya sea la red de GSM, UMTS o LTE. La principal ventaja es poder utilizar una red que ya está en servicio. Como principal desventaja encontramos el coste del uso de la red y que no está orientada a bajo consumo.

Se presenta a continuación una comparativa entre los diferentes protocolos:

<table>
<tr><th>Tecnología</th><th>LTE</th><th>LoRaWAN</th><th>SigFox</th><th>WiFI</th><th>ZigBee</th></tr>
<tr><td>Bandas</td><td>Banda LTE</td><td>Sub-bandas regionales</td><td>Sub-bandas regionales</td><td>2.4 – 5.8Ghz</td><td>2.4Ghz</td></tr>
<tr><td>DL</td><td rowspan="2">20Mhz</td><td rowspan="2">125Khz – 500 Khz</td><td>100Hz</td><td rowspan="2">20, 21, 22, 23, 40, 80, 160 MHz</td><td rowspan="2">600 kHz, 1.2 Mhz</td></tr>
<tr><td>UL</td><td>200Hz</td></tr>
<tr><td>Acceso UL</td><td>OFDMA</td><td rowspan="2">CSS</td><td>UNB / FHSS</td><td rowspan="2">OFDM, DSSS, OFDMA</td><td rowspan="2">CSMA/CA</td></tr>
<tr><td>Acceso DL</td><td>SC-FDMA</td><td>UNB / FHSS</td></tr>
<tr><td>Modulación DL</td><td rowspan="2">QPSK 16QAM</td><td rowspan="2">LoRA, (G) FSK</td><td>GFSK</td><td rowspan="2">CCK, BPSK, QPSK, 16-QAM, 64-QAM</td><td rowspan="2">DSSS, BPSK, O-QPSK</td></tr>
<tr><td>Modulación UL</td><td>DBPSK</td></tr>
<tr><td>Máxima transferencia de datos</td><td>1 Mbps</td><td>0.3kbps – 50Kbps</td><td>100bps / 600bps</td><td>11-54-600Mbps</td><td>20 kbps, 40 kbps</td></tr>
<tr><td>Cobertura</td><td>141 dB</td><td>150-157dB</td><td>146 – 162 dB</td><td>200m</td><td>100m</td></tr>
</table>

Tabla 2: Comparativa protocolos de comunicación IoT

2.2. Estándar IEEE 802.15.4

2.2.1 Introducción

El estándar IEEE 802.15.4 es un estándar definido por la entidad IEEE para el uso de redes LR-WAN (Low-Rate Wireless Personal Area Network). Como su nombre indica son redes generalmente dedicadas a redes con sensores donde la tasa de transmisiones será baja en comparación con otros tipos de redes. Se observaran transmisiones periódicas, no muy frecuentes y con tasas de envíos de datos bajas.

Por lo tanto, este estándar no persigue obtener grandes velocidades de datos si no que por el contrario perseguirá un uso eficiente y bajo de la energía para que en el caso en el que los dispositivos estén conectados con batería, el consumo sea muy limitado [12].

A continuación se muestra una tabla con las principales propiedades del estándar IEEEE 802.15.4:

Propiedad	Rango
Rango de transmisión de datos	868 MHz: 20kb/s; 915MHz: 40kb/s; 2.4 GHz: 250 kb/s
Alcance	10 – 20
Latencia	Debajo de los 15 ms.
Canales	868/915 MHz: 11 canales. 2.4 GHz: 16 canales
Bandas de frecuencia	Dos PHY: 868/915 MHz y 2.4 GHz
Direccionamiento	Cortos de 8 bits o 64 bits IEEE
Canal de acceso	CSMA-CA y rasurado CSMA-CA
Temperatura	El rango de temperatura industrial: -40º a +85ºC

Tabla 3: Propiedades del IEEE 802.15.4 [12]

El estándar IEEE 802.15.4 define la capa física y la capa de control de acceso al medio. Se espera que las redes se auto organicen y que se mantengan en funcionamiento por si mismas para reducir costes.

Son varias las topologías soportadas por el estándar IEEE 802.15.4. Entre otras, las topologías más importantes son las topologías en estrella (Star network) o las topologías peer-to-peer [12].

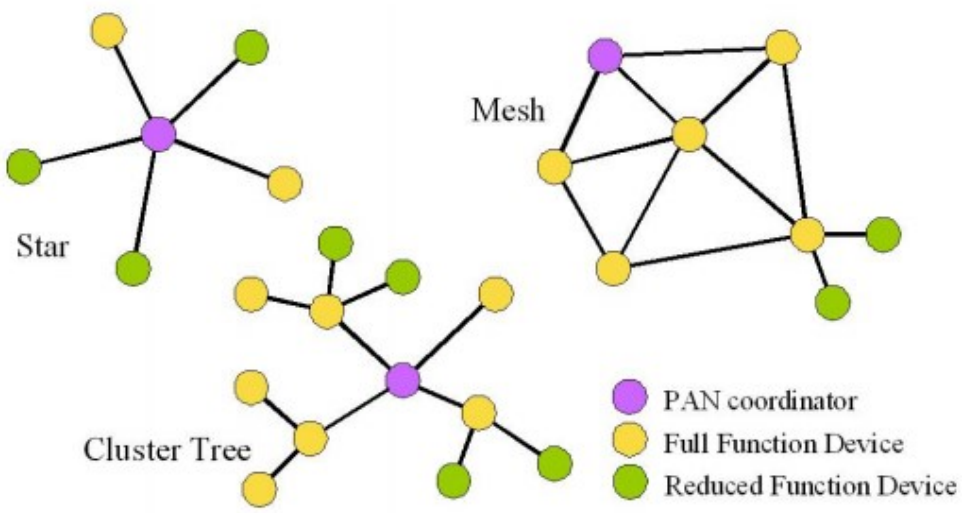


Ilustración 6: Topologías redes en estrella, peer-to-peer y en árbol [13]

Las topologías en estrella se suelen utilizar cuando los dispositivos a conectar requieren utilizar baja potencia. La topología peer-to-peer se usa para premiar la seguridad y por último la red en estructura árbol.

2.2.2 Capa de enlace de datos

La capa de enlace de datos se divide en dos sub capas:

- La subcapa de enlace de acceso a medios MAC (Medium Access Control)
- La subcapa de enlaces lógicos LCC (Logical link control)

Dentro del modelo OSI la capa de enlace de datos queda distribuida de la siguiente forma:

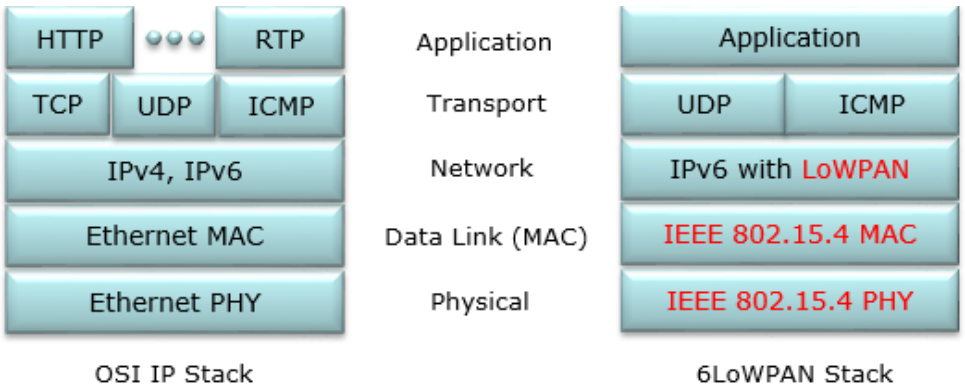


Ilustración 7: Relación IEEE 802.15.4 con el sistema OSI [22]

2.2.3 Formato de las tramas MAC

Lo que se persigue con las tramas MAC es que sea un protocolo simple, fácil de usar y que sea versátil para cualquier tipo de red que requiera su uso.

Las tramas MAC siguen el siguiente formato:

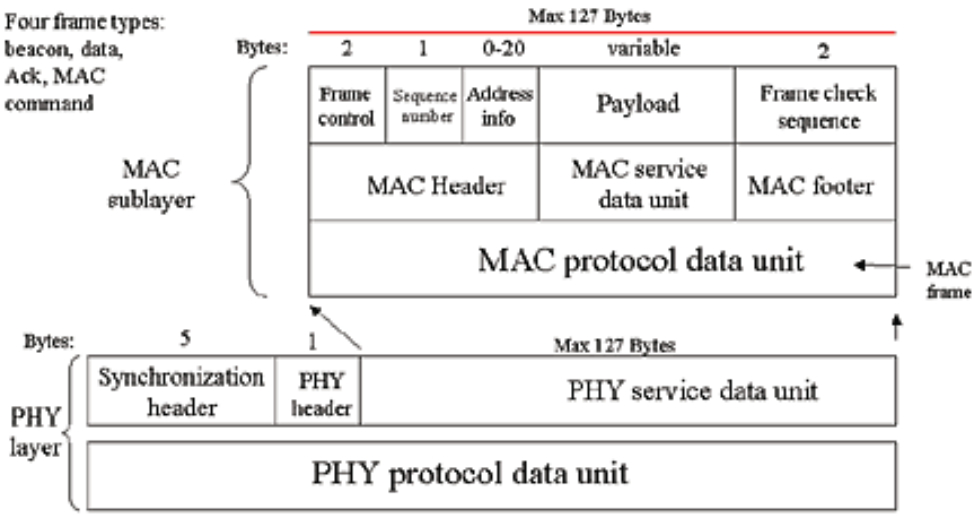


Ilustración 8: Trama MAC IEEE 802.15.4 [12]

Las tramas MAC están formadas por:

- Encabezado (MHR): Indica que se pretende transmitir y la dirección.
- Unidad de servicio de datos MAC (MSDU):
- Pie de MAC (MFR)

Dentro del campo unidad de servicio se define el campo Payload de longitud variable de hasta 127 bytes [12].

2.3. Estándar IEEE 802.15.4e

El IEEE 802.15.4e es un estándar definido por la entidad IEEE que mediante un añadido a la capa MAC incorpora nuevas funcionalidades concretas para las redes de sensores.

Las principales mejoras respecto al estándar IEEE 802.15.4 son las siguientes:

- Respecto al consumo energético: En IEEE 802.15.4e se implementan mejoras MAC para solucionar el problema del

coste energético provocado por que los nodos intermedios en IEEE 802.15.4 requerían de estar siempre conectados.

- Baja tasa de paquetes: Requiere una sincronización previa entre los nodos que forman la red. Esto provoca que la tasa de entrega de paquetes sea baja lo que es muy útil pasa sistemas de bajo consumo pero no para sistemas en tiempo real o con aplicaciones críticas.

- Protección del canal: En IEEE 802.15.4e se incluye una funcionalidad para proteger el canal frente a interferencias y desvanecimientos, que lo hace ideal para entornos IoT.

Estas son las principales ventajas de IEEE 802.15.4e frente a otras alternativas. Para la red que se propone en este TFM, se premiará el bajo consumo frente a otras alternativas que no son críticas en este tipo de redes, como por ejemplo, la respuesta en tiempo real que no es un requisito para la monitorización y actuación de un red de domótica.

2.3.1 Modos de operación

En el estándar IEEE 802.15.4e fue publicado en abril de 2012 e incorpora nuevas mejoras y así, nuevos modos de funcionamiento que permiten su uso para aplicaciones más específicas, dotando al estándar con mayor robustez y versatilidad. Los diferentes modos de operación son los siguientes:

- DSME (Deterministic & Synchronous Multi-Channel Extension): Especial para aplicaciones con alta disponibilidad, eficiencia, escalabilidad y robustez.

- LLDN (Low Latency Deterministic Network): Para aplicaciones en tiempo real o con latencias muy bajas, como por ejemplo: uso en robot, industrial, etc.

- TSCH (Time Slotted Channel Hopping): Se utiliza para aplicaciones de procesos de automatización. Se garantiza el ancho de banda y la latencia. Se pueden comunicar todo los nodos al mismo tiempo con el uso de diferentes canales.

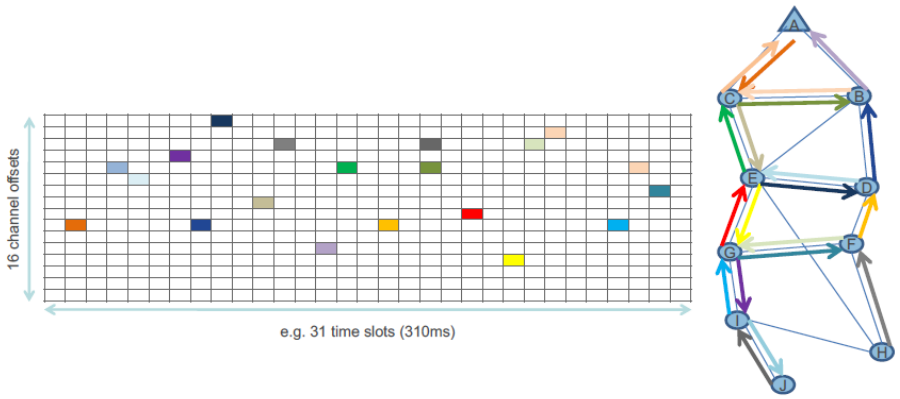


Ilustración 9: Timeslot trama TSCH [9]

Existe una compensación directa entre el rendimiento, la latencia y el consumo de energía. Y una programación de comunicación sin colisiones típico para aplicaciones industriales. Se utiliza también el algoritmo RR (Round Robin) para la retransmisión de paquetes.

- RFID Blink (Radio Frecuency Identification Blink): Se utiliza para la identificación de personas u objetos y de la localización de los mismos.

- AMCA (Asynchronous multi-channel adaptation): Se utiliza para redes más amplias o distribuidas en una región geográfica más grande. Por ejemplo: Para ciudades “Smart Cities” o en procesos de control y monitorización [10].

2.3.2 Low Energy

Low Energy (LE) es un protocolo que se define en el estándar IEEE 802.15.4e. LE permite bajar el consumo de los dispositivos por medio del duty cycle, para ellos se utilizan los mecanismos de CSL (Coordinated Sampled Listening) y Receiver Initiated Transmissions (RIT).

El mecanismo de CSL consigue ahorrar energía incorporando un mecanismo que va comprobando periódicamente y ha recibido una trama de wakeup. Cuando se reciba trama de wakeup el receptor sabrá cuando despertarse y comenzar a recibir datos del emisor [11].

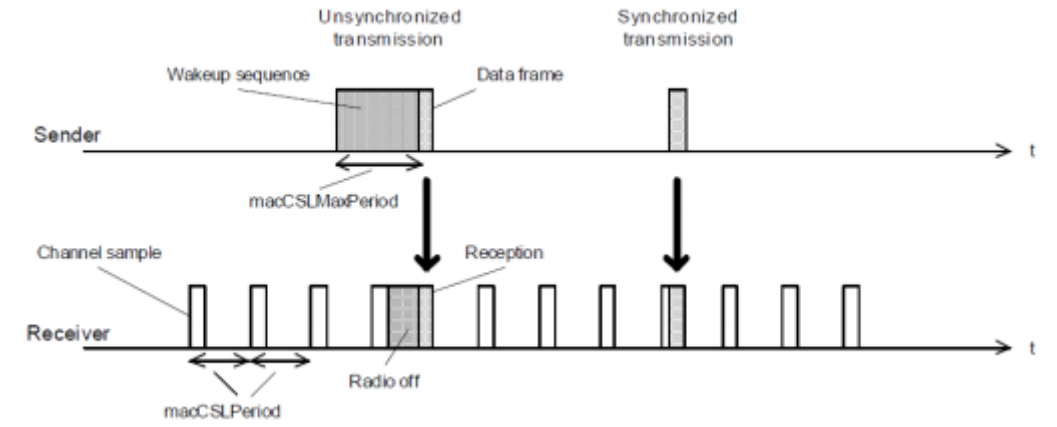


Ilustración 10: Mecanismos CSL [11]

Por otra parte, el mecanismo RIT, se utilizado para redes PAN (Personal Area Network) que no utilizan beacons. De esta forma se envían tramas datareq utilizando CSMA/CA de forma periódica. Se escuchará el canal durante un periodo corto después de enviar las tramas para así recibir las transmisiones, una vez recibidas las tramas datareq se comenzará a enviar los datos [11].

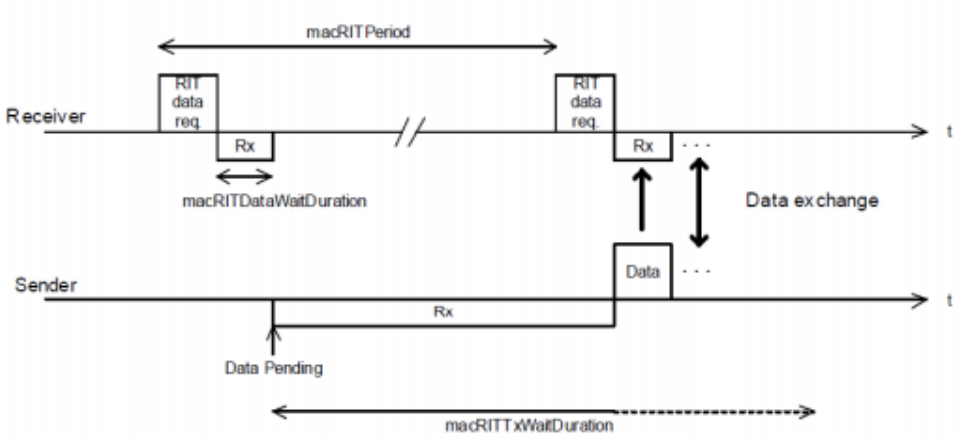


Ilustración 11: Mecanismo RIT [11]

2.3.3 Trama IEEE 802.15.4e

Como se puede observar en la siguiente figura la trama IEEE 802.15.4e tiene la siguiente estructura:

Octets: 1/2	0/1	0/2	0/1/2/8	0/2	0/1/2/8	0/1/5/6/10/14	variable	variable	2
Frame Control	Sequence Number	Destination PAN Identifier	Destination Address	Source PAN Identifier	Source Address	Auxiliary Security Header	Information Elements	Frame Payload	FCS
		Addressing fields					Header IEs	Payload IEs	
MHR							MAC Payload		MFR

Ilustración 12: Trama IEEE 802.15.4e [11]

En la trama de la estructura IEEE 802.15.4e está formada por tres partes importantes: La MAC Header (MHR), la MAC Payload y la MAC Footer MRF y dentro de cada parte existen diferentes campos para cada uso [11]:

- Frame control: Indica el tipo de trama.
- Sequence Number: Etiqueta a cada trama.
- Addressing fields: Para especificar la dirección de destino.
- Auxiliary Security Header: Campo auxiliar para la seguridad.
- Information Elements: Que encapsulan la información.

2.4 Computing layers

2.4.1 Cloud Computing

Cloud computing nace de la necesidad de ofrecer servicios a través de Internet. La computación en la nube ofrece el acceso a recursos de software bajo demanda.

Las principales características del Cloud Computing son las siguientes; **Agilidad**: Capacidad de mejora para ofrecer recursos tecnológicos al usuario por parte del proveedor; **Costo**: Reducción de los costes de la computación**; Escalabilidad y elasticidad**: Aprovechamiento de recursos para una respuesta en tiempo real; **Independencia entre el dispositivo y la ubicación:** Acceso independiente, sin depender desde donde te conectes; **La tecnología de virtualización** permite compartir servidores y dispositivos de almacenamiento y una mayor utilización; Rendimiento: Los sistemas en la nube controlan y optimizan el uso de los recursos de manera automática; **Seguridad**: Llegando a ser incluso mejor que en sistemas tradicionales; **Mantenimiento**: Al ser aplicaciones instaladas en la nube no es necesario un mantenimiento del usuario.

Los modelos de servicio en Cloud Computing son los siguientes:

- Software as a Service (SaaS): Aplicación en línea disponible para múltiples usuarios bajo demanda. Estas aplicaciones pueden ser accesibles vía web. No es necesaria la instalación en local liberando así recursos del equipo local. Ejemplo: Google Docs, Salesforce.

- Platform as a Service (PaaS): Plataforma para desplegar aplicaciones que se pueden escalar bajo demanda. Se permite a los usuarios desarrollar y gestionar aplicaciones sin la necesidad de tener que mantener la infraestructura. Ejemplo: Google AppEngine, cloudfoundry.

