Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels
UNIVERSITAT POLITÈCNICA DE CATALUNYA

TRABAJO FINAL DE GRADO

TÍTULO DEL TFG: Sistema distribuido para medida y procesado en tiempo real de sensores IoT

TITULACIÓN: Grado en Ingeniería de Sistemas de Telecomunicación

AUTORES: Cristian Cano Rigol
Alejandro Guerrero Nicolas

DIRECTOR: Oscar Casas Piedrafita

FECHA: 24 de octubre de 2017

Título: Sistema distribuido para medida y procesado en tiempo real de sensores IoT

Autores: Cristian Cano Rigol

Alejandro Guerrero Nicolas

Director: Oscar Casas Piedrafita

Fecha: 24 de octubre de 2017

Resumen

El Internet de las Cosas (IoT), comúnmente conocido como *Internet of Things*, es un término que actualmente utilizamos para describir todas aquellas tecnologías que hacen posible que elementos físicos y digitales de nuestro mundo puedan interactuar hoy en día. IoT está muy presente en nuestro día a día conectando todo tipo de elementos, desde dispositivos que utilizamos a diario en nuestra casa hasta máquinas que se usan en fábricas o tiendas, de tal manera que el humano pueda interactuar con ellas y las pueda modificar a su antojo. En IoT, los elementos están pensados para poder recoger todo tipo de información para posteriormente procesarla, analizarla y darle valor y así poder crear aplicaciones y ofrecer servicios innovadores. Estos servicios y aplicaciones, poco a poco están cambiando el mundo en el que vivimos haciéndolo cada vez mejor y haciendo que cambie el concepto de la tecnología tal y como la conocemos hoy en día.

Este proyecto tiene como objetivo, implementar una aplicación IoT usando una red de sensores inalámbrica (WSN). Cada sensor se encargará de recoger una serie de información, en nuestro caso temperatura, batería y dos sensores externos a elección del usuario, para proceder con su análisis y procesado. La información se le presentará al usuario a través de dos interfaces: una página web y una aplicación Android. En esta aplicación, el usuario podrá consultar los datos en diferentes formatos para su posterior análisis. Para llevar a cabo este proyecto, se ha usado una Raspberry Pi y 4 nodos Zolertia Z1. Este tipo de sensor utiliza el microcontrolador de bajo consumo MSP430F261. Contiki es el sistema operativo usado para que estos nodos funcionen. El papel que implementa Raspberry en este escenario es de *border router* que permite la comunicación entre la red WSN y una red externa que funcione con IP.

Para la comunicación de estos nodos, se han desarrollado una serie de programas que permiten a los nodos enviar datos usando UDP como protocolo de transporte. Por otro lado, para recibir esta información, se ha implementado un servidor UDP.

Finalmente, se ha ido un paso más allá y se ofrece la opción de que el usuario pueda escoger con qué tipo de sensor quiere trabajar.

Title: Distributed system for measurement and real-time processing of IoT sensors

Authors: Cristian Cano Rigol

Alejandro Guerrero Nicolas

Director: Oscar Casas Piedrafita

Date: October 24 th 2017

Overview

Internet of things (IoT) is a term we use today to describe all those technologies that enables interconnection of digital with physical world. IoT is present in our everyday life allowing to connect all kind of elements. They can be domestic devices, industrial or merchant ones, in such a way that the human can interact and modify them. In IoT, things are enabled to collect information from their surroundings, to later process it, analyze it and give it value and thus be able to create applications and offer innovative services. This services and applications are bit by bit changing the world, making it better and changing the concept of technology as we know it today.

This project aims to implement an IoT application using a wireless sensor network (WSN). Each sensor collects some information, in our case, temperature, battery level and the values of two external sensors, chosen by the user, connected to the mote to proceed with its analysis and processing. The information will be presented to the user through two interfaces: a web page and an Android application. The user can view data in different formats for further analysis. To carry out the network implementation, a Raspberry Pi 3 Model B and four Zolertia Z1 motes were used. Z1 motes feature a low power microcontroller MSP430F2617. Contiki was used as operating system for motes. Raspberry Pi plays the role of a border router that allows the communication between the WSN and external IP network.

To send the information from the nodes, we have developed a serie of programs that allow the sensor to send data using UDP as a transport protocol. On the other hand, to receive the information sent from the nodes, we have developed a UDP server.

Finally, we have gone one step forward and we offer to the user to choose what kind of sensor want to work with.

ÍNDICE

Introducción **1**

1. Fundamentos Teóricos **4**
1.1 Internet de las cosas (IoT) 4
1.2 Red Inalámbrica de Sensores (WSN, Wireless Sensor Network) 6
1.3 IPv6 8
1.4 6LoWPAN 10
1.5 Estándar IEEE 802.15.4 11
1.6 Sistemas operativos para redes WSN 12

2. Diseño e implementación de la red IoT **14**
2.1 Raspberry Pi 3 Modelo B 16
2.2 Zolertia Z1 17
2.3 Diseño funcional 18
2.4 Router de frontera o border router 20
2.5 Programación de nodos 30
2.6 Servidor UDP 33
2.7 Base de datos 34
2.8 Servidor web 35

3. Resultados Obtenidos **37**
3.1 Test de conectividad 38
3.2 Muestra de datos 40
3.3 Futuras implementaciones 44

Conclusiones **46**

Acrónimos **48**

Bibliografía **49**

Anexo I. Instalación de Contiki **51**
I. Instalando Instant Contiki 51
II. Compilar una aplicación en Contiki 51

Anexo II. Programación de los nodos **53**
I. Cliente UDP 53
II. Ejemplo de uso del sensor externo ZIG001 57
III. Ejemplo de uso del sensor externo ZIG002 62

Anexo III. Implementación Servidor UDP **67**
I. UDP Python Server 67

Anexo IV. Desarrollo para la visualización en Android **72**
I. Scripts PHP para proporcionar el servicio web 72
II. Código Java para inicializar la aplicación 79
III. Código Java para mostrar los datos gráficamente 87
IV. Código Java para descargar archivos CSV 101

Anexo V. Código para la implementación WEB **108**
I. Código Grafico Web 108
II. Script PHP export de datos CSV 113
III. Script PHP export de datos en JSON 114

Anexo VI. Tablas Comparativas 115
I. Tabla comparativa nivel 1.....115
II. Tabla comparativa nivel 2.....116
III. Tabla comparativa nivel 3.....117
IV. Tabla comparativa nivel 4.1.....118
V. Tabla comparativa nivel 4.2.....119
VI. Tabla comparativa nivel 5.....120
VII. Tabla comparativa de software nivel 1-5.....121

Lista de figuras

Fig. 1.1 “Dimensiones” que aporta el Internet de las cosas 4
Fig. 1.2 Previsión de IHS de dispositivos IoT conectados en 2017 5
Fig. 1.3 Inversión mundial en IoT (Fuente: www.forbes.com) 5
Fig. 1.4 Estructura de una red de sensores 7
Fig. 1.5 Modelo OSI en 6LoWPAN 10
Fig. 2.1 Infraestructura de la red a implementar 14
Fig. 2.2 Raspberry Pi 3 Modelo B 17
Fig. 2.3 Zolertia Z1 18
Fig. 2.4 Flujo de información en el sistema 19
Fig. 2.5 6LBR en modo router (Fuente: Github 6LBR wiki) 21
Fig. 2.6 6LBR en modo bridge (Fuente: Github 6LBR wiki) 21
Fig. 2.7 6LBR en modo router (Fuente: Github 6LBR wiki) 22
Fig. 2.8 6LBR en modo smart bridge (Fuente: Github 6LBR wiki) 22
Fig. 2.9 6LBR en modo transparent bridge (Fuente: Github 6LBR wiki) 23
Fig. 2.10 6LBR interfaz modo raw ethernet (Fuente: Github 6LBR wiki) 24
Fig. 2.11 6LBR interfaz en modo bridge (Fuente: Github 6LBR wiki) 24
Fig. 2.12 6LBR interfaz en routing mode (Fuente: Github 6LBR wiki) 26
Fig. 2.13 Servidor web de 6LBR 29
Fig. 2.14 Flujo de funcionamiento de los nodos 30
Fig. 2.15 Formato de un paquete de datos 32
Fig. 2.16 Diagrama de flujo Servidor UDP 33
Fig. 2.17 Estructura de la tabla Nodes 34
Fig. 2.18 Estructura de la tabla Measures 35
Fig. 3.1 Estructura de la red implementada 37
Fig. 3.2 Página de sensores en el servidor web de 6LBR 38
Fig. 3.3 Nodos vecinos y rutas en la red de sensores 38
Fig. 3.4 Configuración NAT64 39
Fig. 3.5 Pantalla de inicio y visualización de medidas 41
Fig. 3.6 Pantalla de descarga de datos históricos 42
Fig. 3.7 Página web donde escoger nodo 43
Fig. 3.8 Página web donde consultar los datos 44
Fig. 3.9 Inicialización de los sensores 45
Fig. 3.10 Funciones de lectura de los sensores 45

Lista de tablas

Tabla 2.1 Características principales de Raspberry Pi 3 Modelo B [9] 17
Tabla 2.2 Características principales de los nodos Zolertia Z1 [10] 18
Tabla 2.3 Estructura de datos 31

Introducción

Actualmente la tecnología está en constante evolución y uno de los campos más afectados son las comunicaciones. El acceso móvil a internet ha revolucionado todos los ámbitos y ha provocado grandes progresos. Uno de ellos es la aparición de una nueva "red", el internet de las cosas (IoT, *Internet of Things*).

Nos referimos al internet de las cosas como la interconexión digital de objetos cotidianos a través de internet con el objetivo de crear espacios inteligentes que mejoren nuestra calidad de vida. En el internet de las cosas todos los objetos son únicos en cuanto a identificación y están equipados con dispositivos de detección o sensores para interactuar entre ellos o con el entorno tomando información de su alrededor y mandándola a un analizador o reaccionando por ellos mismos. Esto conlleva el tratamiento de una gran cantidad de información que podemos usar para crear aplicaciones y servicios con el objetivo de mejorar en términos de eficiencia humana, energética y efectividad. Según varios estudios, como el de IHS Markit o IDC, se calcula que en 2017 habrá entre 20,7 y 28,1 billones de dispositivos IoT conectados. Y toda la información recopilada por estos dispositivos será accesible desde cualquier lugar, en cualquier momento y desde cualquier dispositivo.

Con todo esto, existen un gran número de ámbitos que se verán influenciados por el internet de las cosas, a nivel general como sociedad y también a nivel individual. Algunos de ellos pueden ser, por ejemplo, las ciudades inteligentes o *smart cities*, domótica, industria 4.0 o industria inteligente, transporte, salud, complementos, ambientes inteligentes (sistemas en los que la computación es usada para crear ambientes interactivos), etc.

Este proyecto se basa en la **implementación de una red de sensores** que captan cualquier tipo de señal, dependiendo de los sensores que se usen y de las limitaciones de ancho de banda del sistema, por lo que no está orientado a un ámbito en concreto de los expuestos arriba, sino que es aplicable a cualquiera de ellos. Los datos recogidos serán enviados a un dispositivo donde se almacenarán para su visualización en **tiempo real o en cualquier otro momento**.

Previamente a la elección de los dispositivos que se usarán para la implementación de la red se realizará una búsqueda de soluciones de bajo coste para encontrar los más adecuados según los requerimientos genéricos de la aplicación que se desea realizar. Para llevar a cabo este objetivo se ha usado una Raspberry Pi 3 modelo B y cuatro sensores Zolertia Z1. La Raspberry Pi hará de elemento central de la red. Llevará a cabo la función de *router* y servidor. A ella se conectarán los sensores para enviarle la información que recojan y también nos podremos conectar con otro dispositivo, desde una red IP local o externa, para consultar los datos que tenga almacenados en la base de datos. Para desarrollar la funcionalidad de *router* es necesario instalar CETIC 6LBR, una plataforma preparada para la interconexión de redes IP y 6LoWPAN. Requiere una interfaz ethernet en el lado IP y una interfaz 802.15.4 en el lado 6LoWPAN.

Los sensores Z1 de Zolertia están equipados con el microcontrolador MSP430F2617, de bajo consumo y con una interfaz radio compatible con el estándar IEEE 802.15.4, hecho para sistemas con una tasa de datos baja, bajo consumo y un rango de pocos metros. Para conectar dispositivos regidos por este estándar es necesaria una red IP punto a punto. Para hacer esto posible y además soportar IPv6 dentro de la red de sensores necesitamos implementar el protocolo 6LoWPAN. Este tipo de sensores puede trabajar con el sistema operativo Contiki, muy utilizado en el ámbito de IoT, el cual admite la comunicación IP mediante 6LoWPAN.

Para el intercambio de información entre los sensores y la Raspberry Pi se nos ha proporcionado varios programas desarrollados en un proyecto anterior. Estos permiten enviar la información recolectada por los sensores usando los protocolos de transporte UDP o TCP. La otra característica que diferencia los programas es el tiempo que tenemos la interfaz radio de los sensores encendida dependiendo si la queremos tener activa en todo momento o solo cuando se cumplan ciertos requisitos como tener un cierto volumen de información acumulada para enviar.

Por otro lado, para la Raspberry Pi, tenemos dos programas, ambos desarrollados en Python. Uno de ellos recibe los datos con el protocolo de transporte UDP y el otro recibe datos bajo el protocolo TCP. Ambos guardan la información que les llega en la base de datos.

Finalmente, con todos los dispositivos de la red programados se procederá a hacer los test de conectividad, comprobando que existe una conexión dentro de la red de sensores y que también podemos acceder a la red y a los datos desde una red IP externa. Una vez confirmada la conexión de todos los elementos deberemos garantizar que la información que recogen los sensores se almacena y se muestra correctamente.

El documento está estructurado como se expone a continuación. El primer bloque proporciona información teórica general. Se explica el concepto de internet de las cosas detalladamente, así como su arquitectura y los elementos necesarios para poder trabajar con este tipo de redes. También se explica conceptualmente los componentes y las tecnologías usadas para hacer posible este proyecto como lo son una red de sensores, 6LoWPAN o IPv6. En el segundo bloque se muestra el diseño y desarrollo de la parte experimental del proyecto. Se describen más detalladamente las principales características de los dispositivos utilizados y la configuración aplicada a estos con el fin de obtener el funcionamiento deseado de todo el conjunto. Por último, se presentan los resultados obtenidos de los test de conexión y la representación final de la información recaudada, junto con las conclusiones sobre estos resultados.

Para una información más detallada, en los anexos se adjunta información técnica sobre la instalación del software necesario en los dispositivos, así como su configuración y el código usado en los programas para el funcionamiento de los sensores, el servidor y las herramientas de visualización para mostrar los datos.

1. Fundamentos Teóricos

1.1 Internet de las cosas (IoT)

Internet de las cosas o IoT es un concepto que se refiere a la interconexión digital de todo tipo de objetos con internet, convirtiéndose así en objetos inteligentes, capaces de interactuar entre ellos y/o con las personas intercambiando información que recogen de su alrededor, procesándola, mostrándola e incluso reaccionando por sí mismos. [1] Esto abre un mundo de posibilidades incalculable. Todo pasaría a basarse en lo contextual, logrando una mayor independencia de los humanos y, por lo tanto, una mayor eficiencia y comodidad.

Gracias a que el internet de las cosas permite la conexión de cualquier tipo de objeto esta tecnología es aplicable a infinidad de campos como pueden ser la industria, el transporte (tanto personal como de pasajeros o mercancías), las ciudades inteligentes, la domótica, la salud, etc.

Según la ITU: “IoT es una red disponible en cualquier lugar, en cualquier momento, por cualquier persona y por cualquier cosa”. La conectividad adquirirá una dimensión totalmente nueva. Hoy, los usuarios pueden conectarse en cualquier momento y en cualquier lugar. La red global del futuro, no muy lejano, no sólo consistirá en seres humanos y dispositivos, sino también todo tipo de cosas inanimadas. Esta nueva “dimensión” es un gran avance en el campo de la tecnología de la información por lo que el IoT está considerado como la nueva revolución digital, después de la aparición del internet móvil.

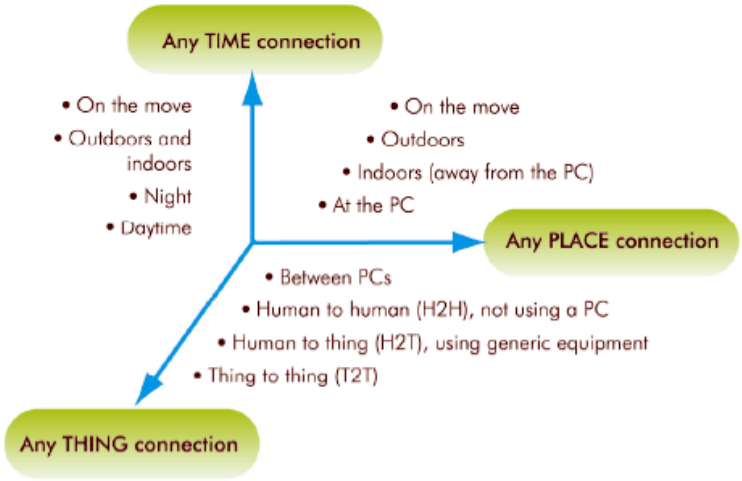


Fig. 1.1 “Dimensiones” que aporta el Internet de las cosas

Para hacernos una idea de la repercusión que tiene actualmente y tendrá el IoT a nivel mundial podemos consultar varios estudios sobre el crecimiento de la red y el aumento de dispositivos inteligentes que forman esta red. Según un informe

de IHS Markit elaborado en enero de 2017 se espera que el incremento en el número de dispositivos conectados a internet de las cosas sea del 15% año tras año, partiendo de un total de 20 mil millones en el año 2017 [2] [3].

Fig. 1.2 Previsión de IHS de dispositivos IoT conectados en 2017

Otra empresa de estudio e investigación, Gartner, realizó también un informe sobre el despliegue de dispositivos IoT con un resultado semejante al anterior estableciendo el número de dispositivos activos de IoT en unos 20,8 mil millones aproximadamente. Esta gran infraestructura supondrá una inversión mundial valorada en 250 billones de euros según Boston Consulting Group (BCG), una firma de consultoría estratégica. El 50% de este gasto será impulsado por la fabricación discreta, el transporte y la logística, y los servicios públicos. [4]

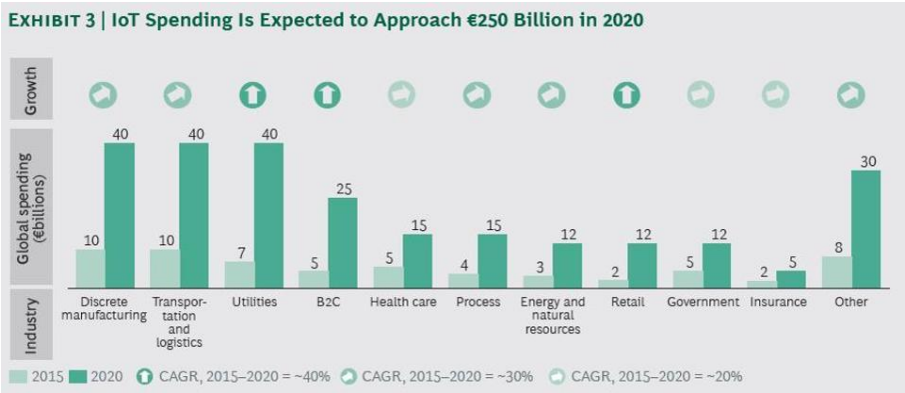


Fig. 1.3 Inversión mundial en IoT (Fuente: www.forbes.com)

1.1.1 Arquitectura del IoT

La base en la que se fundamenta el IoT es una red distribuida. Esto implica que vamos a tener un gran número de objetos, dispositivos y máquinas separadas físicamente y conectadas entre sí por una red de comunicaciones. Cada componente con su propio software y hardware. La arquitectura debe ser capaz de mostrar todos los componentes como un único sistema a los ojos de los usuarios y desarrolladores.

Como sugirió la ITU, esta infraestructura esencial se construirá alrededor de una arquitectura multicapa en la que los objetos inteligentes se utilizarán para ofrecer diferentes servicios a través de cuatro capas principales: una capa de dispositivo, una capa de red, una capa de soporte o procesado de datos y la capa de aplicación. [5]

- Capa de dispositivo: en este bloque encontramos los sensores, actuadores y *gateways* o interfaces de comunicación utilizados para recoger las lecturas del sensor y para su procesamiento adicional.
- Capa de red: proporciona las capacidades de transporte y de red necesarias para encaminar los datos de IoT a los lugares de procesamiento y también permitir la conectividad entre dispositivos de la capa más baja. Según el objetivo para el que se usan los dispositivos esta conexión tiene que ser segura, robusta y tolerante a fallos a fin de que recoja la información obtenida de los dispositivos y a la vez, se puedan gestionar. Existen multitud de protocolos y tecnologías de comunicación como por ejemplo WIFI, Bluetooth, ZigBee o MQTT.
- Capa de procesado de datos: es una capa de *middleware* que sirve para ocultar la complejidad de las capas inferiores a la capa de aplicación y proporciona servicios específicos y genéricos como almacenamiento en diferentes formas (sistemas de gestión de bases de datos o sistemas de computación en nube).
- Capa de aplicación: aplicaciones propiamente dichas, nativas o web, amigables para el ser humano, que permiten manejar y visualizar la información captada por la red de sensores. Algunas aplicaciones pueden tener la capacidad además de modificar los parámetros para cambiar el comportamiento de los sistemas.

1.2 Red Inalámbrica de Sensores (WSN, *Wireless Sensor Network*)

Las redes inalámbricas de sensores (WSN a partir de ahora) [6], se basan en dispositivos de bajo coste y consumo (nodos), distribuidos espacialmente, que son capaces de obtener información de su entorno, procesarla localmente, y comunicarla a través de enlaces inalámbricos hasta un nodo central de coordinación. Los nodos son unidades autónomas que constan de un microcontrolador, una fuente de energía (casi siempre una batería), un transceptor radio (RF) y un elemento sensor. El hecho de tener una batería como

fuente de alimentación obliga a que los nodos tengan un consumo energético bajo, característica definitoria de una red de sensores y en consecuencia del IoT.

1.2.1 Arquitectura y elementos de una red inalámbrica de sensores

Como ya hemos visto, las WSN se componen de pequeños dispositivos, autónomos y distribuidos geográficamente llamados sensores que tienen la capacidad de captación y almacenamiento de información, y añadiéndoles las llamadas motas conseguimos las capacidades de procesamiento y comunicación. Este envío de información fuera de la propia WSN se realiza vía *gateway*, a una estación base, donde la información pueda ser almacenada y procesada para finalmente darle valor.

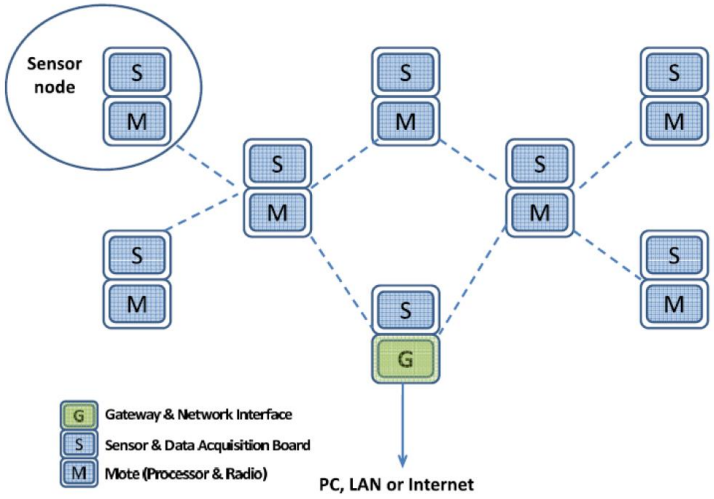


Fig. 1.4 Estructura de una red de sensores

Por lo tanto, en una WSN deberemos tener los elementos nombrados:

- Nodos sensor: se compone de una mota y una placa de sensores. Mota es la entidad compuesta de un procesador y los dispositivos de radio. La placa del sensor es una tarjeta de adquisición de datos conectado a la mota a través de un conector de expansión, que incluye un conjunto de sensores. Algunos modelos de nodo de sensores los incorporan en la propia mota (por ejemplo, algunas de la familia Zolertia). Algunos ejemplos de sensores son el de temperatura, o el acelerómetro.

- *Gateway* o pasarela (puerta de enlace): elemento destinado a la interconexión de redes con protocolos y arquitecturas diferentes a

todos los niveles de comunicación como lo son la red de sensores y una red de datos (por ejemplo, TCP/IP).

- Estación base: actúa como recolector final de datos basado en un ordenador común o un sistema integrado.

1.2.2 Características de las WSN

Las principales características de este tipo de redes son las siguientes:

- Despliegue Ad-hoc y a gran escala sobre la superficie.

- No se utiliza una estructura de red para operar ya que sus nodos pueden actuar de emisores, receptores o *router*.

- Se utilizan tecnologías inalámbricas de corto alcance; el encaminamiento entre dos nodos sin visión directa se realiza mediante comunicaciones de multiple salto.

- Topología dinámica: nodos auto-configurables, tolerancia a fallos y presentan una elevada fiabilidad.

- Dispositivos de pequeño tamaño y un reducido consumo de energía y memoria. Tienen una larga autonomía y pueden operar sin mantenimiento durante meses.

- Coste muy bajo.

1.3 IPv6

La proliferación de dispositivos que disponen de conexión a internet es constante y con la aparición del IoT este incremento se convierte en exponencial. Para poder conectarse a internet cada uno de estos dispositivos necesita una dirección IP única y dado que IPv4 usa direcciones de 32 bits “solo” puede proporcionar 4 294 967 296 (2^{32}) y este valor puede no ser suficiente en los próximos años.

IPv6 en cambio usa direcciones de 128 bits lo que da lugar a 340 sextillones de direcciones (2^{128}), teóricamente suficiente para las próximas décadas. IPv6 además aporta nuevas características en comparación al IPv4 que permiten configurar dispositivos de forma más sencilla y una mejora en movilidad, seguridad y calidad. Todo esto hace que IPv6 sea el protocolo de internet óptimo para el internet de las cosas.

IPv6 especifica un nuevo formato de paquete, diseñado para minimizar el procesamiento del encabezado del paquete en los *routers*. Debido a que los

encabezados de paquetes IPv4 y paquetes IPv6 son diferentes, los dos protocolos no son interoperables. Aunque, en la mayoría de los aspectos, IPv6 es una extensión de IPv4, por lo que gran parte de los protocolos de transporte y capa de aplicación necesitan pocos o ningún cambio para operar a través de IPv6. Esta no interoperabilidad y las nuevas características que aporta IPv6 descritas a continuación conllevan a que este sea el protocolo de internet estándar en un futuro no muy lejano, no solo en el ámbito de IoT, si no a nivel global habrá una migración total del IPv4 al IPv6.

Algunas de las características que provocarán este cambio son las siguientes:

- Capacidad extendida de direccionamiento: cambiando el prefijo anunciado por unos pocos *routers* es posible en principio reasignar la numeración de toda la red, ya que los identificadores de nodos (los 64 bits menos significativos de la dirección) pueden ser auto-configurados independientemente por un nodo.

