Demostrador de Internet of Things con la tecnologia IEEE 802.15.4e utilizando el sistema operativo *OpenWSN*, OpenSim y theThings.io.

Manuel Márquez Salas

Máster Universitario en Ingeniería de Telecomunicación (UOC-URL)

José López Vicario
Xavier Vilajosana Guillen

11 de Junio 2017

FICHA DEL TRABAJO FINAL

Título del trabajo:	*Demostrador de Internet of Things con la tecnología IEEE 802.15.4e utilizando el sistema operativo OpenWSN, OpenSim y theThings.iO.*
Nombre del autor:	*Manuel Márquez Salas*
Nombre del consultor/a:	*José López Vicario*
Nombre del PRA:	*Xavier Vilajosana Guillen*
Fecha de entrega (mm/aaaa):	*06/2017*
Titulación o programa:	Máster Universitario en Ingeniería de Telecomunicación (UOC-URL)
Área del Trabajo Final:	*Telemática*
Idioma del trabajo:	*Castellano*
Palabras clave	*ioT, Edge, Fog*

Resumen del Trabajo:

Actualmente el término nube nos hace pensar en que solo existe un lugar, o mejor dicho un servidor, donde se alojan nuestros datos, pero lo que no sabemos es que existen innumerables nubes. Es por ejemplo el concepto de Edge (también llamado Fog) el cual tiene unas ciertas ventajas y beneficios y que también puede trabajar en conjunción con el primero. Muchas aplicaciones no se pueden permitir el lujo de esperas de decenas de segundos e incluso de minutos para tomar una decisión que dependen de un análisis de dicha información. Es por ello que nace el concepto del Edge computing como solución a este tipo de requerimiento de baja latencia, optimización del ancho de banda disponible añadiendo mayor seguridad por el hecho de procesar los datos en la misma red local.

En este trabajo se ha desarrollado un demostrador de un sistema doméstico formado por una red de sensores de temperatura, humedad, movimiento y un actuador que permitiría la activación de la calefacción. Para ello se ha utilizado la plataforma *OpenWSN* ejecutándose en un sistema barato y suficientemente potente como es la Raspberry pi 3. Mediante scripts escritos en Python y utilizando la librería de COAP se ha podido extraer información de estos dispositivos simulados, generándose las dos situaciones típicas descritas arriba. En primer lugar la comunicación con el Gateway Edge almacenándose los datos de todas las variables importantes, tomándose las decisiones en tiempo real ante situaciones previamente programadas y subiéndose solamente los datos históricos a la nube, a la plataforma theThings.iO.

Y en segundo lugar una segunda situación, donde los sensores escriben directamente en la nube utilizándose también la plataforma theThings.iO,

usándose en este caso dicha plataforma para tomar las decisiones y comunicar de vuelta el resultado al Gateway implementado con Rpi.

Para acabar se han comparado ambos métodos y se ha mostrado el funcionamiento del demostrador.

Abstract:

Nowadays the term cloud makes us think that there is only one place, or rather a server, where our data is hosted, but what we do not know is that there are innumerable clouds. It is for example the concept of Edge (also called Fog) which has certain advantages and benefits and can also work together. Many applications can not afford any delay of tens of seconds or even some minutes to take a decision that depends on an analysis of data generated by that application. Due to that reason the concept of edge computing is born as a solution to fullfill a low latency requirement, optimizing the available bandwidth by adding more security by processing the data in the same local network.

In this project we have developed a domestic system demonstrator composed by a network of temperature, humidity, movement sensors and an actuator that would allow the activation of the heating system. To reach that goal the *OpenWSN* platform has been used running on an inexpensive system and powerful enough to be able to simulate this scenario as is the Raspberry pi v3 board.

Using scripts written in Python and using the COAP library, it was possible to extract relative data from these simulated devices or motes, generating the two typical situations described above. Firstly, the communication with the Edge Gateway, storing the data of all the important variables, making the decisions in real time before previously scheduled situations and uploading only historical data to the cloud, to the theThings.iO platform.

And secondly, a second scenario where sensors write data directly in the cloud using also the theThings.iO platform also to make the decisions and communicate back the result to the Gateway implemented with Rpi.

Finally, both methods have been compared and the demonstrator has been shown to work.

Agradecimientos

A mi familia por su apoyo continuo y en especial
a mi mujer Merche y a mi hijo Daniel por su ayuda y paciencia.
Sin ellos no hubiese sido posible.

A José López y Xavier Vilajosana por el soporte recibido.

Índice de contenidos

Lista de ilustraciones

Lista de tablas

1.Introducción

1.1 Contexto y justificación

El demostrador de Internet of Things (IoT) pretende simular una situación habitual de sensores desplegados en un entorno doméstico utilizando los protocolos de código libre (*open-source*) basándose en *OpenWSN* y utilizando la tecnología *IEEE 802.15.4e*, estándar en el mundo del Internet de las cosas. En este trabajo se focalizará el esfuerzo en la implementación del *Edge Computing* (también llamado *Fog Computing*), el cual se basa en un procesado previo que se realiza en un módulo llamado comúnmente *Gateway* que es el que procesará los datos recibidos de los nodos sensores, generando datos resumidos a partir de los datos recibidos, subiéndose dicha información optimizada en términos de tamaño y uso del ancho de banda disponible. De esta manera se reducirá la latencia en las comunicaciones para acceder a los sensores y actuadores.

Las soluciones actuales pasan por el uso del Cloud computing para permitir la implementación de aplicaciones como la descrita. A diferencia del Cloud computing en este trabajo el procesado se hará en el Edge, dentro de la misma red local, liberando al Cloud de tiempo de procesado, disminuyendo la latencia de los datos, optimizándose el uso del ancho de banda disponible para todos los servicios y aumentando notablemente la seguridad de red ya que las decisiones se toman localmente.

Dicho *Gateway* se implementará utilizando un hardware basado en la Raspberry Pi, un sistema barato, asequible y flexible que permitirá cubrir las necesidades de gestión de datos y conexión de red. Existen otras soluciones hardware embebidas para implementar dicho Gateway como son la *PandaBoard* [28] pero que tienen un coste elevado en comparación con la relación coste/funcionalidades de una Raspberry Pi. Dicha plataforma ejecutará la pila de protocolos que soportan el estándar *IEEE 802.15.4e* para el Internet de las Cosas, dará potencia de procesado para ejecutar scripts implementando el *Edge Computing* y subirá datos a los servidores en la nube de la plataforma theThings.iO para permitir su monitorización a distancia.

1.2 Motivación

Existen diversos motivos que han hecho que el presente trabajo sea el escogido; son los siguientes:

- Estudiar las tecnologías IoT actuales así como aplicaciones comerciales utilizadas a día de hoy.

- Conocer y utilizar la herramienta de software libre llamada *OpenWSN,* la cual permite el desarrollo de aplicaciones para implementar aplicaciones IoT, tanto de forma física con hardware como simulada. A diferencia de otras plataformas, también libres, algunas con menos funcionalidades, ésta permite la simulación de la pila de protocolos estándar del Internet de las cosas como son el IEEE802.15.4e, 6LoWPAN, RPL y CoAP, soportando el modo TSCH del estándar IEEE802.15.4e.

- Utilizar herramientas Software específicas utilizadas en el desarrollo de aplicaciones y aprender cómo se puede desplegar y distribuir.

- Entender la importancia del Edge Computing. Debido a la ingente cantidad de "cosas" conectadas en red que mandan importantes cantidades de información a la nube, el Edge Computing cobra una importancia relevante.

- Saber cuándo aplicar una solución Edge computing y cuando no. En el caso que ocupa este trabajo se desarrollará una aplicación doméstica donde el tiempo de reacción es crucial y donde la seguridad y disponibilidad de la aplicación tienen también una gran importancia, debido a la sensibilidad de los datos que maneja.

- Comparar resultados entre la solución Cloud y Edge con la presente aplicación.

- Aumentar los conocimientos sobre bases de datos en MySQL y conectarla con la nube en una aplicación real.

- Conocer cómo funciona la plataforma theThings.iO y qué funciones dispone su API, qué tipos de llamadas existen y qué servicios IoT ofrecen.

1.3 Objetivos

El objetivo de la presente memoria será el desarrollar un demostrador de IoT basándose en la herramienta *OpenWSN* para simular una situación real y doméstica de despliegue de sensores de temperatura, humedad y movimiento, con un actuador que permite la conexión de la calefacción de forma autónoma en base a la gestión de los datos recogidos y almacenados en una base de datos desarrollada en MySQL. Dichas aplicaciones desarrolladas en Python se ejecutarán en una Raspberry Pi, la cual tomará las decisiones oportunas y subirá la información compilada y resumida a la plataforma *theThings.iO* implementando el *Edge Computing*.

Para llevar a cabo dichos objetivos se desglosarán las actividades en detalle:

- Instalar en la Raspberry Pi el software necesario para llevar a cabo todas las actividades.
- Estudiar el software *OpenWSN* y aprender su funcionamiento interno a nivel de arquitectura y estructura de carpetas y tool chain.
- Aprender a realizar cambios en el modulo *OpenWSN*-fw para poder implementar los sensores referenciados.
- Qué herramientas software dispone *OpenWSN* para desarrollar aplicaciones IoT como son el openvisualizer, openSim, etc.
- Aprender cómo funciona una App, cómo crearlas y entender cómo se configura en *OpenWSN* para que el Openvisualizer la encuentre.
- Ejecutar una simulación, ver como se crea la red WSN y cómo se puede interactuar con ella.
- Desarrollar software en Python utilizando métodos GET y PUT en CoAP para llevar a cabo la captura de datos de los nodos (motes).
- En qué se basa la plataforma theThings.iO y cómo funciona su API.

1.4 Enfoque y método seguido

A partir de los objetivos descritos se ha creado una lista de tareas a llevar cabo dentro de la planificación y entregas necesarias durante el semestre (PAC´s), creando un diagrama de Gantt que se ha ido trabajando durante el desarrollo, añadiendo más actividades a medida que han sido necesarias.

Se ha basado el presente trabajo en los ejemplos que dispone *OpenWSN* para crear nuevos sensores y actuadores, añadiendo nuevas funciones, las cuales se detallarán más adelante.

Como resumen se puede decir que *OpenWSN* permite la simulación de una situación real utilizando la misma pila de protocolos que utilizaría una aplicación que utiliza hardware físico, hecho que permite la depuración del firmware que se ejecutará en todos los nodos reduciendo los costes de desarrollo y una generación de código mucho más depurado, realizándolo en el menor tiempo posible.

1.5 Planificación

Durante el desarrollo del proyecto se ha seguido una planificación inicial que dinámicamente se ha ido adaptando (sobre todo por la necesidad de añadir actividades relacionadas con la creación de scripts utilizando las librerías CoAP y de la API de thethings.iO), a los hitos más importantes a completar. En la presente figura se puede observar las tareas llevadas a cabo:

Tarea

Nombre	Fecha de inicio	Fecha de fin
Planificación TFM	1/03/17	24/06/17
Decisión proyecto	1/03/17	1/03/17
PAC1	8/03/17	8/03/17
PAC2	19/04/17	19/04/17
PAC3	24/05/17	24/05/17
Redacción memoria	8/03/17	10/06/17
Inicio Memoria	8/03/17	8/03/17
Memoria Final	11/06/17	11/06/17
Presentación + Código	18/06/17	18/06/17
Tribunal	19/06/17	24/06/17
Inicio Tribunal	19/06/17	19/06/17
Final Tribunal	25/06/17	25/06/17
Setup Rpi	8/03/17	14/03/17
Compilación/Instalación OpenWSN	15/03/17	29/03/17
Simulación ejemplo OpenWSN	30/03/17	20/04/17
Disponibilidad OpenMote	21/04/17	21/04/17
OpenWSN y OpenMote hardware Setup	21/04/17	6/05/17
Generación scripts Edge Computing	7/05/17	19/05/17
Establecer y seleccionar datos objetivo	7/05/17	8/05/17
Crear base de datos MySQL	8/05/17	10/05/17
Crear Scripts Python lectura de datos (motes)	9/05/17	11/05/17
Crear Scripts Python almacenaje de datos en MySQL	10/05/17	11/05/17
Crear datos resumen y volcado en MySQL	11/05/17	13/05/17
Setup theThings.io, libreria thethings.iO-python-library	12/05/17	13/05/17
Seleccionar datos para subir al Cloud theThings.io	13/05/17	19/05/17
Situación doméstica: ejemplo práctico	21/05/17	21/05/17
Crear Script Python para accionar Calefacción automatico	21/05/17	21/05/17
Subir datos RESUMEN a theThings.io	21/05/17	21/05/17
Validación	3/04/17	3/04/17

Tabla 1: Planificación del TFM

Diagrama de Gantt

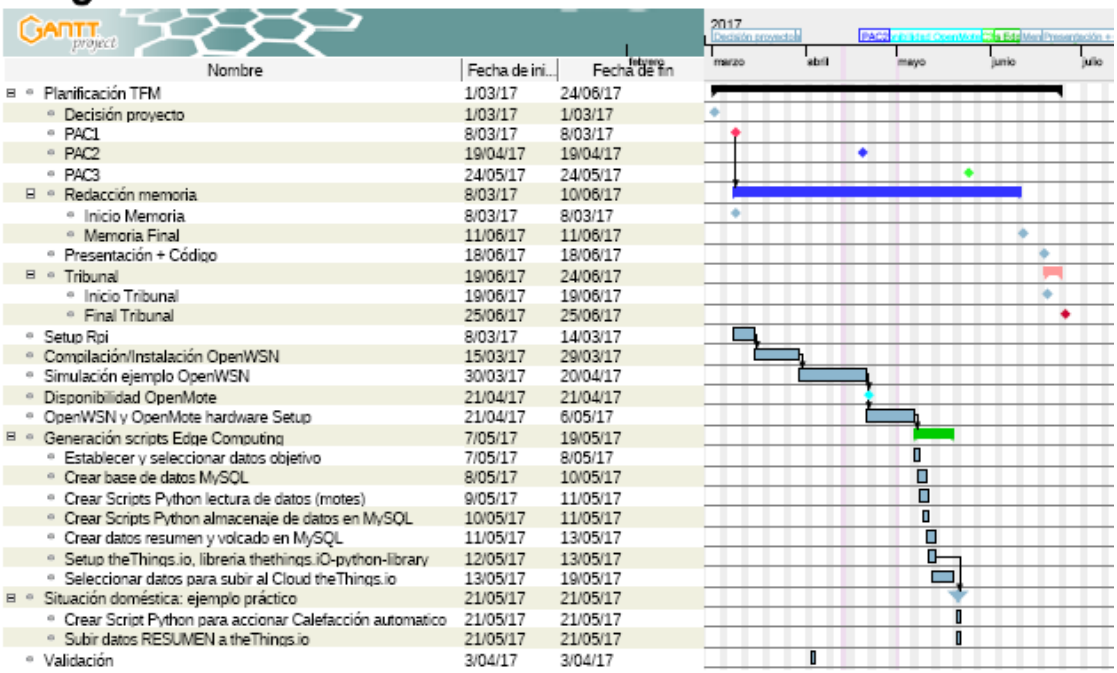


Ilustración 1: Diagrama de Gantt del TFM

1.6 Breve resumen de productos obtenidos

Aquí se detallan los productos que se han desarrollado:

- Aplicaciones (ctempo.c, chumidity.c, cmotion.c y ccalef.c) para implementar los nodos que simulan la generación de datos de temperatura, humedad y detección de movimiento desarrollada en lenguaje C. También incluye la aplicación calefacción para la activación del actuador simulado.

- Aplicaciones en Python para extraer la información de los nodos utilizando la librería CoAP de *OpenWSN*.

- Aplicación en Python de volcado de datos en la base de datos del Gateway en la base de datos diseñada en MySQL.

- Aplicación de procesado en Edge para generar los datos resumen de temperaturas, humedad, movimiento, calefacción y alarmas.

- Generación de scripts que planifican la ejecución de todas las operaciones descritas arriba.

- Aplicación en Python que implementa la comunicación con la plataforma theThings.iO, utilizando la librería para dicho lenguaje para la correcta utilización de su API.

1.7 Descripción de los capítulos de la memoria

Capítulo 2.- El Internet de las cosas (ioT), Cloud y Edge computing: se explica que es el Internet de las cosas (ioT) y los tipos de procesado de los que trata este trabajo que son el *Cloud computing,* el *Edge computing* y los protocolos estándar usados.

Capítulo 3.- Sistemas operativos (SO) para el Internet de las cosas: se presentan las diferentes opciones disponibles en la actualidad y se detalla el *OpenWSN* describiendo sus características principales.

Capítulo 4.- Resumen: herramientas Hardware y Software utilizados: se resumen el hardware y aplicaciones Software escogidas para el desarrollo del presente trabajo.

Capítulo 5.- Herramientas necesarias: instalación y configuración: se muestran en este capítulo los pasos seguidos para realizar el *setup* necesario de otras herramientas.

Capítulo 6.- Desarrollo del Edge Computing con Rpi: aquí se describe el desarrollo de las aplicaciones creadas y de los scripts escritos en Python para

realizar el procesado en Edge. También se desarrolla unos scripts para implementar un procesado en la nube clásico.

Capítulo 7.- Resultados: comparación Cloud vs Edge computing: se presentan los resultados extraídos de este proyecto en base a la comparación de los dos métodos de procesado de datos.

Capítulo 8.- Conclusiones: se finaliza con este apartado resumiendo y comparando lo obtenido en base a lo inicialmente planificado.

2. El Internet de las cosas (ioT), Cloud y Edge computing

2.1 Internet de las cosas (ioT)

El término Internet de las cosas [3] (IoT) nace allá por el año 2005 al publicarse el primer es estudio que hace la ITU [1] (Unión Internacional de Telecomunicaciones) sobre esta temática, añadiéndose el concepto de conectividad para *anything* a las conocida como *anytime, any place and for everyone*. La idea era poder conectar a Internet cualquier objeto, que podían tener cualidades más o menos grandes de computación calificándolos como objetos inteligentes. Dichos dispositivos son una realidad actualmente debido al reducido coste que tienen los procesadores, también las memorias que permiten alojar código y por supuesto las reducidas dimensiones que tienen los sensores y actuadores con posibilidades de conexionado a Internet.

Es por tanto muy importante en un dispositivo ioT que tenga las siguientes cualidades:

- Baratos y con posibilidad de aumentar sus prestaciones.
- Tamaño, peso y consumo reducido.
- Conectividad de bajo coste, escalable y adaptativa.
- Servicios y aplicaciones basados en la computación en la (*Cloud Computing*) actuando en una red de dispositivos colaborativos.

Por tanto si comparamos el actual Internet con el Internet de las cosas podemos encontrar las siguientes diferencias:

- Hardware es reducido comparándolo a nivel funcional con un equipo tradicional.
- Diferencias a nivel de dispositivos conectados (Billones vs Millones) por eso se basa en IPv6 para permitir su registro sin conflictos en asignación de direcciones IP.
- Tamaño de datos reducido en los dispositivos ioT en comparación con los terminales clásicos y redes de acceso que no paran de crecer en términos de ancho de banda para poder dar servicio correctamente a los

usuarios que demanda cada vez servicios de más alta calidad y más diversos.
- En IoT los servicios están centrados en el hardware no en los usuarios.
- Se pone el foco en crear el enlace entre dispositivos para compartir datos de forma autónoma y no en la comunicación en sí, permitiendo detectar y modificar su entorno.

El Internet de las cosas aglutina una gran variedad de tecnologías que permiten que Internet pueda abarcar el mundo de las "cosas" conectadas como por ejemplo podrían ser la identificación de objetos, comunicaciones inalámbricas de corto alcance (NFC), la geolocalización de dispositivos y la obtención de datos a través de un despliegue de sensores conectados en red.

Ilustración 2: Elementos del ioT [2]

La arquitectura de un sistema ioT se basa en tres capas [2]:

1. Capa de información: es la encargada de identificar los elementos y recoger la información del entorno físico. Los nodos o dispositivos que forman parte de esta capa pueden formar parte de una red (por ejemplo, una red de sensores)
2. Capa de Red: incluye las pasarelas por cable o inalámbricas, redes de acceso y troncales, permitiendo básicamente la transmisión, enrutamiento y control entre capas de información y la de aplicaciones.
3. Capa de aplicación: es la que almacena y procesa la información recibida, ejecutando tareas de control.

Aplicaciones ioT pueden implementar sistemas M2M y las ciudades inteligente o *Smart cities* sobretodo en sectores como la agricultura, automoción, control de procesos en la industria, demótica, sector sanitario y un largo etcétera.

2.2 Computación en la nube (*Cloud computing*)

Se puede definir la computación en la nube [3] como la tecnología que ofrece servicios y aplicaciones de cálculo a través de Internet con la idea de liberar el ordenador de los usuarios de la carga que supone tener un programa o aplicación instalado, ofreciendo al usuario final los recursos necesarios para poder realizar las tareas previstas bajo demanda utilizando los servidores dispuestos para tal fin.

Podemos definir como el objetivo principal del *Cloud computing* es el de distribuir tareas de cálculo en varios ordenadores distribuidos y no en ordenadores locales o servidores remotos. Los beneficios de este tipo de servicio son:

- Eficiencia: reduciéndose el coste de hardware y software necesario para ejecutar una tarea por el hecho de no estar instalado en el terminal del usuario, reduciéndose a cero también su mantenimiento. El usuario paga por lo que usa, bajo demanda con uso ilimitado mejorando la utilización y eficiencia de los equipos.

- Potencia: escalable en cualquier momento dependiendo de las necesidades.

- Flexibilidad: independiente de la localización del usuario.

- Escalable: facultad de proveer servicios y aplicaciones e incrementar la infraestructura necesaria bajo demanda.

- Abstracción: interfaz sencilla para el usuario desentendiéndose de la infraestructura, plataforma de hardware o sistema operativo.

También hay que remarcar que el *Cloud computing* no está exento de riesgos como es la seguridad de los datos y programas, fiabilidad (SLA, *Service level agreement*) y calidad (QoS, *Quality of service*) del servicio. Es por tanto la integración del ioT usando este tipo de procesado en la nube una tarea no sencilla debido a los siguientes factores:

- Sincronización entre diferentes proveedores presenta un reto ya que los servicios están creados en una capa por encima de las plataformas en la nube.
- Estandarización entre diferentes proveedores de servicios.
- Balanceado entre servicios de procesado en la nube y los requerimientos de los dispositivos ioT es un reto debido a las diferencias en infraestructura.
- La Fiabilidad y seguridad de los ioT es también un reto debido a las diferencias en las políticas de seguridad.
- La gestión de los sistemas ioT y del Cloud computing es también una tarea difícil de gestionar debido a que cada uno tiene diferentes recursos y componentes.
- Es necesario validar los servicios ioT para alcanzar los requerimientos demandados por los usuarios.

Un aspecto importante de las plataformas Cloud computing es la posibilidad de interactuar con diferentes protocolos de aplicación. Un ejemplo de proveedores de Cloud services son ThingWorx, OpenIoT, Google Cloud y Amazon.

El *Cloud computing* puede ofrecerse de las siguientes maneras:

- Nube privada: La infraestructura de la nube solo está disponible para una organización. Por tanto, la gestión de dicha nube la realiza la propia organización.

- Nube comunitaria: La infraestructura es compartida por varias organizaciones y ofrece los mismos niveles de requisitos de seguridad, QoS, SLA y política de servicio.
- Nube pública: la infraestructura puede usarse por el público en general y es propiedad de una organización que vende los servicios de cálculo en la nube.
- Nube híbrida: La infraestructura de la nube es una composición de dos o más nubes que permiten la portabilidad de datos y aplicaciones de una nube a otra mediante tecnologías específicas.

2.3 Computación en el borde de la red (E*dge computing*)

El *Edge computing*, o también llamado *Fog computing* [8] [9] [6], pueden actuar como un puente entre los dispositivos inteligentes y el *Cloud computing* y servicios de almacenamiento. A través del *Edge computing* es posible acercar el cálculo del *Big data* cerca de los dispositivos ioT que por ejemplo forman parte de una red de sensores, ofreciendo unos mejores servicios en términos de latencia (evitando retardos en tiempos de carga), optimización del consumo de las redes troncales de banda ancha y añadiendo más seguridad. Por tanto los requerimientos del Edge computing son los siguientes:

- Minimizar la latencia: dependiendo del tipo de proceso a controlar la diferencia de unos pocos milisegundos importan. El hecho de que el procesado de la información que generan los dispositivos esta cerca del lugar donde se procesan estos datos hace posible su reducción.
- Gestionar el ancho de banda disponible de forma eficiente: dependiendo de la red de sensores a controlar estos pueden generar miríadas GB a la semana.
- Seguridad: los datos deben ser protegidos tanto durante su tránsito como cuando están estáticos.
- Fiabilidad operativa: muchas de las redes de cosas conectadas manejan datos y funciones críticas que afectan a la seguridad de las personas e infraestructuras vitales, por tanto la disponibilidad e integridad de los datos ha de ser un punto a cumplir siempre.
- Recolección de datos en un área geográfica en diferentes entornos: los dispositivos ioT puedes ser desplegados por un área de centenas de kilómetros cuadrados.
- Procesado de datos ha de estar bien localizado: en el sentido de mejorar la respuesta ante los datos que son recogidos en procesos críticos que requieren un tiempo de respuesta muy bajo. En el otro lado los datos históricos son enviados a la nube para su procesado y almacenamiento.

Las aplicaciones del *Edge computing* [10] [11] son diversas pero básicamente dan servicio a tareas de monitorización de los datos en tiempo real de los dispositivos ioT que forman parte de una red y que necesitan realizar una acción dependiendo de dichos datos.

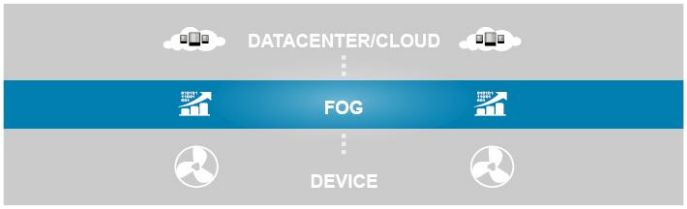


Ilustración 3: Edge acerca el Cloud a los dispositivos IoT que producen los datos [10]

2.3.1 Como funciona el Edge computing

Se trata de que las tareas de procesado criticas [4] se trasladen lo más cerca de la red de dispositivos a monitorizar, por tanto se han de generar aplicaciones que corran en dichos dispositivos o como veremos más adelante en dispositivos que actúan como Gateway para poder ofrecer las características antes descritas.

- Los datos más sensibles al tiempo de reacción son analizados cerca de los dispositivos que los generan [4].
- Los datos que no tienen tanta prioridad, o sea, que pueden esperar algunos segundos o incluso minutos en ser procesados, son enviados a un nodo intermedio para que los analice y actúe.
- Los datos que no son sensibles al tiempo son enviados a la nube para su análisis histórico y almacenado.