- Infrastructure as a Service (IaaS) o Hardware as a service (HaaS): Servidores virtuales y almacenamiento disponible en forma escalable a

través de la red. Se tiene el control de las aplicaciones, sistema operativo y almacenamiento de los archivos. Ejemplos: Amazon web Services, Rackspace [14].

- **Things** as a Service (TaaS). Es un nuevo concepto que se está comenzando a utilizar y persigue la idea de entregar la funcionalidad IoT sin que el usuario final tenga que operar o mantener hardware extenso. Servicios que se pueden entregar en la nube para recibir y procesar los datos generados por las redes de sensores habilitados para IoT.

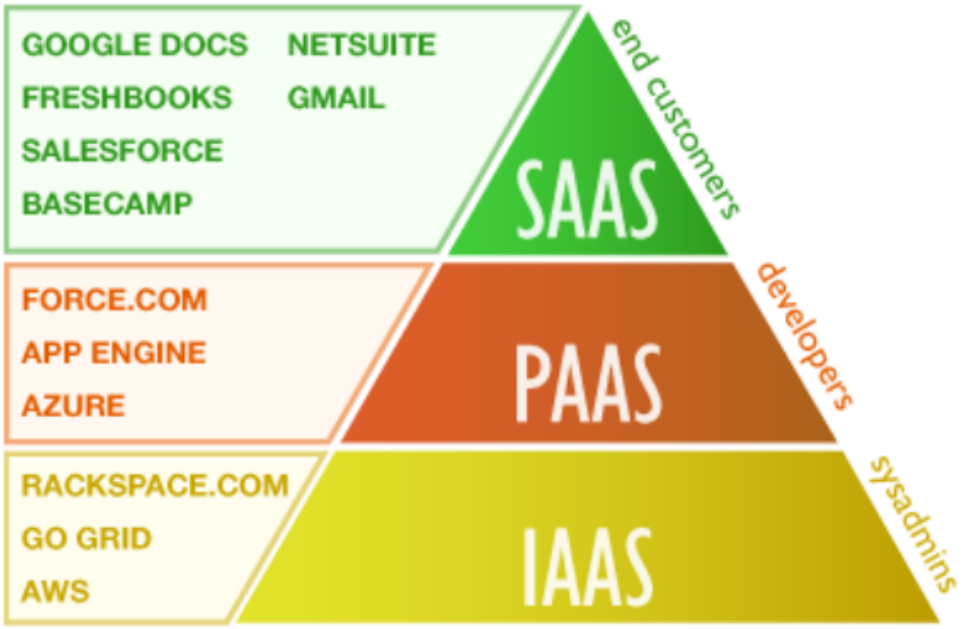


Ilustración 13: Servicios Cloud Computing [15]

2.4.2 Edge y Fog Computing

2.4.2.1 Introducción

"Fog Computing" o la niebla es un término que extiende el concepto de la nube para estar más cerca de las cosas que nos rodean y actúan sobre los datos de IoT.

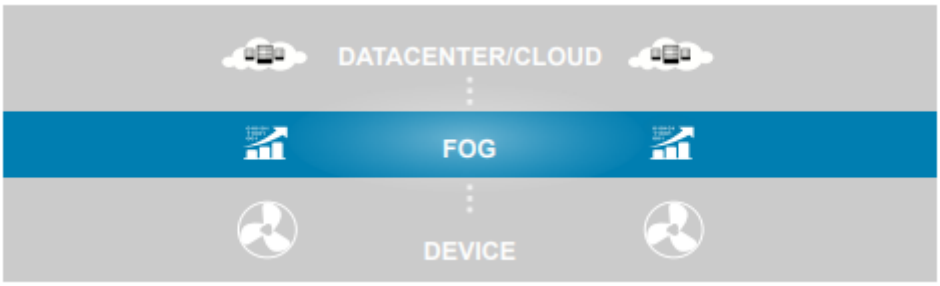


Ilustración 14: Fog Computing [16]

Estos dispositivos, denominados "fog nodes", pueden implementarse en cualquier lugar con una conexión de red: en una fábrica, en la parte superior de un poste de energía, junto a una vía férrea, en un vehículo o en una plataforma petrolera. Cualquier dispositivo con computación, almacenamiento y conectividad de red puede ser un nodo. A diferencia del "Cloud Computing" que son servidores muy alejados de los usuarios, en el internet de las cosas (IoT) entra en juego el concepto de "Fog Computing" donde estaremos rodeados por miles de sensores, conectados y transmitiendo información [16].

Entre los ejemplos podemos encontrar controladores industriales, conmutadores, enrutadores, servidores integrados y cámaras de video vigilancia. Las posibilidades son infinitas [16].

2.4.2.2 Beneficios Fog Computing

Las ventajas del uso de "Fog Computing" son las siguientes:

- Mayor agilidad empresarial: Con las herramientas adecuadas, los desarrolladores pueden desarrollar rápidamente aplicaciones para ser implementadas donde sea necesario. Es posible generar aplicaciones para cualquier necesidad del cliente.

- Mejor seguridad: Se pueden proteger todo los nodos "fog" utilizando la misma política, controles y procedimientos que en todas las partes de del entorno de TI.

- Gestión de la Información, con control de privacidad: Análisis de datos confidenciales localmente en lugar de enviarlos a la nube para su análisis. Se puede monitorear y controlar los dispositivos que recopilan, analizan y almacenan datos.

- Control del gasto operativo: Se puede procesar los datos seleccionados localmente y después enviar lo necesario a la nube [16].

Se muestra a continuación la arquitectura para "Fog Computing" basada en IoT.

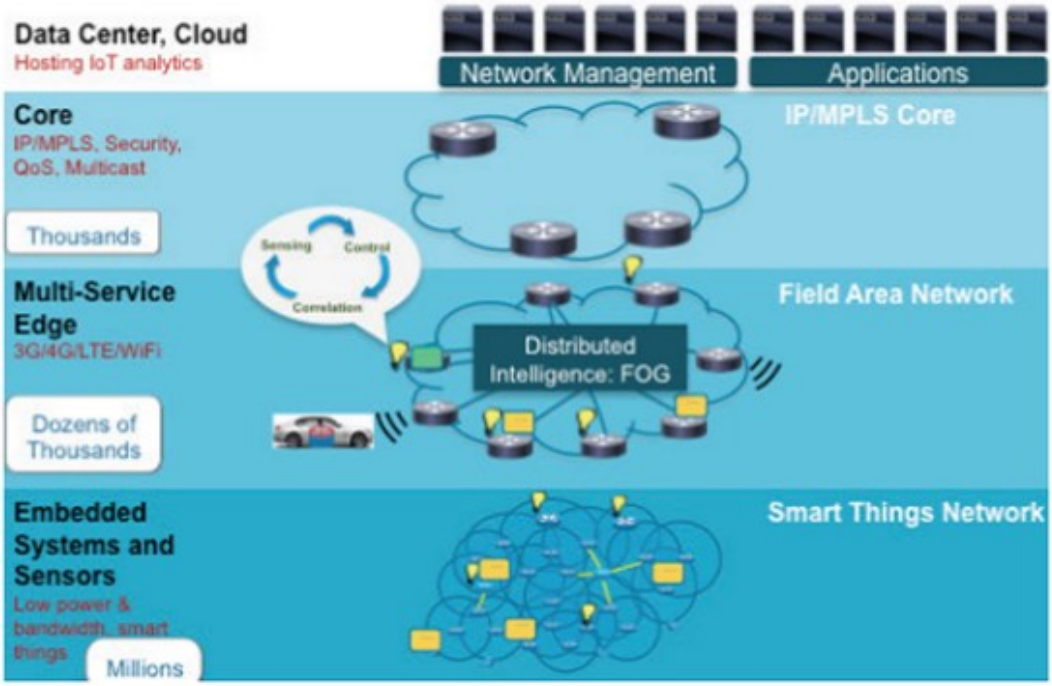


Ilustración 15: Arquitectura Fog Computing en IoT [16]

2.4.2.3 Edge Computing vs Fog Computing vs Could Computing

"Fog Computing" y "Edge Computing" parecen similares ya que ambas implican acercar la inteligencia y el procesamiento a la creación de datos. Sin embargo, la diferencia clave entre los dos radica en dónde se procesa la información. Un entorno "fog" ubica la inteligencia en la red de área local (LAN). Esta arquitectura transmite datos desde los puntos finales a una puerta de enlace, donde luego se transmite a las fuentes para el procesamiento y la transmisión de destino. Mientras que en "Edge computing" se realiza gran parte del procesamiento en plataformas informáticas integradas que interactúan directamente con sensores y controladores. Sin embargo, esta distinción no siempre es clara, ya que las organizaciones pueden ser muy variables en su enfoque del procesamiento de datos [17].

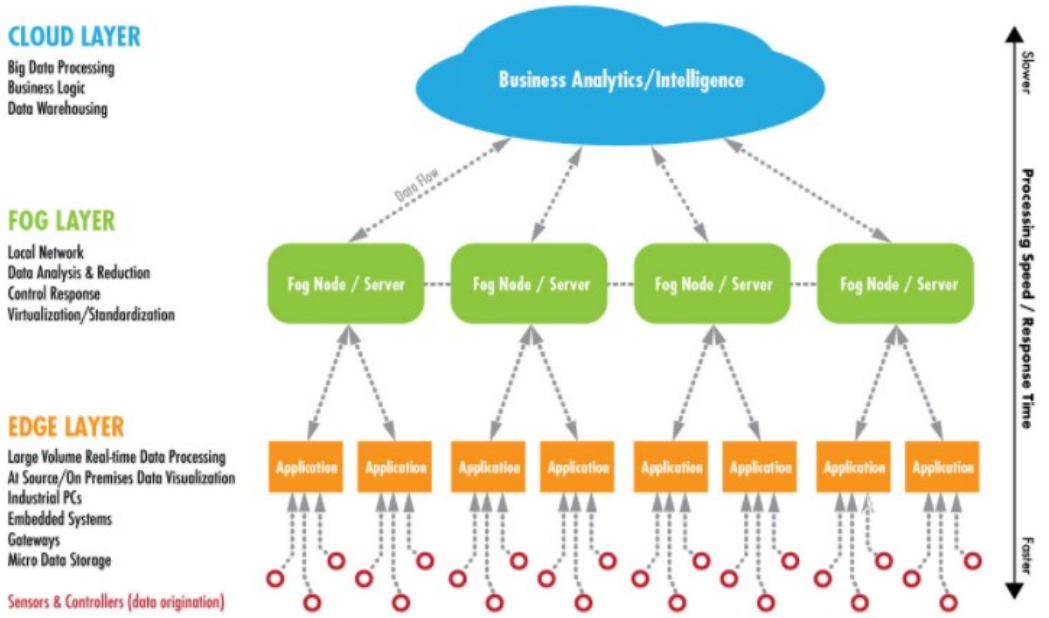


Ilustración 16: Capas en el procesado de datos IoT [17]

Comparativa Fog Computing vs Cloud Computing

Como ya se ha comentado, “Cloud Computing”, se define como un grupo de computadoras y servidores conectados a través de Internet para formar una red. Mientras que empresas y organizaciones comienzan a introducir el Internet de las cosas, la necesidad de acceder a grandes cantidades de datos de manera más rápida y local está en constante crecimiento. Aquí es donde entra en juego el concepto de "Fog Computing".

“Fog Computing”, o niebla, es una infraestructura distribuida en la que ciertos procesos o servicios de aplicaciones son administrados en el borde de la red (“edge”) por un dispositivo inteligente, mientras que otros requieren que se administren en la nube. Para ello, es también necesario, una capa intermedia entre la nube y el hardware para permitir un procesamiento de datos, análisis y almacenamiento más eficientes, lo que se logra al reducir la cantidad de datos que se deben transportar a la nube.

Se presenta a continuación una tabla comparativa con las principales características de cada tipo de computación.

	Cloud Computing	Fog Computing
Latencia	Alta	Baja
Retardo Jitter	Alta	Muy bajo
Localización del servicio	Dentro de internet	En el borde de las redes locales
Distancia entre el	Múltiples saltos	Un salto

cliente y el servidor		
Seguridad	Indefinida	Se puede definir
Posibilidad de ataques	Alta	Baja
Localización definida	No	Si
Geo distribución	Centralizada	Distribuida
Número de nodos	Pocos	Muchos
Soporte para la movilidad	Limitada	Soportada
Iteraciones en tiempo real	Soportada	Soportada
Conexión última milla	Línea alquilada	Wireless

Tabla 4: Cloud Computing vs Fog Computing [18]

La mayor limitación de "Cloud Computing" se encuentra en la respuesta en tiempo real para las aplicaciones que así lo requieran.

Cloud Computing	Fog Computing
La información y las aplicaciones son procesadas en la nube, esto conlleva tiempo para grandes cantidades de datos.	Más que presentar y trabajar con un servidor centralizado en la nube, fog computing opera en la capa borde o edge. Por lo que se requiere menos tiempo.
Puede haber un problema de ancho de banda, ya que es necesario enviar cada resultado a la nube.	Requiere un uso menor del ancho de banda, ya que esta información es trata antes de ser enviada.
Una respuesta lenta para escalar posibles problemas ya que los servidores están localizados en servidores remotos.	Puesto que se dispone de pequeños servidores edge accesible para los usuarios, es posible para fog computing reducir el tiempo de respuesta en la escalación de incidencias.

Tabla 5: Cloud Computing vs Fog Computing 2 [18]

Como conclusión, "Fog Computing" juega un papel muy importante en las demandas creadas por IoT, el aumento de dispositivos conectados a internet a la vez y transmitiendo información constantemente hace que la solución de "Cloud Computing" una herramienta insuficiente, donde los tiempos de espera, costes e eficiencia de respuesta del sistema se puede ver comprometida.

Como solución se introduce el concepto de "Fog Computing" donde se procesará la información en servidores más locales y accesibles para el usuario que posteriormente se enviaran a los servidores de "Cloud Computing". Así pues, se puede ver a "Fog Computing" como una forma de complementar a "Cloud Computing" utilizando las ventajas de cada tecnología, más que como una tecnología que pretende sustituirla.

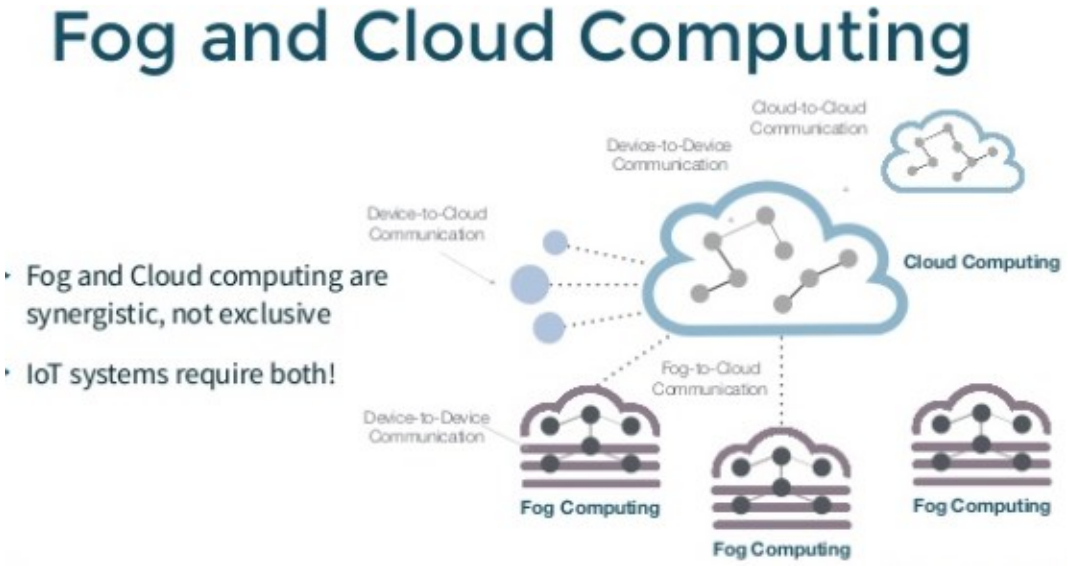


Ilustración 17: Cloud & Fog Computing [23]

3. Descripción de sistema

3.1. Introducción

Con el desarrollo de este proyecto se pretende realizar la elaboración de un hardware y de un software que permita la demostración de un sistema IoT con el estándar IEEE 802.15.4e.

Para ello se utilizará la plataforma OpenMote para el Hardware y OpenWSN como sistema operativo. OpenWSN proporciona una serie de implementaciones de código abierto, con una pila de protocolos completa basada en estándares del internet de las cosas (IoT) con una gran variedad de plataformas software y hardware. Esta implementación la hace muy versátil a la hora de aplicarla a redes IoT lo que ha convertido a OpenWSN en una de las plataformas más versátiles del mercado. Es una herramienta fácil de manejar y de implementar.

Application	CoAP, HTTP
Transport	UDP, TCP
IP/routing	IETF RPL
Adaptation	IETF 6LoWPAN
Médium Access	IEEE802.15.4e
Phy	IEEE802.15.4-2006

Tabla 6: Pila de protocolos OpenWSN [1]

El sistema que se pretende desarrollar es un sistema de control y monitorización de domótica donde se pueda medir y controlar diversos parámetros de medidas que permitan hacer un uso más eficiente de la energía dentro del hogar. Para ellos, se ha implementado un sistema de que es capaz de medir la temperatura y la luminosidad de una vivienda.

Con el control de estas dos magnitudes el usuario podrá conocer en todo momento el estado de su vivienda y podrá decidir si es necesario, activar la climatización, abrir o cerrar persianas en función de la luminosidad o detectar una temperatura excesivamente alta.

Finalmente no ha sido posible implementar en hardware la red por lo que la solución final se presentará como una red simulada de igual o mayor complejidad que una red física. También se pretendía analizar los beneficios del control automático de las persianas en la temperatura de una vivienda para mejorar así el uso energético de la vivienda y la repercusión económica de esta mejora.

Como control software se presenta una aplicación en Python que se encargará del control de la red. Se encargará de comunicarse con los diferentes sensores que hay conectados a red, de almacenar los datos extraídos de los sensores y de comunicarse con ellos para accionar alguna funcionalidad extra (esto equivaldría al control de abrir o cerrar una persiana). La aplicación se encargará además de procesar la información recibida antes de subirla a la nube. Se ha prestado especial interés en desarrollar una aplicación modular que sea fácilmente editable y que soporte la ampliación de la red sin necesidad de tocar el código fuente.

Para el bloque de procesamiento se ha decidido utilizar una Raspberry Pi debido a su reducido coste, tamaño y consumo que lo hacen ideal para controlar el sistema completo. Igualmente, el sistema es perfectamente funcional sobre una plataforma Windows.

Y por último, el último elemento que forma parte de este trabajo es la plataforma THETHINGS.IO, donde se enviaran los datos del sistema y donde se almacenarán para que puedan ser representados fácilmente. También ha utilizado la capacidad que tiene la plataforma THETHINGS.IO de enviar datos a la red para que sirva como mecanismo de control del usuario a la red. Aunque se comentará con más detalle se ha puesto especial atención al número de llamadas que se realizan a la plataforma ya que el uso de la misma es limitado y conlleva coste.

3.2. Arquitectura del sistema

El sistema está formado por tres bloques que se pueden identificar de la siguiente forma:

- El bloque de sensores.
- El bloque central.
- El bloque en la nube con la plataforma THETHINGS.

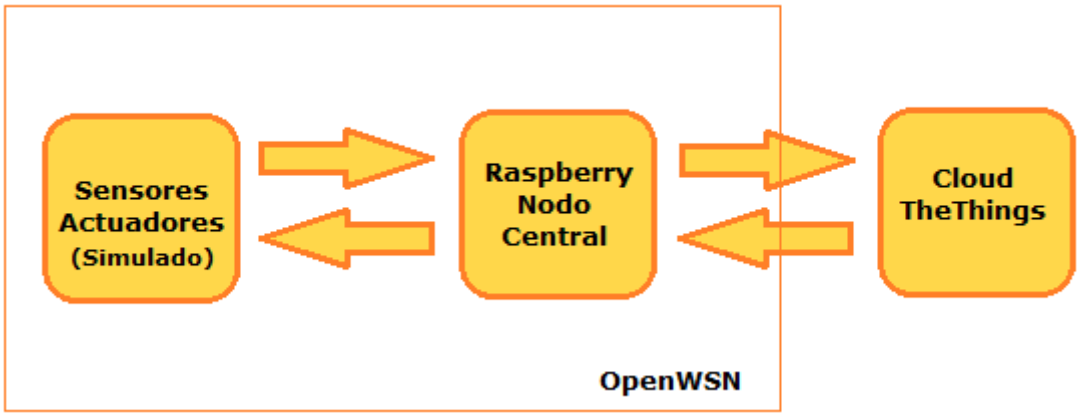


Ilustración 18: Arquitectura del sistema

Bloque de sensores

Este bloque es el encargado de tomar las medidas tanto de temperatura como de luminosidad y que serán enviadas al nodo de control (Raspberry). También será el encargado de recibir desde el nodo central las instrucciones para actuar de una forma u otra. Este bloque debería de estar formado con los diferentes sensores pero en este caso serán sensores simulados desde la plataforma OpenWSN, aunque el funcionamiento es simular.