- Multicast: la habilidad de enviar un paquete único a destinos múltiples es parte de la especificación base de IPv6. Esto es diferente a IPv4, donde es opcional. IPv6 no implementa broadcast, pero puede lograrse el mismo efecto enviando un paquete al grupo de *multicast* de enlace local “todos los nodos” (*all hosts*). Por lo tanto, no existe el concepto de una dirección de *broadcast* y así la dirección más alta de la red (la dirección de *broadcast* en una red IPv4) es considerada una dirección normal en IPv6.

- Autoconfiguración de direcciones libres de estado (SLAAV): Los nodos IPv6 pueden configurarse a sí mismos automáticamente cuando son conectados a una red ruteada en IPv6 usando los mensajes de descubrimiento de *routers* de ICMPv6. Si la autoconfiguración de direcciones libres de estado no es adecuada para una aplicación, es posible utilizar *Dynamic Host Configuration Protocol* para IPv6 (DHCPv6) o bien los nodos pueden ser configurados en forma estática.

- Seguridad de nivel de red obligatoria: *Internet Protocol Security* (IPsec), el protocolo para cifrado y autenticación IP forma parte integral del protocolo base en IPv6. El soporte IPsec es obligatorio en IPv6; a diferencia de IPv4, donde es opcional o fue un agregado posterior. Sin embargo, actualmente no se está usando normalmente IPsec excepto para asegurar el tráfico entre *routers* de BGP IPv6.

- Movilidad: A diferencia de IPv4 móvil (MIPv4), IPv6 móvil (MIPv6) evita el ruteo triangular reduciendo así la latencia de *handover* y por lo tanto haciéndolo tan eficiente como el IPv6 normal. Los *routers* IPv6 pueden soportar también Movilidad de Red (NEMO, por *Network Mobility*), que permite que redes enteras se muevan a nuevos puntos de conexión de *routers* sin reasignación de numeració

- Extras disponibles para dispositivos de bajo consumo: uno de los desarrollos principales de IPv6 aplicable a dispositivos o redes de bajo consumo es el 6LoWPAN, que permite acortar el tamaño de la dirección IPv6 para dispositivos de baja potencia, al tiempo que permite al *border router*, dispositivo que separa la red de sensores de internet, traducir estas direcciones comprimidas en direcciones IPv6 normales.

1.4 6LoWPAN

6LoWPAN (IPv6 *over Low power Wireless Personal Area Networks*) es un estándar que posibilita el uso de manera eficiente de IPv6 sobre redes basadas en el estándar IEEE 802.15.4. Hace posible que dispositivos de baja potencia y corto alcance, como los nodos de una red inalámbrica, puedan comunicarse directamente con otros dispositivos IP. [7]

Para poder usar IPv6 sobre el 802.15.4 pero, es necesario cumplir algunos requisitos como el aumento del tamaño de las direcciones IPv6 y la MTU a 1280 bytes. 6LowPAN soluciona estos inconvenientes introduciendo una capa de adaptación entre la red y la capa de enlace.

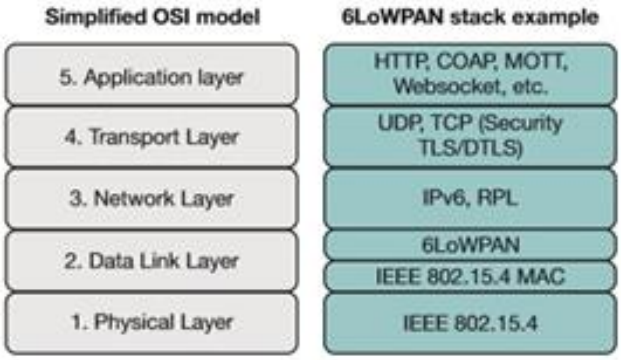


Fig. 1.5 Modelo OSI en 6LoWPAN

Las características principales del 6LoWPAN son:

- Anchos de banda bajos con velocidades de transferencia de hasta 250 Kbps, 40 Kbps y 20 Kbps previamente definidos por la capa física.

- Tamaño de paquete pequeño: 127 bytes en la capa física y en casos de sobrecarga de las capas anteriores, habrá sólo 81 bytes para los datos.
- Soporta direcciones MAC cortas de 16 bits o extendidas de 64 bits.

- Permite la configuración de topologías de malla, estrella o anillo.

- Soporta un gran número de dispositivos conectados a la red.

- Estos dispositivos funcionan con bajo consumo de energía.

- Los dispositivos en una red 6LoWPAN suelen ser largos períodos de tiempo en modo de espera para ahorrar energía.

1.5 Estándar IEEE 802.15.4

IEEE 802.15.4 es un estándar que define el nivel físico y el control de acceso al medio de redes inalámbricas de área personal con tasas bajas de transmisión de datos (*low-rate wireless personal area network*, LR-WPAN). Las características más importantes de este estándar son su flexibilidad de red, bajo coste y bajo consumo de energía.

Los estándares previos al 802.15.4 estaban destinados a aplicaciones con mayores requisitos en cuanto a ancho de banda se refiere. Algunos de ellos son IEEE 802.11 (WiFi), IEEE 502.15.1 (Bluetooth) o IEEE 802.15.3 (WPAN, *Wireless Personal Area Network*). Todos ellos comportan un gran consumo de energía y de ancho de banda o, en el caso del Bluetooth, no está diseñado para soportar comunicación entre redes de varios nodos.

El requisito fundamental del estándar 802.15.4 es un consumo de potencia extremadamente bajo. Su eficiencia energética reside en el uso de tramas "*beacon*" que permiten sincronizar los dispositivos de la red para que puedan permanecer en modo ahorro de energía el mayor tiempo posible. Esto supone una gran ventaja para el desarrollo de WSN que realicen tanto tareas de monitorización como de control. El principal inconveniente de este bajo consumo de energía es que el radio de cobertura se ve reducido, aun así, se ha llegado a establecer comunicaciones de hasta 75 m.

La tecnología inalámbrica basada en IEEE 802.15.4 está diseñada para utilizar bandas de frecuencia sin licencia. Pueden funcionar en las bandas de 868 MHz, 915 MHz y 2,4 GHz, aunque esta última es la más utilizada por las siguientes razones:

- Uso sin licencia disponible en todo el mundo.

- Tasa de datos más alta y mayor número de canales.

- Menor consumo de potencia debido a que se tarda menos tiempo en enviar y recibir porque la tasa de datos es más alta.

- Banda de frecuencias comúnmente empleada en el mercado (también la utilizan el bluetooth y el estándar IEEE 802.11).

La arquitectura definida en el estándar se divide en dos niveles: capa física, la cual actúa como interfaz con el medio físico de transmisión e intercambia bits de datos con el medio y con la capa superior, la subcapa MAC (junto con la subcapa LLC). El conjunto de subcapa MAC y subcapa LLC se conoce como capa de enlace de datos. Como indica su nombre MAC (*medium access control*) proporciona control de acceso a los canales compartidos y servicios para que los dispositivos puedan asociarse o desasociarse de la red.

Las técnicas de acceso al medio que ofrece este estándar para evitar que todos los nodos emitan al mismo tiempo son:

- CSMA-CA: cada nodo debe analizar la red antes de transmitir. Si la energía más alta se encuentra en un nivel específico el nodo espera al transceptor durante un tiempo al azar e intenta de nuevo.

- GTS (*Guaranteed Timeslot*): este sistema utiliza un nodo central (*PAN coordinator*) que da las franjas horarias de tiempo para cada uno de los nodos de modo que cualquier nodo sabe cuándo tiene que transmitir.

1.6 Sistemas operativos para redes WSN

Las necesidades que tiene un nodo de una WSN son totalmente distintas a las que pueda tener cualquier otro dispositivo como puede ser un PC, por lo tanto, estos nodos también deberán tener sus propios sistemas operativos. Los sistemas operativos para WSN son típicamente menos complejos que los de propósito general, tanto debido a los requisitos especiales de las aplicaciones en las que se usan, como a las restricciones de recursos encontrados en las plataformas hardware utilizadas. Además, las restricciones de los recursos en términos de memoria hacen imposible de implementar los mecanismos de memoria virtual.

1.6.1 Contiki

Es un sistema operativo de código abierto, altamente portable y multitarea, desarrollado para sistemas conectados en red, con memoria limitada, enfocado a dispositivos IoT inalámbricos de bajo consumo, desde microcontroladores de 8 bits a sistemas empotrados o embedidos, incluyendo nodos de red sensora. A pesar de incluir multitarea y una pila TCP/IP, Contiki sólo requiere varios kilobytes de código y unos cientos de bytes de RAM. Todo el código está desarrollado en el lenguaje de programación C. Contiki funciona en una gran variedad de plataformas, y es muy común en microcontroladores empotrados, como el MSP430 de Texas Instruments o el AVR de Atmel. [8]

Contiki consiste en un núcleo orientado a eventos para manejar la concurrencia. Hace uso de protohilos sobre los cuales los programas son cargados y descargados dinámicamente. También soporta comunicación entre procesos mediante paso de mensajes a través de eventos. Todos los procesos comparten una misma pila, lo que permite ahorrar memoria. Es la principal ventaja de Contiki.

Soporta las implementaciones de IPv4 e IPv6, incluyendo los nuevos estándares para redes de bajo consumo como 6LoWPAN, RPL y CoAP. También implementa una capa llamada RDC (*Radio Duty-Cycle*) que maneja el período de sueño del transceptor radio. Esta capa decide cuándo se transmitirán los paquetes y se asegurará de que los nodos están despiertos cuando se van a recibir los paquetes.

2. Diseño e implementación de la red IoT

El objetivo principal de este proyecto es la implementación de una aplicación IoT que monitorice cualquier tipo de señal, dentro de unos límites, basada en una red de sensores de bajo consumo. El sistema será lo más genérico posible con el objetivo de extender su uso al máximo de ámbitos posibles como pueden ser el industrial, ambiental, doméstico, médico, etc.

Según el uso que quiera darle el usuario a la aplicación se usará un tipo de sensores u otro. El único factor diferencial entre los diferentes tipos de señales que pueden captar los sensores es el ancho de banda que ocupa dicha señal. Dependiendo de este valor el sistema será capaz de coger muestras de forma adecuada y obtener unos resultados fiables.

Los datos que capten los sensores serán enviados a un elemento concentrador donde se tratará y almacenará toda la información para posterior o simultáneamente mostrarla mediante un dispositivo móvil o una web. El usuario podrá acceder a la red mediante IPv4 o IPv6.

Para el desarrollo de la aplicación necesitamos desplegar una red IP con un estándar de comunicación capaz de actuar sobre IPv6 y dispositivos con procesadores de bajo consumo. Para ello se ha escogido el 6LoWPAN, con el que podremos establecer una comunicación dentro de la WSN. Con el fin de poder interconectar la red de sensores con una red IP externa, dado que cada una de ellas tiene un protocolo de comunicación distinto, es necesario un dispositivo intermedio, un *router* de frontera o *border router*. En la figura 2.1 se puede ver un esquema de la infraestructura necesaria.

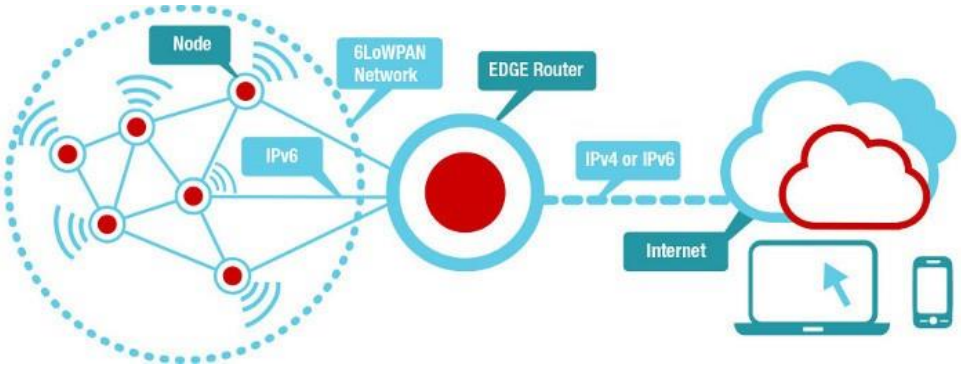


Fig. 2.1 Infraestructura de la red a implementar

Conforme las especificaciones descritas anteriormente y para satisfacerlas en el mayor grado posible, se ha hecho un estudio de búsqueda de varios dispositivos en el mercado para encontrar los más adecuados para crear el escenario. Las

marcas o dispositivos llevados a estudio han sido elegidos por impacto en el mercado y popularidad de los mismos en el ámbito del IoT.

En el Anexo VI se pueden ver las tablas resultado de este estudio con una gran variedad de dispositivos ordenados en 5 niveles situando en el nivel 1 los dispositivos con mejores características técnicas y en el nivel 5 los más modestos. Para hacer esta clasificación nos hemos basado en características de procesamiento como son frecuencia de CPU y memoria RAM. Otras características de diferenciación a tener en cuenta son la conectividad y el consumo.

En el nivel más alto encontramos, por ejemplo, un modelo de Intel, el Intel Joule y dos modelos de la marca UDOO, el Advanced y el Advanced Plus, entre otros. Estos tres serían los de gama más alta dentro de nuestro estudio. Tienen características parecidas en cuanto a procesamiento con unas velocidades de procesador que se sitúan entre los 2,2 y 2,4 GHz y 3-4 GB de memoria RAM. Estos 3 modelos también tienen un sistema gráfico semejante en cuanto a velocidad de GPU y también comparten aspectos de conectividad como puertos de video y audio y puertos USB 3.0. La diferencia entre el que se sitúa en lo más alto de la lista, el Intel Joule, y los modelos de UDOO es la conectividad inalámbrica. El Intel Joule tiene un módulo de WIFI ac integrado mientras que los modelos de UDOO no tienen un módulo WIFI integrado, aunque sí ofrecen la opción de conectar uno externo.

A medida que vamos bajando de nivel, situándonos en los niveles 2 y 3, encontramos otros fabricantes como Orange Pi, Banana Pi o Lattepanda, destacando este último ya que ofrece modelos que trabajan sobre Windows, cuando lo habitual es trabajar sobre Linux. En estos niveles inferiores, podemos encontrar valores de velocidad de procesamiento que oscilan entre 1 y 1,8 GHz y memorias RAM entre 1 y 3 GB. El dispositivo que se use como *router* y servidor a su vez es recomendable que se localice en uno de estos 3 niveles. De no ser así la capacidad de procesamiento podría no ser suficiente de cara a implementar un servicio estable.

En los últimos dos niveles encontramos dispositivos cada vez menos potentes en lo que a procesamiento se refiere, pero manteniendo, en muchos casos, muy buenas características en conectividad. Además, la pérdida de potencia de procesamiento respecto los niveles superiores, implica, normalmente, un menor consumo de energía, por lo que en estos dos niveles inferiores podemos encontrar el dispositivo que satisfaga nuestros requerimientos para llevar a cabo la función de nodo o sensor. Por ejemplo, tenemos el Olimex ESP32-EVB, con una buena relación entre capacidad de procesado y consumo y además tiene módulos integrados de WIFI b/g/n y Bluetooth Low Energy. Por otro lado, si el factor que nos interesa es el consumo energético encontramos los nodos Zolertia Z1, que aun siendo el último dispositivo de la lista por su baja capacidad de procesamiento puede ser muy útil dado su muy bajo consumo de energía.

Si nos fijamos en los resultados, podemos ver que cada fabricante intenta tener un modelo, como mínimo, en cada nivel con el objetivo de satisfacer cualquier necesidad por parte del usuario.

Además de las características físicas también es importante el software con el que podremos manejar cada dispositivo. Para ello se ha hecho otra tabla en la que poder ver de una forma sencilla y ágil los sistemas operativos que soporta cada dispositivo sometido a estudio.

Al elegir el dispositivo que se usará para llevar a cabo el proyecto también debemos tener en cuenta las propias necesidades de cada uno como el volumen de datos que se va a manejar o el presupuesto del que disponemos.

Por todo lo comentado hasta ahora, para implementar el escenario descrito arriba se han elegido los siguientes dispositivos:

- Raspberry Pi 3 Modelo B
- Nodos Zolertia Z1

La Raspberry Pi cumplirá con la función de *router* y de servidor y los nodos Zolertia llevarán a cabo la función de sensor. La primera satisfará nuestros requisitos en cuanto a capacidad de procesamiento y tratado de información y en cuanto a customización del software que correrá. Los nodos Zolertia cumplen con el requisito imprescindible en una WSN de bajo consumo debido al uso del microprocesador MSP430 y además tienen la capacidad de trabajar con el protocolo 6LoWPAN.

Nos hemos decantado por estos dispositivos también por la inmediata disponibilidad de ambos en el departamento – por parte de nuestro tutor.

2.1 Raspberry Pi 3 Modelo B

Raspberry Pi es un ordenador de placa reducida, computador de placa única o computador de placa simple (SBC). Soporta varios sistemas operativos pero el oficial es Raspbian, un sistema basado en Debian, optimizado para Raspberry.

Raspberry Pi es muy usado en el ámbito del IoT por sus características técnicas y su variedad de modelos, los cuales se adaptan perfectamente a cualquier tipo de requerimiento. En este proyecto se usa una Raspberry Pi 3 Modelo B con Raspbian. A continuación, podemos ver el aspecto de este modelo y algunas de sus características.

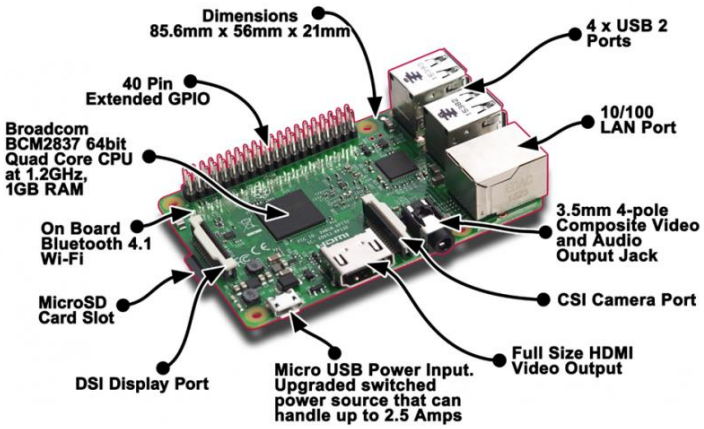


Fig. 2.2 Raspberry Pi 3 Modelo B

Tabla 2.1 Características principales de Raspberry Pi 3 Modelo B [9]

Chip	Broadcom BCM2837
CPU	Quad Core (ARM) - 1,2 GHz
RAM	1 GB DDR2
GPU	Broadcom VideoCore IV – 400 MHz
Conectividad	4 x USB 2.0 Ethernet 10/100 WIFI b/g/n Bluetooth 4.1

2.2 Zolertia Z1

Zolertia Z1 es un dispositivo inalámbrico de bajo consumo que permite conectar sensores, además de los que ya tiene incorporados y que está dotado con un microprocesador y un chip radio lo cual le proporciona la capacidad de procesar información y enviarla.

El microprocesador es el MSP430, de bajo consumo, y el transmisor radio es el CC2240, ambos componentes del fabricante Texas Instruments. Las motas Z1 cumplen con los estándar IEEE 802.15.4 y 6LoWPAN. Puede trabajar sobre varios sistemas operativos entre ellos TinyOS, RIOT y Contiki OS. A continuación, podemos ver su aspecto y sus características principales.

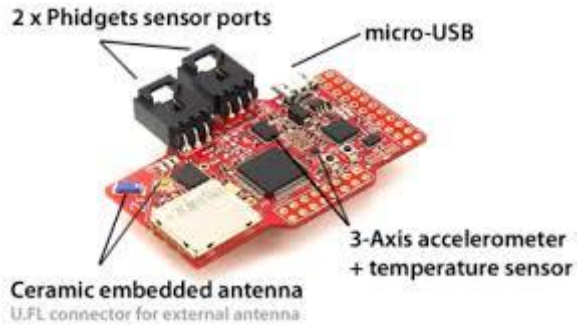


Fig. 2.3 Zolertia Z1

Tabla 2.2 Características principales de los nodos Zolertia Z1 [10]

Chip	TI MSP430F2617: 16-bit Ultra-Low-Power
CPU	16 MHz
RAM	8 KB
Conectividad radio	TI CC2420: 2.4 GHz IEEE 802.15.4 + 6LowPAN
Conectividad (pins)	Conector de expansión de 52 pins
Fuente de energía	Micro USB 2 x AA / AAA
Consumo	365 uA (activo) 0,5 uA (stand by) 0,1 uA (apagado)

2.3 Diseño funcional

Como ya se ha detallado anteriormente el objetivo principal del proyecto es monitorizar cualquier tipo de señal en tiempo real o en modo *data logger* para su posterior visualización. Las motas Z1 de Zolertia lo hacen posible gracias a sus 52 pines disponibles para poder conectar cualquier tipo de sensor externo, digital o analógico, además de los que tiene incorporados de fábrica, un sensor de temperatura, de nivel de batería y un acelerómetro. La única restricción que tiene el sistema es el ancho de banda de la señal que se va a medir. Este ancho de banda no puede ser superior al que admite el convertidor analógico-digital (ADC, *analog to digital converter*) del microprocesador. De ser así, perderíamos información y el sistema no sería fiable.

Para la implementación y test del sistema se usarán los sensores internos de las motas Z1 de Zolertia, pero en el desarrollo de los programas para los nodos se contemplará la opción de trabajar con sensores externos cuyas medidas sean la entrada de uno de los ADC del microprocesador. En el caso de trabajar con

sensores externos, Zolertia proporciona una lista [13] en la que se muestran varios tipos de sensores, con varios modelos dentro de cada tipo, que ya están pre-programados y listos para usar. En el apartado de programación de los nodos se explicará más detalladamente el procedimiento a seguir si nos encontramos en esta situación.

De este modo, mediremos temperatura ambiente y nivel de batería de cada nodo. Cada uno de los sensores usados para conformar la red tomaran dichos valores cada cierto tiempo y mandaran las lecturas a la Raspberry Pi. El protocolo de transporte que se utilizará es UDP, más adecuado para este tipo de aplicaciones que TCP por razones que se detallarán más adelante.

Para escuchar el canal y recibir los datos necesitaremos crear un servidor UDP en la Raspberry Pi. La información recibida será procesada y almacenada en una base de datos local. Por último, la información guardada será recuperada y mostrada de una manera gráfica mediante una web y una aplicación Android. Además, tendremos la opción extra con ambas herramientas de obtener datos pasados, aunque no de manera gráfica, ya que toda la información se almacenará en archivos de tipo CSV a parte de hacerlo en la base de datos.

En la siguiente figura podemos ver representado el flujo de datos desde que los sensores toman medidas hasta que el usuario las consulta.

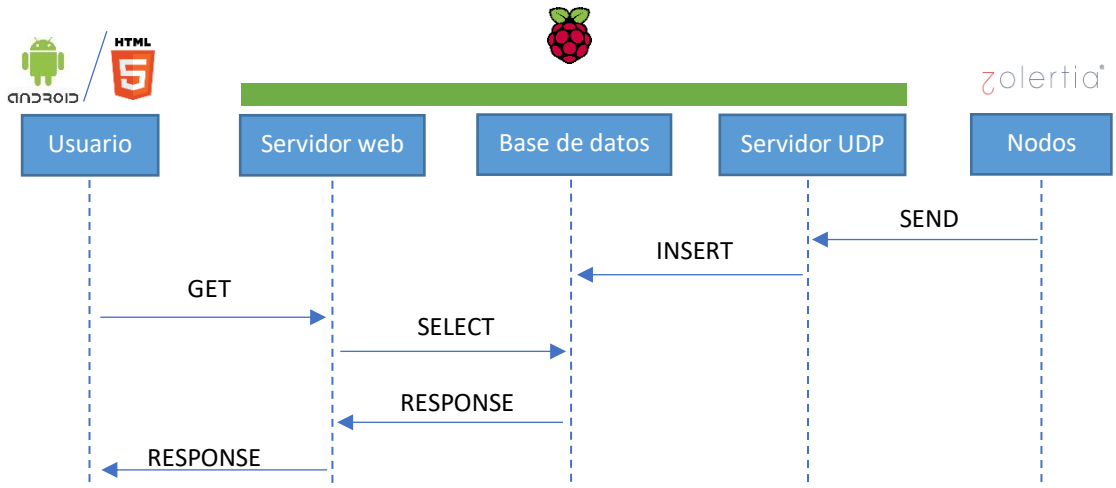


Fig. 2.4 Flujo de información en el sistema

Para implementar el flujo descrito se han desarrollado varios programas y *scripts*, además de instalar el software necesario para cada una de las funcionalidades. Para desarrollar la funcionalidad de *router* es necesario instalar CETIC 6LBR, una plataforma preparada para la interconexión de redes IP y 6LoWPAN. Requiere una interfaz ethernet en el lado IP y una interfaz 802.15.4 en el lado 6LoWPAN. A parte de *router*, la Raspberry Pi también hará la función de servidor

UDP, encargado de recibir y procesar los mensajes procedentes de los nodos. Para esta función se ha desarrollado un programa en Python. Por otro lado, gracias a la capacidad de la Raspberry Pi, también se ha instalado en ella una base de datos MySQL para almacenar los datos y un servidor web para proporcionar los servicios web necesarios al usuario y poder consultar los datos. Para este último propósito se han desarrollado una aplicación Android y una página web.

En lo que respecta a los sensores, se trabajará con Contiki OS puesto que los nodos Z1 de Zolertia soportan este sistema operativo y este, a su vez, está capacitado para trabajar con 6LoWPAN. El programa que se usará en los sensores está escrito en C y es la evolución de una versión que se desarrolló en otro proyecto.

La instalación, configuración y desarrollo del software descrito se presenta en las siguientes secciones.

2.4 *Router* de frontera o *border router*

El *border router* (BR) es el elemento intermedio entre la red de sensores y cualquier otra red IP externa (IPv4 o IPv6) que permite la interconexión entre ambas. Por definición, el *border router* nos permite conectar a internet cualquier red que no soporte una conexión punto a punto de manera sencilla. La red de sensores en cuestión necesita poder estar conectada vía IPv4/IPv6 a internet.

Una solución para implementar en la Raspberry Pi y hacer que esta funcione como *border router* es CETIC 6LBR. Se comporta perfectamente en dispositivos de bajo coste y, además, es ideal ya que puede funcionar en cualquier dispositivo remoto bajo el sistema operativo Linux, como en este caso en una Raspberry Pi que trabaje con Raspbian.

CETIC 6LBR consiste en una solución de código abierto que nos permite implementar un *border router* usando el estándar 6LoWPAN y el protocolo RPL, basada en el sistema operativo Contiki. 6LoWPAN nos permite interconectar una red WSN, basada en 802.15.4 y 6LoWPAN, con una red IPv4/IPv6, sobre ethernet. A partir de la versión 1.4, 6LBR incluye NAT64, que permite acceder al *border router* y nodos usando IPv4. Anteriormente a esta versión solo soportaba comunicaciones en las que se trabajaba con redes IPv6. 6LBR utiliza la capa 2 (Ethernet y 802.15.4) para el enlace de datos, la capa 3 (IPv6/6LoWPAN) en lo que se refiere a la capa de red y, por último, la capa 4 (ICMPv6/RPL) para enrutar los paquetes de datos.