	Nodos Edge cerca de disp. IoT	**Cloud**
Tiempo de respuesta	Milisegundos	Minutos, días, semanas
Ejemplos de aplicaciones	M2M, Telemedicina	Analíticas Big Data
Tiempo de vida de datos	Temporales	Meses o años
Soporte geográfico	Local	Global

Tabla 2: Los nodos Edge acercan el Cloud al borde de la red local [10]

2.3.2 Beneficios del Edge computing

El hecho de acercar la nube y sus servicios al borde de la red, donde operan los dispositivos ioT que pueden formar redes de decenas de miles de nodos emitiendo datos sensibles, en muchos casos datos de importancia y por tanto críticos en su análisis, hacen visibles los beneficios y virtudes de este sistema [6]:

- Oportunidades de negocio: los desarrolladores pueden ofrecer aplicaciones rápidamente cuando sean necesarias.
- Mejora la seguridad: usando las mismas políticas de seguridad de red al estar en el mismo segmento.

- Mejora de la privacidad y control de los datos: se analizan los datos sensibles de forma local evitando el envío de estos a la nube para ser procesados.
- Optimizar el ancho de banda: el hecho de que los datos se procesen localmente hace que no sea necesario enviar cada dato a la nube para su análisis.

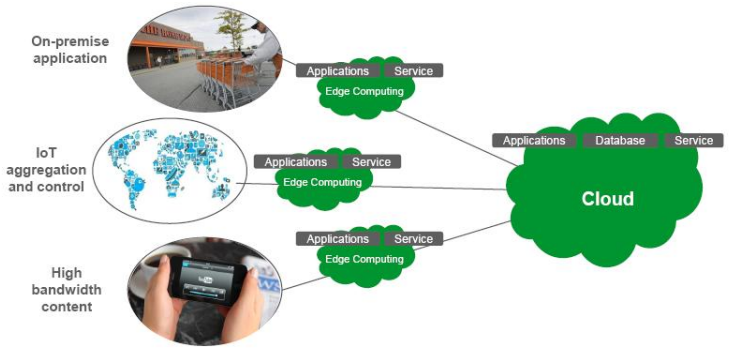


Ilustración 4: Diagrama básico de Cloud computing con dispositivos en el Edge [15]

En referencia a los diseñadores de aplicaciones y servicios, el *Edge computing* es una buena elección debido a los siguientes factores:

- Localización: el Edge está situado entre los dispositivos inteligentes y la nube, por tanto proporcionarán unos retrasos en la comunicación menores.
- Distribución: al estar basados en pequeños dispositivos con almacenamiento, procesado y funciones de red limitadas es posible distribuirlos en una buena cantidad, cerca de los usuarios finales debido a su coste inferior.
- Escalables: permiten a los dispositivos ioT ser más flexibles y escalables al aumentar el número de usuarios.
- Densidad de dispositivos: permite que los servicios tengan un comportamiento elástico con posibilidad de replicación.
- Soporte a la movilidad: actuando como una red móvil al estar cerca de los usuarios finales.
- Tiempo real: mejor soporte a aplicaciones que necesiten reacción en tiempo real.
- Estandarización: los recursos del *Edge computing* pueden operar con varios servicios en la nube.
- Análisis inmediato: pueden recoger datos y enviar parcialmente los datos procesados a la nube para un análisis más completo.
- Por tanto, el *Edge computing* tiene un gran potencial de crecimiento para mejorar el rendimiento de aplicaciones ioT, al ahorrar parte del procesado clásico realizado en la nube utilizando recursos locales.

2.3.3 Tipos de Edge computing

En general existen tres tipos de *Edge computing* que son los que se muestran en la figura.

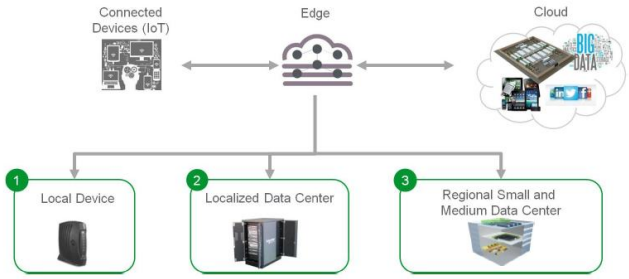


Ilustración 5: Tipos de Edge computing [15]

1.- Dispositivos locales:

Son aquellos que están dimensionados para tener una función específica para que su despliegue sea inmediato, idóneos para el uso doméstico y pequeñas aplicaciones de oficina. Por ejemplo, el dar soporte al sistema de seguridad de un edificio o almacenaje de video local, también el de puerta de enlace o *Gateway* el cual es otro dispositivo local y usualmente un servidor de red que traduce a peticiones a la API del servicio en la nube como SOAP o REST. Dicho Gateway habilita a los usuarios a integrar funciones clásicas del *Cloud computing* en aplicaciones locales sin necesidad de ejecutarlas en la nube.

2.- Data centers centralizados (de 1 a 10 racks):

Estos centros de cálculo proporcionan procesado significativo y capacidades de almacenaje rápidos de instalar, ya que están ya pre-configurados y pre-ensamblados. Están preparados para soportar condiciones extremas a la lluvia, corrosión, humedad, fuego, etc. o disponibles también en un formato de oficina sin tantas protecciones. Dichos centros están dirigidos a aplicaciones de baja latencia con anchos de banda ajustables con posibilidad de añadir características mejoradas de seguridad y disponibilidad del sistema.

Ilustración 6: Data center [15]

3.- *Data centers* regionales (más de 10 racks).

Son los centros de cálculo que están ubicados más cerca del usuario y las fuentes de los datos que los centros *Cloud computing*. Al igual que los anteriores están prefabricados y son flexibles en términos de configuraciones, necesitando también requerimientos específicos de alimentación y refrigeración.

2.3.4 Ejemplos de Edge computing

Se describen a continuación algunos ejemplos [11] [5] de aplicaciones donde se usa el *Edge computing*:

Transporte ferroviario:

- Mejorar la seguridad del pasajero controlando las cámaras de trenes y estaciones. Monitorizar los frenos y ruedas para realizar un mantenimiento preventivo y evitar así accidentes. Enviar fragmentos de video pertenecientes al descarrilamiento de un tren.
- Alertas automáticas a los conductores para prevenir errores humanos.
- Proporcionar conectividad WiFi a los pasajeros haciendo que los nodos Edge actúen como puntos de acceso.

Soporte a industrias productivas:

- Mejorar los cambios de modelo y producto para reducir costes. Los nodos actúan de intermediarios entre la red IP y las redes propietarias de control de maquinaria industrial.
- Reducir los paros de máquina: realizando mantenimiento preventivo detectando averías antes de que se produzcan.
- Asegurar los datos de procesos sensibles a ataques: añadiendo políticas de seguridad de acceso a dichas máquinas.
- Asegurar los sistemas de seguridad: actuar sobre sistemas críticos sin la necesidad de interactuación humana.

Otras utilidades:

- Restaurar la alimentación más rápidamente: analizando la carga en todo momento para detectar posibles fallos.
- Detectar brechas de seguridad: respuesta automatizada ante ciber-ataques.
- Reducir los costes de mantenimiento incrementando la fiabilidad del sistema, al igual que en los casos ya comentados, el *Edge computing* analiza los datos para adelantarse a los acontecimientos, en este caso, realizando un mantenimiento preventivo en caso necesario.

2.3.5 Resumen. Cloud vs Edge computing

En este apartado se resumen las características principales de ambos métodos de computación y cómo pueden coexistir:

Nodos o Gateway en el Edge:

- Reciben datos de los dispositivos ioT usando cualquier protocolo soportado en tiempo real, casi al instante.

- Ejecutan aplicaciones para el control en tiempo real y generación de datos históricos con respuestas de milisegundos.
- Proporcionan almacenaje temporal, dependiendo de la capacidad del sistema, de normalmente 1 o 2 horas.
- Envían los datos resumen a la nube.

Plataforma Cloud:

- Recibe y añade los datos de distintos nodos Edge.
- Proporcionan análisis de datos de otras fuentes para analizarlos para generar oportunidades de negocio.
- Pueden enviar nuevas reglas a los nodos en el Edge basados en los datos históricos.

Para concluir este apartado se resume en la siguiente tabla comparativa [12] los parámetros característicos que han sido discutidos hasta ahora.

Parámetros	Cloud Computing	Edge Computing
Localización Servidor/es	Internet	Dentro de red local (Edge)
Distancia (cliente --> servidor)	Multiples Hops	Un único Hop
Latencia	Alta	Baja
Variación de Latencia	Alta fluctuación	Baja fluctuación
Seguridad	- Segura, indefinida	+ Segura, se puede definir
Información de localización	No	Si
Vulnerabilidad	Alta probabilidad	Baja probabilidad
Distribución	Centralizada	Densa y distribuida
Nº nodos servidores	Pocos	Elevado
Respuesta en Tiempo real	Si	Si
Conexión a Internet	Dedicada	Inalhámbrica
Soporte Mobilidad	Limitada	Si

Tabla 3: Tabla comparativa entre Cloud y Edge computing

2.4 Protocolos existentes para ioT

Existen diferentes y diversos estándares disponibles para facilitar la labor de los programadores e ingenieros que se dedican a desarrollar aplicaciones para el internet de las cosas. Todos ellos han sido creados por diversos grupos como son el W3C (*World Wide Web Consortium*), el IETF (*Internet Engineering Task Force*), el IEEEE (Intitute of electrical and Electronics Engineers) y la ETSI (*European Telecommunications Startandars Institute*) y como se puede ver en la siguiente figura son numerosos.

Application Protocol		DDS	CoAP	AMQP	MQTT	MQTT-SN	XMPP	HTTP REST
Service Discovery		mDNS				DNS-SD		
Infrastructure Protocols	Routing Protocol	RPL						
	Network Layer	6LoWPAN				IPv4/IPv6		
	Link Layer	IEEE 802.15.4						
	Physical/ Device Layer	LTE-A	EPCglobal	IEEE 802.15.4		Z-Wave		
Influential Protocols		IEEE 1888.3, IPSec				IEEE 1905.1		

Tabla 4: Tabla resumen de los protocolos ioT más importantes [2]

Dichos estándares tienen tres cualidades básicas a cumplir:

- Comunicación de bajo consumo: debido al hecho que los dispositivos utilizan baterías para poder funcionar, por lo que los protocolos que se utilicen deben de estar optimizados en este sentido.

- Comunicación fiable: Internet proporciona mecanismos para asegurar la fiabilidad en la comunicación pero para combinar el ioT con Internet es necesario que exista una ecualización y nivelación de ésta, para llegar a tener niveles parecidos.

- Conexión con Internet: los dispositivos ioT deben utilizar el protocolo IP para comunicarse.

En el desarrollo del presente proyecto se han utilizado básicamente el protocolo CoAP y el IEEE 802.15.4e que serán los que se detallen a continuación en los siguientes apartados.

2.4.1 IEEE802.15.4-2006

IEEE802.15.4-2006 es la segunda versión del estándar IEEE802.15.4. Define la capa física y la capa de control de acceso al medio (Medium Access Control, MAC). Está diseñado para Low Power y *Low Rate Wireless Personal Area Networks* (LR-WPAN) y se ha convertido en un estándar para las redes WSN [18]. La base de IEEE802.15.4 es permitir bajo consumo, comunicación de baja velocidad entre dispositivos (hasta 65 mil dispositivos) cercanos con poca o sin infraestructura. Su modo de funcionamiento básico tiene un alcance de 10 metros con tasas de 250 kbps.

2.4.1.1 Capa física

La capa física tiene las siguientes características [18]:

- Bajo consumo de operación.
- Baja velocidad de transmisión 250 a Kbps.
- Soporte multicanal con tiempo de respuesta rápido 192 μs.

- Funciones de gestión del consumo como la calidad del enlace y la detección de energía.

Se definen cuatro capas físicas:

• 868/915 MHz: técnica DSSS con modulación BPSK.
• 868/915 MHz: técnica DSSS con modulación O-QPSK.
• 2450 MHz: técnica DSSS con modulación O-QPSK.
• 868/915 MHz: técnica PSSS con modulación ASK.

La banda de 868 MHz (2 canales) se utiliza en Europa y la banda de 915 MHz (30 canales) en Norte América y Australia. La banda ISM de 2.4 GHz se definen 16 canales de 2 MHz de ancho separados 5 MHz para minimizar interferencias entre canales adyacentes. El estándar requiere que los transmisores de radio tengan una sensibilidad mínima de -85 dBm o mejor, en la práctica, los transceptores tienen sensibilidades entre los valores de -95 dBm y -100 dBm.

2.4.1.2 Capa MAC

Básicamente la capa MAC ha de cumplir [18]:

- Asignación en tiempo real adecuada para la reserva garantizada de intervalos de tiempo.
- Prevención de colisiones a través de CSMA/CA.
- Soporte integrado para comunicaciones seguras (cifrado AES de 128 bits).
- Construcción de topologías en estrella y malladas.
- Apoyo a bajas latencias y dispositivo de direccionamiento dinámico.

Los protocolos MAC definen dos métodos de acceso al canal:

- Sin *Beacon (baliza)*: utiliza el estándar CSMA para resolver los conflictos de acceso al medio. Los paquetes recibidos con éxito son reconocidos positivamente.
- Con *Beacon*: se sigue una estructura de súper-trama, donde un coordinador de la red transmite balizas a intervalos predeterminados. Tras la baliza viene un periodo de contención de acceso (*Contention Access Period* o CAP) donde los vecinos del coordinador pueden hablar con él a través de un esquema de acceso al medio ranurado. Durante este periodo, los nodos pueden solicitar un espacio de tiempo dedicado al periodo *Guaranteed Time Slot* (GTS). Posteriormente, se inicia el *Contention Free Period* (CFP) donde los nodos pueden realizar transmisiones con latencia limitada en su espacio de tiempo dedicado asignado anteriormente. Una vez finalizado el CFP comienza el periodo inactivo donde ningún nodo puede transmitir. Todos los nodos se encuentran en modo *sleep* para ahorrar energía. Pasado un cierto periodo de tiempo se vuelve a transmitir la baliza, repitiéndose todo el proceso antes explicado.

Dentro del protocolo IEEE802.15.4-2006 existen tres tipos de funciones que pueden llevar a cabo loa dispositivos [18]:

1. Coordinador PAN (*Personal Area Network*): coordinador que es el controlador principal de la PAN. Una red sólo tiene un coordinador PAN.

2. Coordinador: nodo final con capacidad de proporcionar coordinación y otros servicios en la red.

3. Nodo final: dispositivo con capacidad de acceder al medio inalámbrico a través de las definiciones del estándar IEEE802.15.4, a nivel físico y MAC.

A parte de estas 3 funciones principales que pueden desarrollar se definen otras dos que son las siguientes [18]:

- *Full Function Device* (FFD): son nodos con capacidad de participar en cualquier topología de red y pueden realizar las funciones explicadas arriba definidas arriba (coordinador PAN, coordinador y nodo final). Además, pueden comunicarse con cualquier otro nodo de la red.

- *Reduced Function Device* (RFD): son nodos simples con recursos de hardware y software limitados. Sólo se pueden comunicar con FFD pero nunca pueden ser coordinadores, son nodos finales. Sólo pueden participar en una red con topología estrella.

Respecto al tipo de topologías existen tres: en estrella (*star*), punto a punto (*peer to peer, P2P*) y en árbol (*cluster tree*) como se puede observar en la figura siguiente:

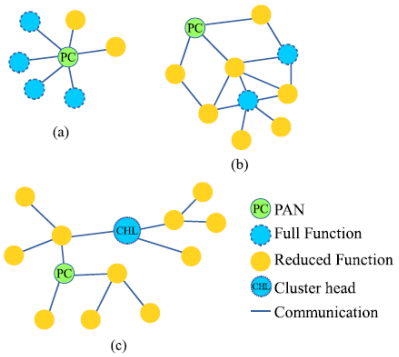


Ilustración 7: Topologías IEEE802.15.4 [2]: (a) Estrella, (b) P2P, (c) Cluster-tree

2.4.2 IEEE802.15.4e

Desarrollado por el grupo de trabajo IEEE 802.15 *WPAN Task Group 4* y define una variante de la capa MAC del estándar IEEE802.15.4-2006 [17] con el objetivo de dar apoyo a los mercados industriales, incorporando nuevos modos de operación para apoyar diferentes tipos de aplicaciones de red como son los siguientes:

- *Deterministic & Synchronous Multi-channel Extension* (DSME): para aplicaciones de alta disponibilidad, eficiencia, escalabilidad y robustez.
- *Low Latency Deterministic Network* (LLDN): para aplicaciones de baja latencia como robots, grúas, etc.
- *Time Slotted Channel Hopping* (TSCH): para aplicaciones destinadas a entornos industriales donde el consumo de energía debe ser reducido y la comunicación debe ser robusta frente interferencias.
- *Radio Frequency Identification Blink* (RFID Blink): para aplicaciones de identificación de objetos o personas, localización o seguimiento.
- *Asynchronous Multi-Channel adaption* (AMCA): para redes de gran tamaño y dispersión geográfica como redes de monitorización de infraestructuras y de control de procesos.

Se define el protocolo *Low Energy* (LE) que permite obtener ciclos de trabajo del orden del 1%, incorpordo dos mecanismos para reducir el consumo de energía que con los modos CSL (*Coordinated Sampled Listening*), donde el receptor escucha el canal periódicamente verificando si hay peticiones de transmisión.

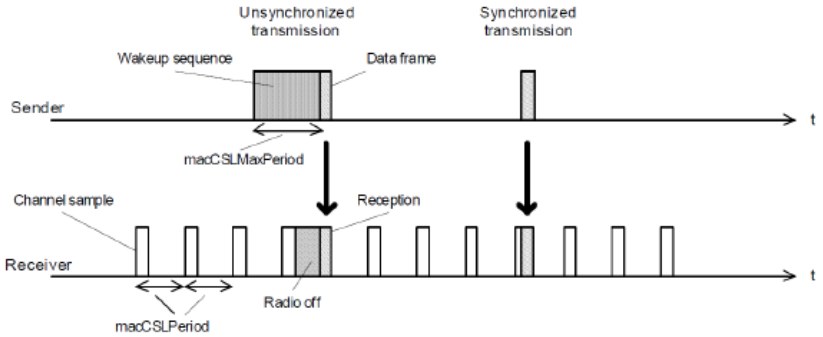


Ilustración 8: IEEE802.15.4e, funcionamiento del modo CSL [17]

Y el modo RIT (*Receiver Initiated Transmission*) que se utiliza en redes PAN que no emplean balizas. Se basa en que el receptor envía tramas *datareq* de forma periódica a través de CSMA/CA no ranurado, después de cada envío escucha el canal durante periodo corto de tiempo. El transmisor escucha el canal a la espera de recibir una trama datareq para transmitir inmediatamente una trama de datos.

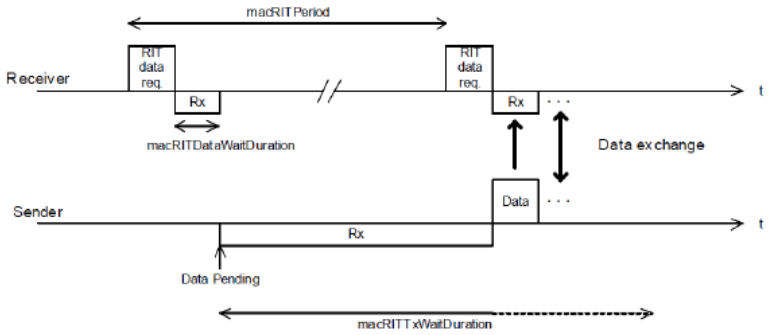


Ilustración 9: IEEE802.15.4e, funcionamiento del modo RIT [17]

El IEEE802.15.4e define los *Information Elements* (IE) como un método flexible y sencillo de implementar para el encapsulamiento de información. Un IE está formado por un campo identificador (ID), un campo de longitud y un campo de contenido. Los IEs proporcionan contenedores de información de sincronización, estado de la red, etc. y permite añadir nuevas definiciones de IEs en futuras versiones del estándar. El contenido de un IE es un campo variable en función del modo de funcionamiento, funcionalidades y modos de comportamiento (CSL, RIT, balizas, etc.). La trama MAC del estándar IEEE802.15.4e está formada por tres partes [17]:

1.- *MAC Header* (MHR): está formada por campos:

- *Frame Control*: tipo de trama.
- *Sequence Number*: número único en la trama.
- *Adressings Fields*: identificadores, direcciones origen y destino de la PAN.
- *Auxiliary Security Header*: opcional, extensión del mecanismos de seguridad de la información de niveles superiores.
- Cabecera *Information Elements* (IEs): datos adicionales orientados al funcionamiento y comportamiento específico de determinados elementos.

2.- *MAC Payload* o *MAC Service Data Unit* (MSDU): datos utiles de los IE y de la trama (puede corresponder con las PDUs de niveles superiores).

3.- *MAC Footer* (MFR): dos bytes para el almacenamiento de la secuencia de control de la trama (Frame Control Sequence o FCS) para verificar y validar la integridad de los datos.

<table>
<tr><td>Octets: 1/2</td><td>0/1</td><td>0/2</td><td>0/1/2/8</td><td>0/2</td><td>0/1/2/8</td><td>0/1/5/6/10/14</td><td colspan="2">variable</td><td>variable</td><td>2</td></tr>
<tr><td rowspan="2">Frame Control</td><td rowspan="2">Sequence Number</td><td>Destination PAN Identifier</td><td>Destination Address</td><td>Source PAN Identifier</td><td>Source Address</td><td rowspan="2">Auxiliary Security Header</td><td colspan="2">Information Elements</td><td rowspan="2">Frame Payload</td><td rowspan="2">FCS</td></tr>
<tr><td colspan="4">Addressing fields</td><td>Header IEs</td><td>Payload IEs</td></tr>
<tr><td colspan="7">MHR</td><td colspan="2">MAC Payload</td><td>MFR</td></tr>
</table>

Ilustración 10: IEEE802.15.4e, trama MAC [17]

El estándar IEEE802.15.4e introduce dos nuevas versiones de los *beacons* que se utilizan en los modos de funcionamiento DSME y TSCH y su contenido está formado a partir de los IE. Estas son:

- *Enhanced Beacon* (EB): proporcionan más flexibilidad en contenido que las balizas del estándar IEEE802.15.4-2006. Los EB tienen el valor "0b10" en campo versión de la trama.

- *Enhanced Beacon Requests* (EBR): se utilizan para pedir un EB especificando filtros en la respuesta que el EB responda sólo la información demanda, reduciéndose así el tamaño de la trama enviada.

2.4.3 CoAP

El Constrained Application Protocol (CoAP) [16] fué creado por el grupo del IETF denominado por las siglas CoRE (Constrained RESTful Environments) para el mundo ioT el cual define un protocolo de la capa de aplicación basado en la transferencia web utilizando la arquitectura REST por encima de la capa con funcionalidades HTTP. Por tanto CoAP permite una via simple de comunicación e intercambio de datos entre clientes y servidores sobre HTTP.

Una de las diferencias entre los protocolos CoAP y HTTP es que el primero envía los mensajes de forma asíncrona a través del protocolo UDP, mientras que HTTP establece conexiones TCP para enviar peticiones y respuestas. El modelo de funcionamiento del CoAP es parecido a la arquitectura cliente/servidor del protocolo HTTP pero la diferencia radica en que en primero las funciones cliente/servicio no están bien definidos debido a que utiliza el protocolo UDP y los dos puntos actúan como clientes y servidores.

Las principales características del protocolo son [16]:

- Asíncrono.
- Cumple con los requerimientos M2M en ciertos entornos.
- Conexión UDP [RFC0768] con fiabilidad opcional que soporta solo solicitudes unicast y multicast.
- Cabecera de los paquetes es reducida.
- Soporta esquema URI.
- Capacidad de chaching y proxy simple.
- Mapeo HTTP sin estado a través del uso de proxies o asignación directa de las interfaces HTTP a CoAP.
- Descubrimiento de recursos opcional.
- Mecanismo Observe para notificaciones asíncronas.
- Seguridad vinculante a DTLS [RFC6347].

2.4.3.1 CoAP: arquitectura interna

La arquitectura interna está formada por dos capas [16]:

1.- Capa de mensajería (messaging layer): Su función es controlar los intercambios de mensajes a través de UDP entre dos endpoints. Las peticiones y las respuestas comparten el mismo formato de mensaje. Los mensajes son identificados por un ID utilizado para detectar duplicidades y para añadir mas fiabilidad. Existen cuatro tipos de mensajes, confirmables (CON) que necesitan confirmación (ACK), No confirmables (NON) que no necesitan confirmación por su baja importancia, mensajes de confirmación (ACK) y mensajes de Reset (RST) como respuesta del receptor cuando no puede contestar a un mensaje CON o NON.

2.- Capa de peticiones/respuesta (Request/Response layer): La estructura de peticiones y respuestas del CoAP se realizan en los mensajes que incluyen un

código de método o de respuesta y opcionalmente la información de la petición y la respuesta, como la URI y el tipo de contenido de los datos (*payload*) se realiza como opciones del CoAP. Un *Token Option* se utiliza para que coincida con las respuestas a peticiones pendientes de confirmar. Como el CoAP se ejecuta sobre UDP, por naturaleza no seguro, los mensajes pueden llegar desordenados, duplicados o perdidos. Se necesita por tanto implementar un mecanismo de fiabilidad:

- Retransmisión fiable *stop-and-wait* con *back-off* exponencial para mensajes tipo CON.
- Detección de mensajes duplicados por los tipos CON y NON.
- Soporte *multicast*.

2.4.3.2 Los mensajes en CoAP

Los mensajes CoAP están codificados en un formato binario simple formado por una cabecera de longitud fija de 4 bytes seguida por el campo Token, después las opciones en el formato TLV (Type-Length-Value) y el contenido del mensaje (payload). Este payload está compuesto por los datos que se encuentran después de la trama de opciones (opcional) y su longitud se calcula a partir de la longitud del datagrama.