Bloque central

Este bloque esta formador por la Raspberry PI 3 aunque también podría estar formado por un ordenador con Windows. Este equipo estará configurado con las herramientas necesarias para el funcionamiento de OpenWSN así como del software necesario para su control. En este nodo central correrá la plataforma OpenWSN y el software desarrollado en Python para su control. Servirá también como base de datos para los datos capturados desde los sensores y se encargará de la comunicación con la plataforma THETHINGS.IO.

Bloque en la nube con la plataforma THETHINGS.

En esta plataforma online, se recogerán los datos enviados desde el nodo central. También se configurará la forma de representar los datos tanto de forma histórica como con la medida actual.

Servirá también para que el usuario pueda comunicarse con la red aplicando un control que se enviará desde THETHINGS a la red.

3.3. Desarrollo software

3.3.1 Introducción

En el desarrollo y configuración del software se pueden diferenciar también tres partes importantes.

- Configuración software de Windows y Linux.
- Desarrollo del firmware que se cargará en las placas OpenMote (Simulado), desarrollado en C.
- Desarrollo de una aplicación de control del sistema desarrollada en Python.

En los siguientes apartados, se detallaran las herramientas utilizadas para la puesta a punto de los equipos tanto en plataformas Windows como en plataformas Linux.

Se detallará también como se ha desarrollado el firmware para los Motes simulados y que hacen las veces de sensores reales. A su vez este bloque de sensores ser controlados por un firmware modelado en C, se encargará de la retransmisión de las medidas mediante el protocolo CoAP.

El módulo principal, modelado en Python, se encargará de hacer las peticiones a los sensores mediante el protocolo CoAP. Para utilizar el protocolo CoAP es necesario instalar previamente las librerías de CoAP. Estas librerías de CoAP de uso libre, son utilizadas por el programa principal para la adquisición de muestras procedentes de los sensores:

```
p = c.GET('coap://[{0}]/i'.format(MOTE_IP))
```

Donde la variable MOTE_IP contendrá la IPv6 del mote al que se le quiera adquirir muestras.

De una forma análoga también se utilizará la función PUT de CoAP para la implementación de actuaciones en los sensores. Esta función podría utilizarse para la apertura de una válvula, para encender o apagar un dispositivo o para que un cierto sensor funcione de una forma u otra. Puesto que en este caso los sensores son simulados, la función PUT se utilizará para hacer un cambio en los valores pseudoaleatorios, los que provocaran un cambio de magnitud en las muestras enviadas desde los sensores. Estos cambios se detallaran más adelante en la descripción del firmware.

La función PUT de CoAP, es ligeramente diferente a la de GET. Es necesario también pasarle como valor la IPv6 del mote al que se quiere mandar el nuevo valor, e indicarle el nuevo valor como “payload”:

```
p = c.PUT('coap://[{0}]/i'.format(MOTE_IP_ACTUATOR),payload =
            [valor_actuador],)
```

Por último, una vez procesadas las muestras adquiridas por los sensores, estas se procesaran para ser enviadas a la plataforma de THETHINGS.IO, plataforma online para su almacenamiento y representación. También se utilizará esta plataforma para que el usuario pueda enviar cambios a los sensores. Para ello, se ha instalado las APIs de THETHINGS.IO que serán llamadas desde el programa principal para utilizar sus funciones de send & received.

A continuación se muestra un esquema de la arquitectura software y como interaccionan todos los elementos entre sí:

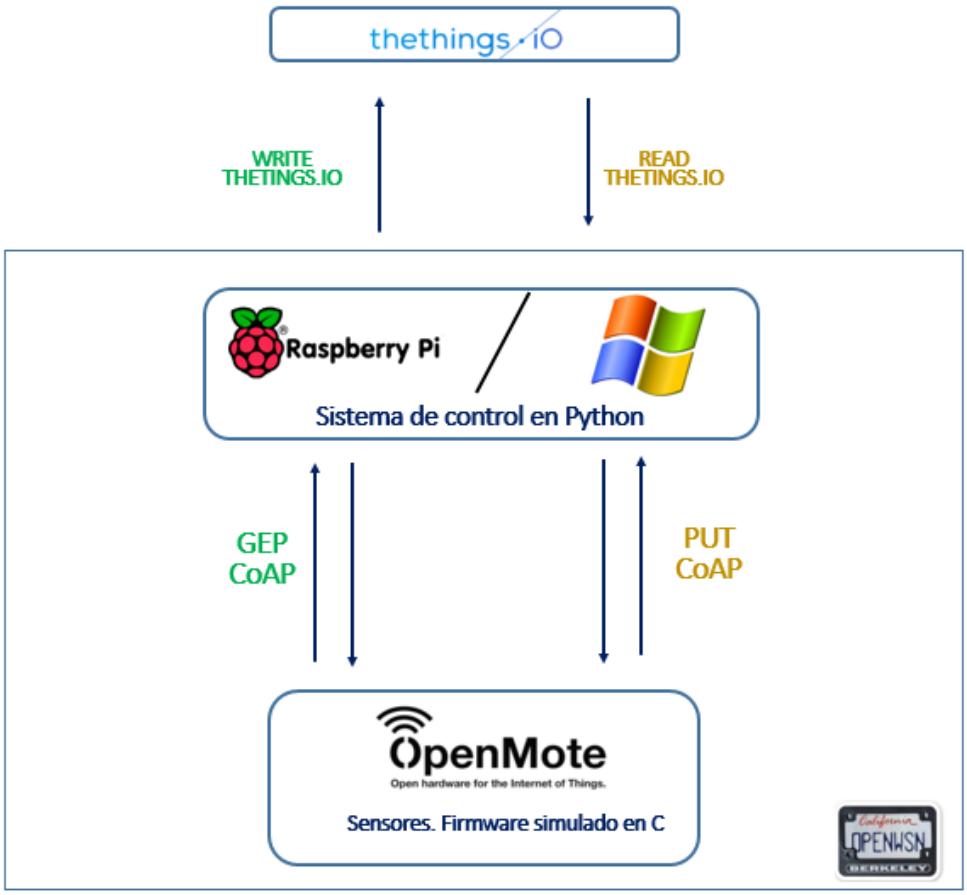


Ilustración 19: Arquitectura software

3.3.2. Herramientas utilizadas

Para la configuración de OpenWSN sobre Windows se ha seguido la referencia [1] del proyecto de Atlassian sobre OpenWSN.

TortoiseGit (https://tortoisegit.org/)

Con esta aplicación se podrá descargar el contenido de Github directamente, simplemente colocando la dirección del repositorio. Es una herramienta muy útil con la que se permite clonar el contenido de Github y con ella se descargará:

- El firmware de OpenWSN para cargar en los sensores OpenMote o para la creación de los sensores simulados: https://github.com/openwsn-berkeley/openwsn-fw

- El software de OpenWSN para la red simulada, además incluye otras herramientas como OpenVisualizer necesario para el funcionamiento del sistema: https://github.com/openwsn-berkeley/openwsn-sw

- Los módulos CoAP implementados en Python que serán necesarios para aplicar CoAP para la comunicación con los sensores: https://github.com/openwsn-berkeley/coap

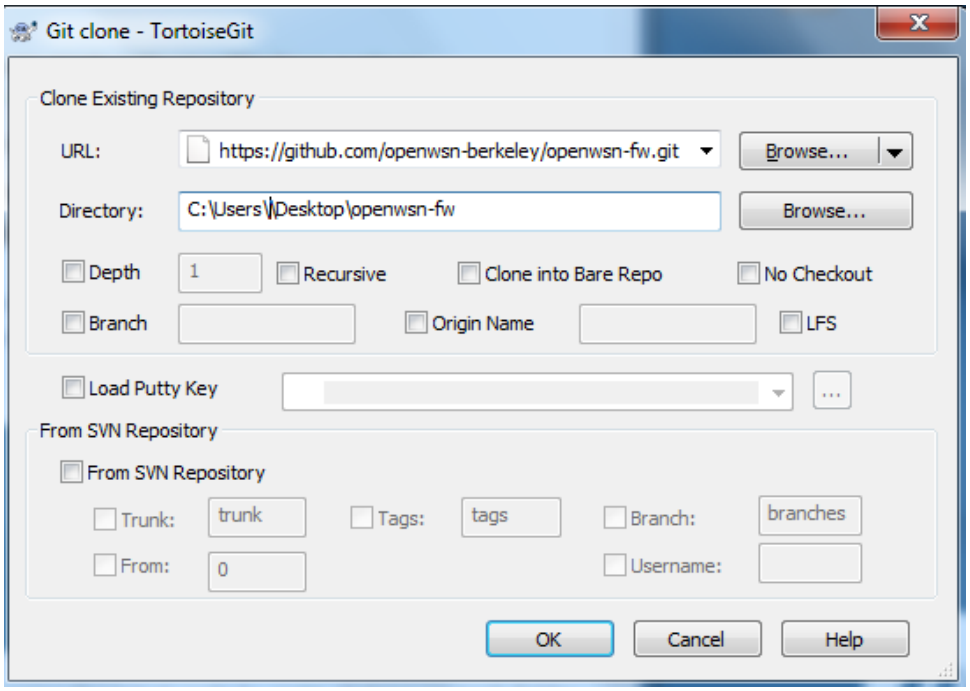


Ilustración 20: Git clone - TortoiseGit

MinGW (https://sourceforge.net/projects/mingw/files/?source=navbar)

Es una colección de compiladores GNU (GCC), con bibliotecas de importación y cabeceras de libre distribución que serán necesarias para la ejecución de algunas plataformas como OpenSim.

Python 27 (http://sourceforge.net/projects/pywin32/)

Módulo de instalación de Python necesario para la compilación de OpenWSN y para el desarrollo de las herramientas necesarias.

Pycharm (https://www.jetbrains.com/pycharm/)

Entorno de desarrollo de Python. Se ha elegido este entorno ya que es un entorno muy popular de Python, con una gran repositorio de librerías y que ya había trabajado con el anteriormente.

SCons

MinGSCons es una herramienta de construcción de software de código abierto, es decir, una herramienta de compilación y de creación de software que requiere la plataforma OpenWSN.

TAP virtual interface (OpenVPN 2.4.5)

OpenVPN es un software que permite la creación de una red simulada que será necesario para la simulación de los motes.

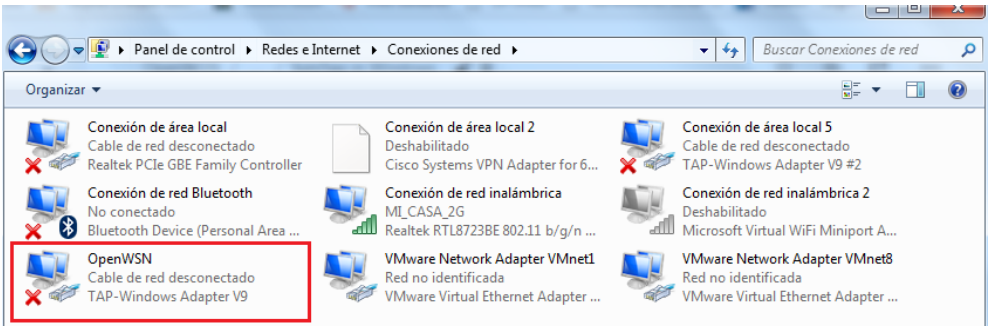


Ilustración 21: Definición TUN [1]

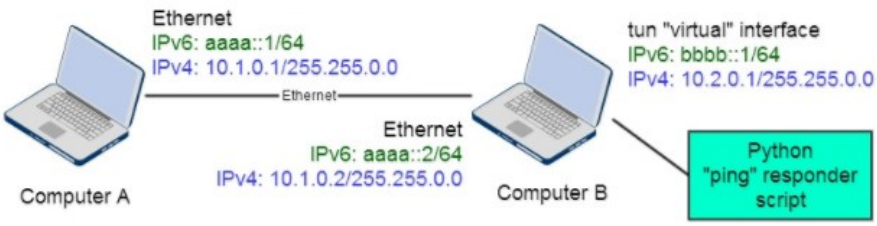


Ilustración 22: Configuración TUN [1]

OpenSim

OpenSim es una herramienta dentro del repositorio de OpenWSN que permite la creación de redes simuladas. Su funcionamiento es similar al de una red con hardware real. La principal diferencia se encuentra en que el firmware se ejecutará desde el nodo principal en vez desde los motes.

OpenSim lo hace compilando el firmware del mote como un módulo de extensión de Python y creando una instancia de la clase resultante para cada mote emulado. Cuando se ejecuta la simulación, estos motes emulados se comunican con el resto de la arquitectura OpenVisualizer.

Como se ilustra en el siguiente diagrama, los motes emulados interactúan de forma similar que con hardware real [1]:

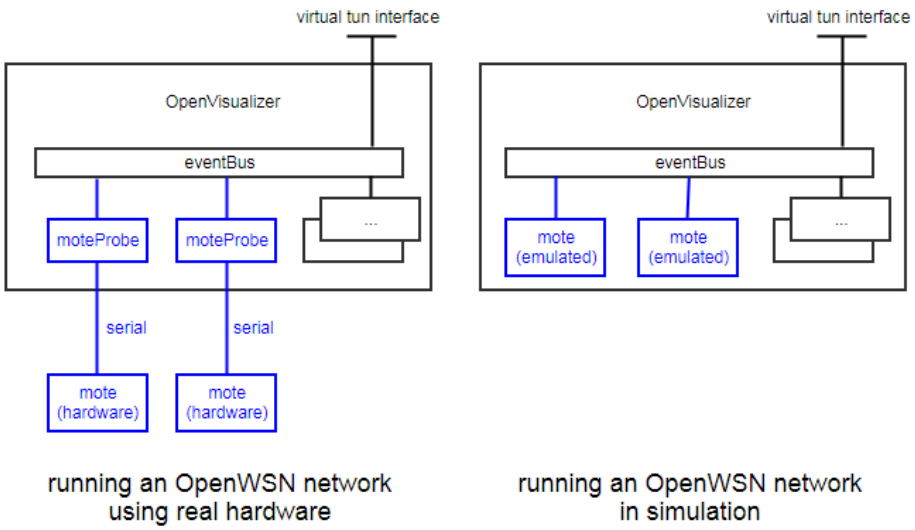


Ilustración 23: OpenWSN con red simulada [1]

OpenVisualizer

OpenVisualizer es una herramienta que está dentro de la plataforma OpenWSN y que es fundamental para definir la red y

que se encargará de la comunicación entre los diferentes elementos.

La inicialización se puede ejecutar directamente por consola o utilizando un interfaz web:

```
scons runweb --sim
```

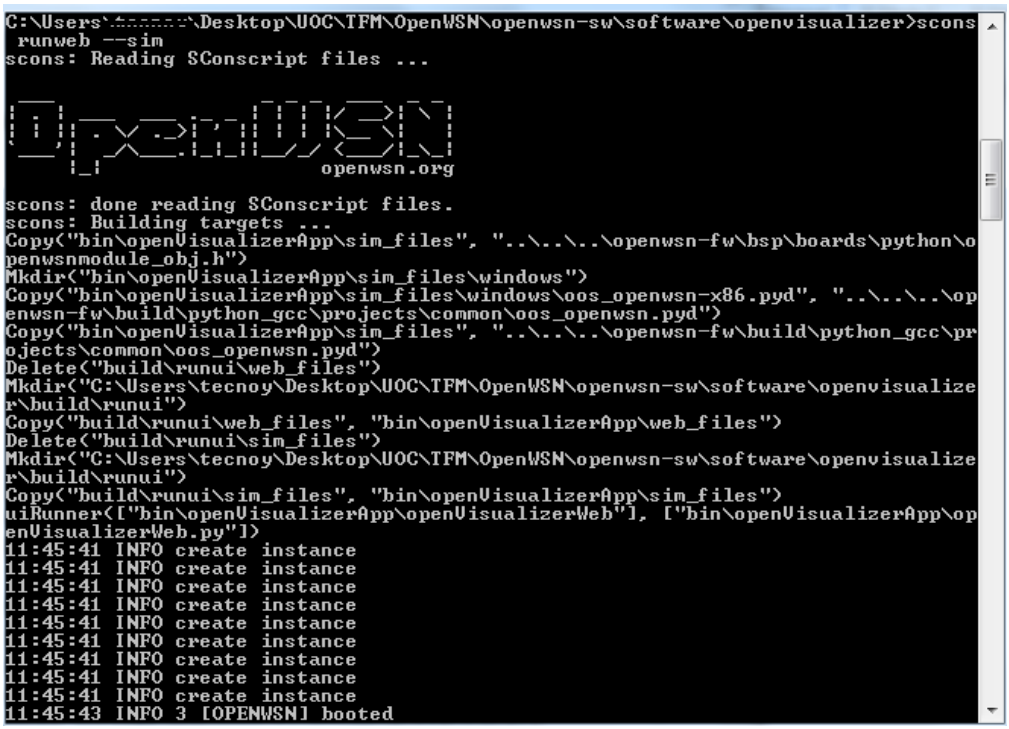


Ilustración 24: OpenVisualizer entorno consola

3.3.3. Configuración en Raspberry Pi 3 Model B

Raspberry Pi es una pequeña placa base que la forman un chip BCM2837, un procesador ARM Cortex-A53 de hasta 1.2GHz y con una RAM de 1GB. A parte a la Raspberry se le incorpora una memoria SD de 32 Gb (externa). Estos pequeños computadores tienen una potencia razonable a un precio bajo, entorno a los 30 euros, clave del éxito de estas placas.

Las aplicaciones de estas placas son diversas y se pueden utilizar para casi cualquier cosa que no requiera una capacidad de procesamiento muy alta. En este TFM se utilizará servidor central del sistema, haciendo de pasarela entre los sensores, procesando la información y retransmitiéndola a la plataforma THETHING.IO.

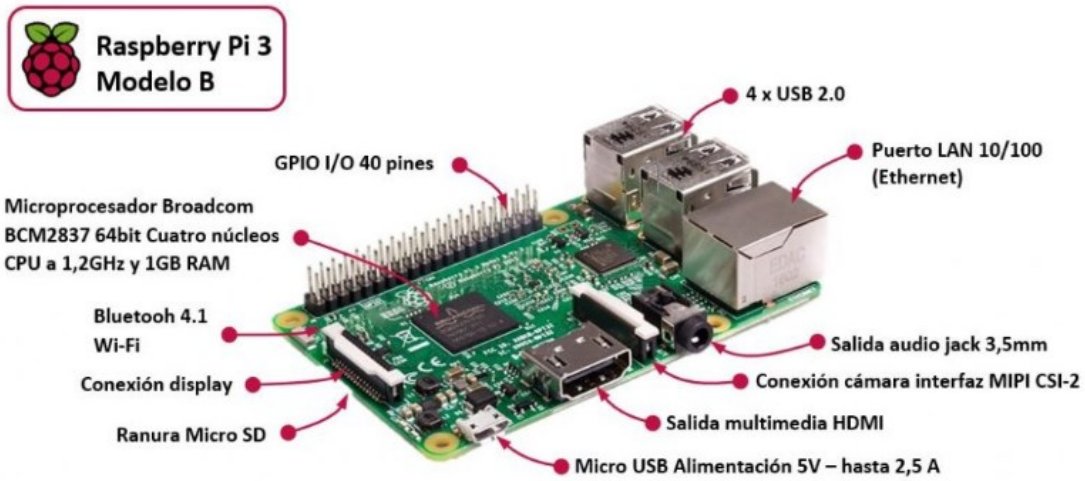


Ilustración 25: Descripción Raspberry Pi 3 Model B [24]

A la Raspberry Pi 3 se le instalará el sistema operativo Rasbian Stretch, que se puede conseguir directamente desde:

https://www.raspberrypi.org/downloads/raspbian/

Raspbian incorpora un pack de software especial para el uso en programación incluyendo módulos en Python que lo hace una opción ideal para el desarrollo del sistema.