2.4.1 Modos de operación

6LBR puede ser usado como *router* o como *bridge*. Cuando hablamos del primer caso, se separan las redes IPv6 y 6LoWPAN en diferentes subredes. Podemos

ver como en la figura 2.5, la red IPv6 tiene bbbb::/64 como prefijo mientras que la red 6LoWPAN tiene aaaa::/64 como prefijo. En cambio, sí se está utilizando en modo *bridge*, se considerará a ambas redes como una única subred común como podemos ver en la figura 2.6.

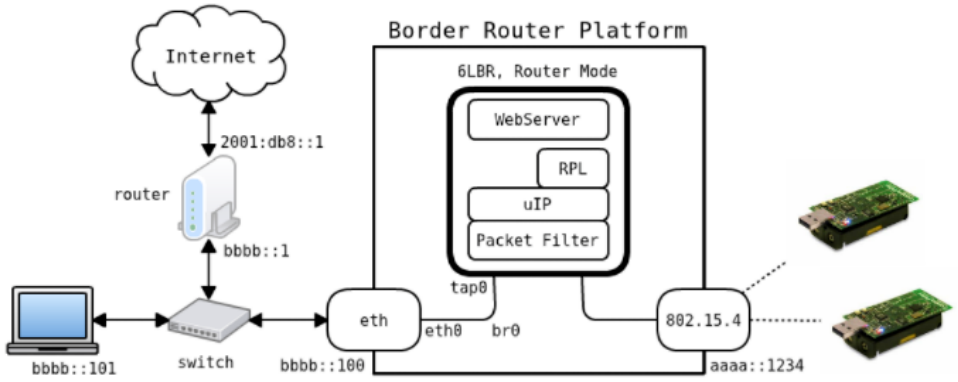


Fig. 2.5 6LBR en modo *router* (Fuente: Github 6LBR wiki)

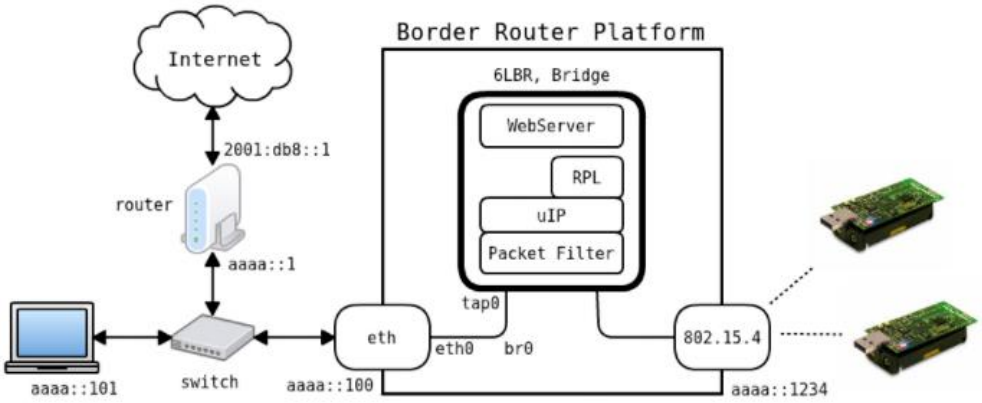


Fig. 2.6 6LBR en modo *bridge* (Fuente: Github 6LBR wiki)

La configuración de 6LBR puede modificarse para que pueda soportar varios tipos de topologías de redes distintas, concretamente funciona en tres modos de categorías: *bridge*, *router* y *transparent bridge*. Los modos principales son los siguientes:

- *Router mode*

  En este modo, 6LBR actúa como como un *router* con todas sus prestaciones y características, interconectando dos subredes IPv6. Respecto a la subred WSN, es gestionada por el protocolo RPL, mientras que la subred de Ethernet es gestionada por IPv6 NDP como podemos ver en la figura 2.7. En este modo, la subred WSN es aislada de otras redes, de esta manera los nodos pueden ser identificados con claridad.

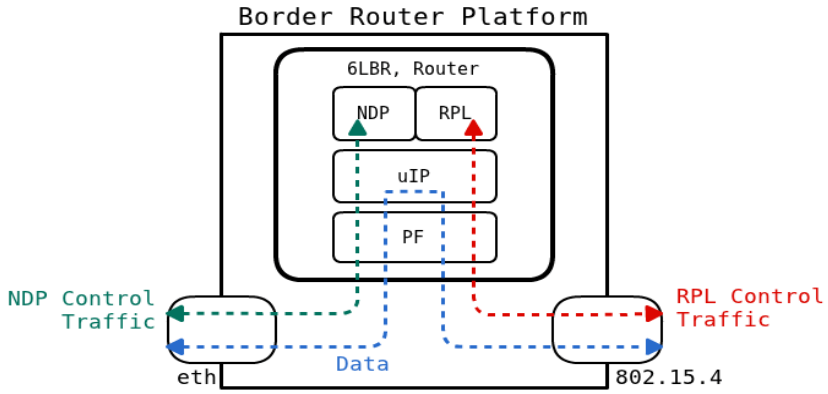


Fig. 2.7 6LBR en modo *router* (Fuente: Github 6LBR wiki)

- *Smart bridge mode*

En este modo, 6LBR permite interconectar redes que trabajan en IPv6 usando RPL basados en mallas WSN. En este modo, el *border router* actúa como un proxy NDP en la parte de Ethernet y usa los parámetros del protocolo NDP para configurar la red de sensores tal y como podemos observar en la figura 2.8.

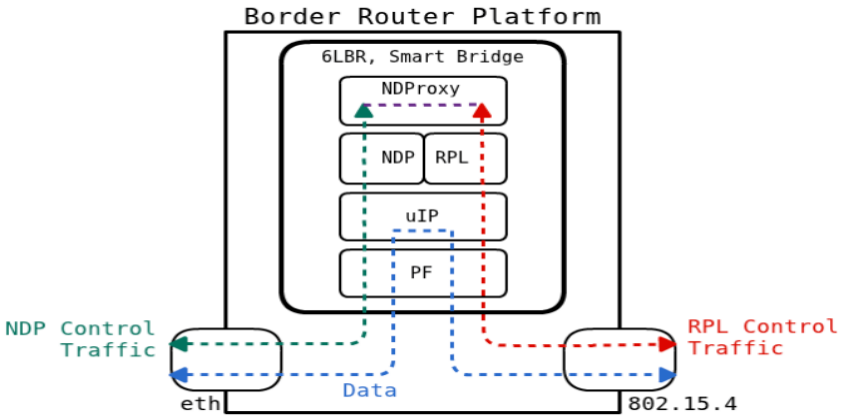


Fig. 2.8 6LBR en modo *smart bridge* (Fuente: Github 6LBR wiki)

- *Transparent bridge modes*

En estos modos, 6LBR actúa como un puente independiente (*standalone bridge*), proporcionando capacidades básicas de commutación. Todos los paquetes que vayan dirigidos a la interfaz 802.15.4 o los paquetes multicast que tengan como origen la interficie ethernet, serán enviados hacia el segmento de la red de sensores. De la misma forma, todos los

paquetes que vayan dirigidos a la interfaz Ethernet o paquetes multicast que tengan como origen la interficie 802.15.4 serán enviados al segmento Ethernet. El funcionamiento de este modo se puede observar en la figura 2.9.

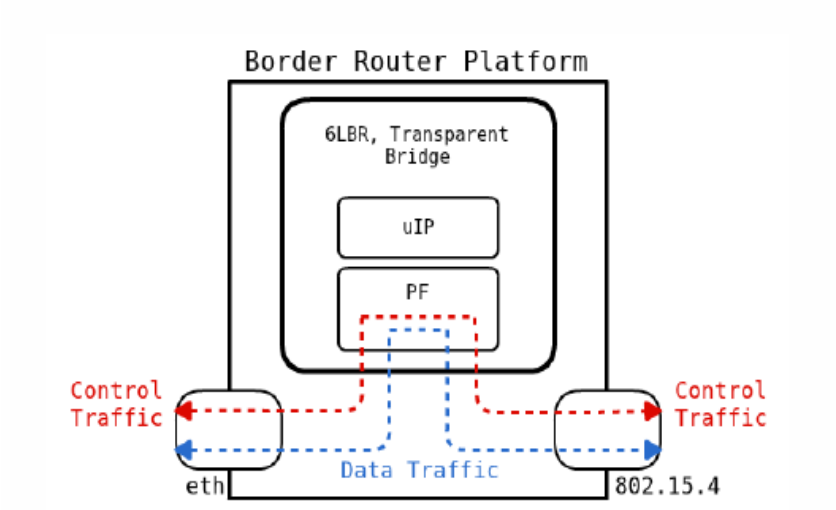


Fig. 2.9 6LBR en modo *transparent bridge* (Fuente: Github 6LBR wiki)

2.4.2 Configuración de la interfaz

Cuando usamos 6LBR en plataformas bajo Linux, es como si estuvieramos una máquina virtual funcionando en una máquina remota que trabaja con Linux. Para poder enlazar la aplicación 6LBR a la red de área local (LAN), tenemos tres opciones distintas: Raw ethernet, bridge mode y routing mode.

Esta configuración es totalmente distinta a la explicada anteriormente. Esta configuración describe como se comunica 6LBR y la plataforma en la que se aloja Linux (Host Linux), en este caso una Raspberry Pi. Se puede encontrar como “Interface Configuration” en la documentación de 6LBR.

- *Raw Ethernet*

  En este modo, el *Linux Host* no será capaz de comunicarse directamente con el 6LBR y la red WSN. 6LBR en este caso, utilizará la interfaz Ethernet (eth0) para mandar y recibir los paquetes tal y como podemos observar en la figura 2.10.

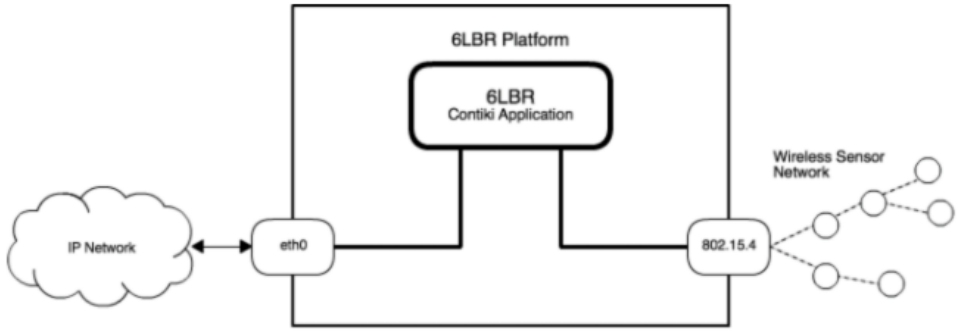


Fig. 2.10 6LBR interfaz modo *raw ethernet* (Fuente: Github 6LBR wiki)

Para llevar a cabo la configuracion en este modo solo tendríamos que cambiar las siguientes líneas en el archivo 6lbr.conf:

```
RAW_ETH=1
BRIDGE=0
DEV_ETH=eth0
```

- *Bridge Mode*

En este modo, el *Linux Host* no podrá comunicarse con la aplicación 6LBR. Para ello 6LBR creará una interfaz Ethernet virtual, a la que normalmente se le asigna el nombre de tap0 como podemos ver en la figura 2.11. Para proporcionar conectividad se deberá enlazar la nueva interfaz con nuestra interfaz anterior, de manera que tengamos conectividad. Una vez conectadas, esta nueva interfaz reemplazará la antigua en todos los aspectos.

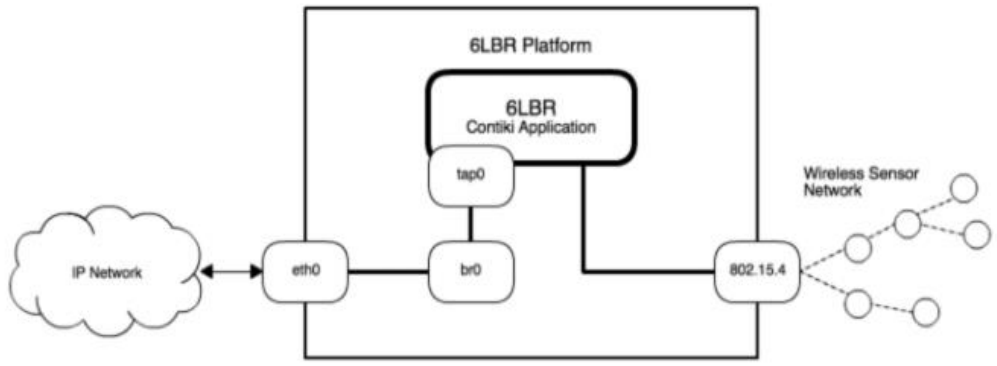


Fig. 2.11 6LBR interfaz en modo *bridge* (Fuente: Github 6LBR wiki)

Para llevar a cabo la configuración en este modo solo tendríamos que cambiar las siguientes líneas en el archivo 6lbr.conf:

```
RAW_ETH=0
BRIDGE=1
CREATE_BRIDGE=0
DEV_ETH=eth0
DEV_BRIDGE=br0
DEV_TAP=tap0
```

Por otro lado, para crear la interfaz “br0” tendremos que tener un archivo en la ruta /etc/network/interfaces con la siguiente configuración y los siguientes comandos:

```
...

iface eth0 inet static
address 0.0.0.0

auto br0
iface br0 inet dhcp
    bridge_ports eth0
    bridge_stp off
    up echo 0 > /sys/devices/virtual/net/br0/bridge/multicast_snooping
    post-up ip link set br0 address `ip link show eth0 | grep ether | awk '{print $2}'`

...
```

Para que el puente que hemos creado entre las dos interfaces funcione de manera idónea, los drivers de la interfaz Ethernet tendran que poder soportar el modo promiscuo, es decir, que capture todo el tráfico que circule por el enlace.

- *Routing mode*

En este modo, en lugar de crear un puente para conectarnos a la interficie tap0 y la interfaz Ethernet, el *Linux Host* conecta las dos interfaces, estén a nivel IP o a nivel Ethernet. Este modo es útil cuando trabajamos con muchas interfaces, cuando el enrutamiento es muy complejo o directamente cuando los drivers de nuestro puerto de entrada no soportan el modo promiscuo. El esquema de este modo lo podemos ver esquematizado en la figura 2.12.

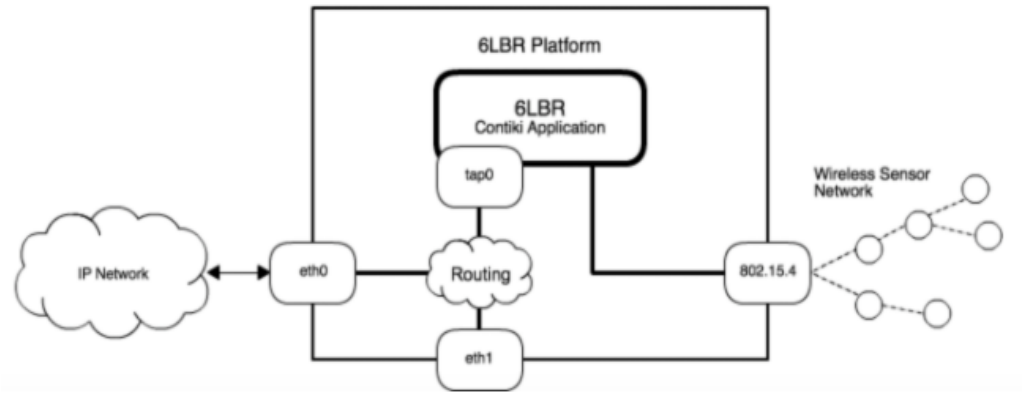


Fig. 2.12 6LBR interfaz en *routing mode* (Fuente: Github 6LBR wiki)

Para llevar a cabo la configuración en este modo solo tendriamos que cambiar las siguientes líneas en el archivo 6lbr.conf:

```
RAW_ETH=0
BRIDGE=0
DEV_ETH=eth0
DEV_TAP=tap0
```

Por otro lado añadir las siguientes dos líneas en /etc/sysctl.conf:

```
net.ipv4.ip_forward = 1
net.ipv6.conf.all.forwarding=1
```

2.4.3 Instalación y configuración del 6LBR

El sistema operativo Raspbian ha sido probado y su compatbilidad con 6LBR y Raspberry Pi a quedado confirmada infinidad de veces, por ello no se debería de tener ningun problema a la hora de instalarlo en el dispositivo. Asumiendo que tenemos instalado Raspbian en la tarjeta SD de la Raspberry Pi, los pasos para proceder con la instalación del 6LBR son los siguientes:

Teniendo en cuenta que Raspberry no tiene una interfaz 802.15.4 para comunicarse con una red WSN, uno de nuestros nodos Z1 tendra que usarse como interfaz. En el nodo que queramos usar como interfaz, tendremos que cargar un firmware llamado slip-radio, que se encarga de la comunicación entre los procesos Linux de 6LBR y el estandar 802.15.4 radio, de esta manera los nodos se podran comunicar con la máquina donde tengamos alojado nuestro servidor y posteriormente poder tratar los datos. Por defecto, la capa MAC esta alojada dentro de los procesos 6LBR y la capa RDC esta alojada dentro del slip-radio. La MAC por defecto es CSMA y la de RDC es nullrdc.

En nuestro caso usamos una versión de desarrollador directamente desde su rama de github, ya que nos proporciona soporte NAT64 y soluciona algunos problemas de compatibilidad. 6LBR se ha configurado en modo *router* y respecto a la configuración de la interfaz, la escogida es la de modo *bridge*. La elección de estas dos configuraciones, se debe a que la red WSN y la red que está cableada estaran aisladas.

Para empezar la instalación desde nuestra Raspberry Pi, necesitaremos la instalación del paquete bridge utils para poder utilizar el modo *tap-bridge*. Para ello ejecutaremos el siguiente comando para instalar dichas dependencias.

```
apt-get install bridge-utils
```

Llegados a este punto, podemos instalar 6LBR de dos modos. El primero de todos es lo que se llama *Binary Package Installation*, que normalmente se suele usar si queremos instalar una versión en concreto. Si queremos usar esta primera opción tendremos que ejectuar lo siguiente:

```
wget https://raw.github.com/wiki/cetic/6lbr/releases/rpi/cetic-6lbr_1.4.0_armhf.deb
```

Y una vez descargado, nos situaremos en el ruta donde se haya descargado y ejecutaremos lo siguiente:

```
dkpg -i cetic-6lbr_1.3.3_armfh.deb
```

Por otro lado, si queremos usar la segunda opción, haremos una instalación donde descargaremos directamente des de la fuente del código. Con esta opción nos aseguramos que estamos instalando la última versión, ya que estaremos descargando todo aquello que esté en la rama master de github. Por otro lado, también podremos descargarnos la rama *develop*, donde puede haber partes que esten en proceso de creación o en cambios constantes y por ello puede ser más inestable.

Para proceder con la instalación desde las fuentes, instalaremos primero de todo dependencias que necesitaremos, git y libncurses, ya que serán esenciales para la instalación. Ejecutaremos el siguiente comando:

```
apt-get install git build-essential libncurses5-dev
```

Una vez hayamos instalado las dependencias procederemos a descargarnos la última versión de 6LBR:

```
git clone https://github.com/cetic/6lbr
cd 6lbr
git submodule update --init --recursive
```

Los siguientes comandos compilan y completan la instalación del 6LBR. Primero de todo ejecutaremos en el directorio principal 6LBR los siguientes comandos:

```
make all
make plugins
make tools
```

Por último, ejecutamos como administrador los siguientes comandos:

```
make install
make plugins-install
update-rc.d 6lbr defaults
```

Una vez esté instalado, el siguiente paso es la configuración del 6LBR. El archivo para modificar la configuración a nuestro gusto se encuentra en /etc/6lbr/6lbr.conf como ya se ha indicado en apartados anteriores. La configuración que usaremos, será la de modo *router* para 6LBR y la de modo *bridge* para la configuración de la interfaz.

```
MODE=ROUTER
BRIDGE=1
CREATE_BRIDGE=0
DEV_ETH=eth0
DEV_BRIDGE=br0
DEV_TAP=tap0
RAW_ETH_FCS=0
DEV_RADIO=/dev/ttyUSB0
BAUDRATE=115200
```

Por otro lado, también es necesario crear la interfaz en modo bridge. El contenido del fichero de configuración para la interfaz se podrá encontrar en la ruta (/etc/network/interfaces) donde lo modificaremos añadiendo las siguientes líneas de texto.

```
iface eth0 inet static
address 0.0.0.0
auto br0
iface br0 inet dhcp
bridge_ports eth0
bridge_stp off
up echo 0 > /sys/devices/virtual/net/br0/bridge/multicast_snooping
post-up ip link set br0 address `ip link show eth0 | grep ether | awk '{print
$2}'`
```

Cada vez que se haga un cambio en cualquiera de los dos ficheros de configuración, será necesario reiniciar el servicio.

Para iniciar o reiniciar el servicio debemos ejecutar respectivamente:

```
service 6lbr start
```

```
service 6lbr restart
```

Si queremos ver el log en tiempo real, por si nos encontramos con algún problema a la hora de iniciar el servicio, ejecutaremos el siguiente comando:

```
tail -f/var/log/6lbr.log
```

Una vez hayamos iniciado el servicio, iremos a nuestro navegador e iremos a la dirección ipv6 bbbb::100, la cual está configurada por defecto para que nos muestre el servidor web o *webserver* de 6LBR, tal y como podemos observar en la figura 2.13.

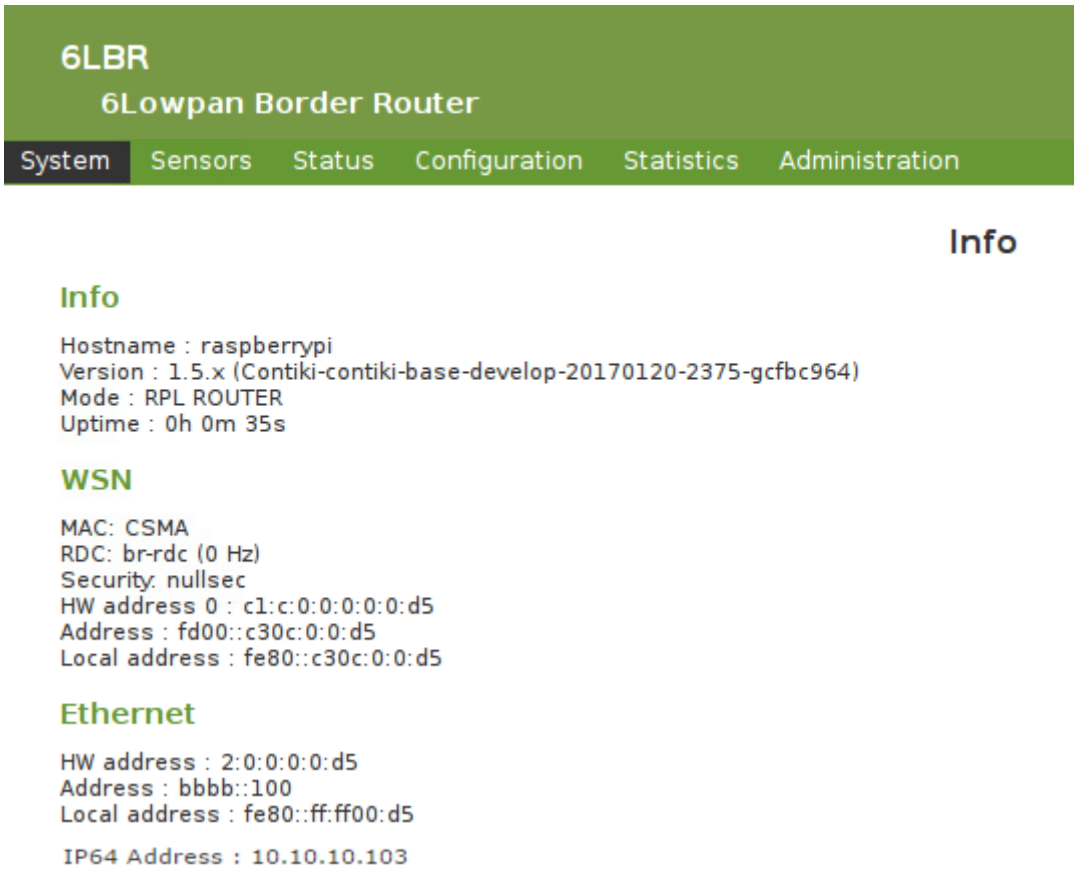


Fig. 2.13 Servidor web de 6LBR

El *webserver* nos permite configurar el *Border Router*, además de poder ver el estado actual de la red de sensores que tengamos desplegada en ese momento, el tráfico que tenemos activo, cambiar la configuración de la red, etc.

Por defecto, 6LBR envía periódicamente mensajes de advertimiento (RA) en la subred Ethernet que contienen el prefijo y la ruta de acceso a la subred WSN. Como la Raspberry Pi se comunica con la red WSN, se configuró de tal manera

que aceptara este tipo de mensaje y que estableciera las rutas a través de la red WSN con el siguiente comando:

```
sysctl -w net.ipv6.conf.br0.accept_ra=2
```

2.5 Programación de nodos

Como ya se ha mencionado anteriormente, las motas Z1 son capaces de medir varios tipos de señales o parámetros gracias a la posibilidad de conectar sensores externos a sus pins a parte de los sensores que lleva integrados la propia mota. Por lo tanto, el programa que se ha desarrollado para los nodos consiste en medir periódicamente todos los sensores disponibles en la mota y mandarlos en un único paquete de información a la Raspberry Pi.

Esta tarea dependerá de dos parámetros, el intervalo entre medidas (t) y el número de medidas necesarias para mandar un paquete (l), los cuales se pueden modificar en el código fuente que se subirá a los nodos. Esto es, el nodo medirá cada t segundos y mandará un paquete a la Raspberry Pi cuando haya leído el valor de los sensores l veces. En el código que veremos a continuación la variable t será *time* y l será *limit* y tendrán los valores de 5 y 1 respectivamente. Hasta que el número de medidas no sea el deseado el nodo guarda las medidas anteriores en una lista, deja pasar un intervalo t y vuelve a medir hasta cumplir con el valor de l establecido. En la figura 2.14 podemos ver el diagrama de flujo del comportamiento del nodo.