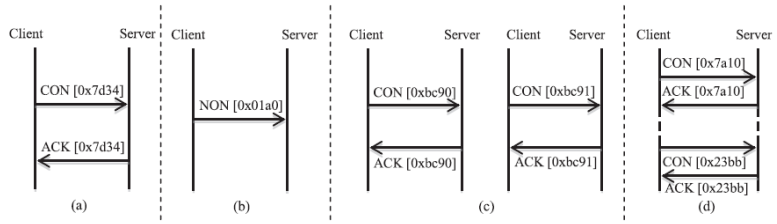


Ilustración 11: Mensajes en CoAP [2]: (a) Confirmable, (b) No Confirmable, (c) Respuesta Piggybacked, (d) Respuesta separada

Los principales campos de los mensajes son [14]:

- Ver (*version*): 2 bits que indican la versión de CoAP. Se establece el valor 1 en la versión actual.
- T (*type*): 2 bits per especificar el tipo de mensaje: CON (0), NON (1), ACK (2) i RST (3).
- OC (*Option Count*): 4 bits que indican la longitud del campo *Token*.
- *Code*: 8 bits para definir si el mensaje es una petición o una respuesta. En el primer caso especifica los métodos de petición (GET, POST, etc.). En el segundo caso indica el código de respuesta (2.01, 4.03, etc.).
- *Message ID*: 16 bits para definir un identificador único, utilizado para la detección de duplicidades y mapear los tipos de mensajes ACK/RST a tipos de mensaje CON/NON.
- *Token*: si existe su longitud es variable hasta 8 bytes y está definida en el campo TKL. El valor del campo *Token* se utiliza para relacionar las peticiones y respuestas.

- *Options*: opciones del mensaje.
- *Payload*: contenido del mensaje.

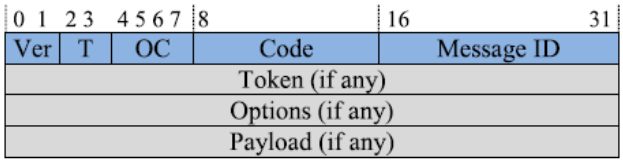


Ilustración 12: Formato del mensaje en CoAP [14]

2.4.3.3 Métodos CoAP

CoAP al estar basado en HTTP utiliza los métodos GET, PUT, POST y DELETE [14] para obtener, pedir, actualizar y borrar datos. Se describen a continuación:

- *GET*: permite recuperar una representación de la información correspondiente al recurso identificado por la petición URI. Si la petición es válida se contestará con un código de respuesta 02:05 (Content) o 2:03 (Valid).

- *POST*: requiere que la representación del recurso incluida en la petición URI sea procesada. Normalmente el resultado es la creación de un nuevo recurso o la actualización del recurso indicado. Si el recurso se ha creado en el servidor contesta con un código de respuesta 02:01 (Created) y debería incluir en la respuesta la URI del nuevo recurso. Si la petición es correcto pero no se ha podido crear el nuevo recurso, el código de respuesta debería ser 02:04 (Changed). Si la petición es válida y el recurso ha sido eliminar, el código de respuesta debe ser 02:02 (Deleted).

- *PUT*: el recurso indicado en la petición URI debe ser actualizado o creado con la representación incluida en el mensaje. El formato de ella representación está definido por el tipo de media incluido en la opción Content-Type. Si el recurso de la petición URI existe, la representación incluida debe modificar la versión de este recurso y responder con el código 02:04 (Changed). Si el recurso no existe, el servidor debe crear un nuevo recurso con esta URI y enviar el código de respuesta 02:01 (Created). Si el recurso no se puede crear o modificar, se envía el código de error correspondiente.

- *DELETE*: el recurso indicado en la petición URI debe ser eliminado. Si se ha podido eliminar el recurso o no existía antes de la petición se envía un código de respuesta 02:02 (Deleted).

Se ha visto como en estos métodos las respuestas están codificadas por de forma similar a los códigos de estado del protocolo HTTP (petición correcta 2.XX, error del cliente 4.XX y error del servidor 5.XX).

2.4.3.4 Esquema URI

Las URIs [14] del CoAP son similares a las de HTTP, identificando cada recurso COAP para proporcionar los medios necesarios para localizarlos. De la misma forma que en las arquitecturas *RESTful*, los recursos están organizados jerárquicamente en un potente servidor que escucha peticiones por un determinado puerto (normalmente el puerto 5683). Un ejemplo de dirección URI es el siguiente:

coap://[bbbb:0:0:0:1415:92cc:0:2]:5683/temperatura

, donde el host es una dirección IP (en este caso IPv6) o un nombre que pueda ser resuelto (DNS), el puerto UDP por donde escucha el servidor y el recurso que se pide (en este caso una petición de medición de temperatura).

3. Sistemas operativos (SO) para el Internet de las cosas

Los dispositivos ioT son sistemas sistemas embebidos que soportan modos de bajo consumo y que los sistemas operativos convencionales no son capaces de explotar sus características al máximo. Es por este motivo que se han desarrollado sistemas para adecuarse a esta arquitectura y que cumplen con algunos (no todos) de los estándares de ioT, de hecho muchos de ellos se han desarrollado en el entorno universitario con licencia de *opensource*. Los más destacados son: TinyOS, Contiki, RIOT, Nimbits y *OpenWSN*.

Operating System	Language Support	Minimum Memory (KB)	Event-based Programming	Multi-threading	Dynamic Memory
TinyOS	nesC	1	Yes	Partial	Yes
Contiki	C	2	Yes	Yes	Yes
LiteOS	C	4	Yes	Yes	Yes
Riot OS	C/C++	1.5	No	Yes	Yes

Ilustración 13: SO usados en IoT [2]

3.1 TinyOS

TinyOS [33] es un sistema operativo de código abierto para redes de sensores inalámbricos. Está escrito en lenguaje de programación nesC, un dialecto de la sintaxis C optimizado para evitar los problemas derivados de las limitaciones de memoria que existen dentro de las redes de sensores. TinyOS es un proyecto conjunto de la Universidad de Berkeley e Intel. Existen herramientas y librerías en C o Java que aumentan sus funcionalidades y oportunidades de uso.

Ilustración 14: Logo de TinyOS

Proporciona implementaciones de capa MAC para redes IEEE802.15.4 y para los estándares 6LoWPAN, RPL i CoAP a través de BLIP2.0 (Berkeley Low-Power IP Stack).

3.2 Contiki

Contiki [36] es un sistema operativo de código abierto para redes WSN desarrollado por el *Swedish Institute of Computer Science* para sistemas del Internet de las Cosas. Permite la conexión de sistemas con microprocesadores de 8-bit o sistemas integrados sobre micro controladores, incluyendo nodos de redes de sensores. Se utiliza en la monitorización de ruidos, medición de energía eléctrica, sistemas de alarma, domótica, vigilancia remota, etc.

Está basado en protocolos y estándares como IPv4, IPv6, 6lowpan, RPL y CoAP. Sus características son:

- Protohilos de ejecución.
- Navegador web.
- Servidor web.
- Conectividad TCP/IP.
- Kernel multitarea.
- Cliente remoto usando VNC (Computación Virtual en Red).

3.3 RIOT

Sus creadores [34] lo definen como “el sistema operativo amigable para el Internet de las Cosas” y está basado en una arquitectura de micro-kernel. Se ejecuta en *hardwares* de 8, 16 y 32 bits y, mediante un puerto nativo, tanto en entornos Linux como en Mac OS. Permite el desarrollo de aplicaciones mediante una programación estándar en lenguajes C y C++ bajo licencia LGPL.

Ilustración 15: Logo de Riot

3.4 Nimbits

Nimbits [35] es una plataforma (PaaS) de registro de datos para la conexión de sensores en la nube.

Ilustración 16: Logo de Nimbits

Es un servicio de código abierto que permite conectarse a redes sociales como Facebook o Twitter, a bases de datos, al motor de conocimiento computacional WolframAlpha. Algunas de sus características fundamentales:

- Usa el framework de desarrollo Spring.
- Dispone de una API REST.
- Se pueden cargar y descargar datos en formato CSV.

3.5 *OpenWSN*

El proyecto *OpenWSN* [19] es una implementación en lenguaje C *open-source* de la pila de protocolos estándar del Internet de las cosas (*IoT*) como son el *6LoWPAN*, *IEEE802.15.4e* y *COAP* utilizando una gran variedad de hardware y plataformas de software que implementa el modo *TSCH* (*Timeslotted Channel Hopping*). El modo *TSCH* permite la sincronización de los motes siguiendo una planificación ofreciendo el ajuste de los paquetes que son generados todo ello para conseguir una buena latencia, robustez de los datos y un bajo consumo al utilizar eficientemente el espectro de radiofrecuencia. *OpenWSN* hace posible el despliegue de redes malladas con fiabilidad y muy bajo consumo con funcionalidades IP.

Ilustración 17: Logo de OpenWSN

Está basado en protocolos y estándares como IPv4, IPv6, IETF 6LoWPAN, IETF RPL, CoAP, HTTP y IEEE802.15.4e para el acceso al medio. También decir como punto a remarcar que es compatible con TinyOS [33], Contiki [37] y RIOT [34].

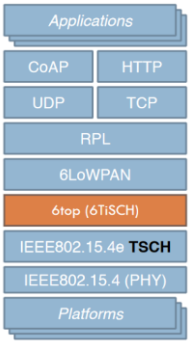


Ilustración 18: OpenWSN, protocolos y estándares soportados [21]

Compatible con las siguientes plataformas de hardware [20]:

- TelosB GINA
- WSN430
- Zolertia Z1
- OpenMote-CC2538
- OpenMoteSTM
- USP Mote CC2538
- IoT-LAB_M3
- AgileFox

3.5.1 Pila de protocolos

En *OpenWSN* se utilizan dos niveles de abstracción [21]:

- *Berkeley Socket*: fue desarrollado como parte del desarrollo del sistema operativo *Berkeley Software Distribution* (*BSD*). El *BSD* fue adoptado por todos los sistemas operativos y hoy es el corazón de Internet. Considera que aplicaciones en dos hosts de Internet se comunican entre ellas a través de un socket identificado de forma unívoca por sus direcciones *IP* y los puertos correspondientes de cada aplicación. La pila *OpenWSN* respeta esta abstracción, por tanto, el desarrollo de una aplicación en la parte superior de la pila *OpenWSN* es muy similar al desarrollo de una aplicación de cualquier host.

- Hardware: consiste en agrupar todas las funciones con acceso a hardware, por ejemplo funciones de escritura de registros, en un grupo de archivos llamados *Board Support Package* (*BSP*). Esto permite que la gran mayoría del código sea compartido entre todas las plataformas. Hay un *BSP* para cada plataforma soportada, el resto del código de la pila se comparte con todas las implementaciones.

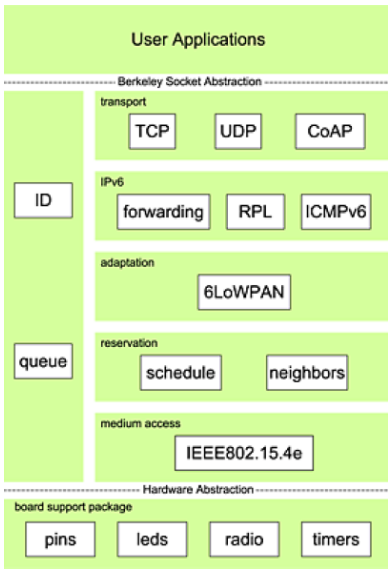


Ilustración 19: Niveles de abstracción [19]

La pila de protocolos está distribuida en varias capas con la implementación de su respectivo estándar como se puede ver en la siguiente tabla:

application	CoAP, HTTP
transport	UDP, TCP
IP/routing	IETF RPL
adaptation	IETF 6LoWPAN
medium access	IEEE802.15.4e
phy	IEEE802.15.4-2006

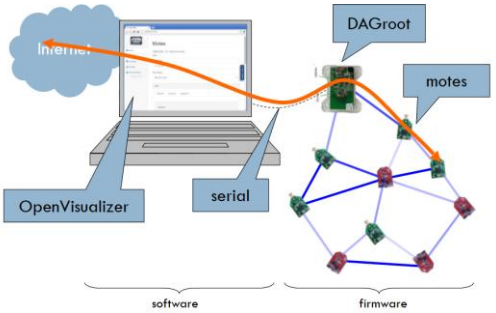


Ilustración 20: Protocolos soportados por OpenWSN y su arquitectura [21]

3.5.2 OpenOS

OpenOS [19] es el planificador principal desarrollado como parte del proyecto *OpenWSN* [19]. Las interrupciones de hardware y de los temporizadores ordenan las tareas en función de su prioridad y se añaden a una lista de tareas. Si existen tareas en la lista, el planificador llamada a la función *callback* asociada a cada tarea y la elimina del listado. En la situación que la lista esté vacía, el planificador informa al micro para que entre en modo de sueño profundo (*deep sleep*), a la espera de una nueva interrupción para ser añadida a la lista de tareas. *OpenOS* es colaborativo significando que el planificador no expulsa a la tarea que se está ejecutando hasta finalizar su tiempo de ejecución. La pila *OpenWSN* no está directamente ligada al planificador *OpenOS*, por tanto, la pila se puede ejecutar como parte de un sistema operativo diferente.

3.5.3 OpenVisualizer

OpenVisualizer [23] es un programa de visualización y depuración desarrollado en lenguaje Python que se ejecuta en un ordenador para interactuar con los nodos *OpenWSN* (motes) conectadas a él. Sus características son:

- Conecta su red *OpenWSN* a Internet a través de una interfaz virtual (Windows y Linux).
- Portable a diferentes sistemas operativos.
- Muestra el estado interno (tabla vecina, tabla de programación, cola, etc.) de cada nodo conectado físicamente al *OpenVisualizer*.
- Muestra los errores reportados por motes.
- Puede funcionar con motes físicas, o motes emuladas (*OpenSim*).
- Permite configurar un mote como *DAG root* (nodo coordinador, actuando como *bridge*).
- Permite compresión *6LoWPAN*.
- Calcula la ruta origen *RPL*.

OpenVisualizer dispone de varias interfaces de gestión:

- Una interfaz gráfica (GUI).
- Una interfaz de línea de comandos (CLI).
- Una interfaz Web.

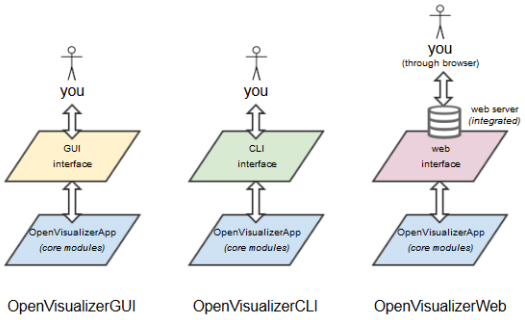


Ilustración 21: Openvisualizer: interfaces disponibles [23]

La comunicación con un nodo (*mote*) se realiza conectando dicho nodo en el ordenador a través del puerto serie o mediante una simulación si no se dispone del hardware (usando *OpenSim*). Esta es una de las ventajas más importantes que dispone *OpenWSN* ya que es posible simular el entorno de comunicación.

Dentro del *OpenVisualizer* se muestra información de red y del estado interno del mote (conectividad, tablas de nodos vecinos y del planificador y estado de las colas), gráficos de conectividad multi-salto (*multi hop*), códigos de error y depuración del mote. En la información de conectividad especifican los datos si la red está sincronizada, *ASN*, el *DAG rank*, en *DAG root*, la dirección *IPv6*, etc. En la tabla de nodos vecinos se muestran datos como la dirección *IPv6*, el *DAG rank*, *RSSI* y *ASN*.

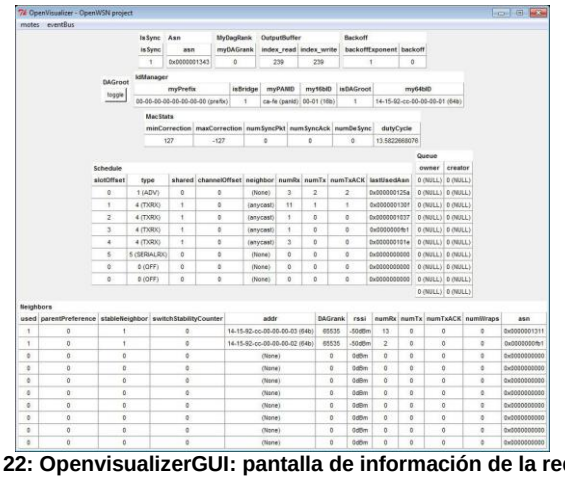


Ilustración 22: OpenvisualizerGUI: pantalla de información de la red WSN [23]

3.5.4 OpenSim

Una red simulada se comporta exactamente como una red con hardware real. Puede interactuar con *OpenVisualizer*, comunicarse con sus nodos desde Internet. La diferencia es que el firmware de cada mote está ejecutándose en su ordenador en lugar de ejecutarse en un dispositivo físico. *OpenSim* [22] lo hace compilando el firmware del mote como un módulo de extensión Python y creando una instancia de la clase resultante para cada mote emulado. Cuando la simulación se está ejecutando, se simula como los motes se comunican con el resto de la arquitectura *OpenVisualizer*.

Los motes simulados interactúan con el *eventBus* de la misma manera que una instancia *moteProbe* (conectada a un mote de hardware real). El *OpenVisualizer* no se preocupa si está hablando con motes simulados o reales, y desde un punto de vista de red, interactuar con los motes simulados es exactamente lo mismo que interactuar con motes reales.

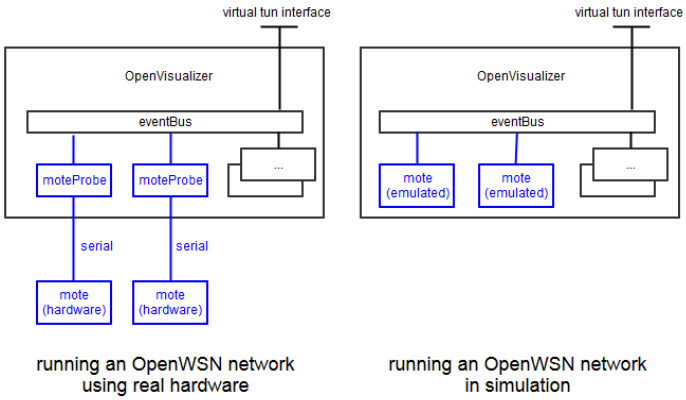


Ilustración 23: interactuación de motes reales y simulados con el eventBus [22]

3.5.5 Librería Python CoAP

Como se introdujo en los objetivos del presente trabajo se utilizará la librería hecha en Python para soportar el protocolo CoAP [24] dentro de *OpenWSN* la

cual implementa un cliente y servidor CoAP con un número arbitrario de recursos la cual permite las siguientes funciones:

- Enviar mensajes de tipo *confirmable* (CON) o *Non-confirmable* (NON).
- Parametrizar el número de reintentos y los *timeouts*
- Soporte de peticiones concurrentes.

La librería no da soporte a funcionalidades de *caching*, *proxying*, seguridad DTLS o *multicast*. En la figura de abajo se puede observar la máquina de estados de la librería cuando debe transmitir una petición de cualquiera de los métodos COAP (GET, PUT, POST y DELETE) en función del tipo de mensaje CON o NON. La transmisión del mensaje tipo CON se basa en esperar durante un periodo de tiempo (*timeout* ACK) a recibir un paquete ACK. Si dentro de este periodo se recibe el paquete ACK se devuelve la respuesta. Si pasado el *timeout* no se ha recibido el paquete ACK se vuelve a repetir la transmisión del mensaje tipo CON. La repetición de la transmisión del mensaje tipo CON finaliza al llegar al número máximo de reintentos. La transmisión del mensaje tipo NON consiste también en esperar para recibir un paquete ACK durante el *timeout* ACK. Pero en este caso no hay repetición de la transmisión sino llega el paquete ACK.

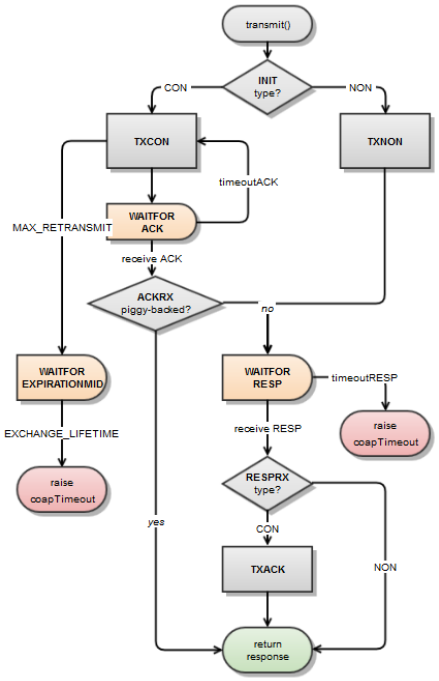


Ilustración 24: Máquina de estados de librería CoAP [24]

3.5.6 Ejecutando una simulación con OpenSim

El entorno OpenSim [22] combina elementos de los siguientes repositorios:

https://github.com/*OpenWSN*-berkeley/*OpenWSN*-fw

https://github.com/*OpenWSN*-berkeley/*OpenWSN*-sw

El entorno OpenSim asume que dichos repositorios han sido clonados al mismo nivel. Esto significa que necesitamos tener los directorios *openwsn*-sw y o*penwsn*-fw en la misma ubicación en el ordenador.

Requerimientos de instalación del OpenSim [22]:

- Debe poder ejecutar Openvisualizer, así que nos tenemos que asegurar de instalar los elementos necesarios para que *Openvisualizer* se ejecute.
- El ordenador necesita tener el compilador *gcc* instalado para poder compilar el firmware como un módulo de extensión de Python. En Linux, debería ser el caso por defecto. En Windows se recomienda usar http://www.mingw.org/

- Para poder compilar el firmware (*openwsn*-fw), el compilador necesitará tener acceso al archivo de encabezado Python.h.

- Los archivos de encabezado de Python deben estar presentes de forma predeterminada en Windows. En Linux, debemos instalar el paquete python-dev con el siguiente comando:

apt-get install python-dev

3.5.6.1 Compilación del firmware

Antes de poder ejecutar una simulación necesitamos compilar el firmware *OpenWSN* como módulo de extensión Python. Para ello tenemos que ir al directorio *openwsn*-fw/ y teclearemos el comando:

scons board=python toolchain=gcc oos_OpenWSN

Este comando creará el siguiente modulo de extensión en Python.

OpenWSN-fw/firmware/openos/projects/common/oos_OpenWSN.pyd

3.5.6.2 Ejecutar una simulación

Ejecutar una simulación es exactamente como ejecutar *Openvisualizer*, pero especificando en este caso, que se trata de una simulación. Al igual que con el *Openvisualizer*, hay varias interfaces disponibles. Tendremos que ir a la carpeta:

openwsn-sw/software/openvisualizer/bin/openVisualizerApp/

e introduciremos los siguientes comandos dependiendo de qué interface queremos usar:

python openVisualizerGui.py --sim [--simCount=<número de nodos simulados>*]*

python openVisualizerCli.py --sim [--simCount = <número de nodos simulados>]

python openVisualizerWeb.py --sim [--simCount = <número de nodos simulados>]

También podemos utilizar el comando *scons* que es una herramienta de código abierto para facilitar la instalación de software utilizando scripts en Python (como las herramientas *make* y *autoconf* de Unix para compilar e instalar código) desde *openwsn*-sw/software/openvisualizer/:

scons rungui --sim [--simCount=<número de nodos simulados>]

scons runcli --sim [--simCount = <número de nodos simulados>]

scons runweb --sim [--simCount = <número de nodos simulados>]

Los parámetros entre corchetes son opcionales y no son necesarios cuando se ejecuta un comando, pero por ejemplo, para ejecutar una simulación de 5 motes (nodos) en red utilizaríamos el siguiente comando para el caso de uso de interfaz gráfica (para el modo CLI y web el uso de este parámetro sería igual):

scons rungui --sim --simCount=5

Para forzar una topología en concreto con un número determinado de nodos se necesita pasar el parámetro *–simCount* y *–simToplogy* con los valores del numero de nodos y el tipo de red a crear, *linear* o *fully-meshed* (en malla) creándose la red WSN.

scons rungui --sim --simCount=5 –simTopology=linear
scons rungui --sim --simCount=5 –simTopology=fully-meshed

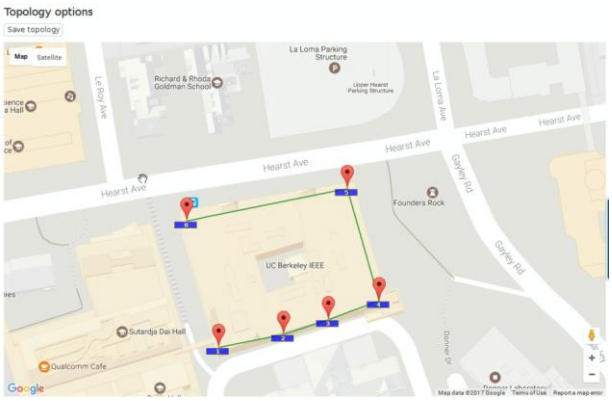


Ilustración 25: Ejemplo de topología lineal

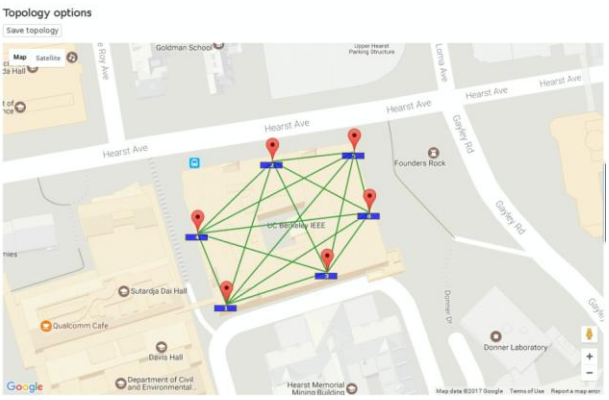


Ilustración 26: Ejemplo de topología fully-meshed (malla)

En el trabajo desarrollado se ha empleado la topología *fully-meshed* (malla) ya que es la que ha incurrido en menos problemas de sincronización entre nodos y fallos en la simulación (cierres inesperados). También se ha comprobado, al menos con la plataforma Rpi utilizada, no aumentar el número de nodos a un valor superior a cinco ya que la simulación termina sin previo aviso.