Para la instalación del sistema operativo Raspbian se utiliza la herramienta **Win32 Disk Imager** de uso libre.

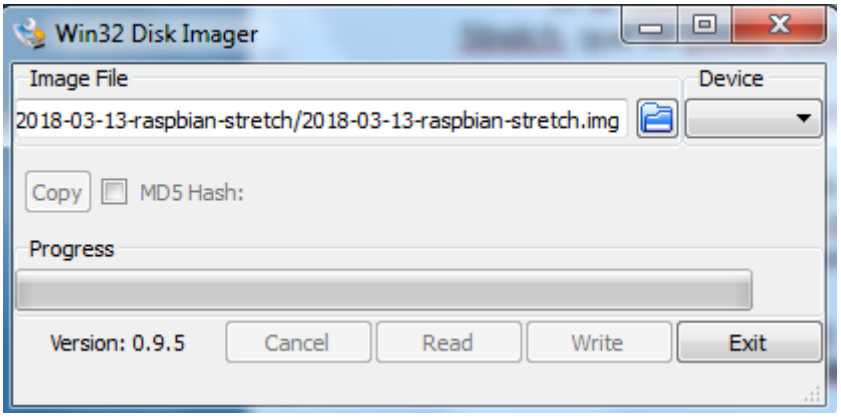


Ilustración 26: Win32 Disk Imager

3.3.4. Definición de la red

3.3.4.1 Introducción

Como ya se ha comentado, se va a definir una red formada por un sensor de temperatura, por un sensor de luminosidad y por un actuador. De este modo se puede utilizar tanto la función GET como la función PUT de CoAP y demostrar la funcionalidad y la capacidad que tiene OpenWSN en el control de sensores.

Puesto como la red al final ha sido una red simulada, el mote actuador que podría hubiera sido un motor que cierra o abre persianas (o cualquier otro tipo de mote que controle aperturas o encendidos) no es real, se ha decidido implementar un sistema para que el mote simulado sea capaz de detectar un valor definido por usuario e implementar un cambio de magnitud en las medidas generadas.

Por lo tanto, la red simulada la formarán dos motes simulados y el modo actuador que recibe las peticiones PUT de CoAP formará parte de estos dos motes simulados. La red final está formada por el nodo principal, y dos motes adicionales para la simulación. La configuración de la red se elabora desde la plataforma OpenVisualizer de OpenWSN.

Para definir la red se ha utilizado OpenVisualizer que está disponible dentro de la plataforma OpenWSN. Se puede acceder a OpenVisualizer mediante la IP de loopback: http://127.0.0.1:8080/.

3.3.4.2 Topología

Para la creación de la red se ha definido una estructura peer-to-peer donde todos los motes están conectados unos con otros. De esta forma todos los nodos están conectados entre sí pudiendo intercambiar información si fuera necesario o en caso de corte de una de las líneas siempre existiría una forma de acceder al mote de destino.

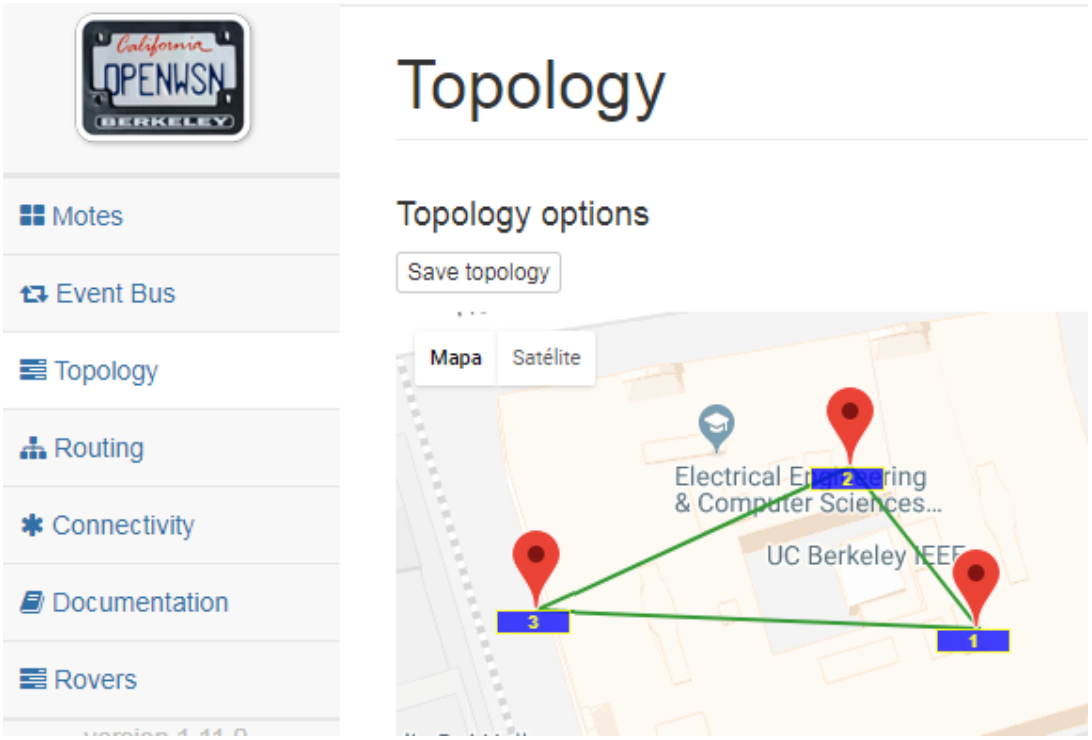


Ilustración 27: Topology red simulada

3.3.4.2 Motes

Se ha definido también el nodo 2 como el nodo de control o DAGRoot que será el nodo principal y el encargado de gestionar la red. De esta forma en el nodo DAGRoot es donde se ejecuta el programa principal.

Los nodos 1 y 3 son los motes simulados y desde donde se realizará la adquisición de datos.

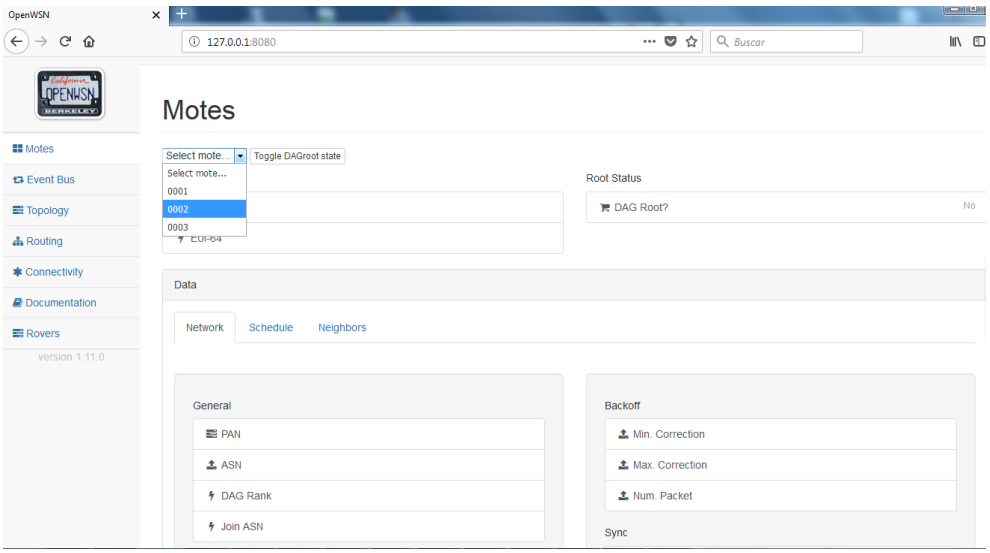


Ilustración 28: OpenWSN - Motes

Es posible controlar los niveles de Una vez seteado el “DAGRoot” y este se ha sincronizado correctamente, se puede observar cuáles son sus nodos vecinos, su dirección MAC, niveles de señal (RSS), etc.

3.3.4.3 Conectividad

Se muestra también la relación de los sensores con el DAGRoot:

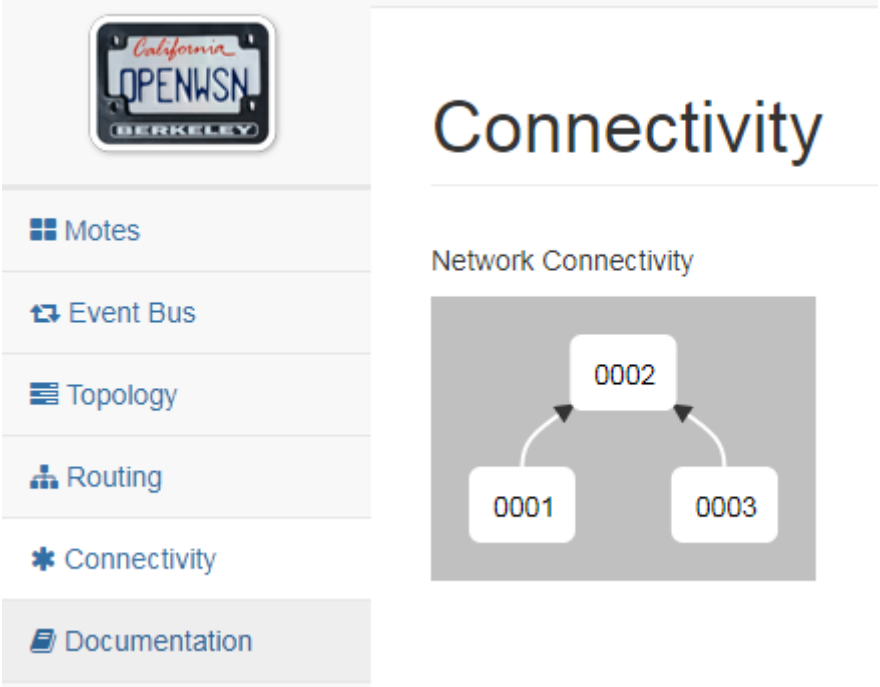


Ilustración 29: Connectivity red simulada

3.3.5. Bloque de sensores

Una de las parte del desarrollo del software consiste en la modificación del firmware de OpenWSN. El objetivo de esta modificación es la de las placas de OpenMote puedan tomar medidas correctamente y el tratamiento de las señales recibidas.

En este TFM, se modificará el firmware para que genere una batería de datos aleatorios que puedan ser recibidos desde el programa principal.

El firmware de control para los motes se puede obtener libremente desde el repositorio https://github.com/openwsn-berkeley/openwsn-fw.

Este firmware esta modelado en C y puede ser compilado a la vez que se carga en la plataforma de OpenWSN. Para la carga y compilación en OpenWSN se realiza con la siguiente llamada:

```
scons board=python toolchain=gcc oos_openwsn
```

Esta llamada realizará la compilación completa del firmware o si ya esta cargado solo compilará y cargará la parte que haya sido modificada:

“

```
C:\Users\\Desktop\openwsn-fw>scons  board=python  toolchain=gcc
oos_openwsn
.
.
.
Compiling
build\python_gcc\bsp\boards\python\supply_obj.o
Compiling
build\python_gcc\bsp\boards\common\aes_cbc.o
Compiling
build\python_gcc\bsp\boards\common\aes_ccms.o
Compiling
build\python_gcc\bsp\boards\common\aes_ctr.o
Compiling
build\python_gcc\bsp\boards\common\aes_ecb.o
Compiling
build\python_gcc\bsp\boards\common\firmware_crypto_engine.o
Compiling
build\python_gcc\bsp\boards\common\dummy_crypto_engine.o
Archiving          build\python_gcc\bsp\boards\libbsp.a
Indexing           build\python_gcc\bsp\boards\libbsp.a
gcc -shared -o
build\python_gcc\projects\common\oos_openwsn.pyd
build\python_gcc\projects\common\03o
os_openwsn\03oos_openwsn_obj.o
build\python_gcc\projects\common\03oos_openwsn\openwsnmodule_o
bj.o -L
C:\Python27\libs -Lbuild\python_gcc\bsp\boards -
Lbuild\python_gcc\kernel\openos -Lbuild\python_gcc\d
rivers -Lbuild\python_gcc\openstack -
Lbuild\python_gcc\openapps -lopenstack -lopenapps -lkernel -
ldr
ivers -lbsp -lpython27 -Wl,--out-
implib,build\python_gcc\projects\common\liboos_openwsn.a
scons: done building targets.”
```

3.3.5.1 Modificación del firmware

Con la modificación del firmware se intenta realizar la emulación de dos sensores, uno de temperatura y otro de luminosidad. Para ello se ha modificado cinfo.c para que esta pueda enviar datos pseudoaleatorios según la hora actual.

Esta aplicación además tendrá la capacidad de recibir datos desde el programa principal y poder modificar así los valores enviados.

La preparación de datos para que el programa principal pueda adquirirlos se elavora con la función COAP_CODE_REQ_GET.

La aplicación detectará la hora actual y según la hora se enviará un dato u otro.

```
131     case COAP_CODE_REQ_GET:
132       //=== reset packet payload (we will reuse this packetBuffer)
133       msg->payload                    = &(msg->packet[127]);
134       msg->length                     = 0;
135
136       //=== prepare  CoAP response
137
138       // Load messager
139
140         // Se toma la hora actual para generar una temperatura y luminosidad definida por hora
141         time_t tiempo = time(0);
142         struct tm *tlocal = localtime(&tiempo);
143         char t_output[127];
144         strftime(t_output,127,"%H",tlocal);
145         temp = atoi(t_output);
146
147         //Con el siguiente swich se le asignará a cada hora una temperatura y luminosidad
148         switch (temp) {
149             case 0: temp = 16 + cambio_lectura; luminosidad = cambio_lectura + 10;  break;
150             case 1: temp = 14 + cambio_lectura; luminosidad = cambio_lectura + 10;  break;
151             case 2: temp = 13 + cambio_lectura; luminosidad = cambio_lectura + 10;  break;
152             case 3: temp = 12 + cambio_lectura; luminosidad = cambio_lectura + 10;  break;
153             case 4: temp = 12 + cambio_lectura; luminosidad = cambio_lectura + 10;  break;
154             case 5: temp = 11 + cambio_lectura; luminosidad = cambio_lectura + 10;  break;
155             case 6: temp = 11 + cambio_lectura; luminosidad = cambio_lectura + 10;  break;
156             case 7: temp = 11 + cambio_lectura; luminosidad = cambio_lectura + 50;  break;
157             case 8: temp = 11 + cambio_lectura; luminosidad = cambio_lectura + 60;  break;
158             case 9: temp = 12 + cambio_lectura; luminosidad = cambio_lectura + 70;  break;
159             case 10: temp = 13 + cambio_lectura; luminosidad = cambio_lectura + 70;  break;
160             case 11: temp = 14 + cambio_lectura; luminosidad = cambio_lectura + 80;  break;
161             case 12: temp = 15 + cambio_lectura; luminosidad = cambio_lectura + 90;  break;
162             case 13: temp = 16 + cambio_lectura; luminosidad = cambio_lectura + 100; break;
163             case 14: temp = 17 + cambio_lectura; luminosidad = cambio_lectura + 110; break;
```

Ilustración 30: Modificación firmware CoAP_CODE_REQ_GET

Donde la variable “cambio_lectura” será la que el sensor reciba procedente del programa principal por medio de la función PUT de CoAP: COAP_CODE_REQ_PUT.

```
95      case COAP_CODE_REQ_PUT:
96
97        // change the LED's state
98        if (msg->payload[0]==1) {
99           cambio_lectura = 1;
100       } else if (msg->payload[0]==2) {
101          cambio_lectura = 2;
102       } else if (msg->payload[0]==3) {
103          cambio_lectura = 3;
104       } else if (msg->payload[0]==4) {
105          cambio_lectura = 4;
106       } else if (msg->payload[0]==5) {
107          cambio_lectura = 5;
108       } else if (msg->payload[0]==6) {
109          cambio_lectura = 6;
110       } else if (msg->payload[0]==7) {
111          cambio_lectura = 7;
112       } else if (msg->payload[0]==8) {
113          cambio_lectura = 8;
114       } else if (msg->payload[0]==9) {
115          cambio_lectura = 9;
116       } else {
117          cambio_lectura = 0;
118       }
```

Ilustración 31: Modificación firmware CoAP_CODE_REQ_GET

De esta forma, el programa principal recibirá dos baterías de datos según la hora y dato del actuador cargado en la variable “cambio_lectura”.

Se presenta a continuación un resumen de los posibles datos que será generamos en la modificación del firmware, para el sensor de temperatura y para el sensor de luminosidad:

Temperatura	cambio_lectura									
Hora	0	1	2	3	4	5	6	7	8	9
0:00	16	17	18	19	20	21	22	23	24	25
1:00	14	15	16	17	18	19	20	21	22	23
2:00	13	14	15	16	17	18	19	20	21	22
3:00	12	13	14	15	16	17	18	19	20	21
4:00	12	13	14	15	16	17	18	19	20	21
5:00	11	12	13	14	15	16	17	18	19	20
6:00	11	12	13	14	15	16	17	18	19	20
7:00	11	12	13	14	15	16	17	18	19	20
8:00	11	12	13	14	15	16	17	18	19	20
9:00	12	13	14	15	16	17	18	19	20	21
10:00	13	14	15	16	17	18	19	20	21	22
11:00	14	15	16	17	18	19	20	21	22	23
12:00	15	16	17	18	19	20	21	22	23	24
13:00	16	17	18	19	20	21	22	23	24	25
14:00	17	18	19	20	21	22	23	24	25	26

15:00	18	19	20	21	22	23	24	25	26	27
16:00	19	20	21	22	23	24	25	26	27	28
17:00	20	21	22	23	24	25	26	27	28	29
18:00	19	20	21	22	23	24	25	26	27	28
19:00	18	19	20	21	22	23	24	25	26	27
20:00	17	18	19	20	21	22	23	24	25	26
21:00	16	17	18	19	20	21	22	23	24	25
22:00	15	16	17	18	19	20	21	22	23	24
23:00	15	16	17	18	19	20	21	22	23	24

Tabla 7: Posibles valores para el sensor de temperatura

luminosidad	cambio_lectura									
Hora	0	1	2	3	4	5	6	7	8	9
0:00	10	11	12	13	14	15	16	17	18	19
1:00	10	11	12	13	14	15	16	17	18	19
2:00	10	11	12	13	14	15	16	17	18	19
3:00	10	11	12	13	14	15	16	17	18	19
4:00	10	11	12	13	14	15	16	17	18	19
5:00	10	11	12	13	14	15	16	17	18	19
6:00	10	11	12	13	14	15	16	17	18	19
7:00	50	51	52	53	54	55	56	57	58	59
8:00	60	61	62	63	64	65	66	67	68	69
9:00	70	71	72	73	74	75	76	77	78	79
10:00	70	71	72	73	74	75	76	77	78	79
11:00	80	81	82	83	84	85	86	87	88	89
12:00	90	91	92	93	94	95	96	97	98	99
13:00	100	101	102	103	104	105	106	107	108	109
14:00	110	111	112	113	114	115	116	117	118	119
15:00	110	111	112	113	114	115	116	117	118	119
16:00	110	111	112	113	114	115	116	117	118	119
17:00	100	101	102	103	104	105	106	107	108	109
18:00	90	91	92	93	94	95	96	97	98	99
19:00	80	81	82	83	84	85	86	87	88	89
20:00	50	51	52	53	54	55	56	57	58	59
21:00	40	41	42	43	44	45	46	47	48	49
22:00	30	31	32	33	34	35	36	37	38	39
23:00	20	21	22	23	24	25	26	27	28	29

Tabla 8: Posibles valores para el sensor de luminosidad

Estos son los mapas de posibles valores que se va a recibir desde la aplicación principal, según la hora y según el parámetro que se cargue al actuadora. Esta tabla también servirá más adelante para la sección de pruebas para comprobar el correcto funcionamiento del sistema.