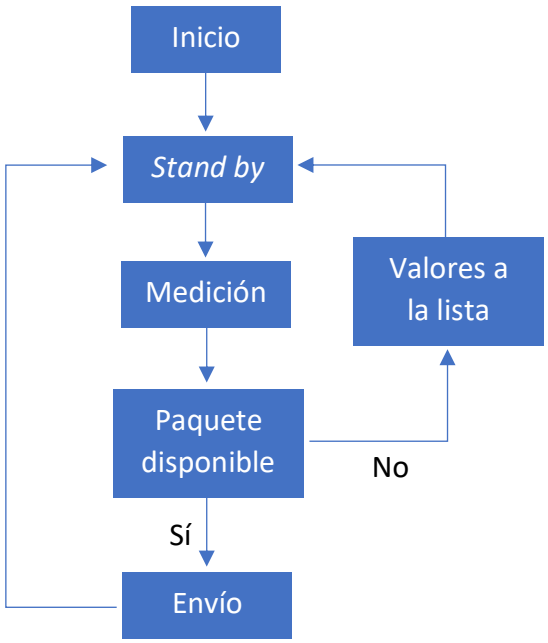


Fig. 2.14 Flujo de funcionamiento de los nodos

Para guardar las lecturas de los nodos se usa una estructura de datos con el siguiente aspecto:

Tabla 2.3 Estructura de datos

Datos	
counter	uint16_t
temperature	uint16_t
battery	uint16_t
external 1	uint16_t
external 2	uint16_t

El valor de *counter* indicará el total acumulado de lecturas del nodo y los valores de *temperature* y *battery* guardarán las medidas tomadas de temperatura y batería respectivamente. Los campos *external1* y *external2* serán aquellos en los que guardaremos las medidas de los sensores externos. Estos no siempre tendrán datos ya que podemos o no tener conectados sensores externos a la mota.

En el Anexo II se puede encontrar todo el código desarrollado para programar los sensores y a continuación se explicarán las funciones principales de las que consta dicho código.

La función *log_packet*() lee los valores de todos los sensores y los guarda en la lista a la espera de más medidas o de la formación del paquete para enviar los datos.

```
/* This function read values the sensors and store this data in the linked list*/
static void
log_packet(void)
{
        uint32_t aux;
        uint32_t aux1;
        uint32_t aux2;
        counter++;
        struct msg_ll *e;
        e =  malloc(sizeof(struct msg_ll));
        if(e != NULL) {
                e->msg.counter = counter;
                e->msg.temperature = tmp102_read_temp_x100();
                /* Convert the battery reading from ADC units to mV (powered over USB)*/
                aux = battery_sensor.value(0);
                aux *= 5000;
                aux /= 4095;
                e->msg.battery = aux;
                //PRINTF("%s",aux);
                /* External sensor 1 read from ADC */
                aux1 = battery_sensor.value(0);
                aux1 *= 5000;
                aux1 /= 4095;
                e->msg.external1 = aux1;
                //PRINTF("%s",aux1);
                /* External sensor 2 read from ADC */
                aux2 = battery_sensor.value(0);
                aux2 *= 5000;
                aux2 /= 4095;
                e->msg.external2 = aux2;
                //PRINTF("%s",aux2);

                list_add(measures_list, e);
        }
        else {printf("No memory available");}
}
```

Como podemos observar el valor de la temperatura se trata de una manera distinta al resto. Esto se debe a que este sensor es digital, por lo que no necesita pasar por la etapa de conversión analógico a digital. Los demás sensores, por el contrario, sí que necesitan esta conversión. En este caso, la salida de los sensores estará conectada a la entrada de uno de los ADC del microprocesador.

Cuando las condiciones para enviar un paquete se cumplen, es decir, el número de lecturas es igual a l o *limit*, se procede a llamar la función *send_packet*(). Esta función coge los parámetros t y l para conocer en todo momento, por parte del usuario, cuáles son sus valores y a continuación concatena todas las medidas de los sensores guardadas en la lista. Toda esta cadena se introduce en un buffer y se envía a la Raspberry Pi.

```
static void
send_packet(void)
{
        char buffer_ll[2*sizeof(uint16_t)+limit*sizeof(struct my_msg_t)];
        char *buffer_llPtr = buffer_ll;
        /*Copy the values of the current parameters in the buffer*/
        memcpy(buffer_llPtr, &time, sizeof(uint16_t));
        memcpy(buffer_llPtr+sizeof(uint16_t), &limit, sizeof(uint16_t));
        /*Copy sensor readings in the buffer*/
        int j = 0;
        struct msg_ll *s;
        for(s = list_head(measures_list); s != NULL; s = list_item_next(s)) {
                memcpy(buffer_llPtr+2*sizeof(uint16_t)+j*sizeof(struct my_msg_t), &s->msg,
sizeof(struct my_msg_t));
                j++;
        }
        uip_udp_packet_sendto(client_conn, &buffer_ll, sizeof(buffer_ll),
                        &server_ipaddr, UIP_HTONS(UDP_SERVER_PORT));
}
```

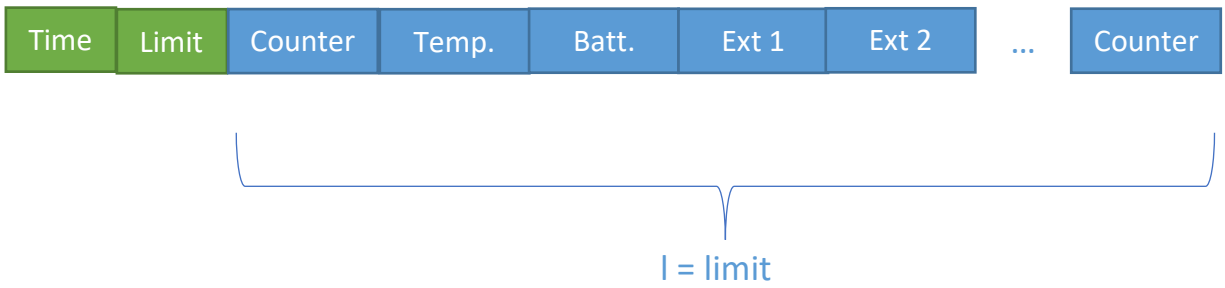


Fig. 2.15 Formato de un paquete de datos

Para una posible mejora en eficiencia y ahorro de energía y dependiendo del uso que se le quiera dar a la aplicación se puede añadir un parámetro más de control, el cual se podría enviar al nodo y así cambiar su valor desde la web o la aplicación de Android. Se trataría de una variable de control *on-off* con la que podríamos apagar el nodo parcialmente cuando no se precise medir y volver a despertarlo cuando fuera necesario.

Para el envío de paquetes se usará el protocolo UDP mucho más apropiado que TCP para aplicaciones en tiempo real. TCP aporta más control sobre los paquetes asegurándose que llegan al destino y reenviándolos en el caso de pérdidas. Esto ralentizaría mucho el proceso de transmisión de datos y en este caso es preferible la velocidad que puede aportar UDP asumiendo que algún paquete se puede perder lo que no supone un gran problema, ya que estaremos enviando más de un paquete por segundo y la pérdida de información es ínfima.

2.6 Servidor UDP

La función que hará este servidor será la de recibir los datos que envían los nodos y almacenarlos en una base de datos. Para ello se creará un servidor UDP utilizando Python como lenguaje de desarrollo. La elección de este tipo de lenguaje entre otros fue por su ventaja a la hora de tratar con todo tipo de datos, su capacidad de optimización de código y su sencillez.

Cuando el servidor empieza a funcionar, lo primero de todo es crear un socket enlazado a un puerto que escuchará todas las posibles conexiones que puedan aparecer de cualquier nodo. Cuando llega un nuevo mensaje, el servidor recibe los datos de los nodos y los procesa. Considerando que parte de los datos que recibe el servidor no es útil, necesitaremos separar los datos útiles y posteriormente proceder a su guardado. Las medidas y los datos de cada uno de los nodos estarán alojados en dos tablas distintas de la misma base de datos. En la figura 2.16 podemos ver el diagrama de flujo del servidor UDP.

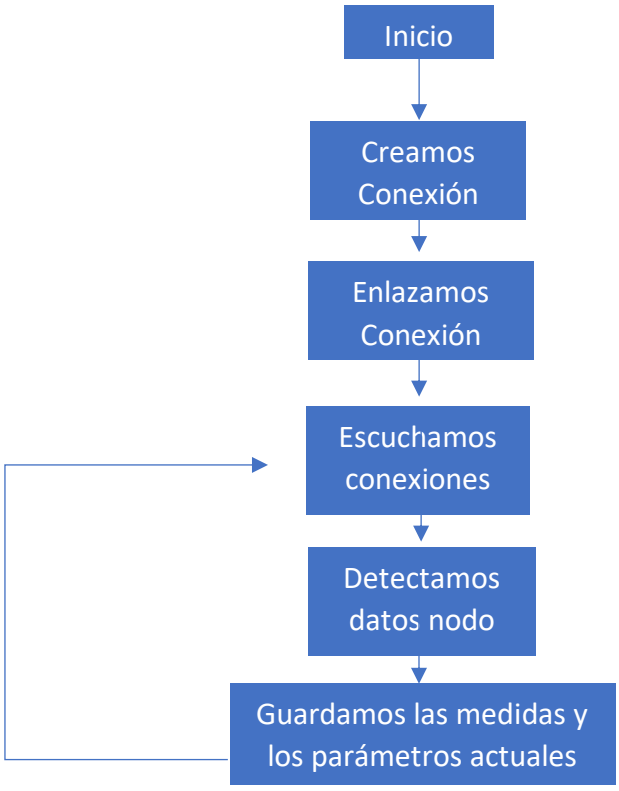


Fig. 2.16 Diagrama de flujo Servidor UDP

Además de guardar las medidas que llegan de los nodos en las tablas de la base de datos se ha desarrollado una función con la que guardaremos las medidas en un archivo CSV diferenciando cada nodo y el día en que se toman las medidas, es decir, tendremos un archivo CSV por nodo y día. De este modo tendremos un *backup* de los datos el cual podremos descargar desde las herramientas de visualización y así consultar datos pasados.

El Anexo III contiene el código completo para desarrollar el servidor.

2.7 Base de datos

Para alojar una base de datos y poder administrarla se ha instalado MySQL y phpMyAdmin en la Raspberry Pi. PhpMyAdmin proporciona una interfaz gráfica y visual para administrar la base de datos de una forma mucho más sencilla que si lo hiciéramos por comandos en la consola.

La base de datos consta de dos tablas. En la primera de ellas, llamada *Nodes*, se guarda la información básica de los nodos que conforman la red. Tenemos un campo para el identificador del nodo y otro con su estado, nodo activo (Y) o inactivo (N). Esta tabla la usaremos para saber los nodos que han recogido datos en algún momento y que por lo tanto podemos consultar.

El campo del estado del nodo está pensado para futuras evoluciones de optimización. Dependiendo del uso que se quiera hacer del sistema el estado en el que estarán los nodos por defecto será *sleep*, en el que los sensores no toman medidas y el transmisor radio está apagado, con lo que bajamos el consumo de energía considerablemente. En el momento en que el usuario quiera consultar los datos se actualizará este campo de la base de datos y se mandará como nuevo parámetro de control en los nodos, de los que ya se ha hablado anteriormente. De la misma manera, cuando el usuario ya no desee consultar los datos este parámetro se actualizará con el valor N, con el que se indica que el nodo no está activo, se mandará a los nodos y estos volverán al estado *sleep*.

Por otro lado, tenemos una segunda tabla, *Measures*, donde se guardarán todas las medidas que recoja cada nodo. En esta tabla encontramos un campo con el número identificador de nodo para poder diferenciar el origen de los datos, un campo para cada tipo de medida recogida por los sensores (temperatura, batería, etc.) y el instante de tiempo en el que se toma dicha medida que también aprovecharemos para poder ordenarlas en el gráfico de consultas.

Nombre	Tipo	Cotejamiento	Atributos	Nulo	Predeterminado
node_id	text	latin1_swedish_ci		No	*Ninguna*
is_active	varchar(1)	latin1_swedish_ci		No	N

Fig. 2.17 Estructura de la tabla *Nodes*

#	Nombre	Tipo	Cotejamiento	Atributos	Nulo	Predeterminado	Extra
1	id	int(11)			No	*Ninguna*	AUTO_INCREMENT
2	node_id	text	latin1_swedish_ci		No	*Ninguna*	
3	temperature	decimal(4,2)			Sí	*NULL*	
4	battery	decimal(4,0)			Sí	*NULL*	
5	external1	decimal(4,0)			Sí	*NULL*	
6	external2	decimal(4,0)			Sí	*NULL*	
7	time	timestamp		on update CURRENT_TIMESTAMP	No	CURRENT_TIMESTAMP	ON UPDATE CURRENT_TIMESTAMP

Fig. 2.18 Estructura de la tabla *Measures*

2.8 Servidor web

El servidor web es el componente que permitirá y gestionará las peticiones de acceso a los datos por parte del usuario. Para llevar a cabo esta funcionalidad se ha procedido con la instalación de *Apache Web Server* en la Raspberry Pi, un servidor web HTTP de código abierto para la creación de páginas y servicios web. Es un servidor multiplataforma, gratuito, muy robusto y que destaca por su seguridad y rendimiento. Tiene un archivo principal de configuración, apache2.conf, en el que podemos configurar las características principales y una carpeta específica localizada en el path /var/www/html en la que debemos alojar todos los archivos, *scripts*, páginas, imágenes, etc. que queramos que sean accesibles para las webs o aplicaciones que requieran de estos archivos o programas.

Para hacer accesible el servidor desde cualquier red externa debemos cumplir dos requerimientos, hacer el *forwarding* del puerto en nuestro *router* y proveer una dirección IP estática al servidor. El primer requerimiento consta en hacer el *forwarding* de uno de los puertos de nuestro *router* hacia el servidor, de esta manera, nos aseguramos que el *router* sabrá interpretar hacia dónde va el tráfico, tanto entrante como saliente, relacionado con los servicios que da el servidor web. Con el segundo punto, haremos que la IP del servidor sea siempre la misma. De no ser así cada vez que desconectemos el servidor de la red y volvamos a conectarlo la IP cambiaría y no sabríamos hacia donde direccionar el tráfico. Además de establecer una dirección IP estática, hemos creado un *host* “tfghost.hopto.org” para acceder de una manera más sencilla al servidor en lugar de poner la dirección IP en el navegador.

El *web server*, se encuentra a la espera de que algún navegador le haga alguna petición, como, por ejemplo, acceder a una página web y responder a la petición enviando código HTML mediante una transferencia de datos en red. Por otro lado, el servidor web también puede proporcionar servicios web, tecnología que utiliza un conjunto de protocolos y estándares que sirven para intercambiar datos entre aplicaciones. Distintas aplicaciones de software desarrolladas en lenguajes de programación diferentes, y ejecutadas sobre cualquier plataforma, pueden utilizar los servicios web para intercambiar datos.

En este caso se han desarrollado varios *scripts* para ofrecer los servicios web relacionados con la consulta de datos que los nodos han recopilado. Para ello se ha usado PHP como lenguaje de programación. Hay dos maneras generales de trabajar con *scripts* de PHP, crear un solo *script* y establecer en él todas las funciones y servicios web que se van a ofrecer o hacer varios, uno por función o por clase de datos, por ejemplo. Nosotros nos hemos decantado por esta segunda opción ya que esto nos permite una mejor organización y en el caso de tener que modificar algún parámetro es más sencillo. Los *scripts* que proporcionan los servicios web y la estructura de los mismos son diferentes dependiendo de la plataforma desde donde se consultan los datos.

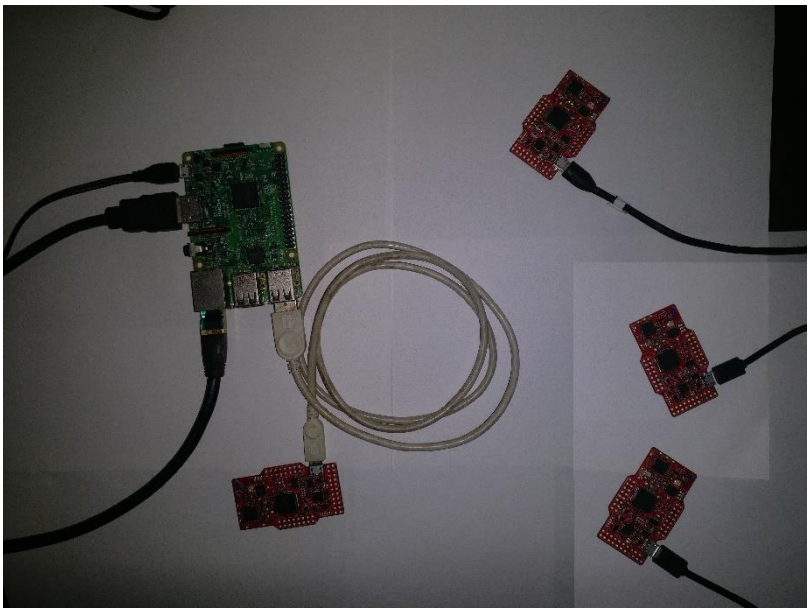
En el caso de acceder a los datos desde la aplicación Android tenemos dos *scripts* principales que son los que establecen la conexión y el manejo de la base de datos. En un segundo nivel encontramos los *scripts* por clase. Para este escenario tendremos dos clases, la clase Nodos y la clase Medidas, relacionadas con las tablas *Nodes* y *Measures* respectivamente de la base de datos explicadas en el punto anterior. En ellos se establecen todas las funciones necesarias de consulta de datos. Por último, se ha creado un *script* por cada función, de esta manera no hace falta llamar al *script* de clase directamente si no que lo hacemos con este *script* de intermediario. Los *scripts* de este último nivel, son los que tratan los datos obtenidos de la base de datos y los codifican en formato JSON que más tarde la herramienta de visualización se encargará de decodificar e interpretar.

En cambio, si accedemos desde la página web tendremos una primera parte del código en PHP que se ocupa de establecer la conexión a la base de datos y gestionar los datos. Una vez conectados a la base de datos, haremos una petición para que nos devuelva en cada caso, aquella información que nos interese. A continuación, creamos un bucle donde se irá rellenando un *array* que posteriormente será consultado por la función que utilizamos para graficar. Las tablas que consultamos durante todo este proceso, serán las mismas que en la aplicación Android. Por último, respecto a la exportación de los datos en formato CSV y JSON, tenemos dos scripts en PHP, que serán llamados desde la parte HTML de nuestro código.

Todos los *scripts* en su totalidad se pueden consultar en los Anexos IV y V.

3. Resultados Obtenidos

En este punto, la red ya está lista para poder tratar los datos de cada uno de los sensores. A continuación, podemos ver una imagen de la red que se ha desarrollado.



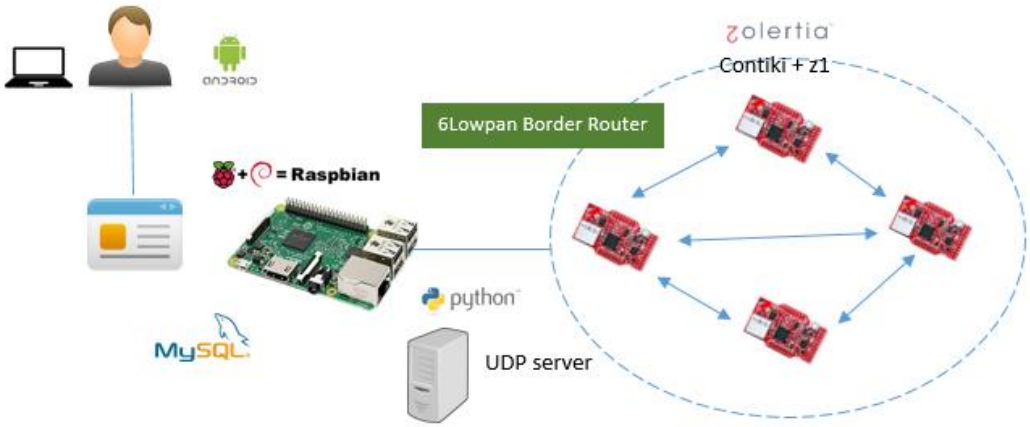


Fig. 3.1 Estructura de la red implementada

Durante el periodo de test y uso del sistema, se ha utilizado la configuración por defecto de Contiki. Algunos de los parámetros importantes que cabe destacar son los siguientes. Por un lado, se ha usado el canal 26 del medio en el que se trabaja con el estándar 802.15.4, ya que, trabajando en este canal, evitamos la interferencia producida por la señal wifi. Por otro lado, la potencia de transmisión

se ha fijado en 0 dBm (1 mW), siendo este el nivel máximo de potencia con la que puede trabajar el transmisor radio CC2420.

3.1 Test de conectividad

Para verificar la conectividad de la red que se ha desarrollado, primero de todo se comprobará la conectividad en la red WSN. 6LBR lleva integrada una opción donde podemos ver toda la conectividad de los nodos (*Sensors Page*). En esta página, podemos ver una lista de todos los nodos sensores diferenciados por su identificador, el tipo, el estado, etc. Por otro lado, también aparece la lista de los nodos vecinos, así como la conectividad que hay entre ellos, es decir, las rutas que seguirá el tráfico saliente de cada nodo para llegar hasta el *border router*.

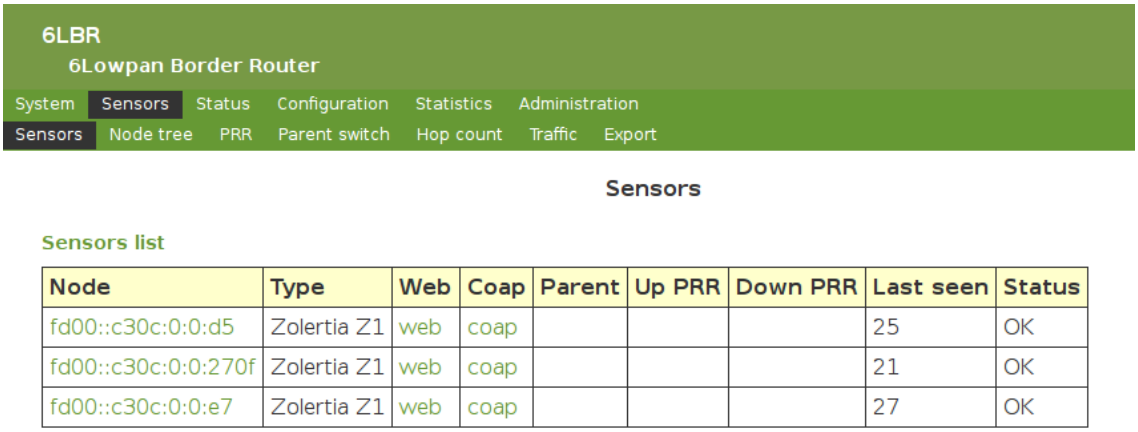


Fig. 3.2 Página de sensores en el servidor web de 6LBR

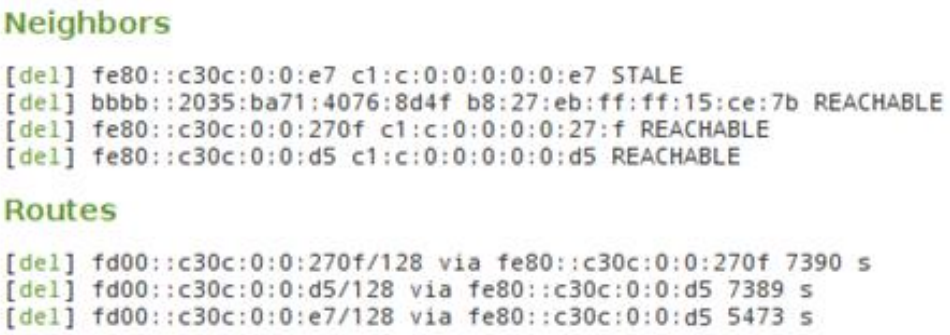


Fig. 3.3 Nodos vecinos y rutas en la red de sensores

Como se puede observar Contiki usa RPL, por ello la red WSN es lo que comúnmente se llama una red *multi-hop*, un tipo de comunicación en redes de radio en las que el área de cobertura de red es mayor que el rango de radio de los nodos individuales, por motivos físicos de distancia u otros como el *fading*. Por lo tanto, para llegar a algún destino, un nodo puede usar otros nodos como *relays* o repetidores.

Las rutas son dinámicas por lo que no siempre aparecerán las mismas para llegar al mismo nodo. En la figura 3.3 podemos observar que en este caso la red cumple con esta característica. La última ruta que tiene como destino el nodo con id “e7” es accesible a través del nodo con id “d5” y no directamente como en los otros dos casos con los nodos “d5” y “270f”.

Por defecto, los valores usados en la configuración del 6LBR, los nodos de la red están auto configurados con la dirección IPv6 de la subred aaaa::/64. Respecto a la parte de Ethernet, el prefijo configurado es el bbbb::/64. El *border router* envía periódicamente un RA (*Router Adverstising*), un mensaje que contiene el prefijo de la subred y la ruta de acceso a la subred de la WSN.

Por último, si habilitamos NAT64, podremos acceder al *border router* y a todos los nodos de la WSN usando IPv4 en lugar de IPv6. De forma inversa, el *border router* y los nodos pueden acceder a cualquier dirección IPv4 externa usando uno de los siguientes prefijos, ::ffff:0:0/96 o 64:ff9b::/96. Para poder acceder a un nodo desde un host IPv4, NAT64 y el mapeo de puerto estático tendrá que estar habilitado. En la figura 3.4 podemos ver la configuración aplicada.

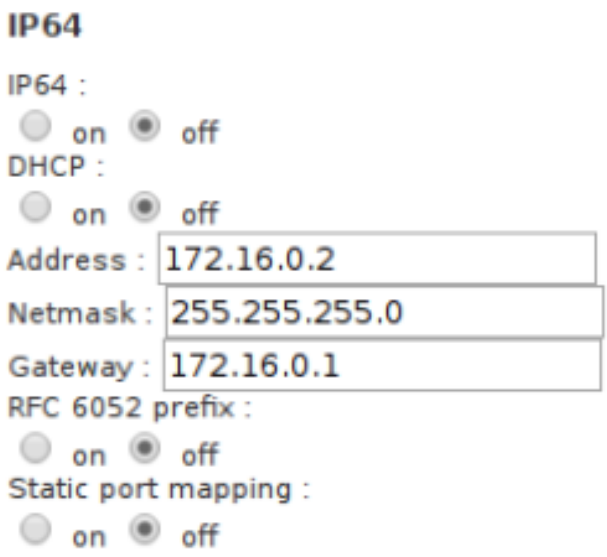


Fig. 3.4 Configuración NAT64

3.2 Muestra de datos

Con el objetivo de mostrar los datos que ha tomado o está tomando el sistema de sensores se han desarrollado dos herramientas de visualización, una página web y una aplicación Android. De esta manera tenemos dos alternativas diferentes, complementarias entre ellas, accesibles desde cualquier dispositivo con acceso a internet mediante las que el usuario podrá consultar los datos en cualquier momento.

Como ya se ha explicado anteriormente, el sistema tiene una limitación en cuanto a la adquisición de señales, y es que el ancho de banda de la señal que se va a medir no puede ser mayor al ancho de banda del ADC del microprocesador integrado en el nodo de Zolertia, ya que estaríamos perdiendo información.

Por su parte, las herramientas de visualización también tienen una limitación respecto al ancho de banda de la señal que se va a medir y en este caso más restrictiva que la del ADC.