Una vez explicado los parámetros necesarios para ejecutar una simulación se mostrará una en detalle. Ejecutamos el openvisualizer en modo web y simulación con el comando (sin indicar ningún parámetro más) en modo *superuser* ya que al crearse la interfície *tun0* necesita estos privilegios:

sudo scons runweb –sim

Abrimos la web que se dirige al servidor web que se crea en el puerto 8080, en este caso la IP local es la 192.168.1.40:

http://192.168.1.40:8080/

Mostrándose la siguiente página web:

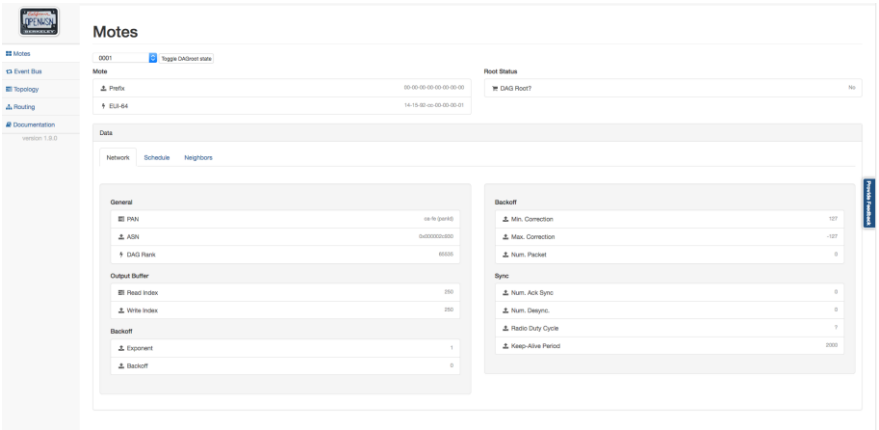


Ilustración 27: Pagina web del servidor creado con openvisualizer

AL pulsar en la opción *Topology* se puede visualizar la topología que se crea (automáticamente cuando existe al menos un nodo DAG root) en la simulación:

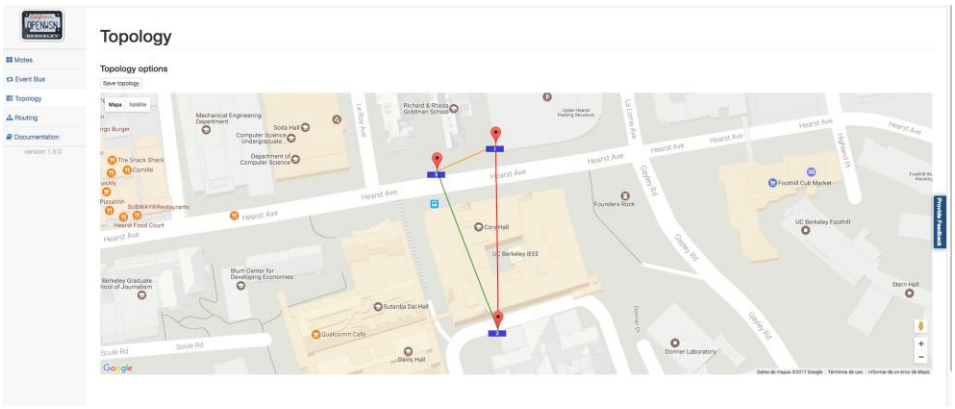


Ilustración 28: Topología de la red

En la figura de arriba se muestra como se crean los enlaces, en rojo y naranja aquellos enlaces en los que los datos no llegan con suficiente señal debido a reintentos o si la señal está degradada debido a obstáculos (*PDR*, *Packet Delivery Ratio*, inferior al 1). Para la simulación modificamos todos los *PDR* a valor 1 para que los nodos se sincronicen sin problemas. Al hacerlo veremos cómo los enlaces cambian a color verde indicándose la máxima visibilidad y recepción de señal entre nodos.

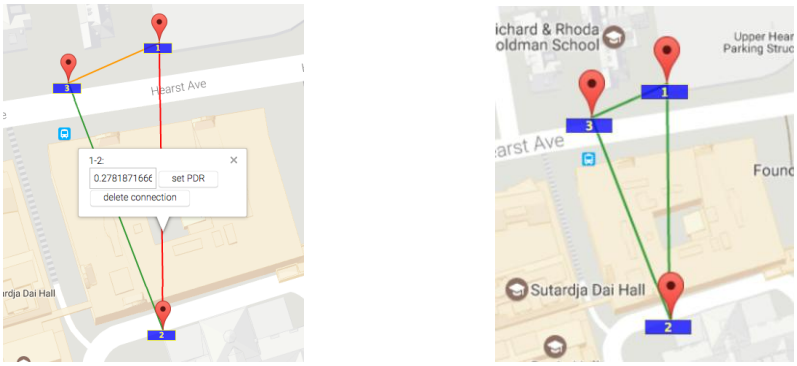


Ilustración 29: Detalle del PDR de un enlace (izquierda), enlace con PDR = 1 (derecha)

En cualquier red WSN ha de existir un gestor de enlace o *DAGroot* que actúa en modo *bridge* por lo que hay que definir uno seleccionando en este caso el nodo 001 y se pulsa el botón "*Toggle DAGroot State*" que activa/desactiva este modo.

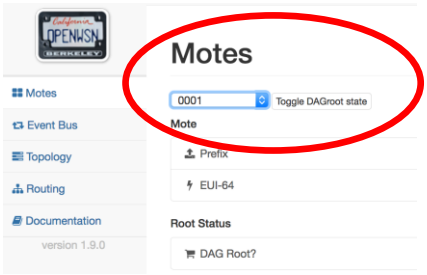


Ilustración 30: DAG root: un mote ha de actuar como puente (bridge)

Al pulsar veremos en la ventana de terminal como se van activando los enlaces y como se asignan las direcciones IPv6 a cada *mote* (con prefijos bbbb::/64), etc. En cualquier caso en el *openVisualizer* es posible ver toda esta información también a parte de mucha otra información a nivel de protocolo (pulsando en la opción *Event Bus*). En esta ventana también se puede activar el modo *debug* para utilizar el programa *Wireshark* para ver todo el tráfico en la interfaz virtual que crea *openSim* llamada "*tun0*".

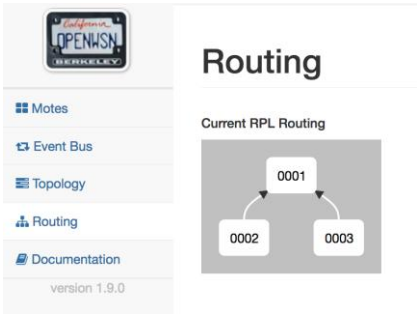


Ilustración 31: Detalle del enlace creado entre motes o nodos

Hacemos un ping (ping6 es el comando lanzar una petición de ECHO ICMP para ipv6) a uno de los motes (bbbb:0:0:0:1415:92cc:0:2):

ping6 bbbb:0:0:0:1415:92cc:0:2

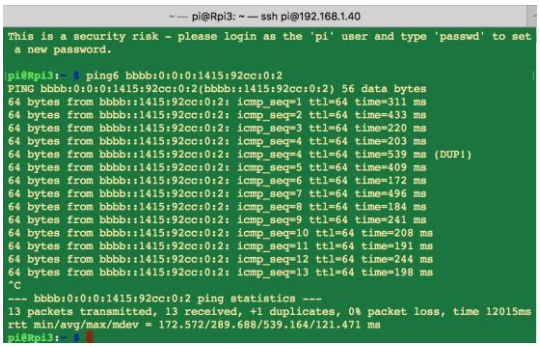


Ilustración 32: Ping al mote con IPv6 bbbb:0:0:0:1415:92cc:0:2

4. Resumen: herramientas Hardware y Software utilizados

En este apartado se darán a conocer las plataformas de hardware y software escogidas y utilizadas para poder implementar el Gateway Edge. Hasta ahora se han expuesto y representado diferentes tipos de procesado de datos y diferentes protocolos de comunicación destinados a gestionar dispositivos ioT, así como los sistemas operativos comúnmente utilizados en este mundo de las "cosas".

En primer lugar se escogerá la Raspberry Pi 3 modelo B [26] debido a su relación coste/funcionalidad, ya que por 40 € tendremos un hardware con una potencia de procesado suficiente para llevar a cabo la ejecución de los scripts

escritos en Python, al contrario de otras soluciones como el uso de la placa PandaBoard [28] ya mencionada en los objetivos del presente trabajo. También se escoge esta solución debido a la gran cantidad de software disponible para la distribución *Jessie*, basada en Debian y a su soporte en Internet.

A nivel de sistema operativo utilizaremos *OpenWSN* [19] ya que implementa la pila de protocolos estándar para el Internet de las cosas y porque es el único que dispone de un simulador (OpenSim [22]), que es vital para la consecución de este proyecto al no disponer de hardware dedicado a implementar la red inalámbrica de sensores domésticos (como ejemplo sería posible utilizando la plataforma hardware *OpenMote* [20]). Específicamente utilizaremos la librería CoAP que implementa *OpenWSN* [24] para la comunicación de las aplicaciones encargadas de recoger los datos y procesarlos.

Finalmente utilizaremos la plataforma Cloud Computing llamada theThings.iO [29] para almacenar los datos característicos de la solución Edge y también almacenar la información generada por una aplicación más típica para poder comparar los resultados de ambas.

4.1 Raspberry pi (Rpi)

Se ha utilizado en este proyecto un ordenador de pequeño tamaño y de coste muy contenido (entre 7 € y 40 €) desarrollado en el Reino Unido por la fundación Raspberry Pi [25] con el objetivo principal de acercar la enseñanza de las ciencias de la computación a los más jóvenes dentro de las escuelas. A diferencia de otras plataformas hardware como la PandaBoard [28], basada en un procesador OMAP4430 Cortex-A9 que tiene un coste alrededor de los 200 €, la RpI es una solución que se ajusta a la aplicación a desarrollar en términos de funcionalidad y coste. La Raspberry Pi es un producto de propiedad registrada pero que es de uso libre en lo que a hardware se refiere. Su sistema operativo en un desarrollo *opensource* basándose en una versión adaptada de la distribución Linux Debian que se la denomina *Raspbian,* aunque permite la instalación de otros SO como por ejemplo Windows10.

Su hardware estaba formado originalmente en el 2006 en un micro de Atmel (ATmega644) y no fue hasta mediados del 2011 cuando se lanzó el primer prototipo utilizando un BCM2835 que es un SoC (*System on a chip*) compuesto por microprocesador (CPU) ARM1176JZF-S a 700MHz. Debido al éxito y demanda del público se lanza comercialmente dicho ordenador al mercado, puesto a la venta por los canales de Farnell y RS a principios del 2012 siendo un éxito.

En la actualidad se han lanzado diferentes versiones de esta placa de tamaño de una tarjeta de crédito, la cual ha ido evolucionando para añadir nuevas características para adaptarse a la demanda del mercado.

Como se ha explicado antes la Rpi utiliza como procesador principal un procesador de Broadcom tipo ARM que varía dependiendo de la versión con un Broadcom VideoCore IV,,61 (soportando OpenGL ES 2.0, MPEG-2 y VC-1, 1080p30 H.264/MPEG-4 AVC) con un lector de tarjetas SD (o microSD) a

modo de disco donde se instala el SO con una interesantísima gama de interfaces entrada/salida (GPIO) y puertos de comunicación como UART, SPI e I2C. Dispone de salida de video por HDMI y video compuesto (CVBS), una salida de audio (jack 3,5mm) y conectores especiales donde conectar una cámara de video y una pantalla táctil, todo ello alimentado a 5V con un consumo que dependerá de la versión que es como máximo 2,5 A.

Raspberry Pi:	Rpi 1 Modelo A+	Rpi 1 Modelo B	Rpi Modelo B+	Rpi 2 Modelo B	Rpi 3 Modelo B
Precio:	29€ (no disponible)	26 (no disponible)	30€ (no disponible)	46 €	39 €
SoC:	Broadcom BCM2835			Broadcom BCM2836	Broadcom BCM2837
Procesador:	ARMv6 single core			ARMv7 quad core	ARMv8 quad core
Frecuencia reloj:	700 MHz			900 MHz	1.2 GHz
Consumo	700mA @ 5V			800mA @ 5V	
GPU:	Dual Core VideoCore IV Multimedia Co-Processor				
Tamaño:	65x56mm	85x56 mm			
Memoria:	256 MB SDRAM @ 400 MHz	512 MB SDRAM @ 400 MHz		1 GB SDRAM @ 400 MHz	
Almacenamiento	Micro SD Card	SD Card	Micro SD Card	Micro SD Card	Micro SD Card
GPIO:	40	26	40		
USB 2.0:	1	2	4		
Ethernet:	-	10/100MB Ethernet RJ45			
Wifi					802.11n
Bluetooth					BT 4.1
Audio:	Audio Multi-Canal sobre HDMI, salida estéreo (3.5mm) Jack				

Tabla 5: Raspberry pi, tabla comparativa entre diferentes versiones

Respecto a los sistemas operativos disponibles de distribución libre, a fecha de la redacción de la presente memoria, se dispone de los siguientes [25]:

- *Raspbian Jessie* (con Pixel), con software pre-instalado para el sector educativo y su uso general. It has Python, Scratch, Sonic Pi, Java, Mathematica and more.
- UBUNTU Mate
- Windows 10 ioT Core
- OSMC (MediaCenter opensource)
- LibreELEC (MediaCenter para Kodi)
- PINet (Sistema para el uso en las aulas)

4.1.1 Raspberry pi 3 modelo B

De la familia Raspberry pi esta [26] es la más potente de todas, tal y como se ha explicado anteriormente y es la que se ha escogido para el desarrollo del trabajo final de máster, ya que dispone de innumerables recursos software.

Raspberry pi 3 modelo B

La distribución del SO escogida ha sido la *Jessie* (con Pixel) que viene con diferentes herramientas de desarrollo de software necesarias para el desarrollo de este proyecto.

Ilustración 33: Sistema operativo Jessie

4.2 Plataforma ioT thethings.iO

Como se ha dicho thethings.iO [29] es una plataforma de Cloud computing para los dispositivos ioT la cual tiene las características siguientes:

Ilustración 34: Logo de theThings.iO

- Datos guardados con protección accesibles fácilmente con poca latencia.
- Soporte a los protocolos REST API, MQTT, CoAP y WebSockets.
- Interoperabilidad de los objetos.
- Proporciona herramientas analíticas potentes que se pueden personalizar.
- Alta disponibilidad, flexible y escalable.
- Acceso gratuito a desarrolladores durante un periodo de dos semanas.

4.2.1 Funcionamiento de thethings.iO

Para comenzar a trabajar con esta plataforma debemos activar una cuenta de usuario para empezar a disfrutar de dos semanas de prueba. Una vez acabado este periodo se nos ofrecen varias opciones para seguir utilizando el servicio que son los planes *Prototyping, Advanced* y *Corporate* que tienen un precio de 29 €, 169 € y 499 € respectivamente.

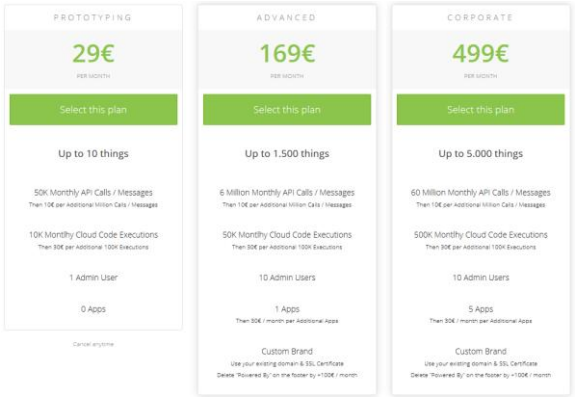


Ilustración 35: Modalidades de uso y características de la plataforma

Una vez activada nuestra cuenta procederemos a registrar nuestra dispositivo activándolo [30] mediante un código de activación que está compuesto por un Token *(thingToken*) y que para la versión de prueba solo se dispone de uno.

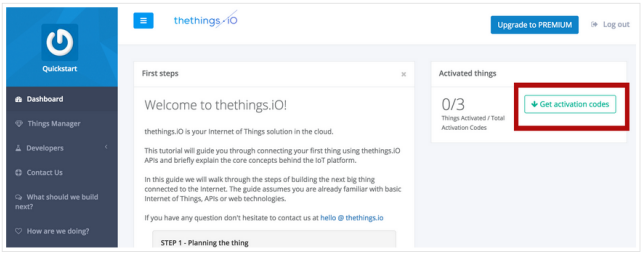


Ilustración 36: Página principal de nuestra cuenta

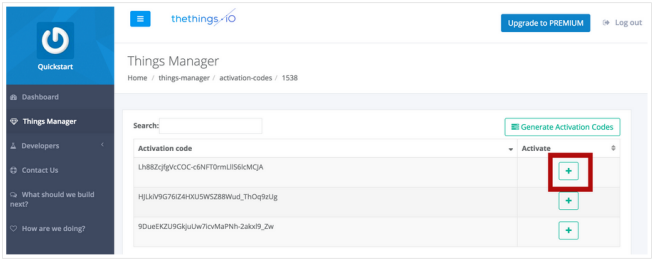


Ilustración 37: Proceso de registro y generación del Token

Una vez activado se nos muestra una ventana de confirmación.

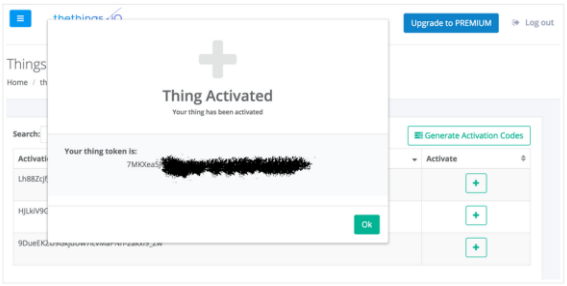


Ilustración 38: Dispositivo registrado

Si ahora nos dirigimos a la opción *things* podemos ver el detalle de los dispositivos registrados y su histórico de peticiones.

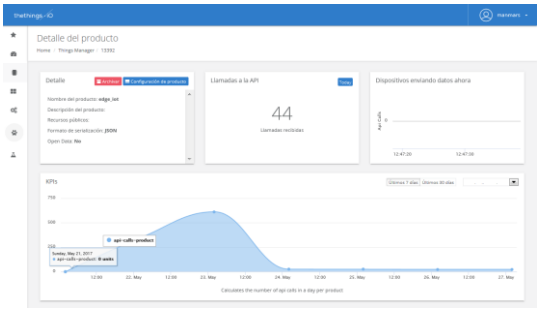


Ilustración 39: Resumen de llamadas de nuestros dispositivos al servidor

4.2.2 Python API para thethings.iO

Para el desarrollo de este trabajo hemos empleado la librería de Python que implementa las llamadas y métodos hacia la API de thethings.iO [31] necesarios en este proyecto, en vez de utilizar otras librerías disponibles como son la de llamadas mediante CoAP utilizando Node.js.

Aquí abajo se indica un ejemplo de escritura de un valor utilizando dicha librería. Como se observa se genera un número aleatorio que es el enviado hacia la API con el dato *"hello*" y un valor aleatorio que se genera con la función *randrange* de Python. Dicho valor es almacenado en la nube dentro del histórico de nuestro dispositivo activado previamente. Esta asignación se realiza mediante la línea de código número cuatro, donde se indica nuestro *thingToken*.

```
1  from thethings import ThethingsAPI
2  from random import randrange
3
4  thethings = ThethingsAPI("yourThingToken")
5  thethings.addVar("hello", randrange(0, 9))
6  thethings.write()
7
```

Ilustración 40: Librería theThings.iO para Python, ejemplo

La librería permite otras funciones enumeradas en los siguientes métodos en Python [31]:

- activate(actCode):
  Activa el dispositivo si no estaba activado previamente.

- activateSync(resource, act/false, auxResource=None):
  Activa/desactiva la función Subscribe para mantener abierta y sincronizada una conexión con un dispositivo.

- clear():
  Borra el buffer de escritura.

- getToken():

Se obtiene el Token utilizado actualmente.

- internalLogs(act/false):
  Activa la grabación de registros util para desarrolladores.

- read(key, limit=1, startDate=None, endDate=None, to=10):
  Realiza la petición a la API de la pareja {key : value} (o valores, *key*) en un intervalo de tiempo, un límite de muestras (limit) o ambas opciones definiéndose un *timeout* (to) para resolver la petición.

- subscribe():
  Función para generar una cola de lectura de un dispositivo.

- write():
  Realiza la petición de escritura de la pareja {key : value}. Va ligada con la función addVar para que realmente se haga la escritura en la nube.

- addVar(key, value, dt=None):
  Añade la pareja de datos {key : value} a la lista de datos pendiente de escribir en thethings.iO.

4.2.3 Otras aplicaciones utilizadas

A parte de estas se han empleado otras para poder llevar a cabo el presente trabajo:

- GanttProject para la generación de la planificación.
- Wireshark 2.2.1 para Windows x64.
- Wireshark 2.0.3 para Linux (*Jessie*)
- Editor Geany para Linux (*Jessie*).
- Compilador gcc.
- Editor IDLE para Python en Linux (Jessie).
- Firefox con plugin Copper (Cu) 1.0.1 con soporte CoAP URI [32].

5. Herramientas necesarias: instalación y configuración

Aquí se detallarán las herramientas y programas que se necesitaron para implementar el Edge computing.

5.1 Instalación de *OpenWSN* en Rpi3

Básicamente se trata de bajar los últimos repositorios del Github que están disponibles en este link, ejecutando los siguientes comandos:

https://*OpenWSN*.atlassian.net/wiki/display/OW/Kickstart+Linux,

git clone https://github.com/*OpenWSN*-berkeley/*OpenWSN*-fw.git

git clone https://github.com/*OpenWSN*-berkeley/*OpenWSN*-sw.git
git clone https://github.com/*OpenWSN*-berkeley/coap.git

En cualquier momento podemos verificar si tenemos la última versión subida y aprobada en github con los siguientes comandos:

```
~$ cd Desktop/
~/Desktop$ cd openwsn/
~/Desktop/openwsn$ cd openwsn-sw/
~/Desktop/openwsn/openwsn-sw$ git pull
Already up-to-date.
~/Desktop/openwsn/openwsn-sw$ cd ..
~/Desktop/openwsn$ cd openwsn-fw/
~/Desktop/openwsn/openwsn-fw$ git pull
Already up-to-date.
```

Ilustración 41: Github, comandos para actualizar la versión de OpenWSN

5.2.- Interactuando con los nodos en CoAP con Firefox para Rpi

CoAP, como ya se ha explicado, es un protocolo de comunicación que esta soportado en cada nodo (*motes*) en su firmware el cual implementa un servidor web que es accesible desde la red local. Para poder acceder a este servidor web se debe instalar el navegador *Firefox* y el *plugin Copper* (Cu) [32] para soportar esta función. Una vez instalado tecleamos en el navegador directamente (con una simulación previamente arrancada en *OpenWSN*):

coap://[bbbb::1415:92cc:0:2]:5683/.well-known/core

y podremos ver el servidor web que implementa el firmware del mote simulado dentro de *OpenWSN*.

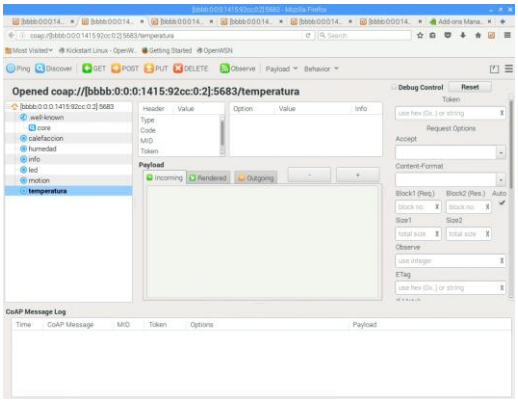


Ilustración 42: Plugin Copper para Firefox [32], visualización del servidor web

5.3 Instalación de Wireshark en Rpi

Para instalar esta herramienta de *sniffing* de red tendremos que bajar el paquete de instalación desde la siguiente dirección:

https://*OpenWSN*.atlassian.net/wiki/download/attachments/29196302/wireshark.sh?version=2&modificationDate=1465479039042&cacheVersion=1&api=v2

y utilizamos el script de instalación:

```
~/Desktop$ chmod +x ./wireshark.sh
~/Desktop$ ./wireshark.sh
```

Para ejecutar *Wireshark* utilizamos el siguiente comando (con privilegios de superuser) sobre la interfaz virtual "*tun0*" que crea *openSim* añadiendo un filtro para encontrar paquetes del protocolo *zep* (*ZigBee*), activando previamente la opción *WireShark Debug* dentro del *openVisualizer*:

```
./sudo  wireshark
```

Una vez instalado ya podremos utilizar la herramienta para ver el tráfico en la interficie *tun0* o en la principal *eth0 (eth en Windows)*. Esta herramienta fue útil para saber el tamaño de los paquetes de datos útiles mandados a la plataforma theThings.iO.

5.4 Instalación de MySQL en Rpi

Para instalar el servidor de base de datos en MySQL utilizaremos en comando *apt-get* para instalar directamente de los repositorios disponibles del sistema operativo *Jessie* para Rpi:

```
sudo apt-get install mysql-server python-mysqldb
```

Se nos pedirá un nombre de usuario y contraseña para poder crear bases de datos y tablas:

```
user = root
Pass = raspberry
```

Para entrar dentro de la aplicación y poder crear la base de datos:

```
pi@Rpi3:~ $ mysql -u root -p
Enter password:
```

Esta es la línea de comando de MySQL que aparece una vez hemos hecho el *login*:

```
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 37
Server version: 5.5.54-0+deb8u1 (Raspbian)
Copyright (c) 2000, 2016, Oracle and/or its affiliates. All rights reserved.
Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
```

Creamos la base de datos para dar soporte al *Edge computing* que se llamará *edge_db* y su propietario será el usuario *edge_user*.

```
mysql> CREATE DATABASE edge_db;
mysql> USE edge_db;
mysql> CREATE USER 'edge_user'@'locahost' IDENTIFIED BY 'edge_user';
mysql> GRANT ALL PRIVILEGES ON edge_db.* TO 'edge_user'@'locahost';
mysql> FLUSH PRIVILEGES;
```

6. Desarrollo del Edge Computing con Rpi

Se trata de realizar el procesado en Edge de una red simulada con *OpenWSN* de sensores desplegados en un entorno doméstico, para ello se necesitaron crear los sensores en el entorno de simulación, llamadas *apps,* dentro de la estructura *OpenWSN*.

Actualmente, como ya se ha explicado, existen las opciones de Cloud y Edge computing para gestionar y procesar la inmensa cantidad de datos que generan las redes de dispositivos ioT. Es precisamente el Edge computing una forma de dar solución a los problemas que conlleva el uso intensivo de las redes y de aprovechamiento del ancho de banda de una forma más eficiente. En la aplicación que se ha desarrollado cobra más importancia la respuesta desde dentro de la misma red local, para tener una latencia mucho menor debido al hecho de no utilizar ningún servidor fuera de dicha red sometida a su disponibilidad y alto coste de subscripción, dependiendo del servicio contratado. Es por ello que se ha realizado la aplicación de lectura de varias variables físicas (simuladas) como son: la temperatura, humedad, unos sensores de movimiento situados en distintas localizaciones dentro de una casa y un actuador sobre un sistema de calefacción. Dichos datos serán almacenados periódicamente en una base de datos dentro de la SD card de la Rpi 3, y se generarán también unos datos resumidos (cada 3 horas) que serán subidos al Cloud (theThings.iO), de esta manera se optimizará el uso de red y el consumo de potencia del sistema aumentando a la vez la seguridad de los datos, ya que se procesan dentro de la misma red local y sólo se envían los valores máximo, mínimo y medias de temperatura y humedad, así como los valores del estado de la calefacción y sensores de presencia.