Por último, también es necesario modificar la aplicación openapps.c para inicializar cinfo.c y ponerlo en funcionamiento:

```
void openapps_init(void) {
   //-- 04-TRAN
   opencoap_init();     // initialize before any of the CoAP applications

   // CoAP
   //c6t_init();
   cinfo_init();
   cleds_init();
   //cjoin_init();
   cwellknown_init();
   //cstorm_init();
   //cexample_init();
   //rrt_init();

   // UDP
   //uecho_init();
   //uinject_init();
   //userialbridge_init();
   //uexpiration_init();
   //umonitor_init();
}
```

Ilustración 32: Inicialización en Openapps.c

3.3.6. Bloque central y aplicación en Python.

El programa principal se desarrollara en Python y será el encargado de llamar a cada módulo del sistema y de gestionar las peticiones, procesar las medidas y de enviar a la nube para su posterior representación.

El bloque central de procesamiento se puede ejecutar tanto en una plataforma Windows como en una plataforma Linux, siempre y cuando estén correctamente configuradas con las herramientas previamente descritas.

Se ha elegido Python por ser un lenguaje muy potente e intuitivo, es un lenguaje muy versátil y donde prácticamente todos los proveedores disponen de librerías para ser usadas en Python, como por ejemplo: Las librerías de CoAP están desarrolladas en Python, las librerías de la plataforma THETHINGS.IO, las librerías de twilio.com para el envío de SMS también están disponibles en Python. Además se dispone una gran cantidad de librerías y de repositorios libres. En resumen, es un lenguaje muy potente que facilita la integración de la mayoría de los sistemas.

Este TFM no es único trabajo sobre la demostración de una red IoT utilizando la plataforma OpenWSN y el estándar IEEE 802.15.4e. Existen más trabajos relacionados con esta área y desarrollados por estudiantes de la UOC.

Todos estos trabajos parten de la misma base que consiste en la demostración de una red IoT basada en OpenWSN y el estándar IEEE 802.15.4e. La mayor parte de los trabajos consisten en la medición de alguna magnitud, ya sea de corriente, humedad, temperatura, luminosidad o intensidad de señales de RF y su presentación en la plataforma THETHINGS.IO.

En este trabajo fin de master es similar en cuanto a contenido, el estudio de los diferentes estándares, el concepto de IoT, el estudio de las diferentes plataformas de cloud computing y el estudio y puesta a punto del equipo para el uso de OpenWSN es simular a otros trabajos propuestos. Se adjunta los enlaces de trabajos similares:

http://openaccess.uoc.edu/webapps/o2/bitstream/10609/58906/7/pry26dTFM0117memoria.pdf

http://openaccess.uoc.edu/webapps/o2/bitstream/10609/64346/3/rmoragoTFM0617memoria.pdf

http://openaccess.uoc.edu/webapps/o2/bitstream/10609/64505/4/manmarsalTFM0617memoria.pdf

Por lo tanto, en este trabajo fin de master se propone como mejora el desarrollo de un software, como bloque central, para el control del sistema que sea:

- Modulable y configurable para que se puedan añadir o quitar sensores sin necesidad de modificar el código fuente.

- Protección ante posibles indisponibilidades de la red.

- Procesado previo basado en “edge computing” para reducir los paquetes enviados a la nube.

- Añadir la opción de que el usuario pueda actuar en el sistema.

Se presenta a continuación la estructura que tiene el sistema completo a nivel de aplicaciones software:

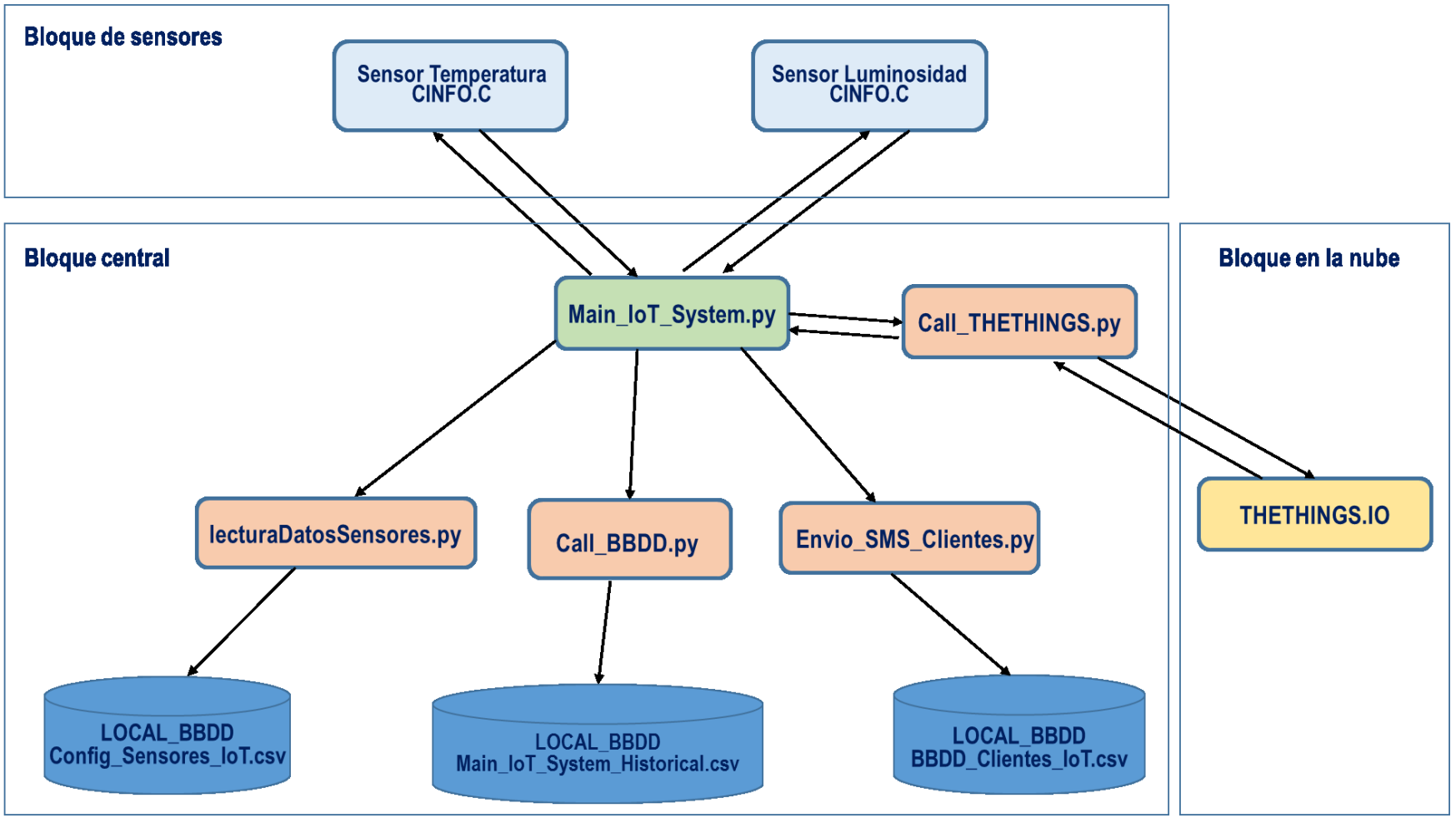


Ilustración 33: Esquema sistema IoT

El bloque de procesamiento central está formado por:

- 5 módulos en Python.
- 3 bases de datos.

♦ lecturaDatosSensores

La idea de la implementación de este módulo es la de dotar al sistema de cierta flexibilidad a la hora de añadir o quitar sensores a la red.

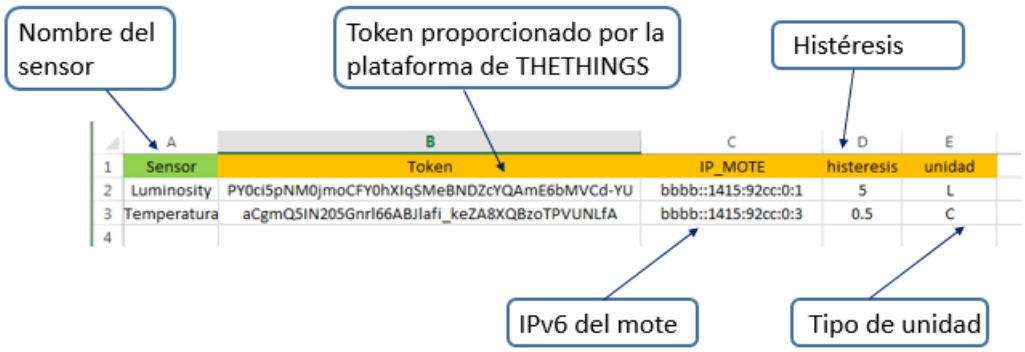
Este módulo, es capaz de leer un archivo de configuración de todos los sensores conectados a la red. De esta forma, si se desea añadir un sensor a la red solo es necesario indicárselo a la base de datos de configuración de sensores.

Esta base de dato está diseñada como un archivo .csv donde en cada línea del mismo se especifican los parámetros necesario de cada que funcione dentro del programa principal.

```
8   #Base de datos con los datos de los sensores, tipo, token, ip del mote e histeresis
9
10  def lecturaDatosSensores ():
11      file = 'Config_Sensores_IoT.csv'
12      respuesta = []
13      n=0
14
15      try:
16          reader = csv.reader(open(file, 'rb'))
17
18          for row in (reader):
19
20              sensorNombre,Token,IP_MOTE,histeresis,unidad = row[0],row[1],row[2],row[3],row[4]
21              sensorLista = [sensorNombre,Token,IP_MOTE,histeresis,unidad]
22
23              if sensorNombre != "Sensor" :
24
25                  respuesta.insert(n,sensorLista)
26                  n=n+1
27
28          return respuesta
29
30
31      except:
32          print "\033[;31m"+"\n### ERROR: No se encuentra la BBDD de los sensores. --> Config_Sensores_IoT.csv <--  ####"
33          print  "\n\033[;37m"
34          sys.exit()
35
```

Ilustración 34: lecturaDatosSensores

El módulo se encargará de leer la base de datos “Config_Sensores_IoT.csv”, los datos necesarios para dar de alta un sensor son los siguientes:



	A	B	C	D	E
1	Sensor	Token	IP_MOTE	histeresis	unidad
2	Luminosity	PY0ci5pNM0jmoCFY0hXIqSMeBNDZcYQAmE6bMVCd-YU	bbbb::1415:92cc:0:1	5	L
3	Temperatura	aCgmQ5IN205Gnrl66ABJlafi_keZA8XQBzoTPVUNLfA	bbbb::1415:92cc:0:3	0.5	C
4					

Ilustración 35: Config_Sensores_IoT.csv

Cada línea representa un sensor, en él se debe de indicar el nombre del sensor que debe de coincidir con el especificado en la plataforma THETHINGS.IO. El token que es facilitado también por la plataforma THETHINGS. La IPv6 que es la ip del mote (en este caso simulado). El tipo de unidad que se va a medir y por último un dato histéresis que es dado también por el usuario y que dependerá del tipo de magnitud que se va a medir.

Aunque se comentará más adelante, el dato histéresis sirve para decidir cuando el dato tiene que ser enviado a la nube y cuando no.

- Call_BBDD

Esta subfunción se encarga de almacenar localmente todas las medidas que se requieran.

```
4
5   def call_BBDD(temp, luz):
6       fecha = time.strftime('%d %b %y')
7       hora=time.strftime('%H:%M:%S')
8       try:
9           f = open ("Main_IoT_System_Historical.csv", "a")
10          f.write(fecha + " " + hora + ",El sensor:," + temp + ",registra:," + luz + "\n")
11          f.close()
12          print "    --> Lecturas almacenadas en BBDD local "
13
14      except:
15          print "\033[;31m" + "\n%%% ERROR: No se ha podido abrir la BBDD Main_IoT_System_Historical %%%\n"
16          print "\n\033[;37m"
```

Ilustración 36: Call_BBDD

La estructura de la subfunción es sencilla y almacenará la hora de la muestra, el sensor que esta almacenando la muestra y la lectura que haya tomado el sensor en ese momento. Los datos se almacenarán en Main_IoT_System_Historical.csv. Si no se puede abrir o crear el archivo se notificará por consola.

	A	B	C	D	E
1	Fecha	*	Sensor	*	Lectur
2	01/05/2018 9:34	El sensor:	Luminosity	registra:	70
3	01/05/2018 9:34	El sensor:	Temperatura	registra:	12
4	01/05/2018 9:35	El sensor:	Luminosity	registra:	77
5	01/05/2018 9:35	El sensor:	Temperatura	registra:	19
6	01/05/2018 9:37	El sensor:	Luminosity	registra:	77
7	01/05/2018 9:40	El sensor:	Luminosity	registra:	77
8	01/05/2018 9:40	El sensor:	Temperatura	registra:	19
9	01/05/2018 9:45	El sensor:	Luminosity	registra:	77
10	01/05/2018 9:45	El sensor:	Temperatura	registra:	19
11	01/05/2018 9:53	El sensor:	Luminosity	registra:	77
12	01/05/2018 10:00	El sensor:	Luminosity	registra:	77
13	01/05/2018 10:00	El sensor:	Temperatura	registra:	19
14	01/05/2018 10:01	El sensor:	Luminosity	registra:	77
15	01/05/2018 10:01	El sensor:	Temperatura	registra:	20
16	01/05/2018 10:10	El sensor:	Luminosity	registra:	77
17	01/05/2018 10:10	El sensor:	Temperatura	registra:	20
18	01/05/2018 10:15	El sensor:	Luminosity	registra:	77

Ilustración 37: Main_IoT_System_Historical.csv

- Envio_SMS_Clientes()

  Con esta subfunción se implementa la posibilidad de enviar un SMS a una lista de clientes predeterminada. De esta forma, se puede avisar al cliente de un exceso en alguna de magnitudes medidas. En este caso, se avisará al cliente de un exceso de temperatura.

  Este tipo de mecanismos es muy interesante para situaciones críticas que requieran de una actuación inmediata. Una de las posibles aplicaciones es por ejemplo el exceso de temperatura que puede avisar cuando se exceda el límite de funcionamiento electrónico de un sistema o incluso avisar de un posible incendio. También tiene otras muchas posibilidades, como exceso de humedad, nivel de agua, etc.

  La idea seguida es la comentada anteriormente. Se desarrolla una base de datos que almacena los datos de los clientes que serán avisados en caso de detectar un exceso de magnitud (puede ser superior o inferior). Gracias a esta base de datos no es necesario tocar el código fuente cuando se quieran añadir, quitar o modificar clientes. Igualmente, los datos de los clientes son privados, no se muestran en el código y pueden ser administrados solo por el personal autorizado. Como posible mejora que no ha sido implementada se puede añadir a este tipo de archivos un cifrado para mayor privacidad.

Para utilizar este tipo de servicios se han utilizado las APIs que proporciona Twilio para Python. Con estas APIs se puede se puede realizar el envío de SMS fácilmente.

```
import os
import sys
import csv
from twilio.rest import Client


#Base de datos con los datos de los clientes y el mensaje a enviar

def Envio_SMS_Clientes (temp):
    file = 'BBDD_Clientes_IoT.csv'

    try:
        reader = csv.reader(open(file, 'rb'))


        for row in (reader):

            account,token,to_mens,from_mens,mensaje = row[0],row[1],row[2],row[3],row[4]

            if account != "account" :

                client = Client(account, token)
                mensaje_send = mensaje + str(temp)
                message = client.messages.create(to=to_mens, from_=from_mens,
                                 body=mensaje_send)
                print "    --> Aviso!. Se ha enviado un SMS de alerta al cliente: " + to_mens

    except:
        print "\033[;31m"+"\n### ERROR: No se encuentra la BBDD de los clientes ####"
        print "\n\033[;37m"
```

Ilustración 38: Envio_SMS_Clientes

De esta forma se podrá enviar un SMS al cliente especificado en el archivo BBDD_Clientes_IoT.csv. Este archivo tiene la siguiente forma y en donde cada línea representa a un usuario:

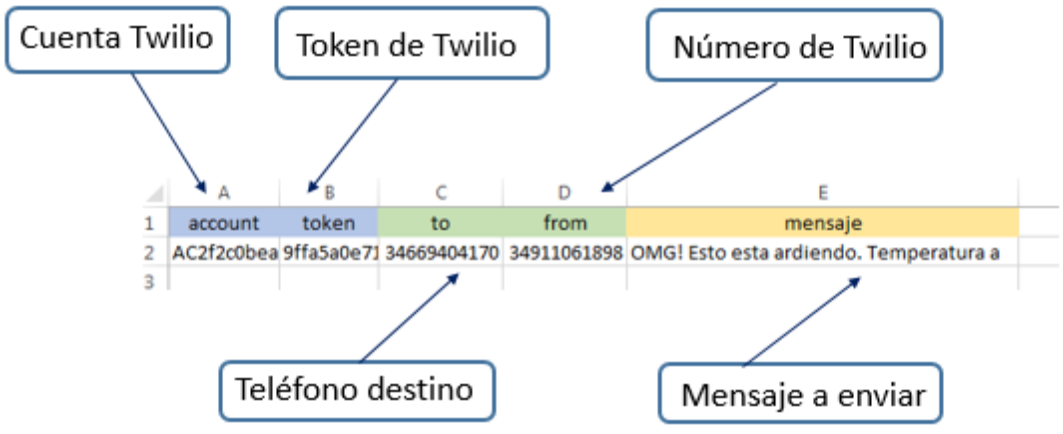


Ilustración 39: BBDD_Clientes_IoT.csv

- Call_THETHINGS

Call_THETHINGS es el módulo encargado de la comunicación con la plataforma THETHINGS. Para trabajar con la plataforma THETHINGS también ha sido necesario instalar las APIs que proporciona la plataforma en lenguaje Python. El módulo está formado por dos funciones, una para enviar datos a la plataforma y la otra para recibir datos desde la plataforma, estos datos a recibir, son los datos del actuador que servirán para demostrar la comunicación entre la plataforma y los sensores.

```
from thethingsAPI import ThethingsAPI
from random import randrange


# Received
def recibir_TT (sensor, TheThingToken):
    thethings = ThethingsAPI(TheThingToken)
    activador = thethings.read(sensor)
    activador_text = str(activador)
    comienzo = activador_text.find("u'value': u'")
    response = activador_text[comienzo +12:comienzo + 13]
    return response

# Send data to THETHING.IO
def escribir_en_TT(sensor, valor, TheThingToken):
    thethings = ThethingsAPI(TheThingToken)
    thethings.clear()
    thethings.addVar(sensor, valor)
    thethings.write()
    print "    --> Comunicacion con THETHING. Se envia " + str(valor) + " al sensor: " + str(sensor) + "."
```

Ilustración 40: Send_THETHINGS

- Main_IoT_System

El programa principal “Main_IoT_System.py”, será el encargado de hacer las llamadas a los diferentes módulos, gestionando cuando hay que solicitar información al sensor, de procesar esta información, de decidir cuándo transmitir esta información a la nube, cuando se almacena o cuando hay que notificar a un cliente por SMS.