En la etapa de test se han hecho pruebas de calidad en la visualización de datos modificando el ancho de banda de la señal que se medía. Se ha comprobado que, con una frecuencia de muestreo en el nodo de 15 Hz, es decir, obteniendo 15 muestras por segundo, lo que implica un ancho de banda máximo de la señal de 7,5 Hz, la calidad de la visualización empieza a decaer. Esto supone que el gráfico no se refresca a la misma velocidad que se adquieren los datos por lo que en pantalla veremos pequeños retrasos de forma esporádica en la visualización de nuevos datos, aunque estos no se perderán y se acabarán mostrando.

Aunque la calidad de visualización sea muy baja podremos consultar los datos, aunque no de forma gráfica, leyendo el archivo CSV en el que podemos asegurar que se guardan todas las medidas. Este segundo método no se ve afectado por este problema ya que el servidor UDP se encarga de registrar todas las medidas de los paquetes que le llegan desde los nodos en el mismo momento de la recepción.

3.2.1 Aplicación Android

Con el fin de que el usuario pueda ver, en tiempo real, los datos que están recogiendo los sensores o acceder a datos que se han tomado con anterioridad se ha desarrollado esta herramienta, enfocada para el uso de dispositivos con sistema operativo Android (versión 4.4 o superior) y programada en Java y Xml. Todo el código de desarrollo de la aplicación, con sus comentarios correspondientes, se puede encontrar en el Anexo IV al final de este documento.

Mediante una interfaz sencilla e intuitiva como se puede observar en la figura 3.5 el usuario puede escoger el nodo que desea consultar usando la lista dinámica con todos los nodos disponibles en la red y también el parámetro que desea consultar utilizando los distintos botones de la pantalla principal. En ese

momento se abrirá la pantalla de consultas en la que aparecerá un gráfico con las medidas requeridas.

Fig. 3.5 Pantalla de inicio y visualización de medidas

El funcionamiento de la aplicación se basa en un proceso que llama a un servicio web determinado, dependiendo del nodo y el parámetro que se desee consultar, alojado en el servidor web. Este servicio web retorna a la aplicación los datos de la base de datos en formato JSON. Un *parser* decodifica estos datos y se pasan al gráfico de muestra, el cual se va actualizando dinámicamente a medida que se van recibiendo datos. Este mecanismo se repite periódicamente hasta que el usuario sale de la pantalla de visualización.

Además de la consulta de datos en tiempo real, la aplicación también tiene la opción de descargar archivos CSV según el nodo y el día que queramos consultar. El usuario puede escoger el nodo de la misma manera que para la visualización en tiempo real y la fecha de la que quiere consultar los datos como se puede observar en la figura 3.6. La aplicación se encarga de abrir una conexión con la base de datos especificando el nombre del archivo que se requiere y de descargarlo. Durante el proceso de descarga una barra de estado muestra el progreso y un mensaje de éxito o de error si no se ha podido completar la descarga. En el caso de que este proceso haya concluido

exitosamente el archivo se alojará por defecto en una nueva carpeta que se creará en el momento de la descarga llamada "*Past measures*" (medidas anteriores) dentro de la carpeta Descargas de nuestro dispositivo. Para poder usar esta funcionalidad de la aplicación el usuario debe otorgar permisos de acceso al contenido multimedia del dispositivo con el fin de permitir que la aplicación guarde el archivo que se descarga en el almacenamiento interno del dispositivo. Estos permisos son solicitados por la aplicación en su primer uso.

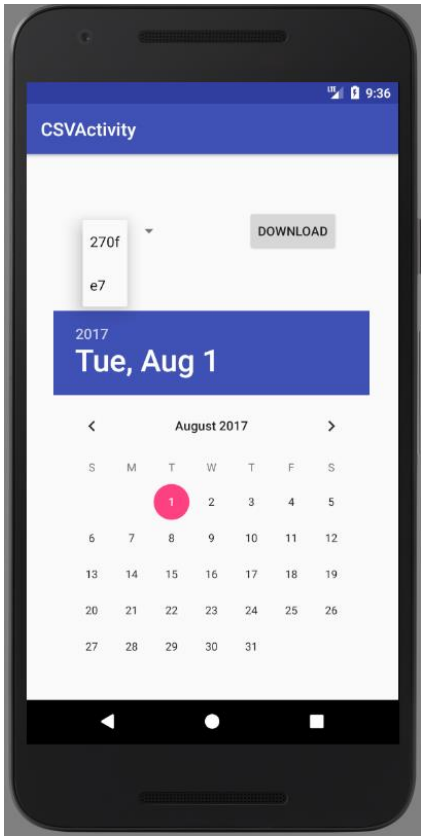


Fig. 3.6 Pantalla de descarga de datos históricos

3.2.2 Web

Por otro lado, se ha desarrollado una interfaz web para el usuario en la que pueda escoger entre los diferentes nodos, y podamos visualizar las variaciones en temperatura y batería que presenten los sensores, tal y como podemos ver en la figura 3.7. Además, se han añadido dos opciones donde podremos visualizar los datos de sensores externos que tengamos conectados. Esta funcionalidad está totalmente preparada para recibir datos de aquellos sensores externos soportados por Zolertia.

Fig. 3.7 Página web donde escoger nodo

Para la implementación de la web, se ha usado principalmente HTML y JavaScript en cuanto al código principal se refiere. El formato y funcionamiento ha sido extraído de la API de Google Charts, la cual posibilita la creación de gráficos interactivos de forma intuitiva y relativamente sencilla. Google charts da la posibilidad de poder escoger entre un abanico de posibilidades en cuanto a tipo de gráfico se refiere. En nuestro caso escogimos el tipo lineal, ya que hemos considerado que es una buena forma de mostrar los datos guardados por los sensores.

El funcionamiento interno es bastante sencillo, utilizando un pequeño script de PHP nos conectamos a nuestra base de datos de MySQL, como ya se ha comentado anteriormente en el punto 2.8. Una vez conectados, extraemos la información que se ha ido almacenando de cada una de las muestras que han ido recopilando los sensores. Por último, cuando ya tengamos estos datos, se mostrarán en forma lineal usando las gráficas de la API.

Por otro lado, se podrán exportar los datos que estemos visualizando en diferentes formatos. Los formatos en los que podremos exportar los datos serán CSV y JSON. En la figura 3.8 podemos observar el resultado la implementación de la web.

El código correspondiente a la web se puede encontrar en el Anexo V, al final de este documento.

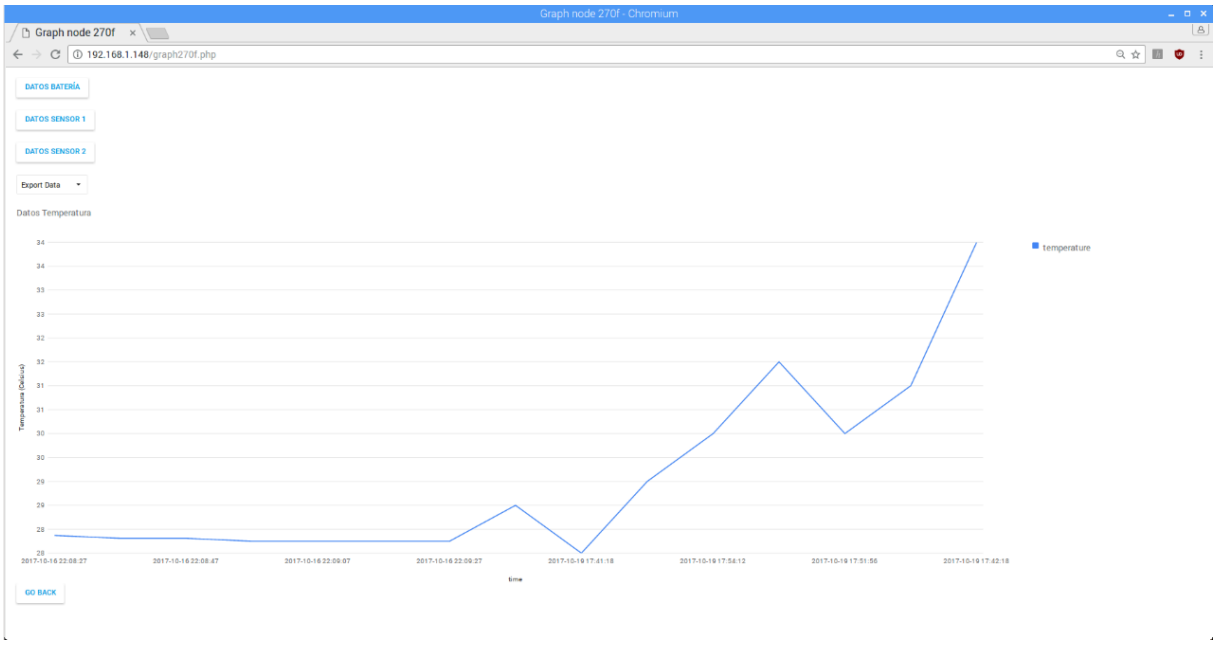


Fig. 3.8 Página web donde consultar los datos

3.3 Futuras implementaciones

Como se ha introducido anteriormente, el sistema está preparado para poder conectar sensores externos a las motas Z1, recibir y procesar los datos que estos mandan y visualizarlos. Sin embargo, no se puede conectar cualquier sensor de una manera ágil y sencilla. Zolertia proporciona una lista de sensores, que ya se ha referenciado en el apartado 2.3, de los que podemos encontrar el código necesario para integrarlos y gestionarlos en nuestro sistema fácilmente. En el caso de usar un sensor que no esté en la lista no tendremos esta ventaja y deberemos desarrollar el código desde cero para gestionarlo.

En cualquiera de los casos, lo que debemos hacer es modificar el archivo .c que se proporciona y que acabaremos subiendo al nodo. Para gestionar cualquier sensor es necesario inicializarlo en el *PROCESS_THREAD*, función principal del programa del nodo, como podemos ver en la figura 3.9 y llamar a la función propia del nodo para que haga una lectura en nuestra función *log_packet()* como se observa en la figura 3.10. Además, deberemos añadir las librerías necesarias para gestionar el sensor en cuestión en la cabecera del archivo.

```
/* Activate the sensors */
tmp102_init();
(battery_sensor);
SENSORS_ACTIVATE(button_sensor);
```

Fig. 3.9 Inicialización de los sensores

```
e->msg.temperature = tmp102_read_temp_x100();
/* Convert the battery reading from ADC units to mV (powered over USB)*/
aux = battery_sensor.value(0);
aux *= 5000;
aux /= 4095;
e->msg.battery = aux;
//PRINTF("%s",aux);
/* External sensor 1 read from ADC */
aux1 = battery_sensor.value(0);
aux1 *= 5000;
aux1 /= 4095;
e->msg.external1 = aux1;
//PRINTF("%s",aux1);
/* External sensor 2 read from ADC */
aux2 = battery_sensor.value(0);
aux2 *= 5000;
aux2 /= 4095;
e->msg.external2 = aux2;
```

Fig. 3.10 Funciones de lectura de los sensores

Durante el periodo de test, y debido a la falta de disponibilidad de sensores externos, se ha usado el sensor de batería de forma reiterada simulando la función de sensor externo con el objetivo de verificar el funcionamiento completo del sistema, lo que quiere decir que el nodo recoge los valores de todos los sensores disponibles y los manda todos de forma correcta. Esto es posible ya que el sensor de batería integrado va conectado a la entrada de un ADC del microprocesador, misma situación en la que nos encontraremos cuando conectemos un sensor externo a los pins de integración de la mota.

En el Anexo II podemos ver dos ejemplos que usaríamos en caso de conexión de los sensores externos ZIG001 *Temperature-Humidity Sensor* y ZIG002 *Light Sensor* de Zolerti

Conclusiones

Este proyecto se basa en la implementación de un sistema IoT formado por una red de sensores que puedan recaudar señales de cualquier tipo. Para lograr este objetivo primero se han expuesto los fundamentos teóricos en los que se sustenta el proyecto. En este apartado tenemos una descripción de las tecnologías que hacen posible el funcionamiento de una red IoT como son las redes de sensores (WSN), IPv6, 6LoWPAN y el sistema operativo Contiki OS, que gestiona los sensores y que permite trabajar con estas tecnologías.

A continuación, se ha implementado una solución para llevar a cabo el objetivo propuesto, que consiste en una red de sensores basada en 6LoWPAN, en la que las motas captan información de parámetros como temperatura, nivel de batería de la mota e información de sensores externos que podemos conectar a la mota. Esta información se manda a un elemento concentrador donde se procesa y almacena para finalmente mostrarla al usuario de forma gráfica en tiempo real. Además, este elemento intermedio entre la red y el usuario tiene la opción de trabajar en modo *data logger* para poder consultar datos pasados en cualquier momento. Para implementar el sistema se ha usado una Raspberry Pi y 4 nodos Zolertia Z1, aplicándoles soluciones de código abierto o desarrollando nuestros propios programas para la configuración y gestión de la red. Dentro de las soluciones de código abierto nos encontramos 6LBR, que juega el rol de *border router* permitiendo la comunicación entre la red de sensores y cualquier otra red IP externa. Por otro lado, se ha llevado a cabo el desarrollo de varios programas para aplicar tanto a los nodos como al dispositivo central que hará de servidor.

Para los nodos se ha desarrollado un programa para tomar medidas periódicamente de los sensores presentes en el propio nodo, sean internos, como temperatura o nivel de batería, o externos, los cuales serán elección de cada usuario, y mandar la información a la Raspberry Pi que hará la función de dispositivo central de la red. Para enviar la información se ha considerado la opción de usar el protocolo de transporte UDP. A parte, se ha desarrollado un servidor UDP para recibir la información que envían los nodos, procesarla y almacenarla en una base de datos local. Estos dos últimos componentes están alojados en la Raspberry Pi. Sobre la Raspberry Pi, también correrá un servidor web, que ofrecerá varios servicios web de consulta de los datos alojados en la base. Para dar uso a los servicios web que se acaban de nombrar, se han desarrollado dos herramientas de visualización gráfica de datos, una página web y una aplicación Android. Ambas, además permiten la opción de descargar archivos CSV con datos pasados, registrados por el servidor UDP mientras trabaja en el modo data logger.

Una vez tenemos todos los dispositivos en funcionamiento, se ha llevado a cabo un test de conectividad para verificar el funcionamiento del sistema, tanto dentro de la red de sensores como la interacción entre la red de sensores y una red externa.

Por último, con el sistema funcionando se han hecho pruebas del conjunto y podemos confirmar el buen funcionamiento del mismo. Los datos siguen el flujo previsto, sin pèrdidas y sin demoras si tenemos en cuenta las limitaciones de las herramientas de visualización. De no sobrepasar estas limitaciones los datos no se muestran en un tiempo real exacto si no que el sistema sufre pequeños retrasos de procesamiento. De todas formas, los datos no se pierden gracias al funcionamiento simultáneo del modo *data logger*. Por otro lado, de no tener en cuenta las limitaciones del propio sistema, haciendo referencia a la limitación que introduce el ADC del microprocontrolador, explicada a lo largo del proyecto, los datos que se toman y que acabamos visualizando no son fiables, es decir, estamos perdiendo información de la señal.

Dentro de los objetivos del proyecto, no se ha tenido en cuenta el consumo de energía del sistema, y más concretamente el consumo de los nodos dentro de la red de sensores, dato crítico en estos escenarios. Aún así, durante el desarrollo se han dejado trazas, tanto en el código como en la estructura del sistema y de los datos, como por ejemplo con el campo *is_active* de la base de datos puediéndolo usar como factor de control de los nodos, para poder hacer una optimización del sistema en futuros desarrollos e implementaciones en lo que a ahorro de energía se refiere.

Finalmente, cabe destacar que todas las aplicaciones desarrolladas en este proyecto pueden ser usadas en otros desarrollos. El programa para los nodos es independiente del hardware y puede usarse en cualquier otra plataforma siempre y cuando dicha plataforma o dispositivo use el sistema operativo Contiki OS. El servidor UDP puede iniciarse en cualquier sistema operativo con el único requerimiento de tener instalado Python y MySQL (en la mayoría de distribuciones de Linux y Windows es posible instalar estos componentes). Lo mismo ocurre con el servidor web. Y, por último, en cuanto a las herramientas de visualización, la página web se puede consultar desde cualquier sistema, al contrario de la aplicación, que debe instalarse obligatoriamente en un dispositivo con sistema operativo Android.

Acrónimos

6LoWPAN	IPV6 Over Low Power Wireless Personal Area Networks
ADC	Analog to Digital Converter
API	Application Programming Interface
BGP	Border Gateway Protocol
CETIC	Centre of Excellence in Information and Communication Technologies
CPU	Central Processing Unit
DHCP	Dynamic Host Configuration Protocol
DSI	Display Interface
GPIO	General Purpose Input/Output
GPRS	General Packet Radio Service
GPU	Graphics Processor Unit
GSM	Global System for Mobile Communications
GTS	Guaranteed Timeslot
HTTP	Hypertext Transfer Protocol
ICMP	Internet Message Control Protocol
ICTs	Information and Communication Technologies
IEEE	Institute of Electrical and Electronics Engineers
IoT	Internet of Things
IP	Internet Protocol
LAN	Local Area Network
LTE	Long Term Evolution
MAC	Media Access Control
MIP	Mobile Internet Protocol
MTU	Maximum Transmission Unit
NAT	Network Address Translation
NDP	Neighbor Discovery Protocol
RA	Router Advertisement
RAM	Random Access Memory
RDC	Radio Duty Cycle
RF	Radio Frequency
RPL	Routing Protocol for Low-Power and Lossy Networks
SBC	Single Board Computer
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
USB	Universal Serial Bus
VM	Virtual Machine
WPAN	Wireless Personal Area Network
WSN	Wireless Sensor Network

Bibliografía

[1] Liñan, A, Vives, A, Bagula, A, Zennaro, M, Pietrosemoli, E, IoT in five Days, 2016.Disponible: https://zolertia.io/docs/IoT_in_five_days-v1.0.pdf

[2] Estudio de IHS Markit. Disponible en: http://electronics360.globalspec.com/article/8032/20-billion-connected-internet-of-things-devices-in-2017-ihs-markit-says

[3] Previsión de futuro del IoT. Disponible en: http://spectrum.ieee.org/tech-talk/telecom/internet/popular-internet-of-things-forecast-of-50-billion-devices-by-2020-is-outdated

[4] Valor del IoT en el mercado e inversión a nivel mundial. Disponible en: https://www.forbes.com/sites/louiscolumbus/2017/01/29/internet-of-things-market-to-reach-267b-by-2020/#5d721693609b

[5] Arquitectura IoT. Disponible en: https://programarfacil.com/podcast/arduino-wifi-proyectos-iot/

[6] Características de las redes de sensores. Disponible en: http://informaticaredes2012.blogspot.com.es/

[7] Definición y características del 6LoWPAN. Disponible en: http://iot6.eu/6lowpan

[8] Contiki OS. Disponible en: http://www.contiki-os.org/

[9] Datasheet del dispositivo Raspberry Pi 3 Modelo B. Disponible en: http://docs-europe.electrocomponents.com/webdocs/14ba/0900766b814ba5fd.pdf

[10] Datasheet del dispositivo Zolertia Z1. Disponible en: http://wiki.zolertia.com/wiki/images/e/e8/Z1_RevC_Datasheet.pdf

[11] Contiki Wiki. Disponible: https://github.com/contiki-os/contiki/wiki

[12] 6lbr Wiki. Disponible: https://github.com/cetic/6lbr/wiki

[13] Phidgets Zolertia: http://zolertia.sourceforge.net/wiki/index.php/Mainpage:allsensors#Phidgets

[14] Sintaxis MySQL: https://dev.mysql.com/doc/refman/5.5/en/sql-syntax.html

[15] Raspberry Pi Documentación: https://www.raspberrypi.org/

[16] Google Charts Examples: https://developers.google.com/chart/interactive/docs/gallery/linechart

[17] Pyhton comunicación protocolo UDP: https://wiki.python.org/moin/UdpCommunication

[18] Documentación PHP: http://php.net/manual/es/index.php

Anexo I. Instalación de Contiki

Contiki es un *software* complejo es su totalidad, tanto por su estructuración general como en su funcionamiento. A continuación, se detallarán los pasos a seguir para su instalación y además se dará unas pequeñas indicaciones de como compilar una aplicación.

I. Instalando Instant Contiki

Instalar Instant Contiki nos facilitará el poder empezar a trabajar des del primer momento. Se deberán de seguir los siguientes pasos para efectuar su instalación:

- Descargar Instant Contiki. Una vez descargado, deberemos descomprimirlo en el directorio que decidamos.
- Descargar e instalar la máquina virtual VMWare Player. Esta máquina virtual es completamente gratuita, lo único que deberemos de hacer será registrarnos. Es posible que se nos pida reiniciar el ordenador para completar la instalación. Por otro lado, necesitaremos una conexión a internet para poder finalizar la instalación.
- Iniciar Instant Contiki ejecutando el archivo descargado anteriormente “InstantContiki3.0.vmx” (última versión en este momento es la 3.0 pero es posible descargar versiones anteriores). Esperaremos a que se inicie la máquina virtual de Ubuntu Linux.
- Iniciar sesión en Instant Contiki. La contraseña por defecto es *user*.

Existe otra versión de Contiki, en la que además del software nos incluye una serie de ejemplos que pueden ayudar al desarrollo de aplicaciones que usen sensores zolertia. Esta versión es la de IOT workshop VM, en la que aparecen resueltos y completamente funcionales, todos los ejemplos del libro IOT in five days.

II. Compilar una aplicación en Contiki

Para poder proceder con la compilación de una aplicación en Contiki, hay 3 archivos esenciales que tenemos que tener en cuenta: el archivo de aplicación Contiki, que se ha mostrado en capítulos anteriores, el cual contiene la aplicación que se usará, por otro lado, tenemos el archivo de configuración de nuestra aplicación y por último el archivo que controlará la estructura de nuestra aplicación cuyo nombre es *Makefile*.

La estructura del archivo *Makefile* es bastante sencilla:

```
CONTIKI = ../../../..
all: app-name
include $(CONTIKI)/Makefile.include
```

La primera línea indica la localización de la ruta raíz de Contiki, la segunda línea específica el nombre de la aplicación que se compilará y la tercera línea incluye el sistema de Contiki por completo, el cual contiene definiciones del núcleo de Contiki, y los puntos para nuestro *Makefile*, además de contener cuál será la plataforma donde tendrá que apuntar.

Por otro lado, tenemos el archivo projectconf.h. Es un archivo opcional el cual no está activo por defecto. Para poder activarlo tendremos que añadir la siguiente línea a nuestro *Makefile* del proyecto:

```
CFLAGS += -DPROJECT_CONF_H=\"../project-conf.h\"
```

Con este archivo podremos modificar o activar configuraciones de Contiki o incluso cambiar parámetros que Contiki tiene configurados por defecto.

Por último, para compilar una aplicación tendremos que tener en cuenta dos cosas: la plataforma a la que queramos apuntar, por ejemplo, en nuestro caso sería z1, y por otro lado el archivo que queramos subir y compilar en dicha plataforma.

Para establecer hacia donde apunta Contiki a la hora de compilar, tendremos que seguir la sintaxis de TARGET = plataforma. En el caso de que no especifiquemos nada, se compilará en una plataforma que está configurada por defecto.

```
make TARGET= z1 nombre_aplicacion
```

Y por otro lado para compilar la aplicación podremos hacer .upload en el caso que queramos subir nuestra aplicación a la plataforma o un punto .login para ejecutarla en la línea de comandos.

```
make TARGET=z1 nombre_aplicacion.upload

make TARGET=z1 nombre_aplicacion.login
```