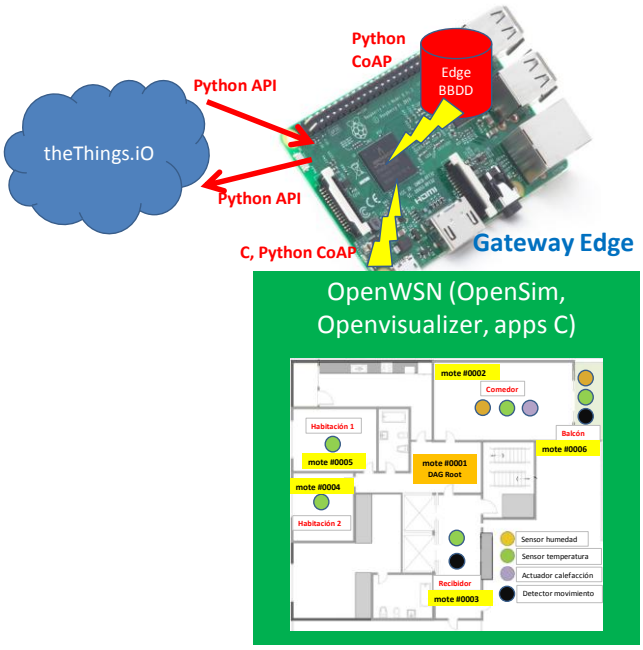


Ilustración 43: Diagrama resumen del flujo de datos, hardware y software utilizado

Finalmente se compararán ambas opciones de procesado en base a unas métricas establecidas como son: el coste de ambas soluciones, tiempo de ejecución de cada procesado, tiempo de uso de red (tiempo en el que se está transmitiendo datos hacia Internet utilizando el máximo ancho de banda disponible) y la seguridad de los datos procesados.

6.1 Estructura del firmware dentro del *OpenWSN*

Todas las funcionalidades de los sensores están ordenadas y ubicadas dentro de la carpeta *openwsn-fw* como se indica en la figura.

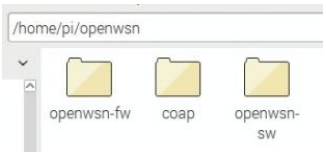


Ilustración 44: openwsn-fw, localización dentro de OpenWSN

La carpeta *openapps* es donde residen todas las apps que soportan los motes dentro de su firmware en forma de código C, ya sea en su memoria (caso del uso de un hardware real) o en forma de simulación, que es el caso que nos ocupa.

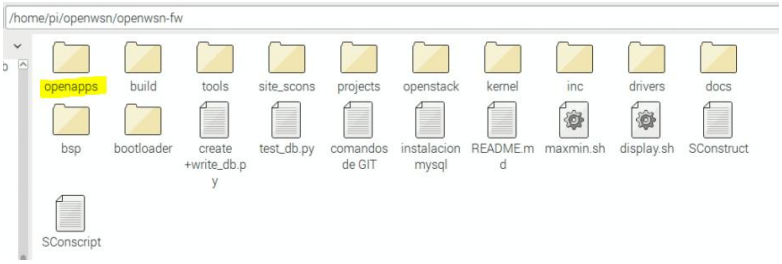


Ilustración 45: openwsn-fw: carpeta openapps, ubicación de apps soportadas

En la figura de abajo se pueden observar las apps desarrolladas remarcadas de color amarillo.

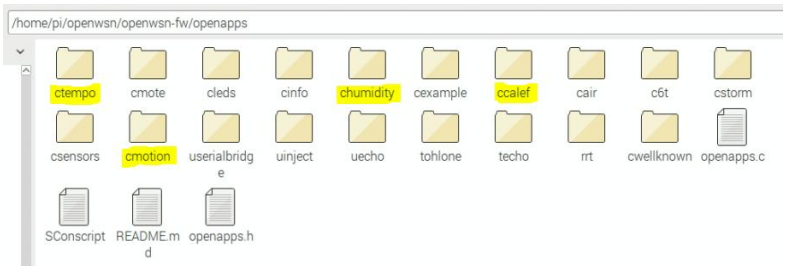


Ilustración 46: openwsn-fw: Localización de apps desarrolladas

- **Sensor de temperatura (ctempo.c, ctempo.h):** app que implementa método GET en CoAP para obtener un valor de temperatura, que al ser

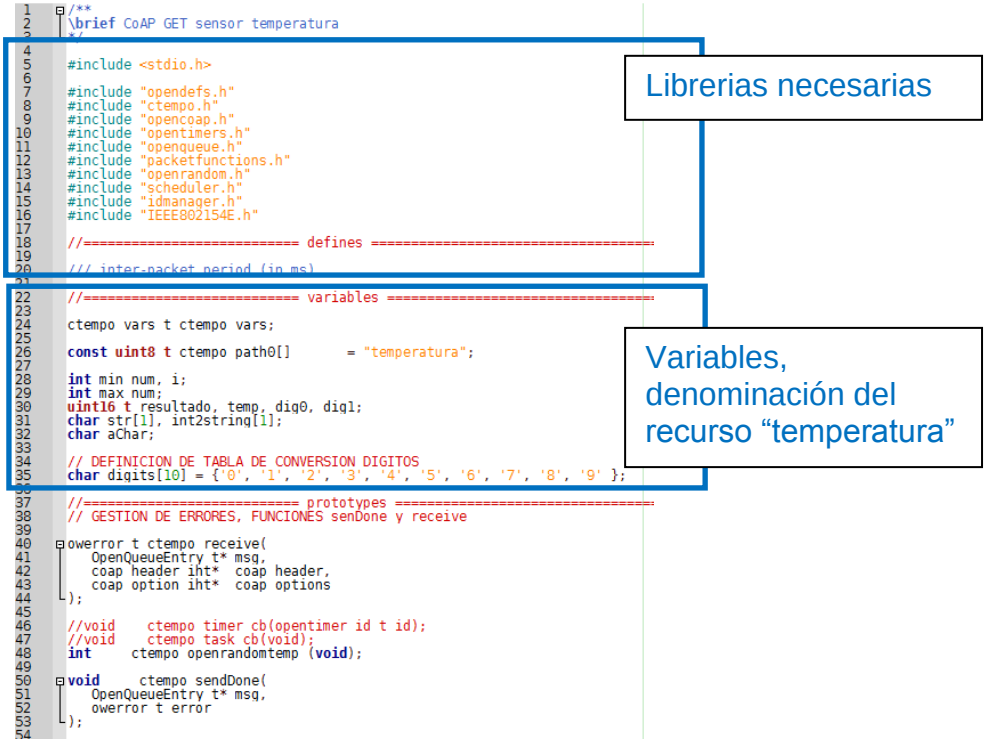
simulado, se crea mediante un generador de número aleatorios entre el margen de los 15 y 25 grados.

- **Sensor de humedad (chumidity.c, chumidity.h):** app que implementa método GET en CoAP para obtener un valor de humedad, que al ser simulado, se crea mediante un generador de número aleatorios entre el margen de los valores de 55 y 95 en tanto por cierto de humedad relativa.
- **Actuador calefacción (ccalef.c, ccalef.h):** app que implementa método GET y PUT en CoAP para obtener el valor del estado del actuador de calefacción y que permite su cambio de estado.
- **Sensor movimiento (cmotion.c, cmotion.h):** app que implementa método GET en CoAP para obtener el valor del estado del sensor de movimiento.

Todos los sensores o motes tendrán dichas funcionalidades dentro de la simulación en el entorno *OpeWSN* [19].

6.2 Sensado de temperatura, ctempo

En esta app se crean las funciones CoAP de recepción de mensajes como por ejemplo utilizando el método GET para obtener el valor de temperatura. Los sensores de temperatura estarán instalados y repartidos por todas las zonas de la casa, o sea, en el comedor, balcón, recibidor y habitaciones.




```
/**
\brief CoAP GET sensor temperatura
*/

#include <stdio.h>

#include "opendefs.h"
#include "ctempo.h"
#include "opencoap.h"
#include "opentimers.h"
#include "openqueue.h"
#include "packetfunctions.h"
#include "openrandom.h"
#include "scheduler.h"
#include "idmanager.h"
#include "IEEE802154E.h"

//=========================== defines ==========================================

/// inter-packet period (in ms)

//=========================== variables ========================================

ctempo_vars_t ctempo_vars;

const uint8_t ctempo_path0[]       = "temperatura";

int min_num, i;
int max_num;
uint16_t resultado, temp, dig0, dig1;
char str[1], int2string[1];
char aChar;

// DEFINICION DE TABLA DE CONVERSION DIGITOS
char digits[10] = {'0', '1', '2', '3', '4', '5', '6', '7', '8', '9' };

//=========================== prototypes ======================================
// GESTION DE ERRORES, FUNCIONES senDone y receive

owerror_t ctempo_receive(
    OpenQueueEntry_t* msg,
    coap_header_iht*  coap_header,
    coap_option_iht*  coap_options
);

//void    ctempo_timer_cb(opentimer_id_t id);
//void    ctempo_task_cb(void);
int     ctempo_openrandomtemp (void);

void    ctempo_sendDone(
    OpenQueueEntry_t* msg,
    owerror_t error
);
```

Ilustración 47: ctempo.c, inicialización de la app

Como parte principal de la app se encuentra la descripción de su inicialización dentro del sistema *OpenWSN* y funcionalidades dentro del protocolo CoAP. Es necesario remarcar la función de generación de números aleatorios llamada

openrandomtemp() la cual devuelve un valor en el margen definido. Como se ha comentado anteriormente al tratarse de una simulación se requirió el uso de una función que pudiera generar números aleatorios dentro de un rango concreto para tener un registro de valores diverso. Se tomó como ejemplo la fu*nción* ya implementada dentro de *OpenWSN* llamada *Openrandom.c* y se simplificó para crear dicha función incluyéndola dentro de cada app que la necesitase.

```
55  //=========================== public ==========================================
56  // INICIALIZACION DE APP
57
58  void ctempo init() {
59
60     if(idmanager getIsDAGroot()==TRUE) return;
61
62     //// prepare the resource descriptor for the /temperatura path
63     ctempo vars.desc.path0len             = sizeof(ctempo path0)-1;
64     ctempo vars.desc.path0val             = (uint8 t*)(&ctempo path0);
65     ctempo vars.desc.path1len             = 0;
66     ctempo vars.desc.path1val             = NULL;
67     ctempo vars.desc.componentID          = COMPONENT CTEMPO;
68     ctempo vars.desc.discoverable         = TRUE;
69     ctempo vars.desc.callbackRx           = &ctempo receive;
70     ctempo vars.desc.callbackSendDone     = &ctempo sendDone;
71
72     // register with the CoAP module
73     opencoap register(&ctempo vars.desc);
74  }
75
76
77  //=========================== private =========================================
78  // GENERACION DE VALORES RANDOM DE TEMPERATURA DENTRO DE UN RANGO
79  // DEFINIDO
80
81  int ctempo openrandomtemp (){
82
83     int resultado = 0, low num = 0, hi num = 0;
84     min num=15;
85     max num=25;
86
87     if (min num < max num)
88     {
89         low num = min num;
90         hi num = max num + 1; // include max num in output
91     } else {
92         low num = max num + 1; // include max num in output
93         hi num = min num;
94     }
95
96     srand(time(NULL));
97     resultado = (rand() % (hi num - low num)) + low num;
98     printf("Min=15 Max=25 TEMP=%d\n",resultado);
99     return resultado;
100
101 }
```

Función de inicialización de la app en CoAP, incluyendo dos funciones que gestionan la llegada y envio de paquetes CoAP (receive, sendDone)

Función *openrandomtemp()* encargada de generar una temperatura aleatoria

Ilustración 48: ctempo.c, inicialización de la app y función openrandomtemp()

Aquí se presenta una captura de la función que se encarga de formar los datos (payload) a partir del valor de temperatura generado aleatoriamente en la captura de código de arriba.

```
104 /**
105 \brief Called when a CoAP message is received for this resource.
106
107 \param[in] msg          The received message. CoAP header and options alre
108    parsed.
109 \param[in] coap header  The CoAP header contained in the message.
110 \param[in] coap options The CoAP options contained in the message.
111
112 \return Whether the response is prepared successfully.
113 */
114 owerror t ctempo receive(
115       OpenQueueEntry t* msg,
116       coap header iht*  coap header,
117       coap option iht*  coap options
118    ) {
119    owerror t outcome;
120
121    switch (coap header->Code) {
122       case COAP CODE REQ GET:
123          // reset packet payload
124          msg->payload                     = &(msg->packet[127]);
125          msg->length                      = 0;
126
127          // add CoAP payload
128          // se define el tamaño del mensaje util (payload)
129          packetfunctions reserveHeaderSize(msg,3);
130          msg->payload[0]                  = COAP PAYLOAD MARKER;
131
132             temp = ctempo openrandomtemp ();
133             dig0 = temp % 10;
134             temp = temp/10;
135             dig1 = temp % 10;
136             aChar = digits[dig1];
137             msg->payload[1]               = aChar;
138             aChar = digits[dig0];
139             msg->payload[2]               = aChar;
140
141          // set the CoAP header
142          coap header->Code                = COAP CODE RESP CONTENT;
143
144          outcome                          = E SUCCESS;
145          break;
146
147       default:
148          outcome                          = E FAIL;
149          break;
150    }
151
152    return outcome;
153 }
154
155 /**
156 \brief The stack indicates that the packet was sent.
157
158 \param[in] msg The CoAP message just sent.
159 \param[in] error The outcome of sending it.
160 */
161 void ctempo sendDone(OpenQueueEntry t* msg, owerror t error) {
162    openqueue freePacketBuffer(msg);
163 }
164
```

Función *encargada de devolver (GET), utilizando el protocolo CoAP, el valor de temperatura generado*

Ilustración 49: ctempo.c, rutina de respuesta del valor de temperatura

6.3 Sensado de humedad, chumidity

Se crean las funciones CoAP de recepción de mensajes utilizando el método GET para obtener el valor de humedad. Dichos sensores estarán ubicados en el comedor y balcón. Como parte principal de la app se encuentra la descripción de su inicialización dentro del sistema *OpenWSN* [19] y funcionalidades dentro del protocolo CoAP.

Importante remarcar la función de generación de números aleatorios llamada *openrandomhumidity()* la cual devuelve un valor en el margen definido. De igual manera que en el caso de la generación aleatoria de temperaturas se incluyó una que generase valores de humedad en un rango comprendido entre 55% y 95%. Se termina dicha app con la función que se encarga de formar los datos (payload) a partir del valor de humedad generado aleatoriamente en la captura de código de arriba al igual que el caso de la app *ctempo.*

6.4 Actuador calefacción, ccalef

Se crean las funciones CoAP de recepción de mensajes usando los métodos GET y PUT para obtener el valor actual del actuador de la calefacción y poder cambiar su estado. Dicho actuador estará ubicado en el comedor.

```
   parsed,
\param[in] coap_header  The CoAP header contained in the message.
\param[in] coap_options The CoAP options contained in the message.

\return Whether the response is prepared successfully.
*/
owerror_t ccalef_receive(
      OpenQueueEntry_t* msg,
      coap_header_iht*  coap_header,
      coap_option_iht*  coap_options
   ) {
   owerror_t outcome;
   switch (coap_header->Code) {
      case COAP_CODE_REQ_GET:
         // reset packet payload
         msg->payload                     = &(msg->packet[127]);
         msg->length                      = 0;
         // add CoAP payload
         packetfunctions_reserveHeaderSize(msg,2);
         msg->payload[0]                  = COAP_PAYLOAD_MARKER;

         if (leds_error_isOn()==1) {
            msg->payload[1]               = '1';
         } else {
            msg->payload[1]               = '0';
         }
         // set the CoAP header
         coap_header->Code                = COAP_CODE_RESP_CONTENT;
         outcome                          = E_SUCCESS;
         break;

      case COAP_CODE_REQ_PUT:

         // change the LED's state
         if (msg->payload[0]=='1') {
            leds_error_on();
         } else if (msg->payload[0]=='2') {
            leds_error_toggle();
         } else {
            leds_error_off();
         }
         // reset packet payload
         msg->payload                     = &(msg->packet[127]);
         msg->length                      = 0;
         // set the CoAP header
         coap_header->Code                = COAP_CODE_RESP_CHANGED;
         outcome                          = E_SUCCESS;
         break;
      default:
         outcome                          = E_FAIL;
         break;
   }

   return outcome;
}
/**
\brief The stack indicates that the packet was sent.

\param[in] msg The CoAP message just sent.
\param[in] error The outcome of sending it.
*/
void ccalef_sendDone(OpenQueueEntry_t* msg, owerror_t error) {
   openqueue_freePacketBuffer(msg);
}
```

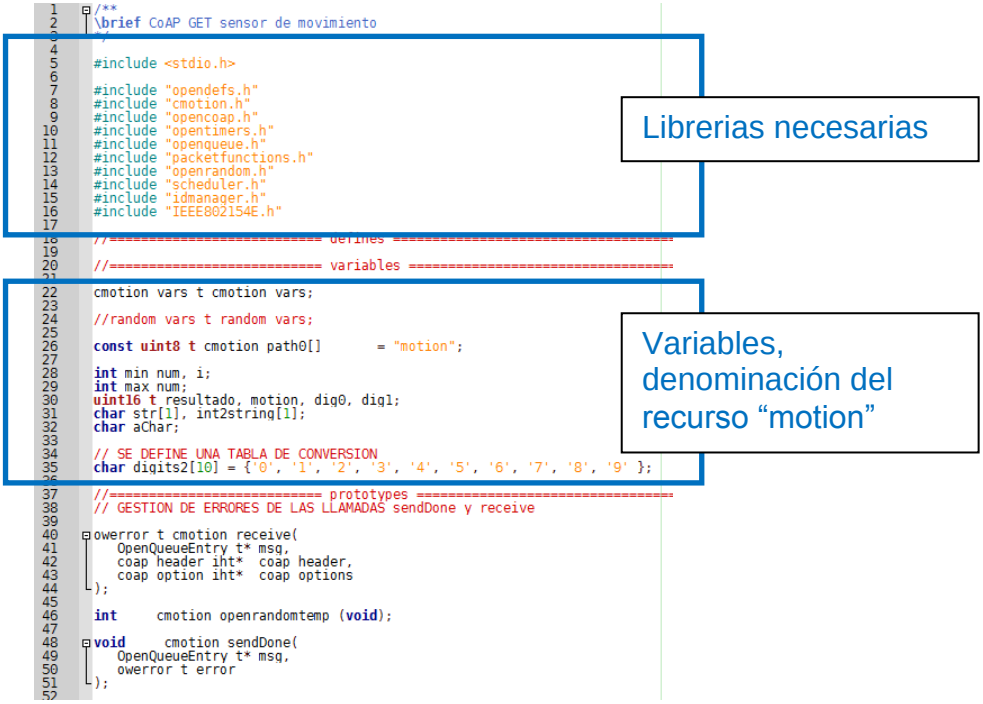
Función *encargada de devolver (GET), utilizando el protocolo CoAP, el valor del actuador calefacción y cambiar su estado (PUT)*

Ilustración 50: ccalef.c, métodos GET y PUT para modificar estado de la calefacción

De igual manera, como en las apps anteriores, se inicializa indicando que funcionalidades CoAP tendrá dentro del sistema *OpenWSN*. A diferencia de las anteriores esta app puede cambiar el estado del actuador calefacción utilizando el método PUT con un valor booleano (el valor booleano “2” tiene la función *toggle* cambiando el estado actual al opuesto).

6.5 Detector de movimiento, cmotion

Se crean las funciones CoAP de recepción de mensajes usando los métodos GET para obtener el valor actual del sensor de presencia que estarán instalados en el balcón y recibidor.




```
/**
\brief CoAP GET sensor de movimiento
*/

#include <stdio.h>

#include "opendefs.h"
#include "cmotion.h"
#include "opencoap.h"
#include "opentimers.h"
#include "openqueue.h"
#include "packetfunctions.h"
#include "openrandom.h"
#include "scheduler.h"
#include "idmanager.h"
#include "IEEE802154E.h"

//=========================== defines =========================================

//=========================== variables =======================================

cmotion_vars_t cmotion_vars;

//random_vars_t random_vars;

const uint8_t cmotion_path0[]       = "motion";

int min_num, i;
int max_num;
uint16_t resultado, motion, dig0, dig1;
char str[1], int2string[1];
char aChar;

// SE DEFINE UNA TABLA DE CONVERSION
char digits2[10] = {'0', '1', '2', '3', '4', '5', '6', '7', '8', '9' };

//=========================== prototypes ======================================
// GESTION DE ERRORES DE LAS LLAMADAS sendDone y receive

owerror_t cmotion_receive(
   OpenQueueEntry_t* msg,
   coap_header_iht*  coap_header,
   coap_option_iht*  coap_options
);

int     cmotion_openrandomtemp (void);

void     cmotion_sendDone(
   OpenQueueEntry_t* msg,
   owerror_t error
);
```

Ilustración 51: cmotion.c, librerías e inicialización de variables

Como las demás apps en el código se inicializa dentro del sistema *OpenWSN* y se definen sus funcionalidades dentro del protocolo CoAP.

También es importante remarcar la función de generación de números aleatorios llamada *openrandommotion()* la cual devuelve un valor en el margen definido (“0” no hay movimiento y “1” movimiento detectado) que está basada de igual manera en la misma idea de generación de valores que en las apps anteriores.

Finalmente la función recibe la petición GET se encargará de formar los datos (payload) a partir del valor de movimiento generado aleatoriamente y se enviará su resultado.

6.6 Modificación del *OpenWSN* añadiendo nuevas funcionalidades

Para que el sistema operativo pueda añadir dichas apps se ha de compilar el *OpenWSN* con su nuevo estado. La manera como se notifica al compilador todos los cambios es mediante la modificación de ciertos ficheros dentro del *toolchain* del sistema *OpenWSN*. Estos ficheros son los que se detallan a continuación:

- openapps.c

- SConscript

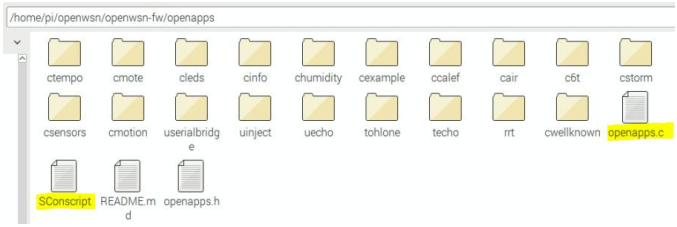


Ilustración 52: openapps.c y Sconscript, localización dentro del firmware de OpenWSN

Se ha de indicar al compilador que appps se añaden, con sus ficheros .h (headers) y haciendo referencia a los procedimientos de inicialización de cada nueva app (*openapps_init(void)*).

```
/**
\brief Applications running on top of the OpenWSN stack.

\author Thomas Watteyne <watteyne@eecs.berkeley.edu>, September 2014.
*/

#include "opendefs.h"

// CoAP
#include "c6t.h"
#include "cinfo.h"
#include "cleds.h"
#include "cexample.h"
#include "cstorm.h"
#include "cwellknown.h"
#include "rrt.h"
#include "ctempo.h"
#include "chumidity.h"
#include "cmotion.h"
#include "ccalef.h"

// TCP
#include "techo.h"
// UDP
#include "uecho.h"
#include "uinject.h"

//=========================== variables =======================================

//=========================== prototypes ======================================

//=========================== public ==========================================

//=========================== private =========================================

void openapps_init(void) {
   // CoAP
   // c6t_init();
   cinfo_init();
   // cexample_init();
   //cleds_init();
   //cstorm_init();
   cwellknown_init();
   //rrt_init();
   ctempo_init();
   chumidity_init();
   cmotion_init();
   ccalef_init();

   // TCP
   techo_init();
   // UDP
   //uecho_init();
}
```

Ilustración 53: openapps.c, cabeceras y funciones de inicialización de cada app

Dentro del fichero *Sconscrip*t se tienen que referenciar todas las apps que se quieren inicializar en la simulación una vez se arranque el *Openvisualizer*.

```
if localEnv['board']=='python':
    defaultApps  += [
        'cexample',   # NO QUITAR
        'cstorm',
        # 'MIS APPS',
        'ctempo',
        'chumidity',
        'cmotion',
        'ccalef',
    ]

# apps which should always be present

# MIS APPS
defaultAppsInit   = [
    'c6t',
    'cinfo',
    'cleds',
    'cwellknown',
    'techo',
    'uecho',
    'rrt',
    #'cexample',
    #'MIS APPS',
    'ctempo',
    'chumidity',
    'cmotion',
    'ccalef',
]
```

Ilustración 54: SConscript, se añaden las apps que por defecto soportará cada mote

- opendefs.h:

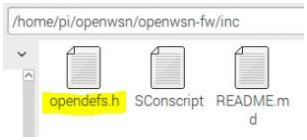


Ilustración 55: opendefs.h, localización dentro del firmware de OpenWSN

Aquí se registra cada app con un número de identificación único necesario dentro del stack *OpenWSN* para saber quien genera los datos.

```
155   // applications
156   COMPONENT_C6T                = 0x1a,
157   COMPONENT_CEXAMPLE           = 0x1b,
158   COMPONENT_CINFO              = 0x1c,
159   COMPONENT_CLEDS              = 0x1d,
160   COMPONENT_CSENSORS           = 0x1e,
161   COMPONENT_CSTORM             = 0x1f,
162   COMPONENT_CWELLKNOWN         = 0x20,
163   COMPONENT_TECHO              = 0x21,
164   COMPONENT_TOHLONE            = 0x22,
165   COMPONENT_UECHO              = 0x23,
166   COMPONENT_UINJECT            = 0x24,
167   COMPONENT_RRT                = 0x25,
168   COMPONENT_SECURITY           = 0x26,
169   COMPONENT_USERIALBRIDGE      = 0x27,
170   COMPONENT_CTEMPO             = 0x28,
171   COMPONENT_CHUMIDITY          = 0x29,
172   COMPONENT_CMOTION            = 0x30,
173   COMPONENT_CCALEF             = 0x31,
```

Ilustración 56: opendefs.h, registro de las apps creadas

- Sconscript.env:

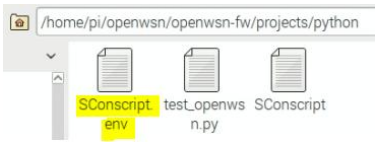
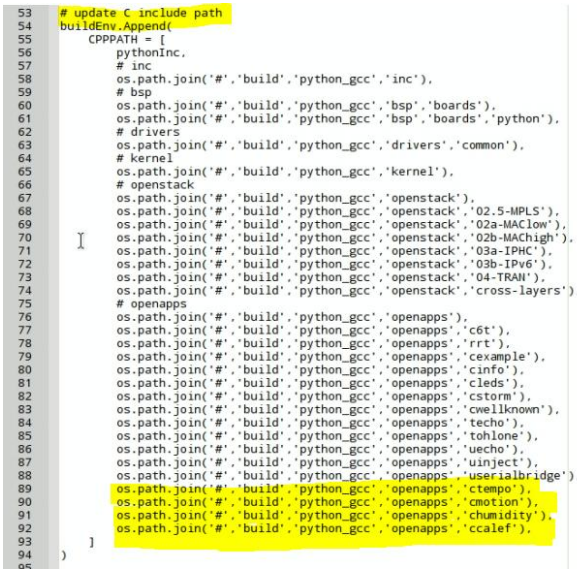