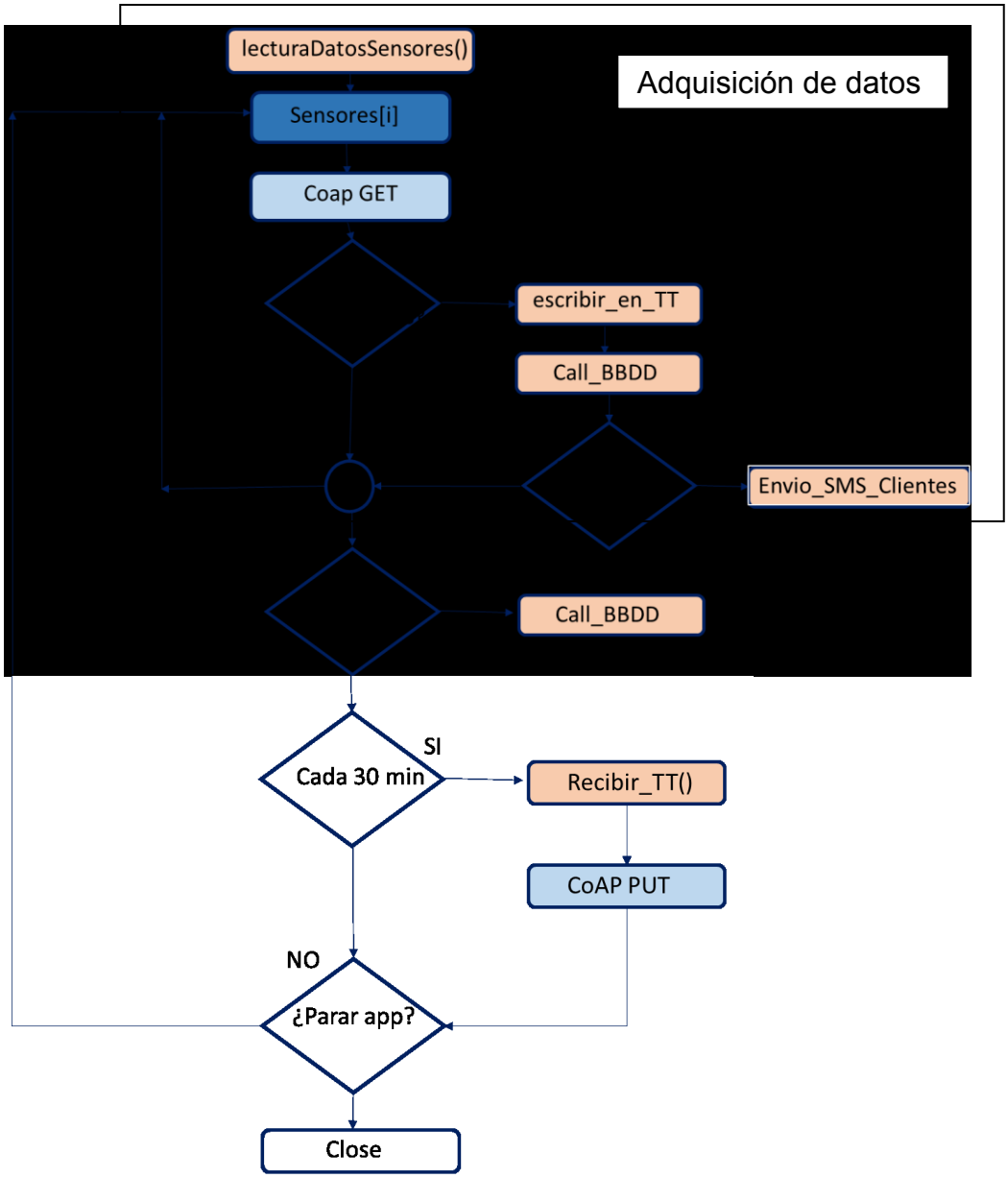


Ilustración 41: Flujograma Main_IoT_System.py

En la ilustración 14 se muestra como es el flujograma del módulo Main_IoT_System. La estructura del programa se puede dividir en dos partes fundamentales:

La primera parte, es la parte de adquisición de datos de los sensores. Para ello, el programa realiza una consulta de los sensores definidos en Config_Sensores_IoT.csv y creará una vector de sensores. El programa recorrerá este vector de sensores secuencialmente para realizar consultas de las medidas de cada sensor. Para realizar las lecturas de las medidas de los sensores, se realizará por medio del protocolo CoAP. CoAP como ya se ha comentado dispone de diversas funciones para la adquisición de datos, en este caso para obtener datos, se usa la función GET.

Una vez adquirida la lectura del sensor en cuestión, el programa decidirá si es un dato relevante o no.

Es una de las partes más importantes del programa. Con la introducción de este mecanismo se intenta que solo se suban a la nube las lecturas importantes reduciendo así el ancho de banda consumido y el número de llamadas a la API de THETHINGS. Esta es una de las mejoras que se propone en este TFM respecto a otros trabajos y que persigue la implementación del concepto Edge Computing. Con este sencillo mecanismo, el programa principal puede descartar las lecturas menos relevantes y reducir el ancho de banda requerido. Aunque para dos sensores el uso del ancho de banda no es crítico, para una red de sensores grande el uso de este tipo de procesado es crítico ya que de otra forma se podría hasta llegar a saturar el ancho de banda disponible.

Por otro lado, la mayoría de las plataformas online están sujetas a un uso limitado, por lo que el número de llamadas a este tipo de aplicaciones tiene que ser limitado y controlado.

El mecanismo consiste en aplicar una histeresis entre lecturas sucesivas. Por ejemplo, para el sensor de temperatura, el sistema comprobará si la lectura obtenida en mayor o menor a la lectura anterior y si lo es con una histeresis de +/- 0.5 ºC. En este caso, se procederá a llamara al módulo que se encarga de transmitir información a THETHINGS y al módulo que se encarga de almacenarlo en BBDD en local y si supera un cierto umbral notificarlo vía SMS a la lista de clientes facilitada en BBDD_Clientes_IoT.csv. De esta forma, se consigue tener siempre actualizada la temperatura en la plataforma online pero con un uso limitado de ella.

Con este sencillo mecanismo, conseguimos reducir el número de llamadas a la API de THETHING, reduciendo así el uso del ancho de banda de la conexión a internet y el consumo de llamadas API. Por lo tanto, solo se realizarán llamadas importantes y cuando las lecturas de los sensores varíen.

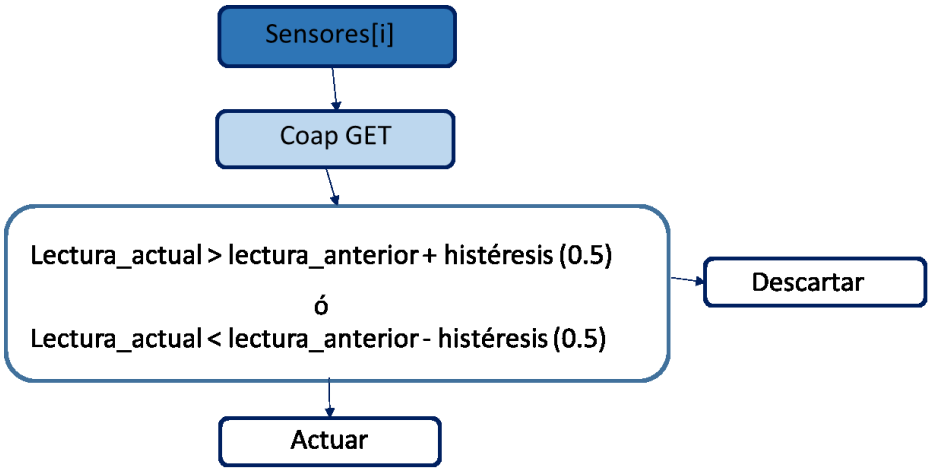


Ilustración 42: Flujograma elección lectura

La segunda parte, es la parte propia del actuador, donde se solicitará a la plataforma de THETHINGS el estado de este actuador para enviar luego este dato al bloque de sensores.

En esta parte, existe una limitación importante ya que no se sabe del dato que está cargado en la plataforma de THETHINGS hasta que no se hace una consulta a la plataforma. La limitación es básicamente que no se puede estar realizando peticiones constantemente a la API de THETHINGS. Por lo tanto, las consultas a la plataforma THETHINGS se realizarán cada 30 minutos. El nuevo dato obtenido de la plataforma THETHINGS se enviará al bloque de sensores mediante la función PUT del protocolo CoAP. Este dato se cargará en la variable "cambio_lectura" ya introducida anteriormente.

Los posibles valores del actuador van del 0 al 9, y modificarán las lecturas de los sensores según lo especificado en las tablas 6 y 7.

Se reserva el 9 como valor para terminar la aplicación y poder así apagar la aplicación en remoto.

Se facilita el código del Main_IoT_System como anexo I debido a su extensión.

3.3.7. Bloque en la nube. Plataforma THETHINGS.IO

THETHING.IO es una plataforma donde se puede almacenar los datos transmitidos por los sistemas IoT y supervisar los mismos.

Para ello es necesario realizar un registro en el siguiente URL: https://theThings.io/.

El registro es sencillo requiere de unos datos básicos:

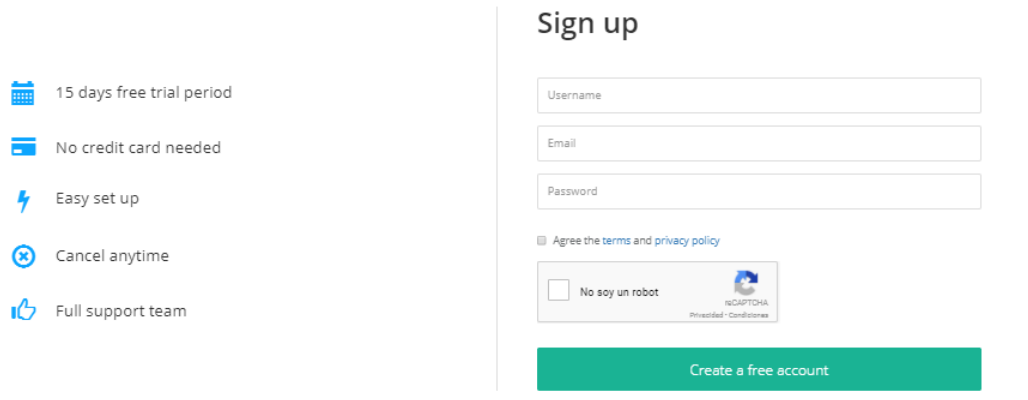


Ilustración 43: Registro THETHINGS.IO

La cuenta dispone de 15 días de prueba después de los cuales la cuenta caducará pasando a ser un servicio de pago. Los precios oscilan entre los 29 euros mensuales a los 499 euros mensuales según los requerimientos del sistema. Se muestra a continuación las características de cada solución.

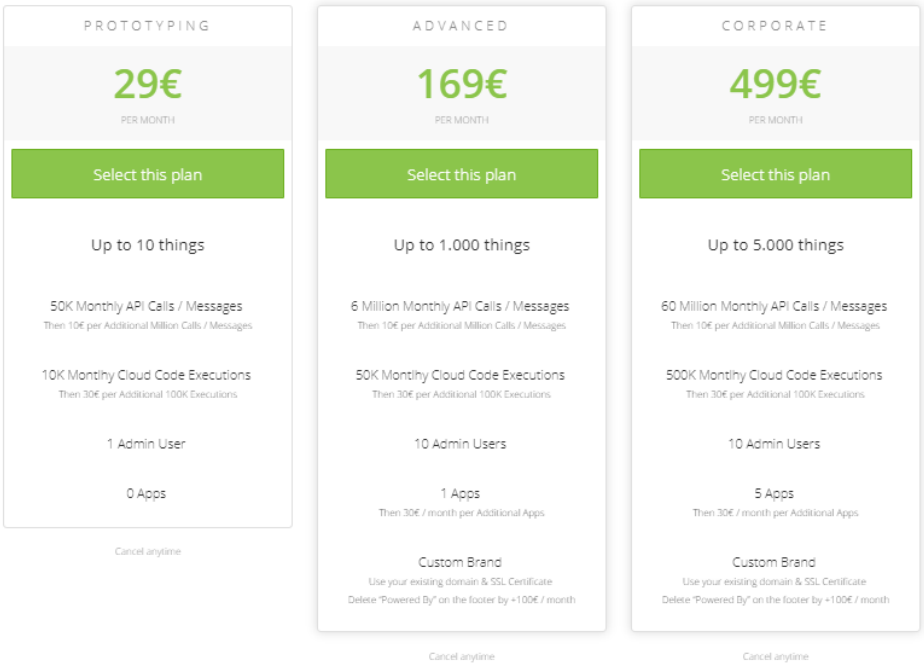


Ilustración 44: Cuenta THETHINGS.IO

Los datos son enviados desde la aplicación en Python que transmite la información haciendo uso de las APIs de THETHINGS. Dentro de la plataforma THETHINGS.IO se configuraran los diferentes dispositivos que irán recibiendo información periódicamente.

La plataforma THETHINGS.IO proporciona el módulo Python necesario para la comunicación del dispositivo con la plataforma:

```
#!/usr/bin/python

from theThingsAPI import TheThingsAPI

tt = TheThingsAPI("YourThingToken")

# Read example
response = tt.read("yourVariable")
print 'value: ' + response[0]['value'] + ' date: ' + response[0]
['datetime']

# write example
tt.addVar('YourVariable_1', 'YourValue_1')
tt.addVar('YourVariable_2', 'YourValue_2')
tt.write();
```

Ilustración 45: THETHINGS.IO módulo Python

Dentro del menú “Things”, se creará un nuevo producto que a su vez funcionará como contenedor de los “Things” que se quieran conectar a la plataforma. La suscripción “Prototyping” que oferta THETHINGS proporciona la posibilidad de activar tres “THINGS”

Ilustración 46: Creación de un nuevo “Thing”

Para el desarrollo de este TFM, como ya se ha comentado anteriormente, se ha propuesto la creación de tres “Things”; Uno para la adquisición de datos de un sensor que mida luminosidad; Otro para un sensor de temperatura; Y por último un “Thing” que sea capaz de funcionar como un actuador, de modo que, cuando a este “Thing” se le cargue un valor predefinido desde la plataforma THETHINGS, el sistema principal pueda recibir esta orden y activar la cualquier función deseada.

Desde el menú “dashboard” se puede configurar como queremos presentar la información transmitida desde la red de sensores. Se pueden presentar gráficas con históricos, valores en tiempo real, mapas y valores que se quiere transmitir a la red, entre otras muchas opciones.

A continuación, se muestra el ejemplo de cómo sería la creación de un nuevo elemento o “Widget”. Para este ejemplo, se está configurando un histórico de medidas de un sensor de temperatura que se representará por medio de líneas.

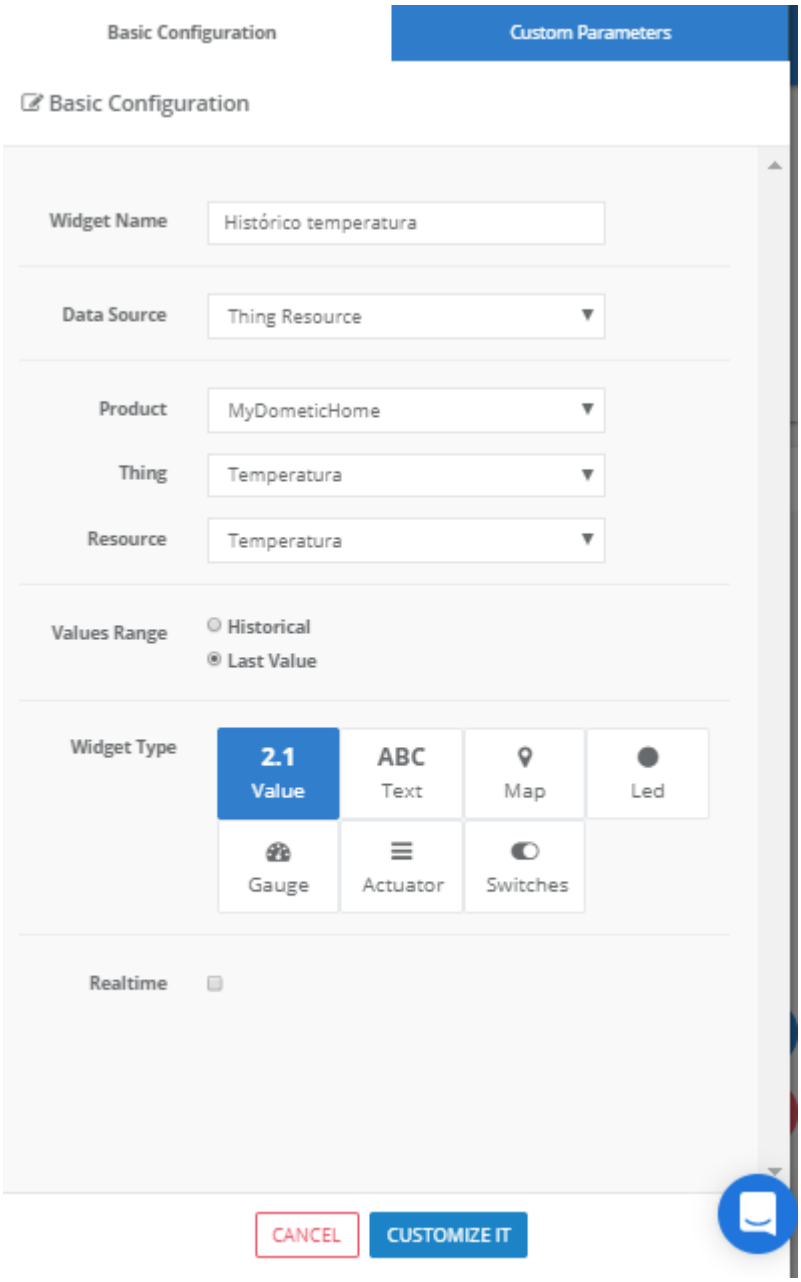


Ilustración 47: Configuración dashboard

A parte se ha creado un widget que muestre siempre la última lectura del sensor, de esta forma se tendrá a la vista la lectura actual del sensor y el histórico del mismo. Se ha configurado un widget análogo para el sensor de luminosidad.

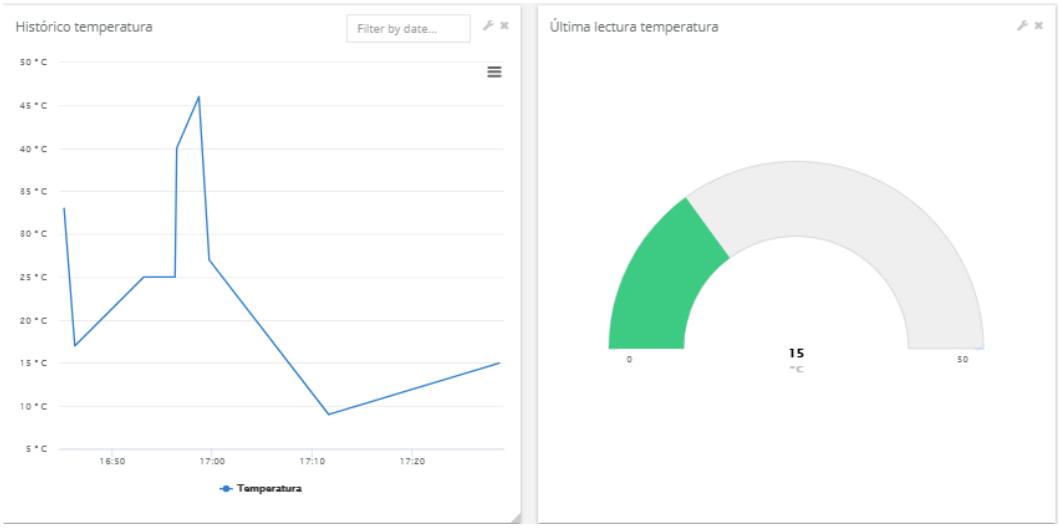


Ilustración 48: Configuración dashboard temperatura

Por otra parte, se ha configurado también un actuador, que servirá para enviar instrucciones al sistema de forma predeterminada. A este actuador se le puede cargar un dato que posteriormente el programa principal leerá mediante la función API "read" proporcionada por THETHINGS.

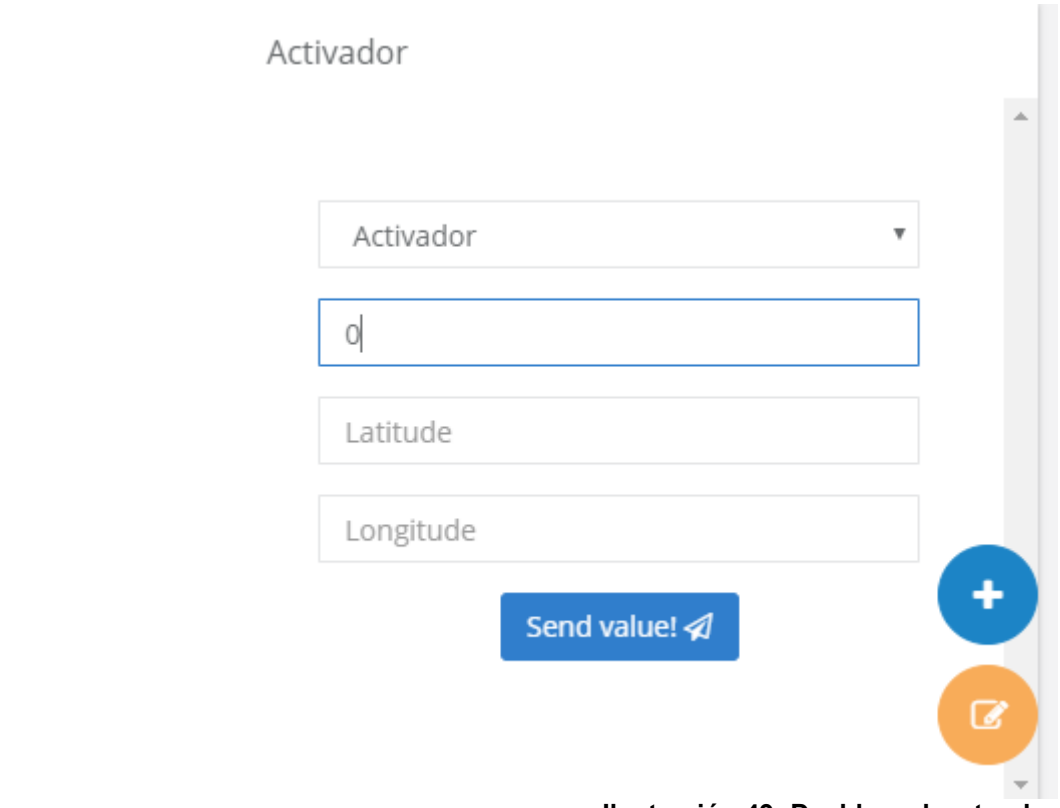


Ilustración 49: Dashboard, actuador

De esta forma, se le podrá enviar el valor 0 al "thing" Activador. Esta opción se puede utilizar para encender (1) o apagar (0) un switch o una válvula o para alguna funcionalidad un poco más compleja que detecte varias acciones dependiendo del valor facilitado. En este caso se

incrementará o reducirá la magnitud de las muestras simuladas y se controlará el apagado de la aplicación con el valor 9.