Anexo II. Programación de los nodos

I. Cliente UDP

```
/*****************************************************************
 *
 * NODE PROGRAM - UDP
 *
*****************************************************************/

/*---------------------------------------------------------------*/
#include "contiki.h"
#include "lib/list.h"
#include "sys/ctimer.h"
#include "net/ip/uip.h"
#include "net/ipv6/uip-ds6.h"
#include "net/ip/uip-udp-packet.h"
#include "../example2.h"
#include <stdio.h>
#include <string.h>

/* Library needed to use sensors in Z1 */
#include "dev/battery-sensor.h"
#include "dev/i2cmaster.h"
#include "dev/tmp102.h"
#include "dev/button-sensor.h"

/*Lib to estimate power consumption */
//#include "powertrace.h"


/*---------------------------------------------------------------
-------*/
/* Enables printing debug output from the IP/IPv6 libraries */
#define DEBUG DEBUG_PRINT
#include "net/ip/uip-debug.h"
/*---------------------------------------------------------------
-------*/
/* Default is to send a packet every 60 seconds */
#define MEASURE_INTERVAL            (5 * CLOCK_SECOND)
static uint16_t limit=1;
static uint16_t time=1;
/*---------------------------------------------------------------
-------*/
/* The structure used in the Simple UDP library to create an UDP
connection */
static struct uip_udp_conn *client_conn;

/* This is the server IPv6 address */
static uip_ipaddr_t server_ipaddr;

/* Keeps account of the number of messages sent */
static uint16_t counter = 0;

static uint16_t counter_send=0;
static struct etimer periodic;
```

```
/*---------------------------------------------------------------------
-------*/
/* Create a linked list of data structures to store the data to be
sent as payload */
LIST(measures_list);
/* Structure for the linked list*/
struct msg_ll {
      struct msg_ll *next;
      struct my_msg_t msg;
};


/*---------------------------------------------------------------------
-------*/
PROCESS(udp_server_process, "UDP server example process");
AUTOSTART_PROCESSES(&udp_server_process);
/*---------------------------------------------------------------------
-------*/

/* Whenever the node receive a packet from the server, this function
is invoked
 * "uip_newdata()" is used to check if there is actually data in the
buffer
 * This function read the new parameters and sets them */
static void
tcpip_handler(void)
{
      char *str;

      if(uip_newdata()) {
            str = uip_appdata;
            str[uip_datalen()] = '\0';

            char *time_cmd = strtok(str," "); //first_part points to
"time"
            char *limit_cmd = strtok(NULL," ");   //sec_part points to
"limit"

            if (time_cmd != NULL || limit_cmd!=NULL){
                  time = atoi(time_cmd);
                  limit = atoi(limit_cmd);
                  counter_send=0;
                  etimer_reset(&periodic);
                  etimer_set(&periodic,
(atoi(time_cmd)*MEASURE_INTERVAL));
                  do {
                        struct msg_ll *e=list_pop(measures_list);
                        free(e);
                  }while(list_length (measures_list) != 0);
            }
      }
}

/*---------------------------------------------------------------------
-------*/
/* If is time to send a packet this function is invoked*/
static void
send_packet(void)
{
      char buffer_ll[2*sizeof(uint16_t)+limit*sizeof(struct
my_msg_t)];
```

```
      char *buffer_llPtr = buffer_ll;
      /*Copy the values of the current parameters in the buffer*/
      memcpy(buffer_llPtr, &time, sizeof(uint16_t));
      memcpy(buffer_llPtr+sizeof(uint16_t), &limit, sizeof(uint16_t));
      /*Copy sensor readings in the buffer*/
      int j = 0;
      struct msg_ll *s;
      for(s = list_head(measures_list); s != NULL; s =
list_item_next(s)) {
            memcpy(buffer_llPtr+2*sizeof(uint16_t)+j*sizeof(struct
my_msg_t), &s->msg, sizeof(struct my_msg_t));
            j++;
      }
      uip_udp_packet_sendto(client_conn, &buffer_ll,
sizeof(buffer_ll),
                        &server_ipaddr, UIP_HTONS(UDP_SERVER_PORT));
}
/*---------------------------------------------------------------------
-------*/
/* This function read values the sensors and store this data in the
linked list*/
static void
log_packet(void)
{
      uint32_t aux;
      uint32_t aux1;
      uint32_t aux2;
      counter++;
      struct msg_ll *e;
      e =  malloc(sizeof(struct msg_ll));
      if(e != NULL) {
            e->msg.counter = counter;
            e->msg.temperature = tmp102_read_temp_x100();
            /* Convert the battery reading from ADC units to mV
(powered over USB) */
          aux = battery_sensor.value(0);
          aux *= 5000;
          aux /= 4095;
          e->msg.battery = aux;
          //PRINTF("%s",aux);
          /* External sensor 1 read from ADC */
          aux1 = battery_sensor.value(0);
          aux1 *= 5000;
          aux1 /= 4095;
          e->msg.external1 = aux1;
          //PRINTF("%s",aux1);
          /* External sensor 2 read from ADC */
          aux2 = battery_sensor.value(0);
          aux2 *= 5000;
          aux2 /= 4095;
          e->msg.external2 = aux2;
          //PRINTF("%s",aux2);

          list_add(measures_list, e);
      }
      else {printf("No memory available");}
}
/*---------------------------------------------------------------------
-------*/

static void
```

```
print_local_addresses(void)
{
      int i;
      uint8_t state;

      PRINTF("Client IPv6 addresses:\n");
      for(i = 0; i < UIP_DS6_ADDR_NB; i++) {
            state = uip_ds6_if.addr_list[i].state;
            if(uip_ds6_if.addr_list[i].isused &&
                        (state == ADDR_TENTATIVE || state ==
ADDR_PREFERRED)) {
                  PRINT6ADDR(&uip_ds6_if.addr_list[i].ipaddr);
                  PRINTF("\n");
                  /* hack to make address "final" */
                  if (state == ADDR_TENTATIVE) {
                        uip_ds6_if.addr_list[i].state = ADDR_PREFERRED;
                  }
            }
      }
}

/*---------------------------------------------------------------------
-------*/
PROCESS_THREAD(udp_server_process, ev, data)
{
      PROCESS_BEGIN();
      //powertrace_start(CLOCK_SECOND*10);
      PROCESS_PAUSE();

      printf("UDP server process started\n");

      /* Raspberry IPv6 address bbbb::388e:bd5:1147:2035*/
      //uip_ip6addr(&server_ipaddr, 0xbbbb, 0, 0, 0, 0x388e, 0xbd5,
0x1147, 0x2035);
      /* Rpi IPv6 address bbbb::4c22:4b04:d9:ef06 */
      uip_ip6addr(&server_ipaddr, 0xbbbb, 0, 0, 0, 0x4c22, 0x4b04,
0xd9, 0xef06);

      /* Print the server's addresses */
      printf("Server address: ");
      PRINT6ADDR(&server_ipaddr);
      printf("\n");
      /* Print the node's addresses */
      print_local_addresses();
      /* Activate the sensors */
      tmp102_init();

      (battery_sensor);
      SENSORS_ACTIVATE(button_sensor);
      /* Create a new UDP connection*/
      client_conn = udp_new(NULL, UIP_HTONS(UDP_SERVER_PORT), NULL);
      if(client_conn == NULL) {
            PRINTF("No UDP connection available, exiting the
process!\n");
            PROCESS_EXIT();
      }
      /* This function binds a UDP connection to a specified local
port*/
      udp_bind(client_conn, UIP_HTONS(UDP_CLIENT_PORT));

      PRINTF("Created a connection with the server ");
```

```
      PRINT6ADDR(&client_conn->ripaddr);
      PRINTF(" local/remote port %u/%u\n", UIP_HTONS(client_conn-
>lport),
                                          UIP_HTONS(client_conn->rport));
      /* Set the timer to default value */
      etimer_set(&periodic, MEASURE_INTERVAL);
      /* Initialize the list used to log measures. */
      list_init(measures_list);
      PRINTF("Entro en el while");
      while(1) {
            PROCESS_YIELD();
            /* Incoming events from the TCP/IP module, when there is a
new packet from the server*/
            if(ev == tcpip_event) {
                  tcpip_handler();
            }
            /* Send data to the server */
            if((ev == sensors_event && data == &button_sensor) ||
                        (ev == PROCESS_EVENT_TIMER)) {
                  counter_send++;
                  log_packet();
                  if (counter_send==limit){
                        send_packet();
                        counter_send=0;
                        do {
                              struct msg_ll *e=list_pop(measures_list);
                              free(e);
                        }while(list_length (measures_list) != 0);
                  }
                  if(etimer_expired(&periodic)) {
                        etimer_reset(&periodic);
                  }
            }
      }
      PROCESS_END();
}
/*---------------------------------------------------------------------
-------*/
```

II. Ejemplo de uso del sensor externo ZIG001

```
/*****************************************************************
 *
 * NODE PROGRAM - UDP
 *
*****************************************************************/

/*-------------------------------------------------------------------*/
#include "contiki.h"
#include "lib/list.h"
#include "sys/ctimer.h"
#include "net/ip/uip.h"
#include "net/ipv6/uip-ds6.h"
#include "net/ip/uip-udp-packet.h"
#include "../example2.h"
#include <stdio.h>
#include <string.h>

/* Library needed to use sensors in Z1 */
#include "dev/battery-sensor.h"
#include "dev/i2cmaster.h"
```

```
#include "dev/tmp102.h"
#include "dev/button-sensor.h"
#include "dev/sht11.h"

/*Lib to estimate power consumption */
//#include "powertrace.h"


/*---------------------------------------------------------------------
-------*/
/* Enables printing debug output from the IP/IPv6 libraries */
#define DEBUG DEBUG_PRINT
#include "net/ip/uip-debug.h"
/*---------------------------------------------------------------------
-------*/
/* Default is to send a packet every 60 seconds */
#define MEASURE_INTERVAL            (5 * CLOCK_SECOND)
static uint16_t limit=1;
static uint16_t time=1;
/*---------------------------------------------------------------------
-------*/
/* The structure used in the Simple UDP library to create an UDP
connection */
static struct uip_udp_conn *client_conn;

/* This is the server IPv6 address */
static uip_ipaddr_t server_ipaddr;

/* Keeps account of the number of messages sent */
static uint16_t counter = 0;

static uint16_t counter_send=0;
static struct etimer periodic;

/*---------------------------------------------------------------------
-------*/
/* Create a linked list of data structures to store the data to be
sent as payload */
LIST(measures_list);
/* Structure for the linked list*/
struct msg_ll {
      struct msg_ll *next;
      struct my_msg_t msg;
};


/*---------------------------------------------------------------------
-------*/
PROCESS(udp_server_process, "UDP server example process");
AUTOSTART_PROCESSES(&udp_server_process);
/*---------------------------------------------------------------------
-------*/

/* Whenever the node receive a packet from the server, this function
is invoked
 * "uip_newdata()" is used to check if there is actually data in the
buffer
 * This function read the new parameters and sets them */
static void
tcpip_handler(void)
{
```

```
      char *str;

      if(uip_newdata()) {
            str = uip_appdata;
            str[uip_datalen()] = '\0';

            char *time_cmd = strtok(str," "); //first_part points to
"time"
            char *limit_cmd = strtok(NULL," ");   //sec_part points to
"limit"

            if (time_cmd != NULL || limit_cmd!=NULL){
                  time = atoi(time_cmd);
                  limit = atoi(limit_cmd);
                  counter_send=0;
                  etimer_reset(&periodic);
                  etimer_set(&periodic,
(atoi(time_cmd)*MEASURE_INTERVAL));
                  do {
                        struct msg_ll *e=list_pop(measures_list);
                        free(e);
                  }while(list_length (measures_list) != 0);
            }
      }
}

/*-------------------------------------------------------------------------
-------*/
/* If is time to send a packet this function is invoked*/
static void
send_packet(void)
{
      char buffer_ll[2*sizeof(uint16_t)+limit*sizeof(struct
my_msg_t)];
      char *buffer_llPtr = buffer_ll;
      /*Copy the values of the current parameters in the buffer*/
      memcpy(buffer_llPtr, &time, sizeof(uint16_t));
      memcpy(buffer_llPtr+sizeof(uint16_t), &limit, sizeof(uint16_t));
      /*Copy sensor readings in the buffer*/
      int j = 0;
      struct msg_ll *s;
      for(s = list_head(measures_list); s != NULL; s =
list_item_next(s)) {
            memcpy(buffer_llPtr+2*sizeof(uint16_t)+j*sizeof(struct
my_msg_t), &s->msg, sizeof(struct my_msg_t));
            j++;
      }
      uip_udp_packet_sendto(client_conn, &buffer_ll,
sizeof(buffer_ll),
                        &server_ipaddr, UIP_HTONS(UDP_SERVER_PORT));
}
/*-------------------------------------------------------------------------
-------*/
/* This function read values the sensors and store this data in the
linked list*/
static void
log_packet(void)
{
      uint32_t aux;
      uint32_t aux1;
      uint32_t aux2;
```

```
      counter++;
      struct msg_ll *e;
      e =  malloc(sizeof(struct msg_ll));
      if(e != NULL) {
            e->msg.counter = counter;
            e->msg.temperature = tmp102_read_temp_x100();
            /* Convert the battery reading from ADC units to mV
(powered over USB) */
          aux = battery_sensor.value(0);
          aux *= 5000;
          aux /= 4095;
          e->msg.battery = aux;
          //PRINTF("%s",aux);
          /* External sensor 1 read from ADC */
          aux1 = (-39.60 + 0.01 * sht11_temp());
          aux1 *= 5000;
          aux1 /= 4095;
          e->msg.external1 = aux1;
          //PRINTF("%s",aux1);
          /* External sensor 2 read from ADC */
          aux2 = sht11_humidity();
          aux2 *= 5000;
          aux2 /= 4095;
          e->msg.external2 = aux2;
          //PRINTF("%s",aux2);

          list_add(measures_list, e);
      }
      else {printf("No memory available");}
}
/*---------------------------------------------------------------------
-------*/

static void
print_local_addresses(void)
{
      int i;
      uint8_t state;

      PRINTF("Client IPv6 addresses:\n");
      for(i = 0; i < UIP_DS6_ADDR_NB; i++) {
            state = uip_ds6_if.addr_list[i].state;
            if(uip_ds6_if.addr_list[i].isused &&
                        (state == ADDR_TENTATIVE || state ==
ADDR_PREFERRED)) {
                  PRINT6ADDR(&uip_ds6_if.addr_list[i].ipaddr);
                  PRINTF("\n");
                  /* hack to make address "final" */
                  if (state == ADDR_TENTATIVE) {
                        uip_ds6_if.addr_list[i].state = ADDR_PREFERRED;
                  }
            }
      }
}

/*---------------------------------------------------------------------
-------*/
PROCESS_THREAD(udp_server_process, ev, data)
{
      PROCESS_BEGIN();
      //powertrace_start(CLOCK_SECOND*10);
```

```
      PROCESS_PAUSE();

      printf("UDP server process started\n");

      /* Raspberry IPv6 address bbbb::388e:bd5:1147:2035*/
      //uip_ip6addr(&server_ipaddr, 0xbbbb, 0, 0, 0, 0x388e, 0xbd5,
0x1147, 0x2035);
      /* Rpi IPv6 address bbbb::4c22:4b04:d9:ef06 */
      uip_ip6addr(&server_ipaddr, 0xbbbb, 0, 0, 0, 0x4c22, 0x4b04,
0xd9, 0xef06);

      /* Print the server's addresses */
      printf("Server address: ");
      PRINT6ADDR(&server_ipaddr);
      printf("\n");
      /* Print the node's addresses */
      print_local_addresses();
      /* Activate the sensors */
      tmp102_init();
      sht11_init();
      (battery_sensor);
      SENSORS_ACTIVATE(button_sensor);
      /* Create a new UDP connection*/
      client_conn = udp_new(NULL, UIP_HTONS(UDP_SERVER_PORT), NULL);
      if(client_conn == NULL) {
            PRINTF("No UDP connection available, exiting the
process!\n");
            PROCESS_EXIT();
      }
      /* This function binds a UDP connection to a specified local
port*/
      udp_bind(client_conn, UIP_HTONS(UDP_CLIENT_PORT));

      PRINTF("Created a connection with the server ");
      PRINT6ADDR(&client_conn->ripaddr);
      PRINTF(" local/remote port %u/%u\n", UIP_HTONS(client_conn-
>lport),
                                          UIP_HTONS(client_conn->rport));
      /* Set the timer to default value */
      etimer_set(&periodic, MEASURE_INTERVAL);
      /* Initialize the list used to log measures. */
      list_init(measures_list);
      PRINTF("Entro en el while");
      while(1) {
            PROCESS_YIELD();
            /* Incoming events from the TCP/IP module, when there is a
new packet from the server*/
            if(ev == tcpip_event) {
                  tcpip_handler();
            }
            /* Send data to the server */
            if((ev == sensors_event && data == &button_sensor) ||
                        (ev == PROCESS_EVENT_TIMER)) {
                  counter_send++;
                  log_packet();
                  if (counter_send==limit){
                        send_packet();
                        counter_send=0;
                        do {
                              struct msg_ll *e=list_pop(measures_list);
                              free(e);
```

```
                        }while(list_length (measures_list) != 0);
                }
                if(etimer_expired(&periodic)) {
                        etimer_reset(&periodic);
                }
        }
    }
    PROCESS_END();
}
/*---------------------------------------------------------------------
-------*/
```

III. Ejemplo de uso del sensor externo ZIG002

```
/*****************************************************************
 *
 * NODE PROGRAM - UDP
 *
*****************************************************************/

/*---------------------------------------------------------------------*/
#include "contiki.h"
#include "lib/list.h"
#include "sys/ctimer.h"
#include "net/ip/uip.h"
#include "net/ipv6/uip-ds6.h"
#include "net/ip/uip-udp-packet.h"
#include "../example2.h"
#include <stdio.h>
#include <string.h>

/* Library needed to use sensors in Z1 */
#include "dev/battery-sensor.h"
#include "dev/i2cmaster.h"
#include "dev/tmp102.h"
#include "dev/button-sensor.h"
#include "dev/light-ziglet.h"

/*Lib to estimate power consumption */
//#include "powertrace.h"


/*---------------------------------------------------------------------
-------*/
/* Enables printing debug output from the IP/IPv6 libraries */
#define DEBUG DEBUG_PRINT
#include "net/ip/uip-debug.h"
/*---------------------------------------------------------------------
-------*/
/* Default is to send a packet every 60 seconds */
#define MEASURE_INTERVAL            (5 * CLOCK_SECOND)
static uint16_t limit=1;
static uint16_t time=1;
/*---------------------------------------------------------------------
-------*/
/* The structure used in the Simple UDP library to create an UDP
connection */
static struct uip_udp_conn *client_conn;

/* This is the server IPv6 address */
```

```
static uip_ipaddr_t server_ipaddr;

/* Keeps account of the number of messages sent */
static uint16_t counter = 0;

static uint16_t counter_send=0;
static struct etimer periodic;

/*---------------------------------------------------------------------
-------*/
/* Create a linked list of data structures to store the data to be
sent as payload */
LIST(measures_list);
/* Structure for the linked list*/
struct msg_ll {
      struct msg_ll *next;
      struct my_msg_t msg;
};


/*---------------------------------------------------------------------
-------*/
PROCESS(udp_server_process, "UDP server example process");
AUTOSTART_PROCESSES(&udp_server_process);
/*---------------------------------------------------------------------
-------*/

/* Whenever the node receive a packet from the server, this function
is invoked
 * "uip_newdata()" is used to check if there is actually data in the
buffer
 * This function read the new parameters and sets them */
static void
tcpip_handler(void)
{
      char *str;

      if(uip_newdata()) {
            str = uip_appdata;
            str[uip_datalen()] = '\0';

            char *time_cmd = strtok(str," "); //first_part points to
"time"
            char *limit_cmd = strtok(NULL," ");   //sec_part points to
"limit"

            if (time_cmd != NULL || limit_cmd!=NULL){
                  time = atoi(time_cmd);
                  limit = atoi(limit_cmd);
                  counter_send=0;
                  etimer_reset(&periodic);
                  etimer_set(&periodic,
(atoi(time_cmd)*MEASURE_INTERVAL));
                  do {
                        struct msg_ll *e=list_pop(measures_list);
                        free(e);
                  }while(list_length (measures_list) != 0);
            }
      }
}
```

```
/*---------------------------------------------------------------------
-------*/
/* If is time to send a packet this function is invoked*/
static void
send_packet(void)
{
      char buffer_ll[2*sizeof(uint16_t)+limit*sizeof(struct
my_msg_t)];
      char *buffer_llPtr = buffer_ll;
      /*Copy the values of the current parameters in the buffer*/
      memcpy(buffer_llPtr, &time, sizeof(uint16_t));
      memcpy(buffer_llPtr+sizeof(uint16_t), &limit, sizeof(uint16_t));
      /*Copy sensor readings in the buffer*/
      int j = 0;
      struct msg_ll *s;
      for(s = list_head(measures_list); s != NULL; s =
list_item_next(s)) {
            memcpy(buffer_llPtr+2*sizeof(uint16_t)+j*sizeof(struct
my_msg_t), &s->msg, sizeof(struct my_msg_t));
            j++;
      }
      uip_udp_packet_sendto(client_conn, &buffer_ll,
sizeof(buffer_ll),
                        &server_ipaddr, UIP_HTONS(UDP_SERVER_PORT));
}
/*---------------------------------------------------------------------
-------*/
/* This function read values the sensors and store this data in the
linked list*/
static void
log_packet(void)
{
      uint32_t aux;
      uint32_t aux1;
      counter++;
      struct msg_ll *e;
      e =  malloc(sizeof(struct msg_ll));
      if(e != NULL) {
            e->msg.counter = counter;
            e->msg.temperature = tmp102_read_temp_x100();
            /* Convert the battery reading from ADC units to mV
(powered over USB) */
          aux = battery_sensor.value(0);
          aux *= 5000;
          aux /= 4095;
          e->msg.battery = aux;
          //PRINTF("%s",aux);
          /* External sensor 1 read from ADC */
          aux1 = light_ziglet_read();
          aux1 *= 5000;
          aux1 /= 4095;
          e->msg.external1 = aux1;
          //PRINTF("%s",aux1);

          list_add(measures_list, e);
      }
      else {printf("No memory available");}
}
/*---------------------------------------------------------------------
-------*/
```

```
static void
print_local_addresses(void)
{
      int i;
      uint8_t state;

      PRINTF("Client IPv6 addresses:\n");
      for(i = 0; i < UIP_DS6_ADDR_NB; i++) {
            state = uip_ds6_if.addr_list[i].state;
            if(uip_ds6_if.addr_list[i].isused &&
                        (state == ADDR_TENTATIVE || state ==
ADDR_PREFERRED)) {
                  PRINT6ADDR(&uip_ds6_if.addr_list[i].ipaddr);
                  PRINTF("\n");
                  /* hack to make address "final" */
                  if (state == ADDR_TENTATIVE) {
                        uip_ds6_if.addr_list[i].state = ADDR_PREFERRED;
                  }
            }
      }
}

/*---------------------------------------------------------------------
-------*/
PROCESS_THREAD(udp_server_process, ev, data)
{
      PROCESS_BEGIN();
      //powertrace_start(CLOCK_SECOND*10);
      PROCESS_PAUSE();

      printf("UDP server process started\n");

      /* Raspberry IPv6 address bbbb::388e:bd5:1147:2035*/
      //uip_ip6addr(&server_ipaddr, 0xbbbb, 0, 0, 0, 0x388e, 0xbd5,
0x1147, 0x2035);
      /* Rpi IPv6 address bbbb::4c22:4b04:d9:ef06 */
      uip_ip6addr(&server_ipaddr, 0xbbbb, 0, 0, 0, 0x4c22, 0x4b04,
0xd9, 0xef06);

      /* Print the server's addresses */
      printf("Server address: ");
      PRINT6ADDR(&server_ipaddr);
      printf("\n");
      /* Print the node's addresses */
      print_local_addresses();
      /* Activate the sensors */
      tmp102_init();
      light_ziglet_init();
      (battery_sensor);
      SENSORS_ACTIVATE(button_sensor);
      /* Create a new UDP connection*/
      client_conn = udp_new(NULL, UIP_HTONS(UDP_SERVER_PORT), NULL);
      if(client_conn == NULL) {
            PRINTF("No UDP connection available, exiting the
process!\n");
            PROCESS_EXIT();
      }
      /* This function binds a UDP connection to a specified local
port*/
      udp_bind(client_conn, UIP_HTONS(UDP_CLIENT_PORT));
```

```
      PRINTF("Created a connection with the server ");
      PRINT6ADDR(&client_conn->ripaddr);
      PRINTF(" local/remote port %u/%u\n", UIP_HTONS(client_conn-
>lport),
                                          UIP_HTONS(client_conn->rport));
      /* Set the timer to default value */
      etimer_set(&periodic, MEASURE_INTERVAL);
      /* Initialize the list used to log measures. */
      list_init(measures_list);
      PRINTF("Entro en el while");
      while(1) {
            PROCESS_YIELD();
            /* Incoming events from the TCP/IP module, when there is a
new packet from the server*/
            if(ev == tcpip_event) {
                  tcpip_handler();
            }
            /* Send data to the server */
            if((ev == sensors_event && data == &button_sensor) ||
                        (ev == PROCESS_EVENT_TIMER)) {
                  counter_send++;
                  log_packet();
                  if (counter_send==limit){
                        send_packet();
                        counter_send=0;
                        do {
                              struct msg_ll *e=list_pop(measures_list);
                              free(e);
                        }while(list_length (measures_list) != 0);
                  }
                  if(etimer_expired(&periodic)) {
                        etimer_reset(&periodic);
                  }
            }
      }
      PROCESS_END();
}
/*---------------------------------------------------------------------
-------*/
```

Anexo III. Implementación Servidor UDP

I. UDP Python Server

```
# ------------------------------------------------------------#
# UDP Server UPDATE 2.0
# ------------------------------------------------------------#
import sys
import datetime
from datetime import datetime, timedelta
import time
from socket import *
from socket import error
from time import sleep
from ctypes import *
import MySQLdb
import csv
from textwrap import wrap

# ------------------------------------------------------------#
ID_STRING = "V0.1"
# ------------------------------------------------------------#
PORT = 5678
CMD_PORT = 8765
BUFSIZE = 1024
# ------------------------------------------------------------#
# If you need to save the received data in a mysql database
# then enable by equal to 1
LOG_IN_MYSQL = 1
PRINT_DATA = 1
# ------------------------------------------------------------#
# Message structure
# ------------------------------------------------------------#

class SENSOR(Structure):
    _pack_ = 1
    _fields_ = [
        ("counter", c_uint16),
        ("temperature", c_int16),
        ("battery", c_uint16),
        ("external1", c_uint16),
        ("external2", c_uint16)
    ]

    def __new__(self, socket_buffer):
        return self.from_buffer_copy(socket_buffer)

    def __init__(self, socket_buffer):
        pass

# ------------------------------------------------------------#


class PARAM(Structure):
    _pack_ = 1
    _fields_ = [
        ("time", c_uint16),
        ("limit", c_uint16)
    ]

    def __new__(self, socket_buffer):
```

```
        return self.from_buffer_copy(socket_buffer)

    def __init__(self, socket_buffer):
        pass


# --------------------------------------------------------------#
# This function prints received data
# --------------------------------------------------------------#
def print_recv_data(msg):
    print "***"
    for f_name, f_type in msg._fields_:
        print "{0}:{1} ".format(f_name, getattr(msg, f_name)),
    print


# --------------------------------------------------------------#
# This function saves received data in the database
# --------------------------------------------------------------#
def save_recv_data(db, node, tdatetime, msg_recv):
    curs = db.cursor()
    node = (node.split(':'))[-1]
    try:
        print "Connected to dababase"
        curs.execute("""INSERT INTO Nodes
                                    (node_id)
                                          SELECT %s
                                          FROM dual
                                          WHERE NOT EXISTS (SELECT *

      FROM Nodes

      WHERE node_id = %s); """, (node, node))

        temp = msg_recv.temperature / float(100)
        batt = msg_recv.battery
        external1 = msg_recv.external1
        external2 = msg_recv.external2

        curs.execute("""INSERT INTO
Measures(node_id,temperature,battery,external1,external2,time)
                                          values(%s, %s, %s, %s, %s,
%s)""", (node, temp, batt, external1, external2, tdatetime))
        db.commit()
        print "Data committed"

    except MySQLdb.Error, e:
        try:
            print "MySQL Error [%d]: %s" % (e.args[0], e.args[1])
        except IndexError:
            print "MySQL Error: %s" % str(e)
        print "Error: the database is being rolled back"
        db.rollback()
    curs.close()

# --------------------------------------------------------------#
# This function saves received data in CSV file
# --------------------------------------------------------------#

def log_rcv_data(db, node, tdatetime, msg_recv):
```

```
    node = (node.split(':'))[-1]
    temp = msg_recv.temperature / float(100)
    batt = msg_recv.battery
    external1 = msg_recv.external1
    external2 = msg_recv.external2

    with open('%s_%s.csv' % (node,time.strftime("%d-%m-%Y")), 'a') as
csvfile:

        writer = csv.writer(csvfile, delimiter=',')


writer.writerow([node,temp,batt,external1,external2,tdatetime])

        print "Data logged in %s_%s.csv" % (node,time.strftime("%d-%m-
%Y"))

# -----------------------------------------------------------#
# This function saves new parameters sent by the user
# -----------------------------------------------------------#
def save_web_request(db, node, time, limit):
    curs = db.cursor()
    try:
        curs.execute("UPDATE nodedat SET time_new=%s, limit_new=%s
WHERE node=%s;", (time, limit, node))
        db.commit()
        curs.close()
    except MySQLdb.Error, e:
        try:
            print "MySQL Error [%d]: %s" % (e.args[0], e.args[1])
        except IndexError:
            print "MySQL Error: %s" % str(e)
        print "Error: the database is being rolled back"
        db.rollback()


# -----------------------------------------------------------#
# This function saves actual parameters sent by the node
# -----------------------------------------------------------#
def save_current_params(db, node, time, limit):
    curs = db.cursor()
    try:
        curs.execute("UPDATE nodedat SET time_current=%s,
limit_current=%s WHERE node=%s;", (time, limit, node))
        db.commit()
        curs.close()
    except MySQLdb.Error, e:
        try:
            print "MySQL Error [%d]: %s" % (e.args[0], e.args[1])
        except IndexError:
            print "MySQL Error: %s" % str(e)
        print "Error: the database is being rolled back"
        db.rollback()


# -----------------------------------------------------------#
# This function updates the parameters in the nodes
# -----------------------------------------------------------#
def change_node_conf(db, s, node):
    curs = db.cursor()
    try:
```

```
        print "Connected to dababase"
        curs.execute("""SELECT node, time_new, limit_new
                                        FROM nodedat
                                        WHERE time_current !=
time_new
                                        OR limit_current !=
limit_new;""")
        node_tb_chgd = [list(i) for i in curs.fetchall()]
        for i in node_tb_chgd:
            send_udp_cmd(s, i[0], i[1], i[2])
            curs.execute("UPDATE nodedat SET time_current=%s,
limit_current=%s WHERE node=%s;", (i[1], i[2], i[0]))
        db.commit()
        s.sendto("", (node, CMD_PORT))
        print "Data committed"
    except MySQLdb.Error, e:
        try:
            print "MySQL Error [%d]: %s" % (e.args[0], e.args[1])
        except IndexError:
            print "MySQL Error: %s" % str(e)
        print "Error: the database is being rolled back"
        db.rollback()
    curs.close()


# ------------------------------------------------------------#
# This function sends the parameters to the nodes
# at any time
# ------------------------------------------------------------#
def send_udp_cmd(s, addr, t, l):
    print "Sending params to " + str(addr)
    try:
        s.sendto(str(t) + " " + str(l), (addr, CMD_PORT))
    except Exception as error:
        print error


# ------------------------------------------------------------#
# This is the main function, execute the main server and call
associate
# ------------------------------------------------------------#
def start_server():
    now = datetime.now()
    print "UDP6 server side application " + ID_STRING
    print "Started " + str(now)
    try:
        s = socket(AF_INET6, SOCK_DGRAM)
        s.setsockopt(SOL_SOCKET, SO_REUSEADDR, 1)
        s.bind(('', PORT))
    except Exception:
        print "ERROR: Server Port Binding Failed"
        return

    print 'UDP6 server ready: %s' % PORT
    print "msg structure size: ", sizeof(SENSOR)

    while True:
            # Connection to Database
        if LOG_IN_MYSQL: db = MySQLdb.connect("localhost", "root",
"root", "TFG_Database")
        data, addr = s.recvfrom(BUFSIZE)
```

```
        now = datetime.now()

        print
"****************************UDP*********************************"
        print str(now)[:19] + " -> " + str(addr[0]) + " PORT " +
str(addr[1]) + " " + str(len(data))
        node = addr[0]
        # This is executed when the data comes from the nodes
        else (len(data) - 4) % 10 == 0 and addr[1] == 8765:
            print "Start"
            params = data[0:4]
            msg_params = PARAM(params)
            if PRINT_DATA:
                print "print_recv_data"
                print_recv_data(msg_params)
            if LOG_IN_MYSQL:
                data = data[4:len(data)]
                split_data = [data[x:x + 10] for x in range(0,
len(data), 10)]
                i = 0

                for data in split_data:
                    msg_recv = SENSOR(data)
                    if PRINT_DATA:
                        print_recv_data(msg_recv)
                    if LOG_IN_MYSQL:
                        tdatetime = (now - timedelta(minutes=i *
int(msg_params.time))).strftime('%Y-%m-%d %H:%M:%S')
                        save_recv_data(db, node, tdatetime, msg_recv)
                        log_rcv_data(db, node, tdatetime, msg_recv)
                        i = i + 1

            else:
                print "Unknown data format"
            if LOG_IN_MYSQL:
                db.close()

# ------------------------------------------------------------#


# MAIN APP
# ------------------------------------------------------------#
if __name__ == "__main__":
    start_server()
```