Ilustración 57: SConscript.env, localización dentro del firmware de OpenWSN

Aquí se indica la ruta hacia los ficheros fuentes de cada app para que el compilador *scons* pueda encontrarlos.



```
53  # update C include path
54  buildEnv.Append(
55      CPPPATH = [
56          pythonInc,
57          # inc
58          os.path.join('#','build','python_gcc','inc'),
59          # bsp
60          os.path.join('#','build','python_gcc','bsp','boards'),
61          os.path.join('#','build','python_gcc','bsp','boards','python'),
62          # drivers
63          os.path.join('#','build','python_gcc','drivers','common'),
64          # kernel
65          os.path.join('#','build','python_gcc','kernel'),
66          # openstack
67          os.path.join('#','build','python_gcc','openstack'),
68          os.path.join('#','build','python_gcc','openstack','02.5-MPLS'),
69          os.path.join('#','build','python_gcc','openstack','02a-MAClow'),
70          os.path.join('#','build','python_gcc','openstack','02b-MAChigh'),
71          os.path.join('#','build','python_gcc','openstack','03a-IPHC'),
72          os.path.join('#','build','python_gcc','openstack','03b-IPv6'),
73          os.path.join('#','build','python_gcc','openstack','04-TRAN'),
74          os.path.join('#','build','python_gcc','openstack','cross-layers'),
75          # openapps
76          os.path.join('#','build','python_gcc','openapps'),
77          os.path.join('#','build','python_gcc','openapps','c6t'),
78          os.path.join('#','build','python_gcc','openapps','rrt'),
79          os.path.join('#','build','python_gcc','openapps','cexample'),
80          os.path.join('#','build','python_gcc','openapps','cinfo'),
81          os.path.join('#','build','python_gcc','openapps','cleds'),
82          os.path.join('#','build','python_gcc','openapps','cstorm'),
83          os.path.join('#','build','python_gcc','openapps','cwellknown'),
84          os.path.join('#','build','python_gcc','openapps','techo'),
85          os.path.join('#','build','python_gcc','openapps','tohlone'),
86          os.path.join('#','build','python_gcc','openapps','uecho'),
87          os.path.join('#','build','python_gcc','openapps','uinject'),
88          os.path.join('#','build','python_gcc','openapps','userialbridge'),
89          os.path.join('#','build','python_gcc','openapps','ctempo'),
90          os.path.join('#','build','python_gcc','openapps','cmotion'),
91          os.path.join('#','build','python_gcc','openapps','chumidity'),
92          os.path.join('#','build','python_gcc','openapps','ccalef'),
93      ]
94  )
```

Ilustración 58: SConscript.env, ruta de los ficheros fuentes de cada app creada

Dentro de *OpenWSN* y de su firmware se han de indicar y referenciar las funciones que tienen cada app y los métodos internos que tiene cada una de ellas, incluido sus ficheros respectivos de cabecera (.h).

```
784     #===== openapps
785     'openapps_init',
786
787     # ctempo
788     'ctempo_init',
789     'ctempo_receive',
790     'ctempo_timer_cb',
791     'ctempo_task_cb',
792     'ctempo_openrandomtemp',
793     'ctempo_sendDone',
794     # cmotion
795     'cmotion_init',
796     'cmotion_receive',
797     'cmotion_timer_cb',
798     'cmotion_task_cb',
799     'cmotion_openrandomtemp',
800     'cmotion_sendDone',
801     # chumidity
802     'chumidity_init',
803     'chumidity_receive',
804     'chumidity_timer_cb',
805     'chumidity_task_cb',
806     'chumidity_openrandomtemp',
807     'chumidity_sendDone',
808     # ccalef
809     'ccalef_init',
810     'ccalef_receive',
811     'ccalef_timer_cb',
812     'ccalef_task_cb',
813     'ccalef_sendDone',
814
```

Ilustración 59: SConscript.env, referencia de cada app y sus funciones

- SConscript (openstack):

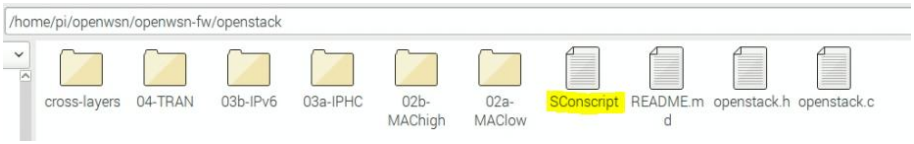


Ilustración 60: SConscript, localización dentro del firmware de OpenWSN

Este fichero es el que contiene las rutas hacia las carpetas de cada apps y que por tanto soporta el firmware de cada mote. Aquí se han de añadir por tanto las rutas de las nuevas apps.

```
else:

    localEnv.Append(
        CPPPATH =  [
            # inc
            os.path.join('#','inc'),
            # kernel
            os.path.join('#','kernel'),
            # drivers
            os.path.join('#','drivers','common'),
            # openstack
            os.path.join('#','openstack'),
            os.path.join('#','openstack','02a-MAClow'),
            os.path.join('#','openstack','02b-MAChigh'),
            os.path.join('#','openstack','03a-IPHC'),
            os.path.join('#','openstack','03b-IPv6'),
            os.path.join('#','openstack','04-TRAN'),
            os.path.join('#','openstack','cross-layers'),
            # openapps
            # TODO: remove once cleaned-up?
            os.path.join('#','openapps','c6t'),
            os.path.join('#','openapps','cexample'),
            os.path.join('#','openapps','cinfo'),
            os.path.join('#','openapps','cleds'),
            os.path.join('#','openapps','cstorm'),
            os.path.join('#','openapps','cwellknown'),
            os.path.join('#','openapps','rrt'),
            os.path.join('#','openapps','techo'),
            os.path.join('#','openapps','tohlone'),
            os.path.join('#','openapps','uecho'),
            os.path.join('#','openapps','uinject'),
            os.path.join('#','openapps','userialbridge'),
            os.path.join('#','openapps','ctempo'),
            os.path.join('#','openapps','cmotion'),
            os.path.join('#','openapps','chumidity'),
            os.path.join('#','openapps','ccalef'),
        ],
    )
```

Ilustración 61: Sconscript, se añaden las rutas hacia las apps creadas

6.7 Creación de estructura de bbdd: MySQL

Para soportar el almacenamiento que necesita el procesado Edge se tuvo que crear una base de datos que contuviera todas las variables que se van leyendo de los sensores y también los datos resumen de los mismos. Se utilizará MySQL para crear esta base de datos, ya que está soportada por el sistema operativo *Jessie* y dispone de muchas funciones y soporte. Por tanto esta base de datos dispondrá de las tablas mostradas en la figura de abajo.

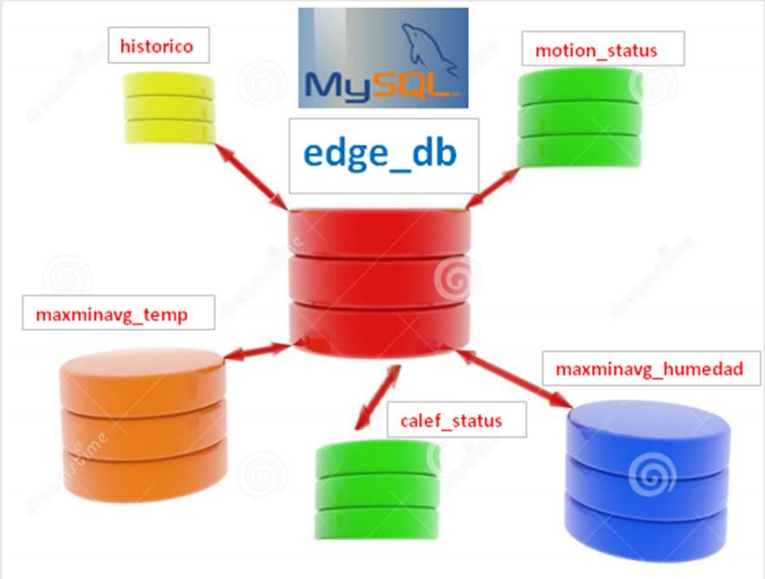


Ilustración 62: Descripción visual de las tablas que soporta edge_db

6.7.1 Creación de las tablas en MySQL

Por tanto, para almacenar los datos que generan los sensores se crean las siguientes tablas:

- **historico**: almacenaje de valores de temperatura, humedad, estado de actuador de calefacción y estado de los sensores de movimiento.
- **maxminavg_temp**: tabla para almacenar los valores máximo, mínimo y media de los valores de temperatura.
- **maxminavg_humedad**: tabla para almacenar los valores máximo, mínimo y media de humedad.
- **calef_status**: tabla para almacenar el estado de la calefacción.
- **motion_status**: tabla para almacenar el estado de los sensores de movimiento.

Utilizamos las siguientes sentencias en lenguaje SQL para crearlas y se añade una captura de su resultado:

```
INSERT INTO edge_db(historico, maxminavg_temp, maxminavg_humedad, calef, motion)
VALUES();
```

```
Database changed
mysql> SHOW tables;
+-------------------+
| Tables_in_edge_db |
+-------------------+
| calef_status      |
| historico         |
| maxminavg_humedad |
| maxminavg_temp    |
| motion_status     |
+-------------------+
5 rows in set (0.00 sec)
```

Tabla 6: Tablas disponibles dentro edge_db

CREATE TABLE historico(sensor VARCHAR(4), dia DATE, hora TIME, valor_temp VARCHAR(4), valor_humedad VARCHAR(4), calefaccion VARCHAR (1), motion VARCHAR(1);

```
mysql> DESCRIBE historico;
+---------------+------------+------+-----+---------+-------+
| Field         | Type       | Null | Key | Default | Extra |
+---------------+------------+------+-----+---------+-------+
| sensor        | varchar(4) | YES  |     | NULL    |       |
| dia           | date       | YES  |     | NULL    |       |
| hora          | time       | YES  |     | NULL    |       |
| valor_temp    | varchar(4) | YES  |     | NULL    |       |
| valor_humedad | varchar(4) | YES  |     | NULL    |       |
| calefaccion   | varchar(1) | YES  |     | NULL    |       |
| motion        | varchar(1) | YES  |     | NULL    |       |
+---------------+------------+------+-----+---------+-------+
7 rows in set (0.00 sec)
```

Tabla 7: Tabla historico

CREATE TABLE maxminavg_temp(sensor VARCHAR(4), dia DATE, hora TIME, temp_max VARCHAR(4), temp_min VARCHAR(4), temp_avg VARCHAR (4));

```
mysql> DESCRIBE maxminavg_temp;
+----------+------------+------+-----+---------+-------+
| Field    | Type       | Null | Key | Default | Extra |
+----------+------------+------+-----+---------+-------+
| sensor   | varchar(4) | YES  |     | NULL    |       |
| dia      | date       | YES  |     | NULL    |       |
| hora     | time       | YES  |     | NULL    |       |
| temp_max | varchar(4) | YES  |     | NULL    |       |
| temp_min | varchar(4) | YES  |     | NULL    |       |
| temp_avg | float      | YES  |     | NULL    |       |
+----------+------------+------+-----+---------+-------+
6 rows in set (0.00 sec)
```

Tabla 8: Tabla maxminavg_temp

CREATE TABLE maxminavg_humedad(sensor VARCHAR(4), dia DATE, hora TIME, humedad_max VARCHAR(4), humedad_min VARCHAR(4), humedad_avg VARCHAR (4));

```
mysql> DESCRIBE maxminavg_humedad;
+-------------+------------+------+-----+---------+-------+
| Field       | Type       | Null | Key | Default | Extra |
+-------------+------------+------+-----+---------+-------+
| sensor      | varchar(4) | YES  |     | NULL    |       |
| dia         | date       | YES  |     | NULL    |       |
| hora        | time       | YES  |     | NULL    |       |
| humedad_max | varchar(4) | YES  |     | NULL    |       |
| humedad_min | varchar(4) | YES  |     | NULL    |       |
| humedad_avg | float      | YES  |     | NULL    |       |
+-------------+------------+------+-----+---------+-------+
6 rows in set (0.00 sec)
```

Tabla 9: Tabla maxminavg_humedad

CREATE TABLE calef_status(sensor VARCHAR(4), dia DATE, hora TIME, calef_status VARCHAR(1));

```
mysql> DESCRIBE calef_status;
+--------------+------------+------+-----+---------+-------+
| Field        | Type       | Null | Key | Default | Extra |
+--------------+------------+------+-----+---------+-------+
| sensor       | varchar(4) | YES  |     | NULL    |       |
| dia          | date       | YES  |     | NULL    |       |
| hora         | time       | YES  |     | NULL    |       |
| calef_status | varchar(1) | YES  |     | NULL    |       |
+--------------+------------+------+-----+---------+-------+
4 rows in set (0.01 sec)
```

Tabla 10: Tabla calef_status

CREATE TABLE motion_status(sensor VARCHAR(4), dia DATE, hora TIME, motion_status VARCHAR(1));

```
mysql> DESCRIBE motion_status;
+--------------+------------+------+-----+---------+-------+
| Field        | Type       | Null | Key | Default | Extra |
+--------------+------------+------+-----+---------+-------+
| sensor       | varchar(4) | YES  |     | NULL    |       |
| dia          | date       | YES  |     | NULL    |       |
| hora         | time       | YES  |     | NULL    |       |
| motion_status | varchar(1) | YES  |     | NULL    |       |
| motion_alarm | varchar(1) | YES  |     | NULL    |       |
+--------------+------------+------+-----+---------+-------+
5 rows in set (0.01 sec)
```

Tabla 11: Tabla motion_status

6.8 Aplicaciones Edge en Python

Una vez creadas las funciones escritas en lenguaje C que contendrá el firmware de cada mote se describirán aquí las funciones a más alto nivel, escritas el Python, las cuales realizan las peticiones CoAP utilizando los métodos GET y PUT antes explicados para consultar el valor de las variables que cada mote soporta. Por tanto aquí se describirán los scripts escritos en Python que formarán la red de sensores desplegados en el entorno doméstico.

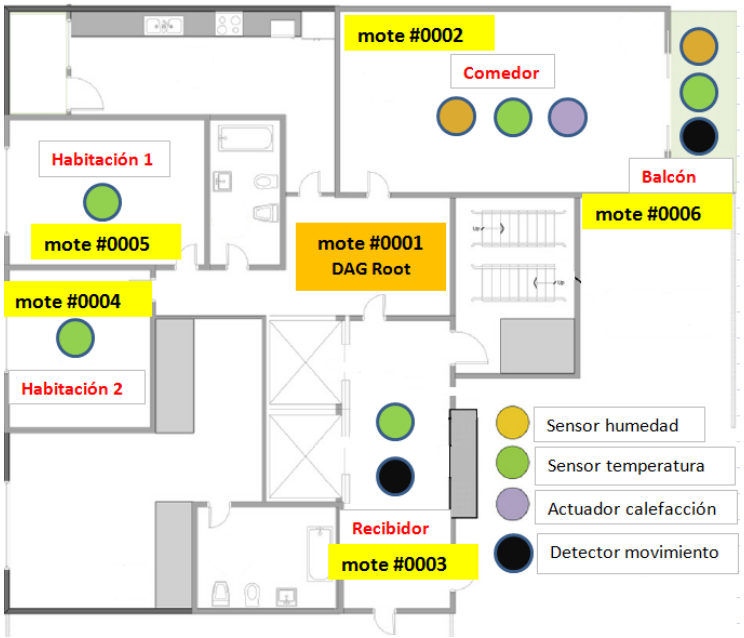


Ilustración 63: Distribución de los motes con sus funciones

Como resumen, una vez creadas las apps descritas en el capítulo 6 las cuales darán soporte a la simulación se deben crear las aplicaciones (desarrolladas en

Python) que se encargaran de consultar y actuar sobre las variables del sistema (temperatura, humedad, movimiento y estado de la calefacción).

En la tabla siguiente se resumen las funciones que tendrá cada mote dentro de la simulación:

mote	*alias*	DAG root	Temperatura	Humedad	Movimiento	Calefacción
#0001	DAG	X	-	-	-	-
#0002	Comedor	-	X	X	-	X
#0003	Recibidor	-	X	-	X	-
#0004	Hab. 2	-	X	-	-	-
#0005	Hab. 1	-	X	-	-	-
#0006	Balcón	-	X	X	X	-

Tabla 12: Tabla resumen de funciones de cada mote

6.8.1 Script edge mote #0002

Se mostrarán las funciones añadidas al mote #0002 del fichero edge_mote2_bbdd.py.

```
1  #!/usr/bin/python
2
3  import os
4  import sys
5  import time
6  import MySQLdb as mdb
7
8  from random import randrange
9  from coap import coap
10
11 here = sys.path[0]
12 print here
13 sys.path.insert(0,os.path.join(here,'..','..','..','..','..','..','coap'))
14
15 MOTE IP 2 = 'bbbb::1415:92cc:0:2'
16 MOTE IP 3 = 'bbbb::1415:92cc:0:3'
17 MOTE IP 4 = 'bbbb::1415:92cc:0:4'
18 MOTE IP 5 = 'bbbb::1415:92cc:0:5'
19 MOTE IP 6 = 'bbbb::1415:92cc:0:6'
20
21 temp = 0;
22 humedad = 0;
23 calef = 0;
24 temp confort = 21
25
26 fecha = time.strftime("%y/%m/%d")
27 hora = time.strftime("%H:%M:%S") #Formato de 24 horas
28
29 c = coap.coap()
30 c.ackTimeout = 1
31 c.respTimeout = 0
```

Librerias necesarias: CoAP, MySQLdb, os, time, sys

Variables: direcciones IPv6 de cada mote, temp, humedad, calef, temp confort

Ilustración 64: Captura del script mote #0002, librerías e inicialización de variables

Aquí se ven como se utilizan los métodos GET en Python para comunicarse con el sensor para recoger la información de temperatura, humedad y valor del actuador de calefacción.

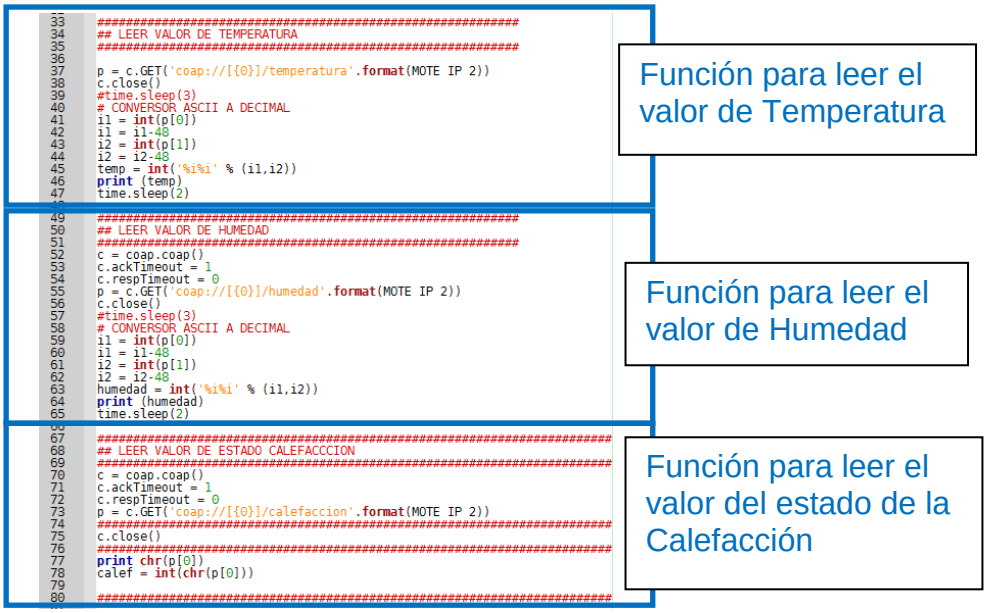


Ilustración 65: Captura de funciones de script del mote #0002, lectura de variables

A partir de un valor de temperatura deseado (*temp confort*) se actúa directamente sobre el actuador, dependiendo de si el actuador estaba activado o no previamente.

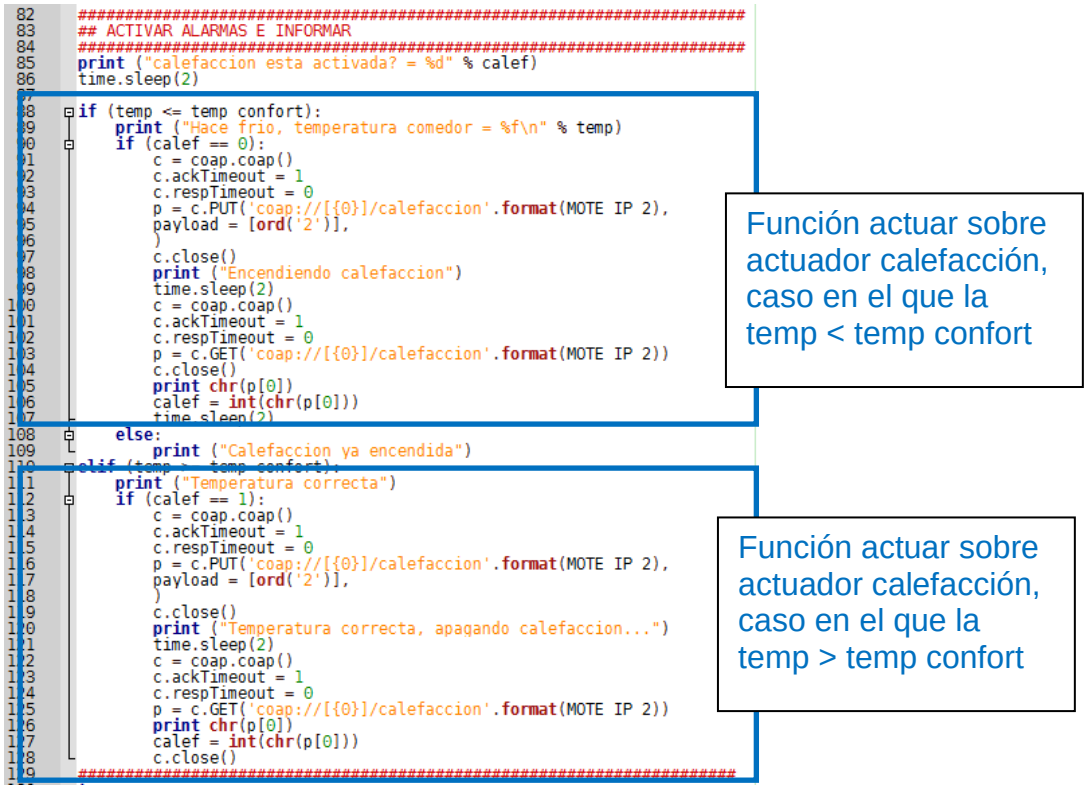


Ilustración 66: Captura de funciones de script del mote #0002, activar/desactivar calefacción

Después del procesado se escribe los valores recogidos en la base de datos en MySQL en la tabla histórico y el valor de la calefacción actual en la tabla *calef_status* junto con un *timestamp*.

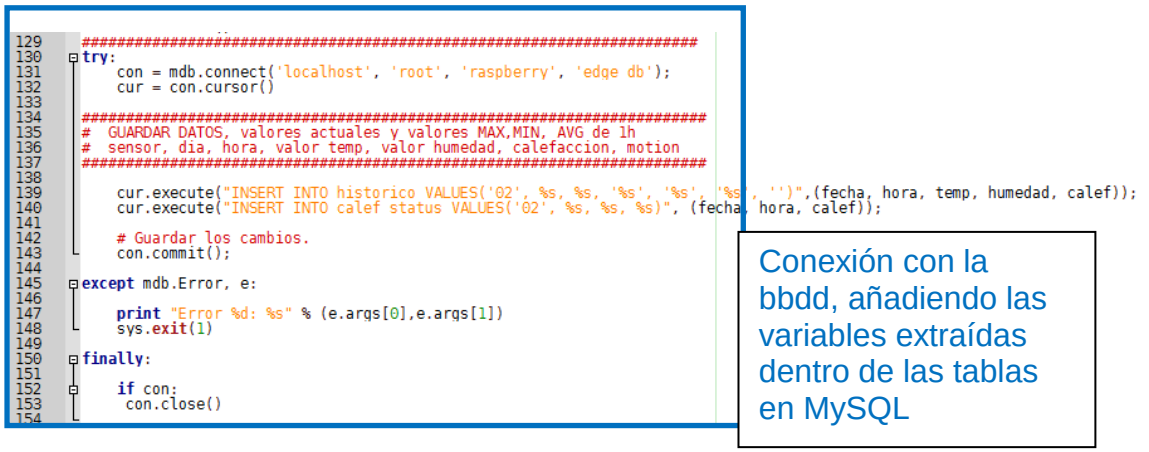


Ilustración 67: Captura de funciones de script del mote #0002, almacenar en bbdd edge_db

6.8.2 Script edge mote #0003

En este script se mostrarán solamente las funciones añadidas al mote #0003 del fichero edge_mote3_bbdd.py a diferencia del anterior.