4. Verificación del sistema

4.1. Introducción

En este capítulo se presentan las pruebas y verificaciones que se han llevado a cabo para comprobar el correcto funcionamiento del sistema. Se presentarán las comprobaciones de comunicación del sistema, la adquisición de datos y la representación de los mismos en la plataforma THETHINGS.IO.

4.2. Verificación formación de la red con OpenWSN

El funcionamiento del sistema se basa en la creación de una red simulada con OpenWSN. Una vez inicializado el sistema y establecido el nodo 2 como DAGroot, se comprueba como el sistema comienza a transmitir y recibir paquete RPL DAO. Estos paquetes se reciben desde los dos nodos predefinidos de la red. Los motes estarán indicados por el último dígito de la cadena bbbb:0:0:0:1415:92cc:0:**X**. En la siguiente figura se muestra como se comienza a recibir estos paquetes:

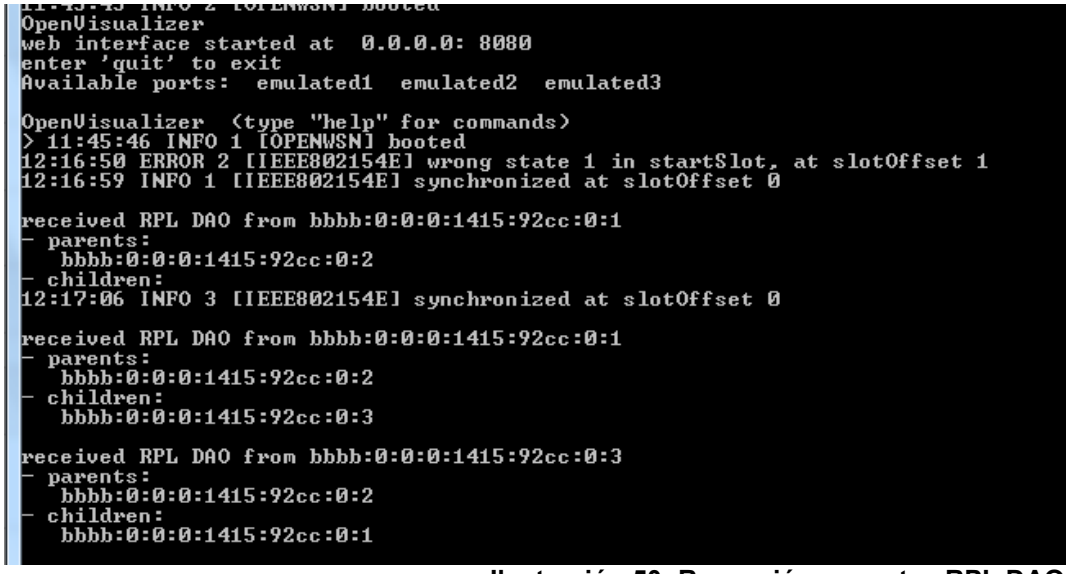


Ilustración 50: Recepción paquetes RPL DAO

Para comprobar el correcto funcionamiento de la red también es necesario comprobar mediante un ping, la comunicación de los motes, para ello solo es necesario hacer un ping a la IPv6 correspondientes:

```
C:\Users>ping bbbb:0:0:0:1415:92cc:0:1

Haciendo ping a bbbb::1415:92cc:0:1 con 32 bytes de datos:
Respuesta desde bbbb::1415:92cc:0:1: tiempo=46ms
Respuesta desde bbbb::1415:92cc:0:1: tiempo=73ms
Respuesta desde bbbb::1415:92cc:0:1: tiempo=77ms
Respuesta desde bbbb::1415:92cc:0:1: tiempo=82ms

Estadísticas de ping para bbbb::1415:92cc:0:1:
    Paquetes: enviados = 4, recibidos = 4, perdidos = 0
    (0% perdidos),
Tiempos aproximados de ida y vuelta en milisegundos:
    Mínimo = 46ms, Máximo = 82ms, Media = 69ms

C:\Users>
```

Ilustración 51: Respuesta ping a motes

Otra comprobación que es necesario hacer es la de comprobar mediante OpenVisualizer si la red se ha formador correctamente. Se puede hacer directamente desde el entorno de OpenVisualizer. En la figura que se muestra a continuación se puede observar cómo se han inicializado los motes 1 y 3 y el nivel de seña recibida.

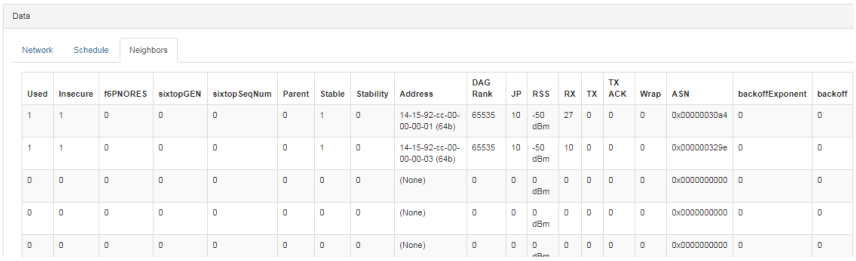


Ilustración 52: Inicialización red OpenWSN

Se muestra también una captura de Wireshark de la red OpenWSN, donde se pueden observar los diferentes paquetes de tráfico de la red. Se muestra en un recuadro rojo un paquete del protocolo CoAP, en concreto de la función GET, se muestra también el valor devuelto por el protocolo, 19, que es el dato que se espera recibir.

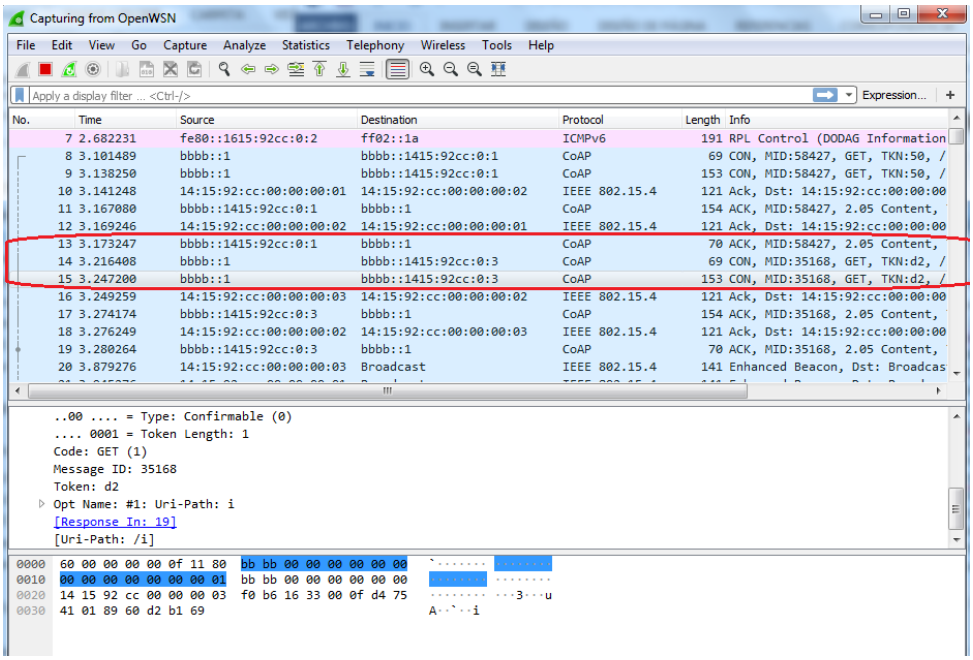


Ilustración 53: Captura Wireshark de la red OpenWSN

4.3. Verificación comunicación bloque central y bloque de sensores

Una vez inicializada la red y hechas las comprobaciones correspondientes, es necesario comprobar que se están recibiendo correctamente los datos desde el programa principal (Main_IoT_System.py).

Para esta comprobación, se ha optado por mostrar por consola los datos recibidos desde los sensores, indicando la IPv6 desde donde se hace la consulta y mostrando el dato capturado.

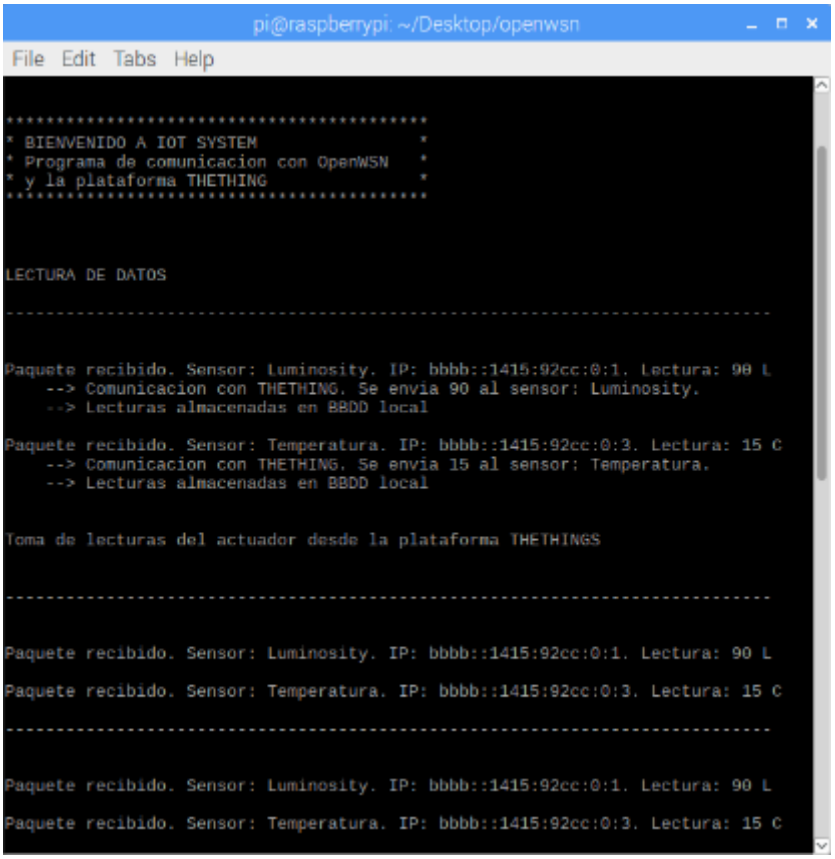


Ilustración 54: Ejecución Main_IoT_System

Se indica también mediantes el símbolo “-->” las llamadas a los diferentes subfunciones del sistema. Cada iteración está separada con los indicadores “----“. En la ilustración 53 se muestra como en el primer ciclo se ha medida 90 L y 15 Cº para el sensor de luminosidad y el sensor de temperatura respectivamente, como es la primera iteración y no ha lecturas anteriores estas lecturas se envían a la plataforma THETHINGS y se almacenan en la BBDD local, seguidamente y puesto que es un ciclo de inicialización se toma el dato que hay cargado en el actuador de la plataforma THETHINGS.

En la siguiente iteración, como las lecturas tomadas de los sensores de luminosidad y de temperatura son las mismas que había

antes, estas lecturas se muestran por consola pero no se almacenan ya que son iguales que las anteriores y no son relevantes.

En el siguiente caso, se ha simulado un corte de transmisión entre los sensores y el módulo principal. Una de las características del diseño es que el sistema añade protecciones y correcciones frente a errores y puede continuar la ejecución aunque se presenten algunos problemas de comunicación entre los sensores y el programa principal.

Ilustración 55: Protección ante errores de comunicación

En la ilustración 54 se presenta un ejemplo de corte de comunicación entre los sensores y el programa principal. En este ejemplo se muestra como la aplicación puede continuar funcionando durante el corte y después del corte sin necesitad de reiniciar la misma.

4.4. Verificación comunicación bloque central con THETHINGS.

Una vez comprobado que la comunicación entre los sensores y el bloque central es correcta, se pasa a comprobar la comunicación entre el bloque central y la plataforma THETHINGS.IO. Para ello se presentan las gráficas obtenidas en la plataforma THETHINGS.IO y se comprueba que los datos recibidos efectivamente son los que se esperan obtener.

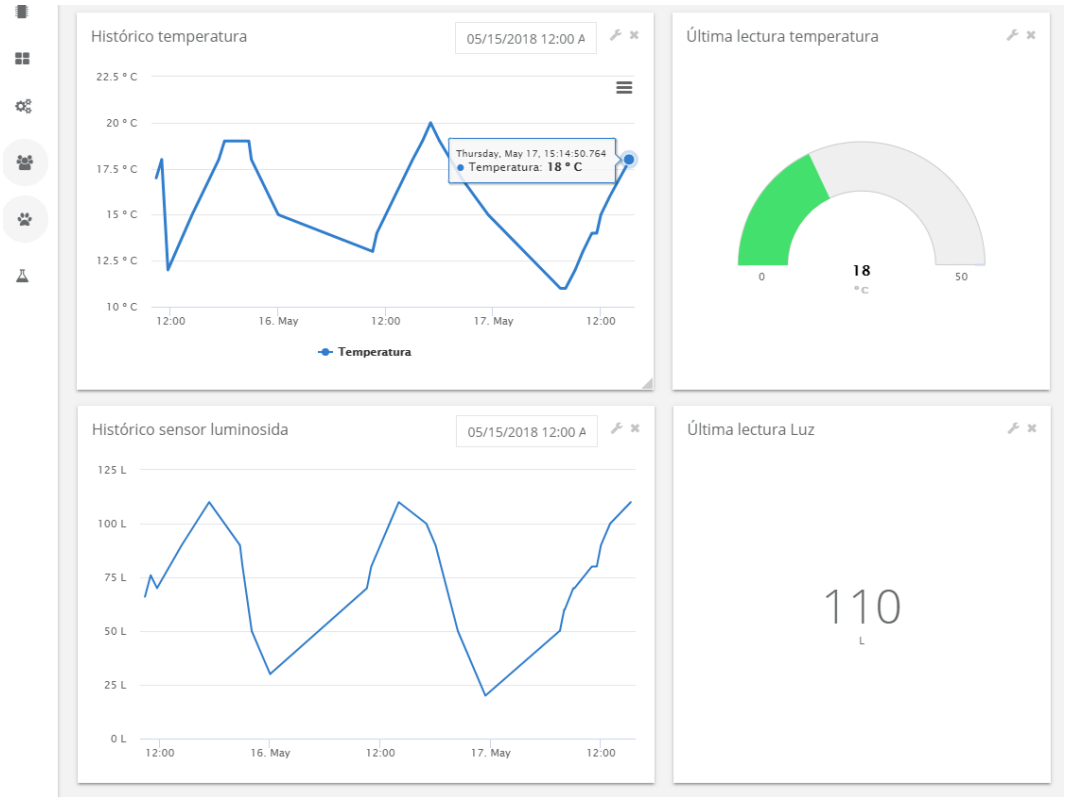


Ilustración 56: Resultados en theThings.io

Se muestra el resultado de aplicar una instrucción desde el actuador en la plataforma THETHINGS y como esta es recibida por el programa principal.

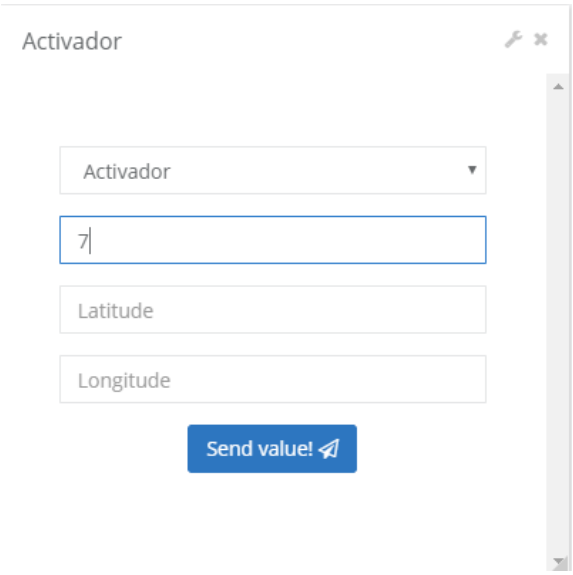


Ilustración 57: Actuador. Cambio de valor

Esta acción es recibida por el bloque central que la transmite al bloque de sensores aplicando un cambio en las medidas obtenidas. Este resultado se presenta a continuación:

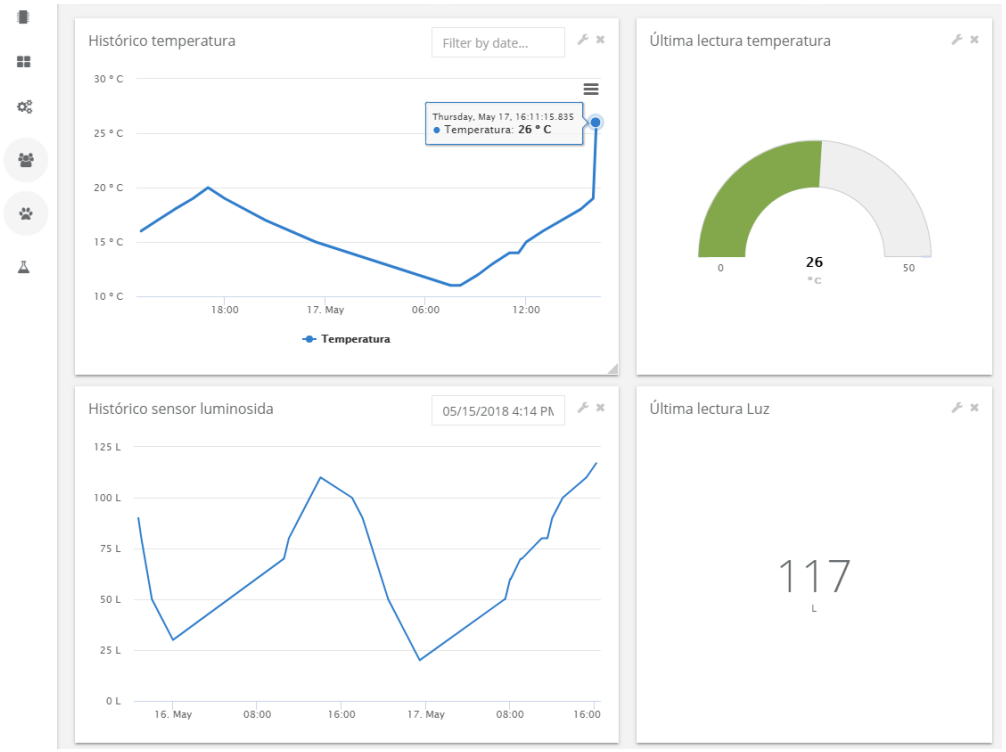


Ilustración 58: Resultados acción actuador

A su vez este pico de temperatura provocará que se active la notificación vía SMS utilizando las API de Twilio

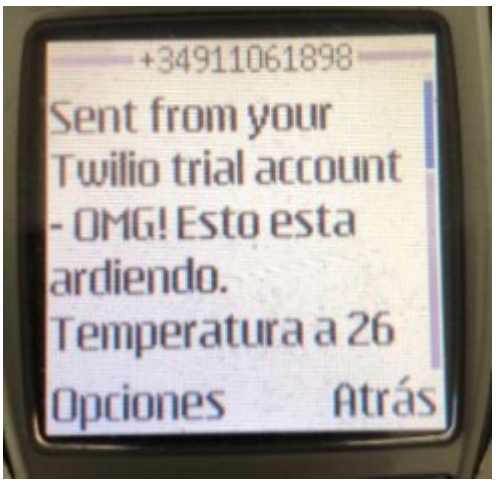


Ilustración 59: Aviso vía SMS. Twilio

Se muestra una captura de Wireshark, en este caso con la comunicación entre el bloque central y la plataforma THETHINGS.IO

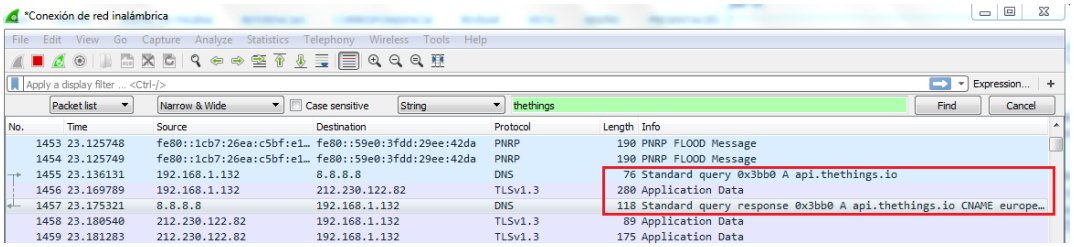


Ilustración 60: Captura Wireshark conexión con internet

4.5. Comparación resultados Fog Computing vs Cloud Computing

En este apartado se presenta la comparativa de aplicar las dos soluciones de "computing" propuestas. Se presentan varias características a tener en cuenta, a la hora de decidir entre fog o cloud computing.