Anexo IV. Desarrollo para la visualización en Android

I. Scripts PHP para proporcionar el servicio web

- Gestión de la base de datos

```
<?php
/**
 * Clase que envuelve una instancia de la clase PDO
 * para el manejo de la base de datos
 */
require_once 'mysql_login.php';
class Database
{
    /**
     * Única instancia de la clase
     */
    private static $db = null;
    /**
     * Instancia de PDO
     */
    private static $pdo;
    final private function __construct()
    {
        try {
            // Crear nueva conexión PDO
            self::getDb();
        } catch (PDOException $e) {
            // Manejo de excepciones
        }
    }
    /**
     * Retorna en la única instancia de la clase
     * @return Database|null
     */
    public static function getInstance()
    {
        if (self::$db === null) {
            self::$db = new self();
        }
        return self::$db;
    }
    /**
     * Crear una nueva conexión PDO basada
     * en los datos de conexión
     * @return PDO Objeto PDO
     */
    public function getDb()
    {
        if (self::$pdo == null) {
            self::$pdo = new PDO(
                'mysql:dbname=' . DATABASE .
                ';host=' . HOSTNAME .
                ';port:63343;', // Eliminar este elemento si se usa
una instalación por defecto
                USERNAME,
                PASSWORD,
                array(PDO::MYSQL_ATTR_INIT_COMMAND => "SET NAMES
utf8")
```

```
            );
            // Habilitar excepciones
            self::$pdo->setAttribute(PDO::ATTR_ERRMODE,
PDO::ERRMODE_EXCEPTION);
        }
        return self::$pdo;
    }
    /**
     * Evita la clonación del objeto
     */
    final protected function __clone()
    {
    }
    function _destructor()
    {
        self::$pdo = null;
    }
}
?>
```

- Conexión a la base de datos

```
<?php
/**
 * Provee las constantes para conectarse a la base de datos
 * Mysql.
 */
define("HOSTNAME", "localhost");// Nombre del host
define("DATABASE", "TFG_Database"); // Nombre de la base de datos
define("USERNAME", "root"); // Nombre del usuario
define("PASSWORD", "root"); // Nombre de la constraseña
?>
```

- Script para la clase Nodos

```
<?php
/**
 * Representa el la estructura de los nodos
 * almacenadas en la base de datos
 */
require 'Database.php';
class Nodes
{
    function __construct()
    {
    }
    /**
     * Retorna todos los nodos de la tabla 'Nodes'
     * @return array Datos del registro
     */
    public static function getAll()
    {
        $consulta = "SELECT node_id FROM Nodes";
        try {
            // Preparar sentencia
            $comando = Database::getInstance()->getDb()-
>prepare($consulta);
            // Ejecutar sentencia preparada
            $comando->execute();
            return $comando->fetchAll(PDO::FETCH_ASSOC);
```

```
        } catch (PDOException $e) {
            return false;
        }
    }
    /**
     * Actualiza un registro de la bases de datos basado
     * en los nuevos valores relacionados con un identificador
     * Activa el nodo
     * @param $node_id      identificador
     */
    public static function activate(
        $node_id,
      $is_active
    )
    {
        // Creando consulta UPDATE
        $consulta = "UPDATE Nodes" .
            " SET is_active=? " .
            "WHERE node_id=?";
        // Preparar la sentencia
        $cmd = Database::getInstance()->getDb()->prepare($consulta);
        // Relacionar y ejecutar la sentencia
        $cmd->execute(array($is_active, $node_id));
        return $cmd;
    }
    /**
     * Actualiza un registro de la bases de datos basado
     * en los nuevos valores relacionados con un identificador
     * Desactiva el nodo
     * @param $node_id      identificador
     */
    public static function deactivate(
        $node_id,
      $is_active
    )
    {
        // Creando consulta UPDATE
        $consulta = "UPDATE Nodes" .
            " SET is_active=?" .
            "WHERE node_id=?";
        // Preparar la sentencia
        $cmd = Database::getInstance()->getDb()->prepare($consulta);
        // Relacionar y ejecutar la sentencia
        $cmd->execute(array($is_active, $node_id));
        return $cmd;
    }
}
?>
```

- Script para la clase Medidas

```
<?php
/**
 * Representa el la estructura de las Medidas
 * almacenadas en la base de datos
 */
require 'Database.php';
class Measures
{
    function __construct()
```

```
    {
    }

    /**
     * Obtiene los campos de un nodo con un identificador
     * determinado
     *
     * @param $id_node Identificador del nodo
     * @return mixed
     */

    public static function getTempById($node_id)
    {
        $consulta = "SELECT node_id, temperature, time
                  FROM Measures
                        WHERE node_id = ?
                        ORDER BY time";
        try {
            // Preparar sentencia
            $comando = Database::getInstance()->getDb()-
>prepare($consulta);
            // Ejecutar sentencia preparada
            //$comando->execute();
            $comando->execute(array($node_id));
            return $comando->fetchAll(PDO::FETCH_ASSOC);
        } catch (PDOException $e) {
            return -1;
        }
    }

    public static function getBatById($node_id)
    {
        $consulta = "SELECT node_id, battery, time
                  FROM Measures
                        WHERE node_id = ?
                        ORDER BY time";
        try {
            // Preparar sentencia
            $comando = Database::getInstance()->getDb()-
>prepare($consulta);
            // Ejecutar sentencia preparada
            //$comando->execute();
            $comando->execute(array($node_id));
            return $comando->fetchAll(PDO::FETCH_ASSOC);
        } catch (PDOException $e) {
            return -1;
        }
    }

    public static function getExt1ById($node_id)
    {
        $consulta = "SELECT node_id, external1, time
                  FROM Measures
                        WHERE node_id = ?
                        ORDER BY time";
        try {
            // Preparar sentencia
            $comando = Database::getInstance()->getDb()-
>prepare($consulta);
            // Ejecutar sentencia preparada
            //$comando->execute();
```

```
            $comando->execute(array($node_id));
            return $comando->fetchAll(PDO::FETCH_ASSOC);
        } catch (PDOException $e) {
            return -1;
        }
    }

    public static function getExt2ById($node_id)
    {
        $consulta = "SELECT node_id, external2, time
                FROM Measures
                      WHERE node_id = ?
                      ORDER BY time";
        try {
            // Preparar sentencia
            $comando = Database::getInstance()->getDb()-
>prepare($consulta);
            // Ejecutar sentencia preparada
            //$comando->execute();
            $comando->execute(array($node_id));
            return $comando->fetchAll(PDO::FETCH_ASSOC);
        } catch (PDOException $e) {
            return -1;
        }
    }
}
?>
```

- Scripts para llamar a las funciones de consulta de datos declaradas en los scripts de clase

```
<?php
/**
 * Obtiene el detalle de un nodo especificado por
 * su identificador "node_id"
 */
require 'Measures.php';
if ($_SERVER['REQUEST_METHOD'] == 'GET') {
    if (isset($_GET['node_id'])) {
        // Obtener parámetro node_id
        $parametro = $_GET['node_id'];
        // Tratar retorno
        $retorno = Measures::getTempById($parametro);
        if ($retorno) {
            $medida["estado"] = 1;        // cambio "1" a 1 porque no
coge bien la cadena.
            $medida["medida"] = $retorno;
            // Enviar objeto json
            print json_encode($medida);
        } else {
            // Enviar respuesta de error general
            print json_encode(
                array(
                    'estado' => 2,
                    'mensaje' => 'No se obtuvo el registro'
                )
            );
        }
    } else {
        // Enviar respuesta de error
```

```
        print json_encode(
            array(
                'estado' => 3,
                'mensaje' => 'Se necesita un identificador'
            )
        );
    }
}
```

```
<?php
/**
 * Obtiene el detalle de un nodo especificado por
 * su identificador "node_id"
 */
require 'Measures.php';
if ($_SERVER['REQUEST_METHOD'] == 'GET') {
    if (isset($_GET['node_id'])) {
        // Obtener parámetro node_id
        $parametro = $_GET['node_id'];
        // Tratar retorno
        $retorno = Measures::getBatById($parametro);
        if ($retorno) {
            $medida["estado"] = 1;        // cambio "1" a 1 porque no
coge bien la cadena.
            $medida["medida"] = $retorno;
            // Enviar objeto json
            print json_encode($medida);
        } else {
            // Enviar respuesta de error general
            print json_encode(
                array(
                    'estado' => 2,
                    'mensaje' => 'No se obtuvo el registro'
                )
            );
        }
    } else {
        // Enviar respuesta de error
        print json_encode(
            array(
                'estado' => 3,
                'mensaje' => 'Se necesita un identificador'
            )
        );
    }
}
```

```
<?php
/**
 * Obtiene el detalle de un nodo especificado por
 * su identificador "node_id"
 */
require 'Measures.php';
if ($_SERVER['REQUEST_METHOD'] == 'GET') {
    if (isset($_GET['node_id'])) {
```

```
        // Obtener parámetro node_id
        $parametro = $_GET['node_id'];
        // Tratar retorno
        $retorno = Measures::getExt1ById($parametro);
        if ($retorno) {
            $medida["estado"] = 1;        // cambio "1" a 1 porque no
coge bien la cadena.
            $medida["medida"] = $retorno;
            // Enviar objeto json
            print json_encode($medida);
        } else {
            // Enviar respuesta de error general
            print json_encode(
                array(
                    'estado' => 2,
                    'mensaje' => 'No se obtuvo el registro'
                )
            );
        }
    } else {
        // Enviar respuesta de error
        print json_encode(
            array(
                'estado' => 3,
                'mensaje' => 'Se necesita un identificador'
            )
        );
    }
}
```

```
<?php
/**
 * Obtiene el detalle de un nodo especificado por
 * su identificador "node_id"
 */
require 'Measures.php';
if ($_SERVER['REQUEST_METHOD'] == 'GET') {
    if (isset($_GET['node_id'])) {
        // Obtener parámetro node_id
        $parametro = $_GET['node_id'];
        // Tratar retorno
        $retorno = Measures::getExt2ById($parametro);
        if ($retorno) {
            $medida["estado"] = 1;        // cambio "1" a 1 porque no
coge bien la cadena.
            $medida["medida"] = $retorno;
            // Enviar objeto json
            print json_encode($medida);
        } else {
            // Enviar respuesta de error general
            print json_encode(
                array(
                    'estado' => 2,
                    'mensaje' => 'No se obtuvo el registro'
                )
            );
        }
    } else {
```

```
        // Enviar respuesta de error
        print json_encode(
            array(
                'estado' => 3,
                'mensaje' => 'Se necesita un identificador'
            )
        );
    }
}
```

```
<?php
/**
 * Obtiene todos los nodos de la base de datos
 */
require 'Nodes.php';
if ($_SERVER['REQUEST_METHOD'] == 'GET') {
    // Manejar petición GET
    $nodes = Nodes::getAll();
    if ($nodes) {
        $datos["estado"] = 1;
        $datos["nodes"] = $nodes;
        print json_encode($datos);
    } else {
        print json_encode(array(
            "estado" => 2,
            "mensaje" => "Ha ocurrido un error"
        ));
    }
}
```

II. Código Java para inicializar la aplicación

```
package com.example.user.sensormonitoring_v30;

import android.Manifest;
import android.content.Intent;
import android.content.pm.PackageManager;
import android.os.AsyncTask;
import android.os.Build;
import android.os.Bundle;
import android.support.v4.app.ActivityCompat;
import android.support.v7.app.AppCompatActivity;
import android.support.v7.widget.Toolbar;
import android.util.Log;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.Spinner;

import org.json.JSONArray;
import org.json.JSONException;
import org.json.JSONObject;

import java.io.BufferedInputStream;
import java.io.BufferedReader;
import java.io.BufferedWriter;
```

```
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.OutputStream;
import java.io.OutputStreamWriter;
import java.net.HttpURLConnection;
import java.net.MalformedURLException;
import java.net.URL;
import java.util.ArrayList;

import static android.R.layout.simple_spinner_item;

public class MainActivity extends AppCompatActivity implements
View.OnClickListener{

    Button temp;
    Button bat;
    Button csv;
    Button ext1;
    Button ext2;
    Spinner spinner;

    // IP (raspberry pi host)
    String IP = "http://tfghost.hopto.org";
    // Web Services
    String GET_NODES = IP + "/get_nodes.php";
    String ACTIVATE = IP + "/activate.php";

    GetNodes connectionthread;
    ActivateNode hilodeconexion;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);

        //Link to visual element on XML
        temp = (Button)findViewById(R.id.temp);
        bat = (Button)findViewById(R.id.bat);
        csv = (Button) findViewById(R.id.csv);
        ext1 = (Button) findViewById(R.id.ext1);
        ext2 = (Button) findViewById(R.id.ext2);
        spinner = (Spinner)findViewById(R.id.spinner);

        //Button listener
        temp.setOnClickListener(this);
        bat.setOnClickListener(this);
        csv.setOnClickListener(this);
        ext1.setOnClickListener(this);
        ext2.setOnClickListener(this);

        //execute getNodes --> set data on the list (spinner)
        connectionthread = new GetNodes();
        connectionthread.execute(GET_NODES); //param for
doInBackground (URL)

        isStoragePermissionGranted();
```

```
    }

    public  boolean isStoragePermissionGranted() {
        if (Build.VERSION.SDK_INT >= 23) {
            if
(checkSelfPermission(android.Manifest.permission.WRITE_EXTERNAL_STORAG
E)
                    == PackageManager.PERMISSION_GRANTED) {
                Log.v("TAG","Permission is granted");
                return true;
            } else {

                Log.v("TAG","Permission is revoked");
                ActivityCompat.requestPermissions(this, new
String[]{Manifest.permission.WRITE_EXTERNAL_STORAGE}, 1);
                return false;
            }
        }
        else { //permission is automatically granted on sdk<23 upon
installation
            Log.v("TAG","Permission is granted");
            return true;
        }
    }

    @Override
    public void onRequestPermissionsResult(int requestCode, String[]
permissions, int[] grantResults) {
        super.onRequestPermissionsResult(requestCode, permissions,
grantResults);
        if(grantResults[0]== PackageManager.PERMISSION_GRANTED){
            Log.v("TAG","Permission: "+permissions[0]+ "was
"+grantResults[0]);
            //resume tasks needing this permission
        }
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if it
is present.
        getMenuInflater().inflate(R.menu.menu_main, menu);
        return true;
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        // Handle action bar item clicks here. The action bar will
        // automatically handle clicks on the Home/Up button, so long
        // as you specify a parent activity in AndroidManifest.xml.
        int id = item.getItemId();

        //noinspection SimplifiableIfStatement
        if (id == R.id.action_settings) {
            return true;
        }

        return super.onOptionsItemSelected(item);
    }

    public class GetNodes extends AsyncTask<String,Void,String> {
```

```
        //Array para guardar los nodos
        ArrayList<String> nodesArray = new ArrayList<>();

        @Override
        protected void onPreExecute() {
            super.onPreExecute();
        }

        @Override
        protected String doInBackground(String... params) {

            String cadena = params[0];
            URL url = null;
            String nodes = "";

            try {
                url = new URL(cadena);
                HttpURLConnection connection = (HttpURLConnection)
url.openConnection(); //Open connection

                int respuesta = connection.getResponseCode();
                StringBuilder result = new StringBuilder();

                if (respuesta == HttpURLConnection.HTTP_OK){

                    InputStream in = new
BufferedInputStream(connection.getInputStream());  // preparo la
cadena de entrada

                    BufferedReader reader = new BufferedReader(new
InputStreamReader(in));  // la introduzco en un BufferedReader

                    // El siguiente proceso lo hago porque el
JSONOBject necesita un String y tengo
                    // que tranformar el BufferedReader a String. Esto
lo hago a traves de un
                    // StringBuilder.

                    String line;
                    while ((line = reader.readLine()) != null) {
                        result.append(line);        // Paso toda la
entrada al StringBuilder
                    }

                    //Creamos un objeto JSONObject para poder acceder
a los atributos (campos) del objeto.
                    JSONObject respuestaJSON = new
JSONObject(result.toString());   //Creo un JSONObject a partir del
StringBuilder pasado a cadena

                    //Accedemos al vector de resultados
                    String resultJSON =
respuestaJSON.getString("estado");   // estado es el nombre del
campo(tag) en el JSON

                    if (resultJSON.equals("1")){      // hay datos a
mostrar
                        JSONArray medidasJSON =
respuestaJSON.getJSONArray("nodes");   // nodes es el nombre del campo
en el JSON
```

```
                        for(int i=0;i<medidasJSON.length();i++){

nodesArray.add(medidasJSON.getJSONObject(i).getString("node_id"));
                        }
                    }

                    else if (resultJSON.equals("2")){
                        nodesArray.add("No hay nodos disponibles en la
red");
                    }

                }
            } catch (MalformedURLException e) {
                e.printStackTrace();
            } catch (IOException e) {
                e.printStackTrace();
            } catch (JSONException e) {
                e.printStackTrace();
            }

            return null;
        }

        @Override
        protected void onProgressUpdate(Void... values) {
            super.onProgressUpdate(values);
        }

        @Override
        protected void onPostExecute(String s) {
            super.onPostExecute(s);

            // Application of the Array to the Spinner
            ArrayAdapter adapter = new
ArrayAdapter<>(getApplicationContext(), simple_spinner_item,
nodesArray);
            spinner.setAdapter(adapter);

adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdo
wn_item); // The drop down view

        }

        @Override
        protected void onCancelled(String s) {
            super.onCancelled(s);
        }

    }

    @Override
    public void onClick(View v) {

        String active = "Y";
        String activate_link = ACTIVATE;

        switch (v.getId()) {
            case R.id.temp:

                hilodeconexion = new ActivateNode();
```

```
                //String link = ACTIVATE; //+ "?node_id=" +
spinner.getSelectedItem().toString();
                //String active = "Y";
                hilodeconexion.execute(activate_link,
spinner.getSelectedItem().toString(), active);

                Intent myIntent = new Intent(MainActivity.this,
ChartActivity.class);
                myIntent.putExtra("spinner_value",
spinner.getSelectedItem().toString()); //Optional parameters
                myIntent.putExtra("measure_type", "1");
                MainActivity.this.startActivity(myIntent);

                break;

            case R.id.bat:

                hilodeconexion = new ActivateNode();
                //String link = ACTIVATE; //+ "?node_id=" +
spinner.getSelectedItem().toString();
                //String active = "Y";
                hilodeconexion.execute(activate_link,
spinner.getSelectedItem().toString(), active);

                Intent miIntent = new Intent(MainActivity.this,
ChartActivity.class);
                miIntent.putExtra("spinner_value",
spinner.getSelectedItem().toString()); //Optional parameters
                miIntent.putExtra("measure_type", "2");
                MainActivity.this.startActivity(miIntent);

                break;

            case R.id.ext1:

                hilodeconexion = new ActivateNode();
                //String link = ACTIVATE; //+ "?node_id=" +
spinner.getSelectedItem().toString();
                //String active = "Y";
                hilodeconexion.execute(activate_link,
spinner.getSelectedItem().toString(), active);

                Intent newIntent = new Intent(MainActivity.this,
ChartActivity.class);
                newIntent.putExtra("spinner_value",
spinner.getSelectedItem().toString()); //Optional parameters
                newIntent.putExtra("measure_type", "3");
                MainActivity.this.startActivity(newIntent);

                break;

            case R.id.ext2:

                hilodeconexion = new ActivateNode();
                //String link = ACTIVATE; //+ "?node_id=" +
spinner.getSelectedItem().toString();
                //String active = "Y";
                hilodeconexion.execute(activate_link,
spinner.getSelectedItem().toString(), active);
```

```
                Intent newIntent2 = new Intent(MainActivity.this,
ChartActivity.class);
                newIntent2.putExtra("spinner_value",
spinner.getSelectedItem().toString()); //Optional parameters
                newIntent2.putExtra("measure_type", "4");
                MainActivity.this.startActivity(newIntent2);

                break;

            case R.id.csv:

                Intent intent = new Intent(MainActivity.this,
CSVActivity.class);
                MainActivity.this.startActivity(intent);

                break;

            default:

                break;
        }
    }

    public class ActivateNode extends AsyncTask <String,Void,String>{

        @Override
        protected void onPreExecute() {
            super.onPreExecute();
        }

        @Override
        protected String doInBackground(String... params) {

            String chain = params[0];
            URL url = null;
            String devuelve ="";

            try {
                HttpURLConnection urlConn;

                DataOutputStream printout;
                DataInputStream input;
                url = new URL(chain);
                urlConn = (HttpURLConnection) url.openConnection();
                urlConn.setDoInput(true);
                urlConn.setDoOutput(true);
                urlConn.setUseCaches(false);
                urlConn.setRequestProperty("Content-Type",
"application/json");
                urlConn.setRequestProperty("Accept",
"application/json");
                urlConn.connect();
                //Creo el Objeto JSON
                JSONObject jsonParam = new JSONObject();
                jsonParam.put("node_id",params[1]);
                jsonParam.put("is_active", params[2]);
                // Envio los parámetros post.
                OutputStream os = urlConn.getOutputStream();
                BufferedWriter writer = new BufferedWriter(
                        new OutputStreamWriter(os, "UTF-8"));
                writer.write(jsonParam.toString());
```

```
                writer.flush();
                writer.close();

                int respuesta = urlConn.getResponseCode();


                StringBuilder result = new StringBuilder();

                if (respuesta == HttpURLConnection.HTTP_OK) {

                    String line;
                    BufferedReader br=new BufferedReader(new
InputStreamReader(urlConn.getInputStream()));
                    while ((line=br.readLine()) != null) {
                        result.append(line);
                        //response+=line;
                    }

                    //Creamos un objeto JSONObject para poder acceder
a los atributos (campos) del objeto.
                    JSONObject respuestaJSON = new
JSONObject(result.toString());   //Creo un JSONObject a partir del
StringBuilder pasado a cadena
                    //Accedemos al vector de resultados

                    String resultJSON =
respuestaJSON.getString("estado");   // estado es el nombre del campo
en el JSON

                    if (resultJSON == "1") {      // hay datos que
mostrar
                        devuelve = "Nodo activado correctamente";

                    } else if (resultJSON == "2") {
                        devuelve = "Nodo no activado";
                    }

                }

            } catch (MalformedURLException e) {
                e.printStackTrace();
            } catch (IOException e) {
                e.printStackTrace();
            } catch (JSONException e) {
                e.printStackTrace();
            }
            return devuelve;
        }

        @Override
        protected void onProgressUpdate(Void... values) {
            super.onProgressUpdate(values);
        }

        @Override
        protected void onPostExecute(String s) {
            super.onPostExecute(s);
        }
    }

}
```