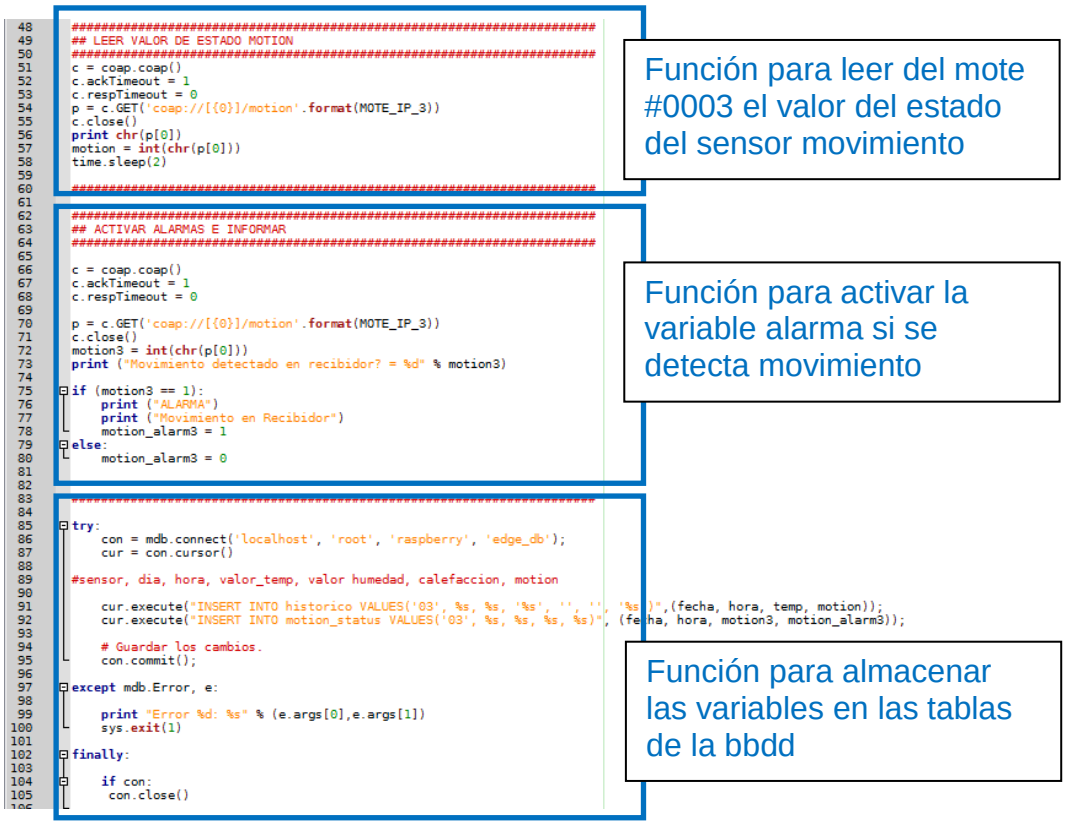


Ilustración 68: Captura de funciones de script del mote #0003, detector presencia y activar alarma

6.8.3 Script edge mote #0004, mote #0005 y mote #0006

En el caso motes #0004 y #0005 solo tienen la función de lectura de temperatura (edge_mote4_bbdd.py y edge_mote5_bbdd.py). El mote #0006 (edge_mote6_bbdd.py) que está ubicado en el balcón del domicilio tiene las funciones de lectura de temperatura, humedad y sensor de presencia, funciones que ya han sido descritas más arriba.

6.8.4 Script edge para almacenaje de variables resumen en el cloud theThings.iO

Este script permite la generación de los datos generados durante 3 horas, que son almacenados en la tabla *historico*, haciendo las llamadas en Python utilizando la librería MySQLdb para generar los valores máximo, mínimo y medias (avg) y subiéndolo a la nube en la plataforma theThings.iO. En esta captura se ve como se define el Token que se nos proporciona en la plataforma theThings.iO para poder conectarnos a sus servidores utilizando su API gracias a la librería *thethingsAPI*.

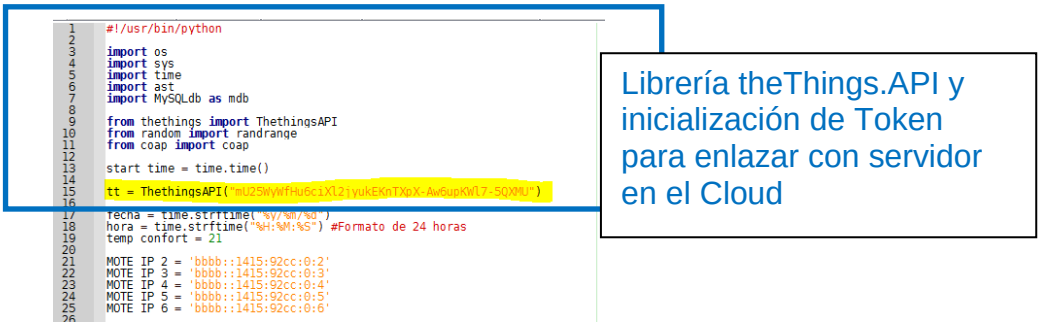


Ilustración 69: Captura del script: librerías e inicialización del Token de theThings.iO

Aquí se muestra como se abre la base de datos edge_db y se seleccionan un rango de datos de tres horas para cacular los valores MAX, MIN y AVG de los valores de temperatura que se encuentran en la tabla *historico*.

Ilustración 70: Consultas para extraer los valores de temperatura resumen de la base de datos

Existen más líneas de código (para cada mote) pero por el bien del tamaño de la memoria y debido a su carácter repetitivo no las incluiré todas.

Abajo se muestra cómo se abre la tabla humedad y se seleccionan un rango de datos de tres horas para calcular los valores MAX, MIN y AVG de los valores de humedad que se encuentran en la tabla *historico.*

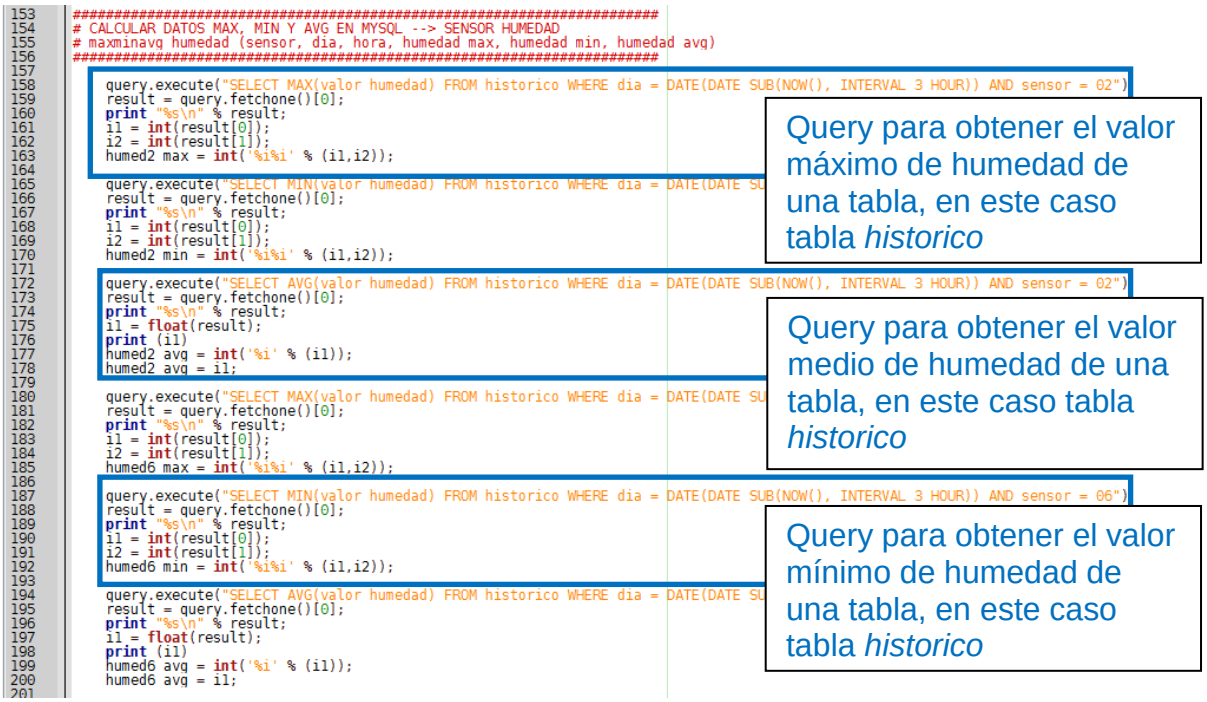


Ilustración 71: Consultas para extraer los valores de humedad resumen de la base de datos

Una vez obtenidos cada valor se guardan los valores resumen dentro de la tabla *maxminavg_temp* y *maxminavg_humedad* de la base de datos *edge_db* en MySQL. Finalmente escribimos en la plataforma theThings.iO utilizando los comandos antes descritos.

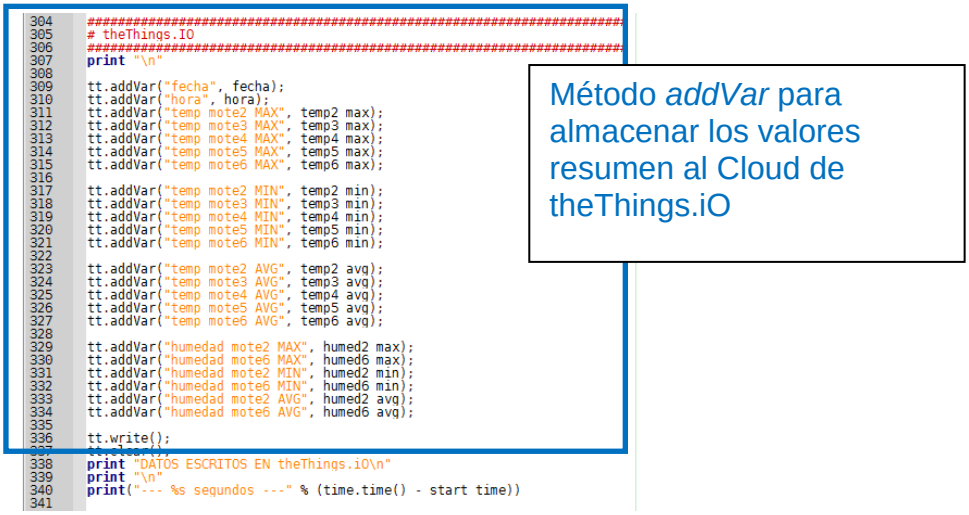


Ilustración 72: Subida de datos resumen al Cloud theThings.iO

6.9 Procesado Edge final

Se representa el esquema final del procesado en el Edge realizado con la plataforma *OpenWSN* y una Raspberry pi 3.

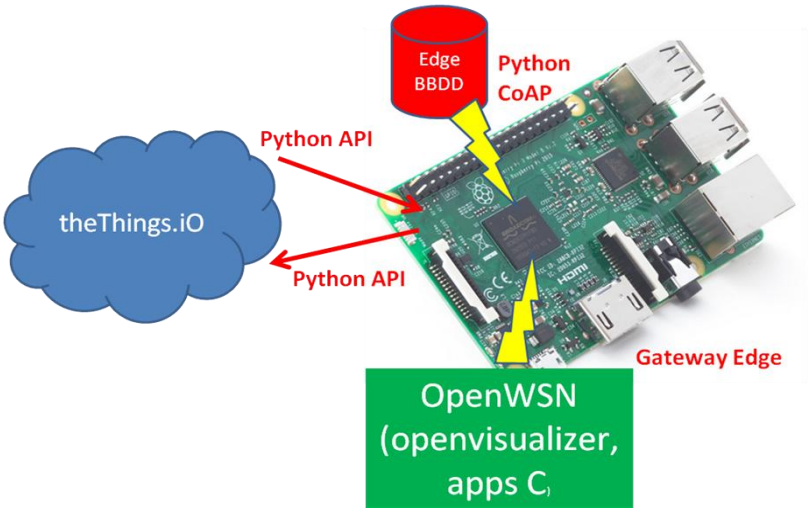


Ilustración 73: Esquema resumen del sistema Edge computing

6.9.1 Temporización del procesado

Se trata de secuenciar en el tiempo cada lectura de los sensores, cuando se graban los datos útiles y resumidos en la base de datos y en qué momento se suben a la nube en la plataforma thethings.iO, para en definitiva implementar el procesado Edge.

Estos son los scripts de lectura de datos, como ya hemos visto anteriormente:

- *edge_mote2_bbdd.py*
- *edge_mote3_bbdd.py*
- *edge_mote4_bbdd.py*
- *edge_mote5_bbdd.py*
- *edge_mote6_bbdd.py*

Se representa más abajo la secuencia creada en *crontab* donde se observa cómo se crea la secuenciación de lectura de cada variable que soporta cada mote y como escribe cada uno los datos obtenidos en la base de datos.

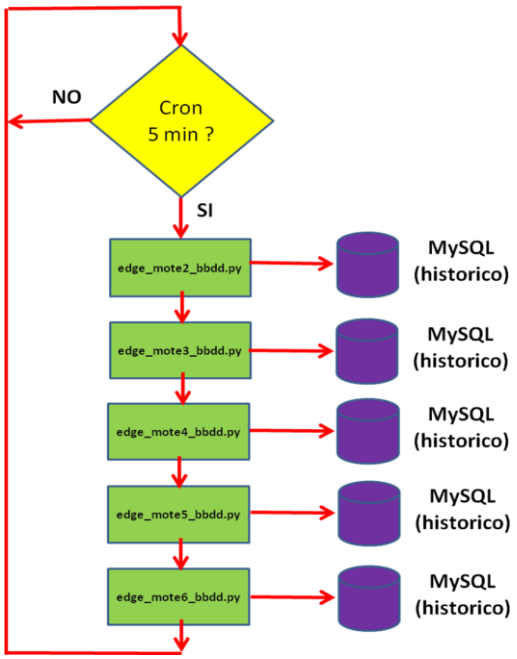


Ilustración 74: Secuencia temporal de petición de lectura de variables de cada mote

El script de escritura de datos resumidos (valores máximos, mínimos y medias) en la base de datos y en la nube se llama *edge_maxminavg2theThingsIO.py. De igual manera se crea con crontab y se observa cómo se llama a este script cada 3 horas para generar los valores resumen.*

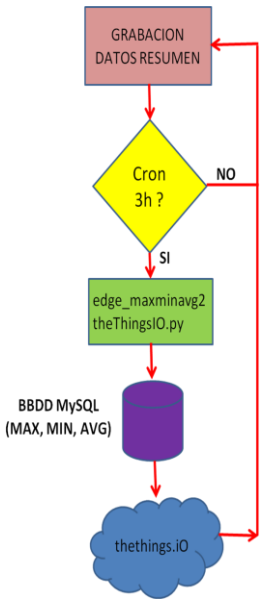


Ilustración 75: Secuencia de petición de lectura de variables resumen y escritura en el Cloud

Una vez llegado a este punto iniciamos la simulación ejecutando el *openvisualizer* para hacer el test final de cómo se recogen los datos almacenando en las tablas internas y se suben a la nube. Vemos en la figura de abajo como se ha creado la red de sensores y como el nodo #0001 actúa como DAG Root.

Ilustración 76: Creación de red WSN de la aplicación Edge, nodo #0001 como DAG Root

En la captura de abajo vemos el tipo de topología creada (malla).

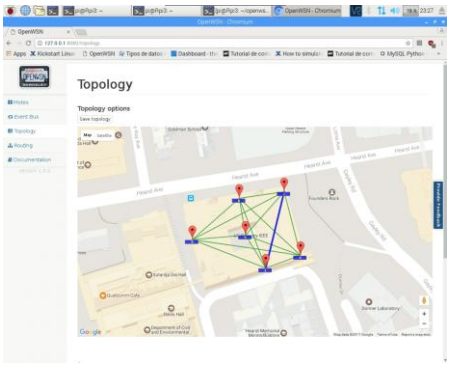


Ilustración 77: Topología de red creada con seis sensores

En las capturas siguientes se observan cómo se sincronizan y como se asignan las IP´s IPv6 a cada nodo y que canal utilizan para comunicarse.

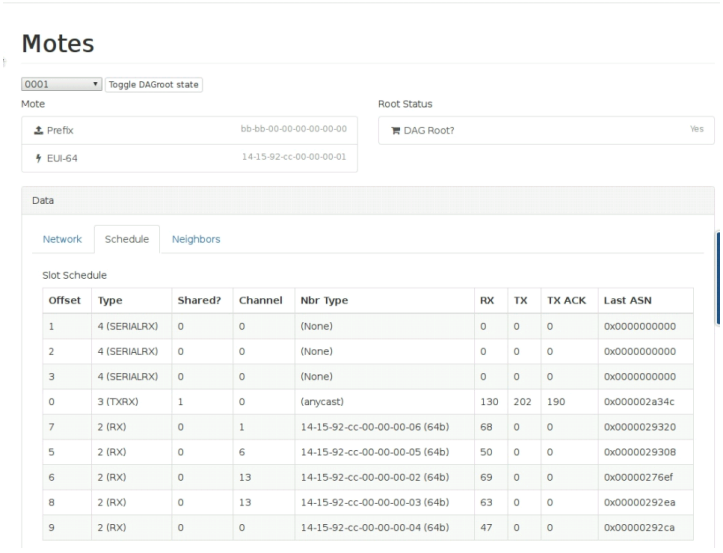


Ilustración 78: Scheduler del OpenWSN de simulación Edge

Una vez introducida la información en el crontab comienza el funcionamiento de la secuencia arriba expuesta y se empiezan a almacenar datos cada cinco minutos y cada tres horas se generan los datos resumen.

```
*/5 * * * *     /home/pi/openwsn/scripts/mote2cloud.sh && /home/pi/openwsn/scripts/delay.sh &&
 /home/pi/openwsn/scripts/mote3cloud.sh && /home/pi/openwsn/scripts/delay.sh &&
/home/pi/openwsn/scripts/mote4cloud.sh && /home/pi/openwsn/scripts/delay.sh &&
/home/pi/openwsn/scripts/mote5cloud.sh && /home/pi/openwsn/scripts/delay.sh &&
/home/pi/openwsn/scripts/mote6cloud.sh

*/180 * * * *   /home/pi/openwsn/scripts/edge_maxminavg_theThingsIO.sh
```

Secuenciación mediante script usando *Crontab*

Ilustración 79: Crontab, secuenciación de scripts

Aquí se pueden ver las capturas de cómo se guardan los datos en la BBDD en MySQL.

```
mysql> SELECT * FROM historico;
+--------+------------+----------+------------+---------------+-------------+--------+
| sensor | dia        | hora     | valor_temp | valor_humedad | calefaccion | motion |
+--------+------------+----------+------------+---------------+-------------+--------+
| 03     | 2017-05-10 | 23:10:05 | 23         | 79            | 0           | 1      |
| 03     | 2017-05-10 | 23:10:05 | 23         | 79            | 0           | 1      |
| 03     | 0000-00-00 | 00:00:00 | 23         | 79            | 0           | 1      |
| 03     | 2017-05-11 | 21:51:58 | 23         | 79            | 0           | 1      |
| 03     | 2017-05-12 | 00:12:17 | 23         | 79            | 0           | 1      |
| 03     | 2017-05-12 | 00:21:40 | 16         | 74            | 4           |        |
| 03     | 2017-05-12 | 00:22:51 | 24         | 90            | 4           |        |
+--------+------------+----------+------------+---------------+-------------+--------+
```

Tabla 13: Valores escritos en tabla historico

Aquí se pueden ver las capturas dentro de la tabla *maxminavg_temp* y como se almacenan los valores del *timestamp* de forma de *día, hora, número de sensor, valor temp_max, temp_min y temp_avg (media)*.

```
mysql> SELECT * FROM maxminavg_temp;
+--------+------------+----------+----------+----------+----------+
| sensor | dia        | hora     | temp_max | temp_min | temp_avg |
+--------+------------+----------+----------+----------+----------+
| 02     | 2017-05-14 | 17:18:50 | 25       | 25       |       20 |
| 02     | 2017-05-14 | 17:19:34 | 25       | 25       |       20 |
| 02     | 2017-05-14 | 17:20:39 | 25       | 25       |       20 |
| 02     | 2017-05-14 | 17:23:30 | 25       | 25       |       20 |
| 02     | 2017-05-14 | 17:24:39 | 25       | 25       |       20 |
| 02     | 2017-05-14 | 17:25:23 | 25       | 25       |       20 |
| 02     | 2017-05-14 | 17:25:57 | 25       | 25       |       20 |
| 02     | 2017-05-14 | 17:26:23 | 25       | 25       |       20 |
| 02     | 2017-05-14 | 17:27:06 | 25       | 25       |       21 |
| 02     | 2017-05-14 | 17:28:08 | 25       | 25       |       21 |
| 02     | 2017-05-14 | 17:30:58 | 25       | 25       |       20 |
| 02     | 2017-05-14 | 17:32:01 | 25       | 25       |       21 |
| 02     | 2017-05-14 | 17:34:02 | 25       | 25       |  20.5455 |
| 02     | 2017-05-14 | 17:34:40 | 25       | 25       |  20.5455 |
| 02     | 2017-05-14 | 17:35:10 | 25       | 25       |  20.5455 |
| 02     | 2017-05-14 | 17:36:16 | 25       | 25       |  20.5455 |
| 02     | 2017-05-14 | 17:37:14 | 25       | 16       |  20.5455 |
| 02     | 2017-05-14 | 17:40:44 | 25       | 16       |  20.5455 |
| 02     | 2017-05-14 | 17:45:36 | 25       | 16       |  20.5455 |
+--------+------------+----------+----------+----------+----------+
```

Tabla 14: Valores escritos en tabla maxminavg_temp

Aquí se pueden ver las capturas dentro de la tabla *maxminavg_humedad* y como se almacenan los valores del *timestamp* de forma de *día, hora, número de sensor, valor humedad_max, humedad_min y humedad_avg (media)*.

```
mysql> SELECT * FROM maxminavg_humedad;
+--------+------------+----------+--------------+--------------+--------------+
| sensor | dia        | hora     | humedad_max  | humedad_min  | humedad_avg  |
+--------+------------+----------+--------------+--------------+--------------+
| 02     | 2017-05-14 | 17:45:36 | 95           | 56           |         75.3 |
| 06     | 2017-05-14 | 17:45:36 | 91           | 56           |         74.4 |
| 02     | 2017-05-14 | 17:47:52 | 95           | 56           |      75.3182 |
| 06     | 2017-05-14 | 17:47:52 | 91           | 56           |      74.4545 |
| 02     | 2017-05-14 | 17:50:20 | 95           | 56           |      75.3182 |
| 06     | 2017-05-14 | 17:50:20 | 91           | 56           |      74.4545 |
| 02     | 2017-05-14 | 17:51:36 | 95           | 56           |      75.3182 |
| 06     | 2017-05-14 | 17:51:36 | 91           | 56           |      74.4545 |
| 02     | 2017-05-14 | 17:53:46 | 95           | 56           |      75.3182 |
| 06     | 2017-05-14 | 17:53:46 | 91           | 56           |      74.4545 |
| 02     | 2017-05-14 | 18:36:51 | 95           | 56           |      75.3182 |
| 06     | 2017-05-14 | 18:36:51 | 91           | 56           |      74.4545 |
| 02     | 2017-05-14 | 18:37:00 | 95           | 56           |      75.3182 |
| 06     | 2017-05-14 | 18:37:00 | 91           | 56           |      74.4545 |
| 02     | 2017-05-14 | 19:00:01 | 95           | 56           |      75.3182 |
| 06     | 2017-05-14 | 19:00:01 | 91           | 56           |      74.4545 |
| 02     | 2017-05-14 | 19:47:49 | 95           | 56           |      75.3182 |
| 06     | 2017-05-14 | 19:47:49 | 91           | 56           |      74.4545 |
| 02     | 2017-05-14 | 20:00:01 | 95           | 56           |      75.0417 |
+--------+------------+----------+--------------+--------------+--------------+
```

Tabla 15: Valores escritos en tabla maxminavg_humedad

Aquí se pueden ver algunas capturas dentro de la tabla *calef_status* y como se almacenan los valores del *timestamp* de forma de *día, hora, número de sensor y valor calef_status* que indica si la calefacción está encendida o apagada).

```
mysql> SELECT * FROM calef_status;
+--------+------------+----------+--------------+
| sensor | dia        | hora     | calef_status |
+--------+------------+----------+--------------+
| 02     | 2017-05-14 | 23:56:13 | 0            |
| 02     | 2017-05-14 | 23:58:09 | 1            |
| 02     | 2017-05-14 | 23:58:40 | 1            |
| 02     | 2017-05-14 | 23:59:26 | 1            |
| 02     | 2017-05-15 | 00:02:28 | 1            |
| 02     | 2017-05-17 | 22:43:16 | 0            |
| 02     | 2017-05-18 | 19:58:42 | 0            |
| 02     | 2017-05-18 | 20:01:39 | 0            |
| 02     | 2017-05-18 | 20:05:36 | 1            |
| 02     | 2017-05-18 | 20:06:34 | 0            |
| 02     | 2017-05-18 | 20:10:59 | 1            |
| 02     | 2017-05-18 | 20:13:30 | 0            |
| 02     | 2017-05-18 | 20:14:58 | 1            |
| 02     | 2017-05-18 | 20:15:49 | 0            |
| 02     | 2017-05-18 | 20:17:16 | 0            |
```

Tabla 16: Valores escritos en tabla calef_status

Aquí se pueden ver las capturas dentro de la tabla *motion_status* y como se almacenan los valores del *timestamp* de forma de *día, hora, número de sensor y valor motion_status y motion_alarm* que indica si ha habido movimiento o no.

```
mysql> SELECT * FROM motion_status;
+--------+------------+----------+---------------+--------------+
| sensor | dia        | hora     | motion_status | motion_alarm |
+--------+------------+----------+---------------+--------------+
| 03     | 2017-05-15 | 00:22:59 | 1             | 1            |
| 06     | 2017-05-15 | 00:22:59 | 0             | 0            |
| 03     | 2017-05-15 | 00:23:13 | 0             | 0            |
| 06     | 2017-05-15 | 00:23:13 | 0             | 0            |
| 03     | 2017-05-15 | 00:23:46 | 0             | 0            |
| 06     | 2017-05-15 | 00:23:46 | 1             | 1            |
| 03     | 2017-05-17 | 22:19:48 | 0             | 0            |
| 03     | 2017-05-17 | 22:36:55 | 1             | 1            |
| 06     | 2017-05-17 | 22:43:40 | 1             | 1            |
| 06     | 2017-05-17 | 22:45:25 | 0             | 0            |
| 03     | 2017-05-18 | 19:58:48 | 1             | 1            |
| 06     | 2017-05-18 | 19:59:01 | 1             | 1            |
| 06     | 2017-05-18 | 20:01:55 | 0             | 0            |
| 06     | 2017-05-18 | 20:05:52 | 1             | 1            |
| 03     | 2017-05-18 | 20:06:40 | 0             | 0            |
| 06     | 2017-05-18 | 20:06:58 | 1             | 1            |
```

Tabla 17: Valores escritos en tabla motion_status

Respecto a la plataforma thethings.iO aquí vemos como se graban los datos.