- **Latencia:** Es el retardo que existe entre que el bloque central envía los datos a la plataforma THETHINGS.IO. Este retardo es de aproximadamente 29ms y aunque depende de la conexión contratado con el proveedor de internet (ISP) sobre todo depende del servidor en la nube y del servicio contratado. Dependiendo del tipo de aplicación que se diseñe este retardo es más o menos crítico. Este tiempo es crítico para sistemas en tiempo real, siendo 29ms un retardo bueno.

```
C:\Users>ping thethings.io

Haciendo ping a thethings.io [104.199.94.203] con 32 bytes de datos:
Respuesta desde 104.199.94.203: bytes=32 tiempo=29ms TTL=60
Respuesta desde 104.199.94.203: bytes=32 tiempo=29ms TTL=60
Respuesta desde 104.199.94.203: bytes=32 tiempo=29ms TTL=60
Respuesta desde 104.199.94.203: bytes=32 tiempo=29ms TTL=60

Estadísticas de ping para 104.199.94.203:
    Paquetes: enviados = 4, recibidos = 4, perdidos = 0
    (0% perdidos),
Tiempos aproximados de ida y vuelta en milisegundos:
    Mínimo = 29ms, Máximo = 29ms, Media = 29ms
```

Ilustración 61: Latencia con theThings.io

- **Seguridad:** En soluciones Fog Computing, el procesado se realiza en local por lo que puede aplicar una protección extra a los datos enviar a la nube, por lo que existe un nivel de seguridad superior. En cloud computing los datos se envían tal y como se reciben por lo que existe esta protección a la hora de enviar los mensajes.

- **Disponibilidad:** Generalmente las plataformas Cloud suelen estar más disponibles que las plataformas locales aunque es cierto que una indisponibilidad de una plataforma Cloud no es escalable, mientras que en las plataformas de Fog Computing es posible dar una solución que permita volver a poner en funcionamiento el sistema. Es una

ventaja por lo tanto disponer un uso de Fog Computing donde los datos siempre estarán accesible.

- **Uso del ancho de banda:** Tal y como se muestra en la ilustración 56, la comunicación entre el bloque central y THETHINGS.Io se realiza por medio de la API que proporciona THETHINGS.IO. Esta comunicación está formada por 2 paquetes con un tamaño de: 76 bytes y 118 bytes unos 194 bytes en total. Para una red pequeña no es un tráfico muy elevado pero al aumentar el número de sensores el consumo de ancho de banda de la red se verá incrementado, pudiendo llegar hasta a saturar la conexión ISP.

  Con la propuesta de procesamiento en local de las lecturas recibidas se consigue una notable reducción del uso del ancho de banda. Los resultados obtenidos se muestran en la tabla 8.

	Cloud Computing					Fog Computing	
Nº Sensores	1 segundo (b)	1 minuto (kb)	1 hora (kb)	1 día (Mb)	1 día (Gb)	40 llamadas API - (Mb)	Reducción de tráfico
1,00	196,00	11,48	689,06	16,15	0,02	0,01	99,95%
2,00	392,00	22,97	1378,13	32,30	0,03	0,01	99,95%
3,00	588,00	34,45	2067,19	48,45	0,05	0,02	99,95%
4,00	784,00	45,94	2756,25	64,60	0,06	0,03	99,95%
5,00	980,00	57,42	3445,31	80,75	0,08	0,04	99,95%
6,00	1176,00	68,91	4134,38	96,90	0,09	0,04	99,95%
7,00	1372,00	80,39	4823,44	113,05	0,11	0,05	99,95%
8,00	1568,00	91,88	5512,50	129,20	0,13	0,06	99,95%
9,00	1764,00	103,36	6201,56	145,35	0,14	0,07	99,95%
10,00	1960,00	114,84	6890,63	161,50	0,16	0,07	99,95%
100,00	19600,00	1148,44	68906,25	1614,99	1,58	0,75	99,95%
1000,00	196000,00	11484,38	689062,50	16149,90	15,77	7,48	99,95%
10000,00	1960000,00	114843,75	6890625,00	161499,02	157,71	74,77	99,95%
100000,00	19600000,00	1148437,50	68906250,00	1614990,23	1577,14	747,68	99,95%

Tabla 9: Tráfico Cloud Computing vs Fog Computing

- **Coste:** En cuando a coste los beneficios de utilizar Fog Computing son notables. Se reduce el ancho de banda utilizado que para redes grandes es más crítico y la reducción de uso de plataformas online también ayuda a reducir estos costes ya que estas plataformas cobran en función de la utilización y del número de dispositivos conectados a ella.

5. Conclusiones y líneas futuras

5.1. Lecciones aprendidas

El objetivo principal de este trabajo fin de máster era el desarrollo de una red IoT utilizando el estándar IEEE 802.15.4e sobre un sistema OpenWSN y con la plataforma THETHINGS.IO. Y el desarrollo HW de la misma red en la plataforma OpenMote y con una Raspberry Pi.

Para el desarrollo de esta red ha sido necesario una primera etapa de investigación y documentación. En primer lugar detallando las características de los diferentes estándares de comunicación que involucran a las redes, estudiando y analizando el estándar IEEE 802.15.4 y el IEEE 802.15.4e, que es el estándar que utiliza OpenWSN, los protocolos de comunicación y las diferentes alternativas del mercado como CoAP, SoAP, REST, etc.

Trabajar con los diferentes conceptos de Edge computing, Cloud computing y Fog computing para decidir cómo trabajar con la información disponible y como transmitir está a la nube.

Por otra parte la puesta a punto de los equipos para que puedan trabajar con OpenWSN, tanto en plataformas Windows como en plataformas Unix (en la Raspberry) y un desarrollo software que permitiese la comunicación de la red, la adquisición de datos y el procesado de los datos.

Por último, la transmisión y representación de estos datos en las plataformas online de Cloud Computing, en este caso en la plataforma de THETHINGS.IO, donde se presentan las lecturas obtenidas.

5.2. Objetivos cumplidos

Se han cumplido la mayoría de los objetivos marcados en el punto 1.2. Se ha conseguido desarrollar un prototipo haciendo uso de la tecnología OpenWSN y usando el estándar IEEE 802.15.4e, desarrollando para ello una aplicación en Python y en C y su representación en la plataforma THETHINGS.IO.

Por cumplir, ha faltado la implementación del prototipo en un red real y utilizando para ello el hardware de OpenMote. No disponer del hardware desde el principio y tener que familiarizase con todas las herramientas que forman parte de este tipo de redes ha provocado un retraso en la demostración del funcionamiento de la misma que ha impedido la posterior implementación en la red en hardware real.

5.3. Seguimiento de la planificación

La planificación inicial se ha visto afectada debido a la necesidad de familiarizarse con las múltiples disciplinas que implica el desarrollo del TFM. El estudio de los estándares en los que está basado el sistema, protocolos y plataformas han requerido un esfuerzo especial que ha provocado el retraso del proyecto.

Por ello, se ha tenido que simplificar algunos aspectos del desarrollo de la aplicación, a la que le falta modularizar algunos aspectos y la implementación hardware que no ha sido posible finalmente.

5.4. Líneas futuras de trabajo

En este apartado se presentan las posibles mejoras que podrían ser añadidas en una versión mejorada del producto:

- **Implementación del sistema en un hardware real**. Una de los objetivos no cumplidos era la implementación del sistema con un hardware real. Con sensores que midiesen las magnitudes reales en una vivienda y con un actuador que puede abrir o cerrar ventanas para comprobar el ahorro energético del proyecto.

- **Implementación de interfaz gráfica del módulo central.** En vez de utilizar un interfaz por consola, se podría implementar un programa con una interfaz gráfica, de esta forma sería mucho más intuitivo para el usuario y permitiría funcionalidades adicionales como mostrar los resultados gráficamente directamente desde la aplicación en local. Además de podrían añadir nuevas funcionalidades.

- **Autodescubrimiento de nuevos sensores en la red**. Se podría implementar un sistema que mediante CoAP auto descubriera nuevos elementos que se conectan a la red, en vez de utilizar una base de datos donde es necesario especificar la dirección de cada elemento.

- Añadir un módulo adicional al software para el elemento actuador para hacerlo modular igual que para los sensores.

6. Glosario

IEEE: Institue of Electrical and Electronics Engineers
IoT: Internet of Things
TFM: Trabajo fin de máster.
M2M: Machine-to-Machine.
CoAP: Constrained Application Protocol
LowPAN: Low Power Area Networks
ETSI: European Telecommunications Standards Institute
OMG: Object Management Group
MQTT: Message Queuing Telemetry Transport
MQTT-SN: MQTT For Sensor Networks
XMPP: Extensible Messaging and Presence Protocol
AMQP: Advanced Message Queuing Protocol
DDS: Data-Distribution Service for Real-Time Systems
LLAP: lightweight local automation protocol
SSI: Simple Sensor Interface
REST: Representational state transfer
SOAP: Simple Object Access Protocol
LR-WAN: Low-Rate Wireless Personal Area Network
MAC: Medium Access Control
LCC: Logical link control
OSI: Open System Interconnection
DAO: Destination Advertisement Object
WSDL: Web Services Description Language
XML: Extensible Markup Language

7. Bibliografía

[1] Tutorial OpenWSN: https://openwsn.atlassian.net/wiki/spaces/OW/pages/29884450/Kickstart+Windows

[2] Arquitectura IoT: https://techseminarforcse.blogspot.com.es/2016/01/technicalseminarforcse.html

[3] Def. del IoT: https://es.wikipedia.org/wiki/Internet_de_las_cosas

[4] Origen del IoT: http://www.bcendon.com/el-origen-del-iot/

[5] http://www.dataversity.net/brief-history-internet-things/

[6] Arquitectura sistemas IoT: https://techseminarforcse.blogspot.com.es/2016/01/technicalseminarforcse.html

[7] Protocolos IoT: https://about.sofia2.com/2014/05/29/protocolos-iot-mqtt-rest-coap-xmpp-y-sofia2/

[8] Protocolos IoT: https://www.postscapes.com/internet-of-things-protocols/

[9] IoT Roadmap: https://www.iotroadmap.com/

[10] From IEEE 802.15.4 to IEEE 802.15.4e http://www.iet.unipi.it/g.anastasi/talks/2014-Guangzhou.pdf

[11] Estándar IEEE 802.15.4e http://standards.ieee.org/findstds/standard/802.15.4e-2012.html

[12] Estándar IEEE 802.15.4 http://catarina.udlap.mx/u_dl_a/tales/documentos/lem/archundia_p_fm/capitulo4.pdf

[13] Estándar IEEE 802.15.4 https://rua.ua.es/dspace/bitstream/10045/1109/1/InformeTecZB.pdf

[14] Cloud computing https://es.wikipedia.org/wiki/Computaci%C3%B3n_en_la_nube

[15] Cloud computing https://www.globaldots.com/cloud-computing-types-of-cloud/

[16] Fog Computing

https://www.cisco.com/c/dam/en_us/solutions/trends/iot/docs/computing-overview.pdf

[17] Fog Computing vs Edge Computing
https://www.winsystems.com/cloud-fog-and-edge-computing-whats-the-difference/

[18] Cloud Computing vs Fog Computing
https://blogs.cisco.com/perspectives/iot-from-cloud-to-fog-computing

[19] Cloud Computing vs Fog Computing
https://openwsn.atlassian.net/wiki/spaces/OW/pages/29196302/Kickstart+Linux

[20] IoT y computación en la nube
https://www.slideshare.net/dipina/internet-del-futuro-internet-de-las-cosas-computacin-en-la-nube-y-la-web-de-datos-27977775

[21] Protocolos de datos IoT
http://repositorio.upct.es/bitstream/handle/10317/4163/pfc5908.pdf;sequence=1

[22] Low Rate - WPAN - 6LoWPAN
http://www.sharetechnote.com/html/IoT/LR_WPAN_6LoPAN.html

[23] Fog Computing
http://comingtechs.com/fog-computing/

[24] Raspberry descripcion
https://ingenierate.com/2017/10/03/raspberry-pi-caracteristicas-aplicaciones/

8. Anexos

8.1. Anexo I: Main_IoT_System

Se adjunta el código desarrollado para la aplicación principal.

```
import os
import sys
import array
import time                                    # Carga para la funcion
delay
from Send_THETHING import escribir_en_TT        # Carga de la
funcion para enviar datos a THETHING
from Send_THETHING import recibir_TT
from BBDD_IoT import call_BBDD
from lecturaDatosSensores import lecturaDatosSensores
from envio_SMS import Envio_SMS_Clientes

# import coap
from coap import coap

# Constantes
max_temp_excedida = 21

sistemaop = os.name

if sistemaop == "nt":
    os.system('cls')    # Para Windows
else:
    os.system('clear')   # Para Linux


print('\n'+'\n')
print ("*****************************************")
print ("* BIENVENIDO A IOT SYSTEM                *")
print ("* Programa de comunicacion con OpenWSN   *")
print ("* y la plataforma THETHING               *")
print ("*****************************************" + '\n' + '\n'
+ '\n')


# MOTEs IP
MOTE_IP_ACTUATOR = 'bbbb::1415:92cc:0:1'    # IPv6 del mote donde
se actuara
UDPPORT = 61615

# THETHINGS Token, para comunicar con el actuador en la plataforma
TheThingTokenActua = "LGQNmm5SMnRq8a9IqNjpYOc2QUbcF1338R_z023-qd4"


print ("LECTURA DE DATOS")


valor_actuador = 0
control_40 = 1
control_10 = 1
histeresis_pre = []
```

```
sensores = lecturaDatosSensores()

# Bucle para la lectura periodica de muestras de los motes
while valor_actuador != 9:

    minutos = time.strftime("%M")
    segundos = time.strftime("%S")
    print("\n-------------------------------------------------------
-----------------------\n")

    # LECTURA DE SENSORES
    for i in range(0, len(sensores)):
        try:
            MOTE_IP = str(sensores[i][2])
            # Open CoAP udpPort=UDPPORT
            c = coap(udpPort=UDPPORT)

            p = c.GET('coap://[{0}]/i'.format(MOTE_IP))

            print ("\nPaquete recibido. Sensor: " +
str(sensores[i][0])) + ". IP: " + format(sensores[i][2]) + ".
Lectura: " + str(p[i]) + " " + sensores[i][4]

            # Se inicializa histeresis_pre e histeresisPost con el
numero de sensores disponibles
            if (len(histeresis_pre)) < (len(sensores)*2):
                histeresis_pre.insert(i*2, float(p[i]))
                histeresis_pre.insert((i*2)+1, 0.0)
            else:
                histeresis_pre[i*2] = float(p[i])

            if (float(histeresis_pre[i*2]) >
float(histeresis_pre[(i*2)+1]) + float(sensores[i][3])) or
(float(histeresis_pre[i*2]) < float(histeresis_pre[(i*2)+1]) -
float(sensores[i][3])):
                # Se llama a la funcion que se encargara de la
emision de datos a la plataforma THETHINGIO
                escribir_en_TT(sensores[i][0], p[i],
sensores[i][1])
                # Se llama a la funcion que almacena las lecturas
en la bbdd local
                call_BBDD(format(sensores[i][0]), str(p[i]))
                histeresis_pre[(i*2)+1] = p[i]
                histeresis_pre[i*2] = p[i]

                # En el caso de que tengamos una nueva lectura y
esta supere un cierto umbral se notificara al cliente por SMS
                if p[i] > max_temp_excedida and sensores[i][0] ==
"Temperatura":
                    Envio_SMS_Clientes(p[i])
                    print "\n Se enviar " + str(p[i]) + "de SMS"

            c.close()

        except:
            print "\033[;31m" + "\n%%% ERROR: No ha sido posible
la comunicacion con el mote en la IP: " + format(sensores[i][2]) +
" %%%%\n"
            print "\n\033[;37m"
            c.close()
```

```
    if minutos == "00" or minutos == "15" or minutos == "30" or
minutos == "45":
        # Se llama a la funcion que almacena las lecturas en la
bbdd local
        print "\n\nSe procede a enviar las lecturas a la BBDD
local"
        call_BBDD("Luminosity", str(p[0]))
        call_BBDD("Temperatura", str(p[1]))
        time.sleep(50)

    # ACTUADOR - Se encarga de enviar datos a los motes

    if int(minutos) > 10 and control_10 == 1 and int(minutos) <=
40:
        print "\n\nToma de lecturas del actuador desde la
plataforma THETHINGS\n"

        new_valor_actuador = int(recibir_TT('Activador',
TheThingTokenActua))
        if int(new_valor_actuador) != int(valor_actuador):
            print "    --> Nueva lectura en ACTUADOR: " +
str(new_valor_actuador) + "\n"
            valor_actuador = new_valor_actuador

            try:
                c = coap(udpPort=UDPPORT)
                p =
c.PUT('coap://[{0}]/i'.format(MOTE_IP_ACTUATOR),payload =
[valor_actuador],)
                print ("\nPaquete enviado a la IP: " +
format(MOTE_IP_ACTUATOR) + ". Cargado: " + str(valor_actuador) +
'\n' + '\n')
                c.close()
            except:
                print "\033[;31m" + "\n%%% ERROR: No ha sido
posible enviar los datos al mote en la IP: " + MOTE_IP_ACTUATOR +
" %%%%\n\n"
                print "\n\033[;37m"
                c.close()

        control_10 = 0
        control_40 = 1

    elif int(minutos) > 40 and control_40 == 1:
        print "\n\nToma de lecturas del actuador desde la
plataforma THETHINGS\n"
        new_valor_actuador = int(recibir_TT('Activador',
TheThingTokenActua))
        if new_valor_actuador != valor_actuador:
            print ("    --> Nueva lectura en ACTUADOR: " +
str(new_valor_actuador) + "\n")
            valor_actuador = new_valor_actuador
            try:
                c = coap(udpPort=UDPPORT)
                p =
c.PUT('coap://[{0}]/i'.format(MOTE_IP_ACTUATOR),payload =
[valor_actuador],)
                print ("\nPaquete enviado a la IP: " +
format(MOTE_IP_ACTUATOR) + ". Cargado: " + str(valor_actuador) +
'\n' + '\n')
                c.close()
```

```
            except:
                print "\033[;31m" + "\n%%% ERROR: No ha sido
posible enviar los datos al mote en la IP: " + MOTE_IP_ACTUATOR +
" %%%%\n\n"
                print "\n\033[;37m"
                c.close()

        control_10 = 1
        control_40 = 0

    # Control del bucle
    time.sleep(10)    # Delay for 10 seconds.

print '\n'
print 'bye bye.'
```