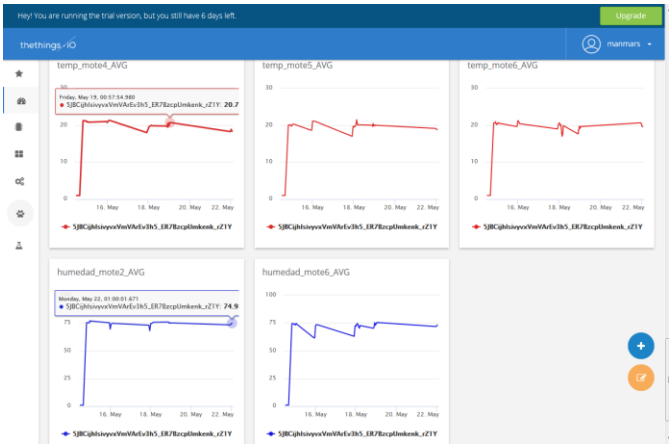


Ilustración 80: Gráficos generados por plataforma theThings.iO: valores medios

La plataforma theThings.iO dispone de una consola de desarrollo donde se puede ver las llamadas a sus servidores. En este caso se muestra la escritura en la plataforma con los datos resumen generado por el script *edge_maxminavg2theThingsIO.py.*

Thing write
5JBCijhIsivyvxVmVArEv3h5_ER7BzcpUmkenk_rZ1Y

[{"value":"17/05/22","key":"fecha"},{"value":"21:49:45","key":"hora"},{"value":25,"key":"temp_mote2_MAX"},
{"value":25,"key":"temp_mote3_MAX"},{"value":25,"key":"temp_mote4_MAX"},{"value":25,"key":"temp_mote5_MAX"},
{"value":25,"key":"temp_mote6_MAX"},{"value":15,"key":"temp_mote2_MIN"},{"value":16,"key":"temp_mote3_MIN"},
{"value":16,"key":"temp_mote4_MIN"},{"value":15,"key":"temp_mote5_MIN"},{"value":16,"key":"temp_mote6_MIN"},
{"value":19.523809523809526,"key":"temp_mote2_AVG"},{"value":21.1875,"key":"temp_mote3_AVG"},
{"value":19.625,"key":"temp_mote4_AVG"},{"value":20.705882352941178,"key":"temp_mote5_AVG"},
{"value":21.266666666666666,"key":"temp_mote6_AVG"},{"value":93,"key":"humedad_mote2_MAX"},
{"value":90,"key":"humedad_mote6_MAX"},{"value":55,"key":"humedad_mote2_MIN"},{"value":58,"key":"humedad_mote6_MIN"},
{"value":75.04761904761905,"key":"humedad_mote2_AVG"},{"value":73.6,"key":"humedad_mote6_AVG"}]

Ilustración 81: Consola de desarrollo, escritura en la plataforma de valores resumen

6.9.2 Procesado clásico en el Cloud

Es necesario contrastar los resultados del Edge computing con una situación cotidiana de subida de datos masivos de una red de dispositivos similar a la mostrada hasta ahora, por lo tanto se modificaron los scripts antes descritos para implementar el procesado en el Cloud. Al igual que pasaba en los apartados anteriores, por el bien de contener la memoria en el tamaño fijado se indican tan solo las diferencias con el anterior tipo de procesado y objetivo de la presente memoria.

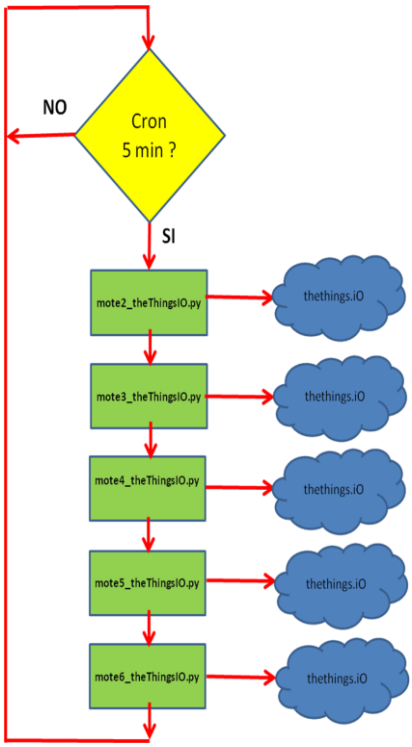


Ilustración 82: Secuencia de petición de lectura de variables resumen y escritura directa en el Cloud

Estos son los scripts de lectura de datos, como ya hemos visto anteriormente:

- *mote2_theThingsIO.py*
- *mote3_theThingsIO.py*
- *mote4_theThingsIO.py*
- *mote5_theThingsIO.py*
- mote6_theThingsIO.py

El script de escritura de datos resumidos (valores máximos, mínimos y medias) en la base de datos y en la nube se llama *tTio_maxminavg2theThingsIO.py. Dichos valores máximos, mínimos y medias son generados a partir de un Cloud code* que permite la plataforma thethings.iO. La actuación sobre la calefacción se realiza con un *Job* añadido a la plataforma para que dependiendo del valor de temperatura leído del mote #0002 se active la calefacción mandando la orden a la Rpi (mediante un tt.read).

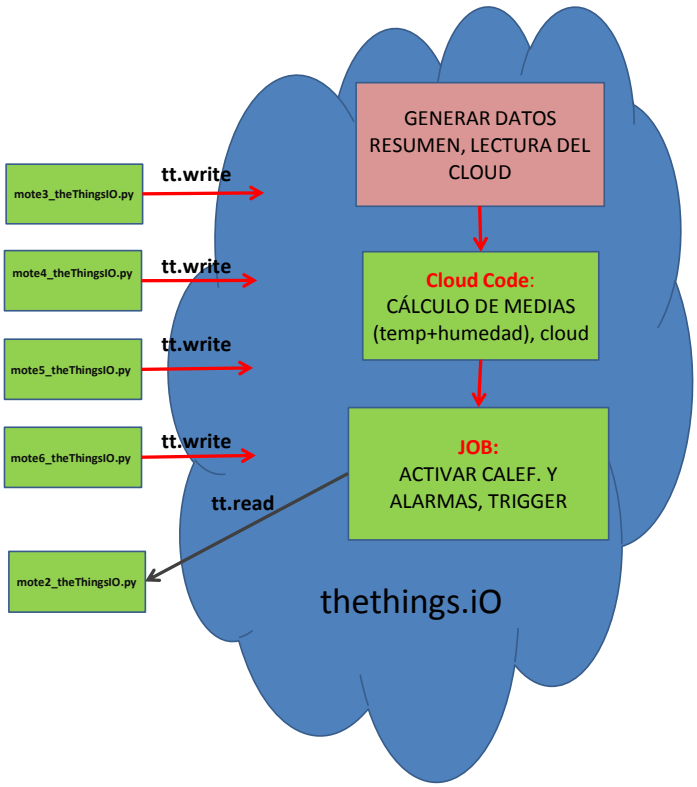


Ilustración 83: Secuencia de petición de datos resumen al Cloud, cálculo de medias y escritura

Se añade la parte del código para poder escribir en thethings.iO y como se almacenan en la nube todos los valores leídos del mote junto a su *timestamp*, en este caso la temperatura, humedad y estado de la calefacción.

```
79  ##########################################################################
80
81  tt.addVar("fecha", fecha);
82  tt.addVar("hora", hora);
83  tt.addVar("temp mote2", temp, dt=None);
84  tt.addVar("humedad mote2", humedad);
85  tt.addVar("calef mote2", calef);
86
87  tt.write();
88  tt.clear();
89
```

Ilustración 84: Escritura de variables en el Cloud theThings.iO

Como se ha comentado anteriormente se añade un *Cloud code* (*trigger*) y el Job en thethings.iO para actuar sobre la calefacción y hacer el cálculo en la nube.

En la figura de abajo se observa el código escrito en Javascript comparando la temperatura actual del mote #0002 con una temperatura fija (21º). Hay que decir que el tiempo de ejecución de este código está limitado a dos segundos dentro de la plataforma por lo que no puede contener código demasiado complejo.

encender_Calefaccion

```
function trigger(params, callback){
  console.log('¡¡ Se ha activado la alarma !!')
  //ignore non write events
  if(params.action !== 'write') return callback()

  var values = params.values
  var thingToken = params.thingToken

  //iterate over the values of the write
  for(var i=0; i<values.length; ++i){
    if(values[i].key === 'temp_mote2' && values[i].value <= 21){
      console.log('hace mucho frio %d/n',values[i].value)
      email(
        {
          service : 'SendGrid',
          auth: {
            api_user: 'manmarsal',
            api_key: 'YOUR API KEY'
          }
        },
        {
          from: 'manmarsal@gmail.com',
          to: 'manmarsal@gmail.com',
          subject : 'La temperatura es muy baja',
          text : 'HACE MUCHO FRIO! \n\n Saludos,\n '+ thingToken
        }
      )
    }
  }
  //end the trigger
  callback()
}
```

Ilustración 85: Cloud code para activar una alarma

Aquí se añade el resultado de la ejecución del trigger programado al recibir un valor de temperatura inferior al fijado.

Thing log

["hace mucho frio"]

Thing log

["¡¡ Se ha activado la alarma !!"]

Thing write

5JBCijhIsivyvxVmVArEv3h5_ER7BzcpUmkenk_rZ1Y

[{"value":"17/05/21","key":"fecha"},{"value":"22:50:01","key":"hora"},{"value":21,"key":"temp_mote2"},
{"value":73,"key":"humedad_mote2"},{"value":0,"key":"calef_mote2"}]

Ilustración 86: Consola de desarrollo: generación de alarma en el Cloud

Un Job es también escrito es Javascript el cual puede ser ejecutado una vez al día o cada hora dentro de la plataforma. Es por tanto una herramienta que permite ejecutar tareas periódicas como en este caso, donde se obtienen los valores máximo y mínimo de un rango de datos previamente almacenados.

Ilustración 87: Job para generar los valores máximo, mínimo

7. Resultados: comparación Cloud vs Edge computing

En este capítulo se presenta la comparación entre las dos plataformas de computación, el Cloud y el Edge computing.

Se han tenido en cuentas las siguientes métricas para poder hacer esta comparativa:

- Tiempo de ejecución.
- Tiempo de ocupación de red.
- Coste de ambas soluciones.
- Seguridad de los datos.

Como se aprecia los tiempos de procesado son similares. En el caso del procesado en Edge se hace escritura en disco en la BBDD en MySQL, por el contrario del caso en el Cloud que se utiliza la plataforma thethings.iO, para almacenar los datos. Se ha observado que los resultados son variables y dependientes de estos factores:

- Disponibilidad del servicio *thethings.iO*. llegando a añadirse un tiempo extra de 10 segundos.
- Colisiones CoAP en la utilización de los métodos GET. Se ha observado que no se pueden realizar peticiones nuevas de forma consecutiva, ya que se provocan colisiones, haciendo que no se obtenga el dato (*timeout*).
- En Edge, cada lectura de los motes se hace procesando localmente y actuando en tiempo real sobre el actuador, a diferencia del caso del Cloud, donde la lectura se realiza de igual manera pero subiendo todos los datos, para que el Cloud los procese y ejecute una acción (del orden de 5 segundos extra).

	theThings.iO	Rpi Edge
Tiempo ejecución x Script (mote)/seg	15	13
Tiempo ejecución x Script general (mote)/seg	29	34
Tiempo ejecución x Script (datos resumen)/seg	3,13	3,2
Tiempo de ocupación red (mote)/seg	2	0
Tiempo de ocupación red (datos resumen)/seg	3	3

Tabla 18: Tabla de comparativa de tiempos

Dichos tiempos se han calculado controlando el tiempo de inicio del script y tomando el tiempo final del mismo para calcular la diferencia (delta), siempre de conexiones efectivas y sin errores.

En términos de coste se ha hecho el siguiente cálculo comparativo. Se ha considerado que los bytes de subida de cada sensor son del orden de 547 bytes (cada paquete subido a la nube) y de 1,5K de datos resumen que fueron obtenidos a partir de varias capturas utilizando la herramienta *Wireshark,* para analizar las tramas enviadas a la nube.

Edge	
bytes datos resumen	1.500
bytes resumen/dia	36.000
bytes resumen/mes	1.080.000
bytes resumen/anuales	12.960.000
Cloud	
bytes/subida sensor /paquete	547
sensores	5
llamadas / hora	30
llamadas / hora / 5 sensores	150
bytes / hora / red 5 sensores	82.050
bytes / dia	1.969.200
bytes /mes	59.076.000
bytes / año	708.912.000

Tabla 19: Tabla comparativa de cantidad de datos generados subidos

Se han estimado un total de 13 MB de datos anuales generados por la red de cinco sensores en la aplicación Edge, donde solo se suben los datos resumen al Cloud. Respecto a la situación clásica de *Cloud computing* se ha considerado una subida de datos cada dos minutos por sensor, haciendo que el tamaño de datos subidos anuales sea del orden de 709 MB.

	theThings.iO	Rpi Edge	Amazon AWS
Llamadas a la API (máx)	50000	n/a	n/a
Cloud code (máx)	10000	n/a	n/a
Llamadas sensor (cadencia/minutos)	2	2	2
Sensores (aplicación)	5	5	5
Llamadas diárias	14400	4800	
Llamadas mensuales	432000	144000	ilimitado
Cloud code diários (valores MAX, MIN, AVG), una cada hora	24	8	24h/dia
Cloud code mensual = instancias (AMAZON)	720	240	720
COSTE (€/mes)	29	-	2906,64

Tabla 20: Tabla comparativa de costes

Se puede ver como los beneficios a nivel de coste del Edge computing son evidentes para una aplicación modesta pero que requiera de una actuación en tiempo real. Comparado con el servicio que ofrece Amazon (AWS) se ve diferencias notables, porque esta última opción va dirigida a aplicaciones donde el número total de dispositivos sea del orden de dos o tres órdenes de magnitud más elevados (coste basado en un equipo dedicado Linux t2-nano con 4GB de almacenamiento de datos). Comparándolo con la plataforma theThings.iO vemos que su coste no es demasiado elevado permitiéndose una cierta cantidad fija de llamadas a su servidor, pero comparándolo con la aplicación desarrollada en este trabajo no permite tanta flexibilidad a la hora de generar procesado (limitada a dos segundos de ejecución de *Cloud code* como máximo), actuación en tiempo real y seguridad de los datos sensibles viajando hacia Internet. Se añaden las capturas de la calculadora de servicios AWS de Amazon donde se ve el coste antes mostrado de forma resumida.

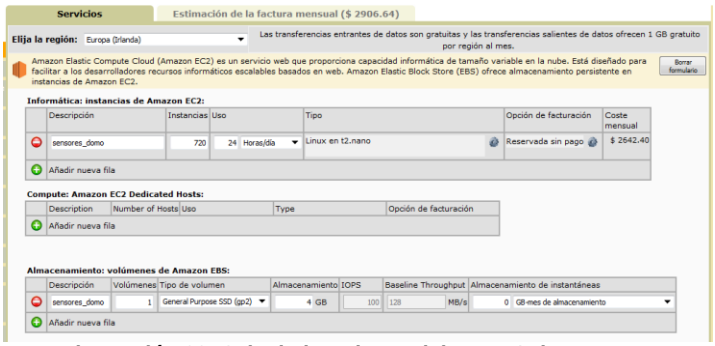


Ilustración 88: Calculadora de servicios AWS de Amazon

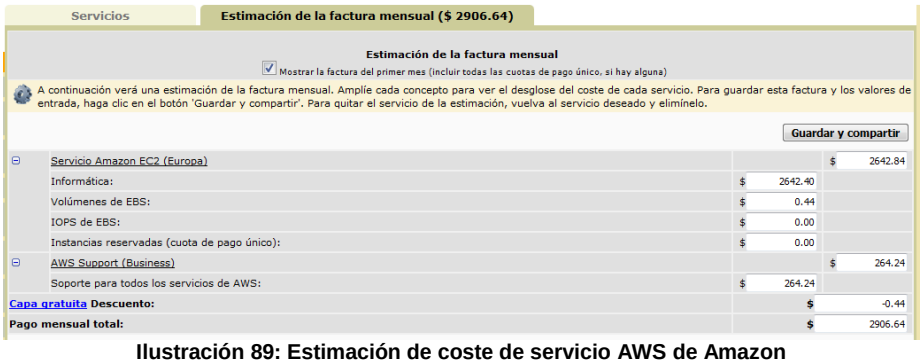


Ilustración 89: Estimación de coste de servicio AWS de Amazon

8. Conclusiones

En la memoria del presente proyecto se han explicado los diferentes conceptos y protocolos relacionados con el mundo del Internet de las cosas, así como los protocolos utilizados en este trabajo.

También se han descrito y comparado las diferentes formas de procesado de datos existentes en el mundo de los ioT, entre ellas el procesado en el borde (Edge) objetivo del presente trabajo final.

Se ha realizado un demostrador de ioT utilizando el sistema operativo *OpenWSN* simulando una situación doméstica de despliegue de sensores y actuadores. Se han visto las virtudes y problemas relacionados con el número total de motes a simular y también el diferente comportamiento de la plataforma, dependiendo del tipo de topología usada siendo la combinación de seis sensores y la topología en malla la más estable.

Se ha aprendido por tanto, el funcionamiento interno del *OpenWSN* para modificar el *firmware* de los motes (en lenguaje C), para añadir las funciones descritas en esta memoria (sensado de temperatura, humedad, detección de movimiento y actuador sobre el sistema de calefacción), así como utilizar las herramientas de que dispone.

También se han sufrido cambios en la planificación original, debido sobretodo a la creación de las funciones antes descritas dentro del firmware de OpenWSN, pero se añadieron las actividades necesarias dentro del margen original sin afectar a la fecha de finalización.

También se ha desarrollado el software en lenguaje Python, utilizando las librerías de CoAP y thethings.iO, para implementar el procesado Edge y se han comparado los resultados con una situación clásica de subida de datos a la nube (*Cloud computing*), utilizando también la plataforma thethings.iO, viendo las ventajas del procesado en el Edge con respecto a procesados típicos.

En lo referente a posibles mejoras, debido al corto plazo de planificación del TFM, se hubiera podido implementar un mejor cálculo del tiempo de ejecución de los scripts, también la de tareas de depuración del código y unificación de los mismos.

9. Glosario

API: *Application Programming Interface*
ARM: *Avanced RISC Machine*
BSP: *Board Support Package*
CoAP: *Constrained Application Procotol*
CON: *Confirmable*
CSMA/CA: *Carrier Sense Multiple Access with Collision Avoidance*
EB: *Enhanced Beacon*
EBR: *Enhanced Beacon Requests*
ETSI: *European Telecommunications Standards Institute*
GPIO: *General-purpose input/output*
GUI: *Graphical user interface*
HTTP: *Hypertext Transfer Protocol*
IEEE: *Institue of Electrical and Electronics Engineers*
IETF: *Internet Engineering Task Force*
IoT: *Internet of Things*
IP: *Internet Protocol*
JSON: *JavaScript Object Notation*
MAC: *Medium Access Control*
M2M: *Machine to Machine*
MTU: *Maximum Transfer Unit*
NFC: *near field communications*
NON: *Non- Confirmable*
PAN: *Personal Area Network*
SoC: *System on Chip*
TCP: *Transmission Control Protocol*
TSCH: *Time Slotted Channel Hopping*
UDP: *User Datagram Protocol*
UIT: *Unión Internacional de telecomunicaciones*
URI: *Uniform Resource Identifier*
WPAN: *Wireless Personal Area Networks*
WSN: *Wireless Sensor Network*

10. Bibliografía

[1] Web de la ITU; http://www.itu.int/es/Pages/default.aspx

[2] Al-Fuqaha, Ala; Guizani, Mohsen; Mohammadi, Mehdi; **Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications**; IEEE Communications Surveys & Tutorials, 2015, Volum 17, Número 4. http://0-ieeexplore.ieee.org.cataleg.uoc.edu/document/7123563/

[3] Helena Rifà Pous; **Contexto actual y evolución hacia las redes de nueva generación.** Biblioteca de la UOC, 06/2017. http://uoc.summon.serialssolutions.com/#!/search?bookMark=ePnHCXMw42JgAfZbUzkZHMHHK1WU5CskgrdCKFQgpIJyXHLm4c15ChmJQKsVgA1EhaLUlNRihZRUhbzS1LJEhXTwactgRdwMum6uIc4euqX5yfHgucPieNBpyMll8WCRlNL4AE8X0Kosc1MzY9DVt6SpBwAMJDqC

[4] Gia, Tuan Nguyen; Jiang, Mingzhe; Rahmani, Amir-Mohammad; **Fog Computing in Healthcare Internet of Things: A Case Study on ECG Feature Extraction**; 2015 IEEE International Conference on Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing, 2015. http://0-ieeexplore.ieee.org.cataleg.uoc.edu/document/7363093/

[5] Condry, Michael W; Nelson, Catherine Blackadar; **Using Smart Edge IoT Devices for Safer, Rapid Response With Industry IoT Control Operations** Proceedings of the IEEE, 2016, Volum 104, Número 5. http://0-ieeexplore.ieee.org.cataleg.uoc.edu/document/7423655/

[6] Dastjerdi, Amir Vahid; Buyya, Rajkumar; **Fog Computing: Helping the Internet of Things Realize Its Potential**; Computer, 2016, Volum 49, Número 8. http://0-ieeexplore.ieee.org.cataleg.uoc.edu/document/7543455/

[7] Hao, Zijiang; Novak, Ed; Yi, Shanhe; **Challenges and Software Architecture for Fog Computing**; IEEE Internet Computing, 2017, Volum 21, Número 2. http://0-ieeexplore.ieee.org.cataleg.uoc.edu/document/7867731/

[8] Tomovic, Slavica; Yoshigoe, Kenji; Maljevic, Ivo; **Software-Defined Fog Network Architecture for IoT**; Wireless Personal Communications, 01/2017, Volum 92, Número 1. http://uoc.summon.serialssolutions.com/#!/search?bookMark=ePnHCXMwdV07DsIwDI0QA59LZGAkEk0aaEbER7CwABJblGKHsRUgcX3spgUW1sh-TaTWdm3neST69N-Koi0gNrfUMq4TUmyhTJFbNesxhemCPkSTX_pMhccdGGRCB5_JA7Q0FNmRDNEr3FGtMVLIBXJb3eQhdUfL5U-eXVJ8J_fVaSzO281ptVPtHAF15aFaKtLjI7tuZ20sHFf2MMzQRvLlBYLLA2hwWOYO3JzMs4kmgHGFBrSAOpqxyBJu57l8nagg_Jf0l8_ouV2Kz-

h5Btck6XQW3oeSkxfX58MzBxo5Sor0Scwksa4M-4VuEDVDLtNLlKBr4A1N_2iR_KJVddpnjYp5A6MVcZQ

[9] Hajji, Wajdi; Tso, Fung; **Understanding the Performance of Low Power Raspberry Pi Cloud for Big Data Electronics**, 06/2016, Volum 5, Número 2. http://www.mdpi.com/2079-9292/5/2/29

[10] Cisco White paper; **Fog Computing and the Internet of Things: Extend the Cloud to Where the Things Are**. https://www.cisco.com/c/dam/en_us/solutions/trends/iot/docs/computing-overview.pdf

[11] Cisco White paper; **Cisco Fog Computing Solutions: Unleash the Power of the Internet of Things**. https://www.google.es/url?sa=t&rct=j&q=&esrc=s&source=web&cd=3&cad=rja&uact=8&ved=0ahUKEwiN8LCEvYHUAhXJ1hoKHZoIADoQFghAMAI&url=https%3A%2F%2Fwww.researchgate.net%2Ffile.PostFileLoader.html%3Fid%3D57bea7b6217e20e33f730969%26assetKey%3DAS%253A398883164311552%254014721125651144&usg=AFQjCNGfa9llwJCgk1nVJg_QA2Eex9BkDA

[12] K.P.Saharan, Anuj Kumar; **Fog in Comparison to Cloud: A Survey,** International Journal of Computer Applications (0975 – 8887) Volume 122, No. 3, July 2015. http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=2E68E19617EBD85C990DDEE4AEE08D86?doi=10.1.1.695.1847&rep=rep1&type=pdf

[13] Alejandro Aguirre**; El trabajo en conjunto del Edge Computing y la Computación en la Nube.** https://es.linkedin.com/pulse/el-trabajo-en-conjunto-del-edge-computing-y-la-nube-alejandro-aguirre

[14] Sitio Web RFC7252; **The Constrained Application Protocol (CoAP).** https://tools.ietf.org/html/rfc7252

[15] Steven Carlini, APC White paper; **The Drivers and Benefits of Edge Computing.** http://www.apc.com/salestools/VAVR-A4M867/VAVR-A4M867_R0_EN.pdf?sdirect=true

[16] Sitio Web CoAP**; Información sobre el protocolo CoAP.** http://coap.technology/

[17] Sitio Web *OpenWSN*. **Estàndard IEEE802.15.4e.** https://*OpenWSN*.atlassian.net/wiki/display/OW/IEEE802.15.4e

[18] Sitio Web *OpenWSN*. **Estàndard IEEE802.15.4-2006**. https://*OpenWSN*.atlassian.net/wiki/display/OW/IEEE802.15.4-2006

[19] Sitio Web *OpenWSN*. **Proyecto *OpenWSN***. https://*OpenWSN*.atlassian.net/wiki/pages/viewpage.action?pageId=688187

[20] Sitio Web *OpenWSN*.**Hardware compatible con *OpenWSN***.
https://*OpenWSN*.atlassian.net/wiki/display/OW/

[21] Sitio Web *OpenWSN*. **Pila de protocolos de *OpenWSN***.
https://*OpenWSN*.atlassian.net/wiki/display/OW/Protocol+Stack

[22] Sitio Web *OpenWSN*. **OpenSim**.
https://*OpenWSN*.atlassian.net/wiki/display/OW/OpenSim

[23] Sitio Web *OpenWSN*. **OpenVisualizer**.
https://*OpenWSN*.atlassian.net/wiki/display/OW/OpenVisualizer

[24] Sitio Web *OpenWSN*. **Màquina de estados, librería Python de CoAP**.
https://*OpenWSN*.atlassian.net/wiki/display/OW/CoAP

[25] Sitio Web Raspberry Pi Foundation; **Productos Raspberry Pi**.
https://www.raspberrypi.org/products/

[26] Sitio Web Raspberry Pi Foundation; **Raspberry Pi 3, model B**.
https://www.raspberrypi.org/products/raspberry-pi-3-model-b/

[27] Sitio Web Wikipedia. **Raspberry Pi wiki.**
https://es.wikipedia.org/wiki/Raspberry_Pi

[28] Sitio Web PandaBoard; **The PandaBoard, an OMAP4430 (Cortex-A9) based low cost development platform.**
http://www.ti.com/devnet/docs/catalog/endequipmentproductfolder.tsp?actionPerformed=productFolder&productId=16961

[29] Sitio Web thethings.iO; **Página principal**. https://thethings.io/

[30] Sitio Web thethings.iO; **Quick Start**.
https://developers.thethings.io/index.html

[31] Sitio Web de Github; **thethings.iO Python library**.
https://github.com/theThings/thethings.iO-python-library

[32] Sitio web Firefox; **Complemento Copper (Cu) soporte CoAP.**
https://addons.mozilla.org/es/firefox/addon/copper-270430/

[33] Sitio Web TinyOS. **TinyOS**.
http://tinyos.stanford.edu/tinyos-wiki/index.php/Main_Page

[34] Sitio web RIOT. **The friendly Operating System for IoT**.
http://www.riot-os.org/#home

[35] Sitio web Nimbits. **Características del sistema operativo.**
https://www.nimbits.com/

[36] Sitio web Contiki. **The open source OS for the IoT.**
http://www.contiki-os.org/