III. Código Java para mostrar los datos gráficamente

```
package com.example.user.sensormonitoring_v30;

import android.content.Intent;
import android.graphics.Color;
import android.os.AsyncTask;
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.support.v7.widget.Toolbar;
import android.util.Log;
import android.view.MotionEvent;
import android.view.View;
import android.widget.Button;

import com.github.mikephil.charting.animation.Easing;
import com.github.mikephil.charting.charts.LineChart;
import com.github.mikephil.charting.components.Legend;
import com.github.mikephil.charting.components.YAxis;
import com.github.mikephil.charting.data.Entry;
import com.github.mikephil.charting.data.LineData;
import com.github.mikephil.charting.data.LineDataSet;
import com.github.mikephil.charting.highlight.Highlight;
import com.github.mikephil.charting.interfaces.datasets.ILineDataSet;
import com.github.mikephil.charting.listener.ChartTouchListener;
import com.github.mikephil.charting.listener.OnChartGestureListener;
import
com.github.mikephil.charting.listener.OnChartValueSelectedListener;

import org.json.JSONArray;
import org.json.JSONException;
import org.json.JSONObject;

import java.io.BufferedInputStream;
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.OutputStream;
import java.io.OutputStreamWriter;
import java.net.HttpURLConnection;
import java.net.MalformedURLException;
import java.net.URL;
import java.util.ArrayList;

import static java.lang.Float.parseFloat;


public class ChartActivity extends AppCompatActivity implements
View.OnClickListener,
        OnChartGestureListener,
        OnChartValueSelectedListener{

    Button back;
    Boolean activo = true;

    // IP de mi Url
    String IP = "http://tfghost.hopto.org";
```

```
    // Ruta al Web Service
    String GET_TEMP = IP + "/get_temp.php";
    String GET_BAT = IP + "/get_bat.php";
    String GET_EXT1 = IP + "/get_ext1.php";
    String GET_EXT2 = IP + "/get_ext2.php";
    String DEACTIVATE = IP + "/deactivate.php";


    GetMeasures hiloconexion;
    String valor = null;
    String type = null;

    DeactivateNode hilothread;

    /*-----------------------------------------------------------------
-----------------------------
            CHART
    -----------------------------------------------------------------
---------------------------*/

    private LineChart mChart;

    /*-----------------------------------------------------------------
-----------------------------
            CHART END
    -----------------------------------------------------------------
---------------------------*/


    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_chart);
        Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);

        Intent intent = getIntent();
        String value = getIntent().getStringExtra("spinner_value");
//if it's a string you stored.
        valor = value;
        String measure_type =
getIntent().getStringExtra("measure_type");
        type = measure_type;

        // Enlaces con elementos visuales del XML
        back = (Button)findViewById(R.id.back);

        // Listener de los botones
        back.setOnClickListener(this);

        /*-----------------------------------------------------------------
---------------------------------
                CHART
        -----------------------------------------------------------------
-------------------------------*/

        mChart = (LineChart) findViewById(R.id.linechart);

        // get the legend (only possible after setting data)
        Legend l = mChart.getLegend();
```

```
        // modify the legend ...
        //l.setPosition(Legend.LegendPosition.LEFT_OF_CHART);
        l.setForm(Legend.LegendForm.LINE);

        mChart.setOnChartGestureListener(this);
        mChart.setOnChartValueSelectedListener(this);

        // no description text
        mChart.setDescription("Sensor Monitoring v3.0");
        mChart.setNoDataTextDescription("No data found.");
        mChart.clear();

        // enable touch gestures
        mChart.setTouchEnabled(true);

        // enable scaling and dragging
        mChart.setDragEnabled(true);
        mChart.setScaleEnabled(true);

        YAxis leftAxis = mChart.getAxisLeft();

        leftAxis.enableGridDashedLine(10f, 10f, 0f);
        leftAxis.setDrawZeroLine(false);

        leftAxis.setDrawLimitLinesBehindData(true);

        mChart.getAxisRight().setEnabled(false);

        mChart.animateX(1500, Easing.EasingOption.EaseInOutQuart);
//2500
        mChart.animateY(1500, Easing.EasingOption.EaseInOutQuart);

        /*----------------------------------------------------------------
---------------------------------
                CHART END
        ----------------------------------------------------------------
------------------------------*/

        hiloconexion = new GetMeasures();

        if (type.equals("1")){
            String cadenallamada = GET_TEMP + "?node_id=" + valor;
            hiloconexion.execute(cadenallamada,"1");
        }
        else if (type.equals("2")){
            String cadenallamada = GET_BAT + "?node_id=" + valor;
            hiloconexion.execute(cadenallamada,"2");
        }
        else if (type.equals("3")){
            String cadenallamada = GET_EXT1 + "?node_id=" + valor;
            hiloconexion.execute(cadenallamada,"3");
        }
        else if (type.equals("4")){
            String cadenallamada = GET_EXT2 + "?node_id=" + valor;
            hiloconexion.execute(cadenallamada,"4");
        }

        /*FloatingActionButton fab = (FloatingActionButton)
findViewById(R.id.fab);
        fab.setOnClickListener(new View.OnClickListener() {
            @Override
```

```
            public void onClick(View view) {
                Snackbar.make(view, "Replace with your own action",
Snackbar.LENGTH_LONG)
                        .setAction("Action", null).show();
            }
        });*/
    }

    @Override
    public void onClick(View v) {

        hilothread = new DeactivateNode();
        String link = DEACTIVATE;
        String active = "N";
        hilothread.execute(link,valor,active);

        activo = false;
        Intent myIntent = new Intent(ChartActivity.this,
MainActivity.class);
        ChartActivity.this.startActivity(myIntent);
    }

    /*----------------------------------------------------------------
----------------------------
            CHART
    ----------------------------------------------------------------
----------------------------*/

    @Override
    public void onChartGestureStart(MotionEvent me,
                                    ChartTouchListener.ChartGesture
                                            lastPerformedGesture) {

        Log.i("Gesture", "START, x: " + me.getX() + ", y: " +
me.getY());
    }

    @Override
    public void onChartGestureEnd(MotionEvent me,
                                  ChartTouchListener.ChartGesture
                                          lastPerformedGesture) {

        Log.i("Gesture", "END, lastGesture: " + lastPerformedGesture);

        // un-highlight values after the gesture is finished and no
single-tap
        if(lastPerformedGesture !=
ChartTouchListener.ChartGesture.SINGLE_TAP)
            // or highlightTouch(null) for callback to
onNothingSelected(...)
            mChart.highlightValues(null);
    }

    @Override
    public void onChartLongPressed(MotionEvent me) {
        Log.i("LongPress", "Chart longpressed.");
    }

    @Override
    public void onChartDoubleTapped(MotionEvent me) {
        Log.i("DoubleTap", "Chart double-tapped.");
```

```
    }

    @Override
    public void onChartSingleTapped(MotionEvent me) {
        Log.i("SingleTap", "Chart single-tapped.");
    }

    @Override
    public void onChartFling(MotionEvent me1, MotionEvent me2,
                             float velocityX, float velocityY) {
        Log.i("Fling", "Chart flinged. VeloX: "
                + velocityX + ", VeloY: " + velocityY);
    }

    @Override
    public void onChartScale(MotionEvent me, float scaleX, float
scaleY) {
        Log.i("Scale / Zoom", "ScaleX: " + scaleX + ", ScaleY: " +
scaleY);
    }

    @Override
    public void onChartTranslate(MotionEvent me, float dX, float dY) {
        Log.i("Translate / Move", "dX: " + dX + ", dY: " + dY);
    }

    @Override
    public void onValueSelected(Entry e, int dataSetIndex, Highlight
h) {
        Log.i("Entry selected", e.toString());
        Log.i("LOWHIGH", "low: " + mChart.getLowestVisibleXIndex()
                + ", high: " + mChart.getHighestVisibleXIndex());

        Log.i("MIN MAX", "xmin: " + mChart.getXChartMin()
                + ", xmax: " + mChart.getXChartMax()
                + ", ymin: " + mChart.getYChartMin()
                + ", ymax: " + mChart.getYChartMax());
    }

    @Override
    public void onNothingSelected() {
        Log.i("Nothing selected", "Nothing selected.");
    }

    /*-----------------------------------------------------------------
----------------------------
            CHART END
    -----------------------------------------------------------------
--------------------------*/

    public class GetMeasures extends AsyncTask<String,Void,String> {

        String cadenallamada = null; //GET + "?node_id=" + valor;
//para llamar al asynctask de refresh

        /*-----------------------------------------------------------
----------------------------------
                CHART
        -----------------------------------------------------------
------------------------------*/
```

```
        //arraylists para el chart
        ArrayList<String> miXArray = new ArrayList<>();
        ArrayList<Entry> miYArray = new ArrayList<>();

        /*-------------------------------------------------------------
---------------------------------
                 CHART END
        -------------------------------------------------------------
------------------------------*/


        @Override
        protected void onPreExecute() {
            super.onPreExecute();
        }

        @Override
        protected String doInBackground(String... params) {

            String cadena = params[0];
            URL url = null;
            String retorno = "";

            if (params[1].equals("1")) { //get temp

                try {
                    url = new URL(cadena);
                    HttpURLConnection connection = (HttpURLConnection)
url.openConnection(); //Open connection

                    int respuesta = connection.getResponseCode();
                    StringBuilder result = new StringBuilder();

                    if (respuesta == HttpURLConnection.HTTP_OK) {

                        InputStream in = new
BufferedInputStream(connection.getInputStream());  // preparo la
cadena de entrada

                        BufferedReader reader = new BufferedReader(new
InputStreamReader(in));  // la introduzco en un BufferedReader

                        // El siguiente proceso lo hago porque el
JSONOBject necesita un String y tengo
                        // que tranformar el BufferedReader a String.
Esto lo hago a traves de un
                        // StringBuilder.

                        String line;
                        while ((line = reader.readLine()) != null) {
                            result.append(line);        // Paso toda
la entrada al StringBuilder
                        }

                        //Creamos un objeto JSONObject para poder
acceder a los atributos (campos) del objeto.
                        JSONObject respuestaJSON = new
JSONObject(result.toString());   //Creo un JSONObject a partir del
StringBuilder pasado a cadena

                        //Accedemos al vector de resultados
```

```
                        String resultJSON =
respuestaJSON.getString("estado");   // estado es el nombre del
campo(tag) en el JSON

                        if (resultJSON.equals("1")) {      // hay
datos a mostrar
                            JSONArray medidasJSON =
respuestaJSON.getJSONArray("medida");   // nodes es el nombre del
campo en el JSON
                            for (int i = 0; i < medidasJSON.length();
i++) {
                                /*-------------------------------------
-------------------------------------------------------
                                        CHART
                                -------------------------------------
-------------------------------------------------------*/


miXArray.add(medidasJSON.getJSONObject(i).getString("time"));
                                miYArray.add(new
Entry(parseFloat(medidasJSON.getJSONObject(i).getString("temperature")
), i));

                                /*-------------------------------------
-------------------------------------------------------
                                        CHART END
                                -------------------------------------
-------------------------------------------------------*/

                                retorno = "temp";

                            }
                        } else if (resultJSON.equals("2")) {
                            miXArray.add("No Data");
                        }

                    }
                } catch (MalformedURLException e) {
                    e.printStackTrace();
                } catch (IOException e) {
                    e.printStackTrace();
                } catch (JSONException e) {
                    e.printStackTrace();
                }

                return retorno;
            }
            else if (params[1].equals("2")){ //get battery
                try {
                    url = new URL(cadena);
                    HttpURLConnection connection = (HttpURLConnection)
url.openConnection(); //Open connection

                    int respuesta = connection.getResponseCode();
                    StringBuilder result = new StringBuilder();

                    if (respuesta == HttpURLConnection.HTTP_OK) {

                        InputStream in = new
BufferedInputStream(connection.getInputStream());  // preparo la
cadena de entrada
```

```
                        BufferedReader reader = new BufferedReader(new
InputStreamReader(in));  // la introduzco en un BufferedReader

                        // El siguiente proceso lo hago porque el
JSONObject necesita un String y tengo
                        // que tranformar el BufferedReader a String.
Esto lo hago a traves de un
                        // StringBuilder.

                        String line;
                        while ((line = reader.readLine()) != null) {
                            result.append(line);        // Paso toda
la entrada al StringBuilder
                        }

                        //Creamos un objeto JSONObject para poder
acceder a los atributos (campos) del objeto.
                        JSONObject respuestaJSON = new
JSONObject(result.toString());   //Creo un JSONObject a partir del
StringBuilder pasado a cadena

                        //Accedemos al vector de resultados
                        String resultJSON =
respuestaJSON.getString("estado");   // estado es el nombre del
campo(tag) en el JSON

                        if (resultJSON.equals("1")) {      // hay
datos a mostrar
                            JSONArray medidasJSON =
respuestaJSON.getJSONArray("medida");   // medida es el nombre del
campo en el JSON
                            for (int i = 0; i < medidasJSON.length();
i++) {
                                /*-----------------------------------
-------------------------------------------------------
                                        CHART
                                -----------------------------------
-------------------------------------------------------*/


miXArray.add(medidasJSON.getJSONObject(i).getString("time"));
                                miYArray.add(new
Entry(parseFloat(medidasJSON.getJSONObject(i).getString("battery")),
i));

                                /*-----------------------------------
-------------------------------------------------------
                                        CHART END
                                -----------------------------------
-------------------------------------------------------*/

                                retorno = "bat";

                            }
                        } else if (resultJSON.equals("2")) {
                            miXArray.add("No Data");
                        }

                    }
                } catch (MalformedURLException e) {
```

```
                    e.printStackTrace();
                } catch (IOException e) {
                    e.printStackTrace();
                } catch (JSONException e) {
                    e.printStackTrace();
                }

                return retorno;
            }

            else if (params[1].equals("3")){ //get external sensor 1
                try {
                    url = new URL(cadena);
                    HttpURLConnection connection = (HttpURLConnection)
url.openConnection(); //Open connection

                    int respuesta = connection.getResponseCode();
                    StringBuilder result = new StringBuilder();

                    if (respuesta == HttpURLConnection.HTTP_OK) {

                        InputStream in = new
BufferedInputStream(connection.getInputStream());  // preparo la
cadena de entrada

                        BufferedReader reader = new BufferedReader(new
InputStreamReader(in));  // la introduzco en un BufferedReader

                        // El siguiente proceso lo hago porque el
JSONOBject necesita un String y tengo
                        // que tranformar el BufferedReader a String.
Esto lo hago a traves de un
                        // StringBuilder.

                        String line;
                        while ((line = reader.readLine()) != null) {
                            result.append(line);        // Paso toda
la entrada al StringBuilder
                        }

                        //Creamos un objeto JSONObject para poder
acceder a los atributos (campos) del objeto.
                        JSONObject respuestaJSON = new
JSONObject(result.toString());   //Creo un JSONObject a partir del
StringBuilder pasado a cadena

                        //Accedemos al vector de resultados
                        String resultJSON =
respuestaJSON.getString("estado");   // estado es el nombre del
campo(tag) en el JSON

                        if (resultJSON.equals("1")) {      // hay
datos a mostrar
                            JSONArray medidasJSON =
respuestaJSON.getJSONArray("medida");   // medida es el nombre del
campo en el JSON
                            for (int i = 0; i < medidasJSON.length();
i++) {
                                /*-----------------------------------
-------------------------------------------------------
                                        CHART
```

```
                                    ------------------------------------
------------------------------------------------------*/


miXArray.add(medidasJSON.getJSONObject(i).getString("time"));
                                    miYArray.add(new
Entry(parseFloat(medidasJSON.getJSONObject(i).getString("external1")),
i));

                                    /*-----------------------------------
------------------------------------------------------
                                            CHART END
                                    ------------------------------------
------------------------------------------------------*/

                                    retorno = "ext1";

                                }
                            } else if (resultJSON.equals("2")) {
                                miXArray.add("No Data");
                            }

                        }
                    } catch (MalformedURLException e) {
                        e.printStackTrace();
                    } catch (IOException e) {
                        e.printStackTrace();
                    } catch (JSONException e) {
                        e.printStackTrace();
                    }

                    return retorno;
                }

                else if (params[1].equals("4")){ //get external sensor 2
                    try {
                        url = new URL(cadena);
                        HttpURLConnection connection = (HttpURLConnection)
url.openConnection(); //Open connection

                        int respuesta = connection.getResponseCode();
                        StringBuilder result = new StringBuilder();

                        if (respuesta == HttpURLConnection.HTTP_OK) {

                            InputStream in = new
BufferedInputStream(connection.getInputStream());  // preparo la
cadena de entrada

                            BufferedReader reader = new BufferedReader(new
InputStreamReader(in));  // la introduzco en un BufferedReader

                            // El siguiente proceso lo hago porque el
JSONOBject necesita un String y tengo
                            // que tranformar el BufferedReader a String.
Esto lo hago a traves de un
                            // StringBuilder.

                            String line;
                            while ((line = reader.readLine()) != null) {
```

```
                            result.append(line);        // Paso toda
la entrada al StringBuilder
                        }

                        //Creamos un objeto JSONObject para poder
acceder a los atributos (campos) del objeto.
                        JSONObject respuestaJSON = new
JSONObject(result.toString());   //Creo un JSONObject a partir del
StringBuilder pasado a cadena

                        //Accedemos al vector de resultados
                        String resultJSON =
respuestaJSON.getString("estado");   // estado es el nombre del
campo(tag) en el JSON

                        if (resultJSON.equals("1")) {      // hay
datos a mostrar
                            JSONArray medidasJSON =
respuestaJSON.getJSONArray("medida");   // medida es el nombre del
campo en el JSON
                            for (int i = 0; i < medidasJSON.length();
i++) {
                                /*-----------------------------------
-------------------------------------------------------
                                        CHART
                                -----------------------------------
-------------------------------------------------------*/


miXArray.add(medidasJSON.getJSONObject(i).getString("time"));
                                miYArray.add(new
Entry(parseFloat(medidasJSON.getJSONObject(i).getString("external2")),
i));

                                /*-----------------------------------
-------------------------------------------------------
                                        CHART END
                                -----------------------------------
-------------------------------------------------------*/

                                retorno = "ext2";

                            }
                        } else if (resultJSON.equals("2")) {
                            miXArray.add("No Data");
                        }

                    }
                } catch (MalformedURLException e) {
                    e.printStackTrace();
                } catch (IOException e) {
                    e.printStackTrace();
                } catch (JSONException e) {
                    e.printStackTrace();
                }

                return retorno;
            }
            return null;
        }
```

```
        @Override
        protected void onProgressUpdate(Void... values) {
            super.onProgressUpdate(values);
        }

        @Override
        protected void onPostExecute(String s) {
            super.onPostExecute(s);

            /*-------------------------------------------------------------
-----------------------------------
                    CHART
            -------------------------------------------------------------
----------------------------------*/

            // add data methods
            setData();
            //setRandomData();

            /*-------------------------------------------------------------
-----------------------------------
                    CHART END
            -------------------------------------------------------------
----------------------------------*/

            if (activo){
                if (s.equals("temp")){
                    cadenallamada = GET_TEMP + "?node_id=" + valor;
                    new GetMeasures().execute(cadenallamada,"1");
                }
                else if (s.equals("bat")){
                    cadenallamada = GET_BAT + "?node_id=" + valor;
                    new GetMeasures().execute(cadenallamada,"2");
                }
                else if (s.equals("ext1")){
                    cadenallamada = GET_EXT1 + "?node_id=" + valor;
                    new GetMeasures().execute(cadenallamada,"3");
                }
                else if (s.equals("ext2")){
                    cadenallamada = GET_EXT2 + "?node_id=" + valor;
                    new GetMeasures().execute(cadenallamada,"4");
                }

            }

        }

        @Override
        protected void onCancelled(String s) {
            super.onCancelled(s);
        }

        /*-------------------------------------------------------------
---------------------------------
                CHART
        -------------------------------------------------------------
--------------------------------*/

        private void setData() {

            ArrayList<String> xVals = miXArray;
```

```
            ArrayList<Entry> yVals = miYArray;

            LineDataSet set1;

            // create a dataset and give it a type
            //set1 = new LineDataSet(yVals, "DataSet 1");
            set1 = new LineDataSet(yVals, "");
            set1.setFillAlpha(110);
            // set1.setFillColor(Color.RED);

            // set the line to be drawn like this "- - - - - -"
            // set1.enableDashedLine(10f, 5f, 0f);
            // set1.enableDashedHighlightLine(10f, 5f, 0f);
            set1.setColor(Color.BLUE);
            set1.setCircleColor(Color.BLACK);
            set1.setLineWidth(1f);
            set1.setCircleRadius(3f);
            set1.setDrawCircleHole(false);
            set1.setValueTextSize(0f);
            set1.setDrawFilled(false);

            ArrayList<ILineDataSet> dataSets = new
ArrayList<ILineDataSet>();
            dataSets.add(set1); // add the datasets

            // create a data object with the datasets
            LineData data = new LineData(xVals, dataSets);

            // set data
            mChart.setData(data);
            mChart.setVisibleXRangeMaximum(40);
            mChart.moveViewToX(data.getXValCount());

            mChart.forceLayout();

        }


        private void setRandomData() {

            LineData data = new LineData();
            mChart.setData(data);

        }

        /*-----------------------------------------------------------------
---------------------------------
                CHART END
        -----------------------------------------------------------------
--------------------------------*/


    }

    public class DeactivateNode extends AsyncTask<String,Void,String>{

        @Override
        protected void onPreExecute() {
            super.onPreExecute();
        }
```

```
        @Override
        protected String doInBackground(String... params) {

            String chain = params[0];
            URL url = null;
            String devuelve ="";

            try {
                HttpURLConnection urlConn;

                DataOutputStream printout;
                DataInputStream input;
                url = new URL(chain);
                urlConn = (HttpURLConnection) url.openConnection();
                urlConn.setDoInput(true);
                urlConn.setDoOutput(true);
                urlConn.setUseCaches(false);
                urlConn.setRequestProperty("Content-Type",
"application/json");
                urlConn.setRequestProperty("Accept",
"application/json");
                urlConn.connect();
                //Creo el Objeto JSON
                JSONObject jsonParam = new JSONObject();
                jsonParam.put("node_id",params[1]);
                jsonParam.put("is_active", params[2]);
                // Envio los parámetros post.
                OutputStream os = urlConn.getOutputStream();
                BufferedWriter writer = new BufferedWriter(
                        new OutputStreamWriter(os, "UTF-8"));
                writer.write(jsonParam.toString());
                writer.flush();
                writer.close();

                int respuesta = urlConn.getResponseCode();


                StringBuilder result = new StringBuilder();

                if (respuesta == HttpURLConnection.HTTP_OK) {

                    String line;
                    BufferedReader br=new BufferedReader(new
InputStreamReader(urlConn.getInputStream()));
                    while ((line=br.readLine()) != null) {
                        result.append(line);
                        //response+=line;
                    }

                    //Creamos un objeto JSONObject para poder acceder
a los atributos (campos) del objeto.
                    JSONObject respuestaJSON = new
JSONObject(result.toString());   //Creo un JSONObject a partir del
StringBuilder pasado a cadena
                    //Accedemos al vector de resultados

                    String resultJSON =
respuestaJSON.getString("estado");   // estado es el nombre del campo
en el JSON
```

```
                    if (resultJSON == "1") {      // hay datos que
mostrar
                        devuelve = "Nodo activado correctamente";

                    } else if (resultJSON == "2") {
                        devuelve = "Nodo no activado";
                    }

                }

            } catch (MalformedURLException e) {
                e.printStackTrace();
            } catch (IOException e) {
                e.printStackTrace();
            } catch (JSONException e) {
                e.printStackTrace();
            }

            return devuelve;
            //return null;

        }

        @Override
        protected void onProgressUpdate(Void... values) {
            super.onProgressUpdate(values);
        }

        @Override
        protected void onPostExecute(String s) {
            super.onPostExecute(s);
        }
    }
}
```

IV. Código Java para descargar archivos CSV

```
package com.example.user.sensormonitoring_v30;

import android.app.ProgressDialog;
import android.os.AsyncTask;
import android.os.Bundle;
import android.os.Environment;
import android.support.v7.app.AppCompatActivity;
import android.support.v7.widget.Toolbar;
import android.util.Log;
import android.view.View;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.DatePicker;
import android.widget.Spinner;
import android.widget.Toast;

import org.json.JSONArray;
import org.json.JSONException;
import org.json.JSONObject;

import java.io.BufferedInputStream;
import java.io.BufferedReader;
```

```
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.OutputStream;
import java.net.HttpURLConnection;
import java.net.MalformedURLException;
import java.net.URL;
import java.util.ArrayList;

import static android.R.layout.simple_spinner_item;

public class CSVActivity extends AppCompatActivity implements
View.OnClickListener{

    Button download;
    Spinner spinnercsv;
    DatePicker datepicker;

    // IP (raspberry pi host)
    String IP = "http://tfghost.hopto.org/";
    // Web Services
    String GET_NODES = IP + "get_nodes.php";
    String EXTENSION = ".csv";

    GetNodesCSV connection;
    DownloadPastMeasures pastconnection;

    String Globalfilename;

    //variables globales para mensaje de error
    String node;
    String dia;
    String mes;
    String ano;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_csv);
        Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);

        //Link to visual element on XML
        download = (Button) findViewById(R.id.download);
        spinnercsv = (Spinner)findViewById(R.id.spinnercsv);
        datepicker = (DatePicker) findViewById(R.id.datepicker);

        //Button listener
        download.setOnClickListener((View.OnClickListener) this);

        //execute getNodesCSV --> set data on the list (spinner)
        connection = new GetNodesCSV();
        connection.execute(GET_NODES); //param for doInBackground
(URL)

    }

    @Override
    public void onClick(View v) {
```

```
        switch (v.getId()) {
            case R.id.download:

                int day = datepicker.getDayOfMonth();
                int month = datepicker.getMonth() + 1;
                int year = datepicker.getYear();

                String sday = String.valueOf(day);
                String smonth = String.valueOf(month);
                String syear = String.valueOf(year);

                if(month < 10){
                    smonth = "0" + smonth;
                }
                if(day < 10){
                    sday  = "0" + sday ;
                }

                String link = IP +
spinnercsv.getSelectedItem().toString() + "_" + sday + "-" + smonth +
"-" + syear + EXTENSION;
                String filename =
spinnercsv.getSelectedItem().toString() + "_" + sday + "-" + smonth +
"-" + syear + EXTENSION;

                Globalfilename = filename;
                node = spinnercsv.getSelectedItem().toString();
                dia = sday;
                mes = smonth;
                ano = syear;

                pastconnection = new DownloadPastMeasures();
                pastconnection.execute(link,filename);

                break;

            default:

                break;

        }
    }

    class GetNodesCSV extends AsyncTask<String,Void,String> {

        //Array para guardar los nodos
        ArrayList<String> nodesArray = new ArrayList<>();

        @Override
        protected void onPreExecute() {
            super.onPreExecute();
        }

        @Override
        protected String doInBackground(String... params) {

            String cadena = params[0];
            URL url = null;
            String nodes = "";
```

```
            try {
                url = new URL(cadena);
                HttpURLConnection connection = (HttpURLConnection)
url.openConnection(); //Open connection

                int respuesta = connection.getResponseCode();
                StringBuilder result = new StringBuilder();

                if (respuesta == HttpURLConnection.HTTP_OK){

                    InputStream in = new
BufferedInputStream(connection.getInputStream());

                    BufferedReader reader = new BufferedReader(new
InputStreamReader(in));

                    String line;
                    while ((line = reader.readLine()) != null) {
                        result.append(line
                    }
                    JSONObject respuestaJSON = new
JSONObject(result.toString());

                    String resultJSON =
respuestaJSON.getString("estado");

                    if (resultJSON.equals("1")){
                        JSONArray medidasJSON =
respuestaJSON.getJSONArray("nodes");
                        for(int i=0;i<medidasJSON.length();i++){

nodesArray.add(medidasJSON.getJSONObject(i).getString("node_id"));
                        }
                    }

                    else if (resultJSON.equals("2")){
                        nodesArray.add("No hay nodos disponibles en la
red");
                    }

                }
            } catch (MalformedURLException e) {
                e.printStackTrace();
            } catch (IOException e) {
                e.printStackTrace();
            } catch (JSONException e) {
                e.printStackTrace();
            }
            return null;
        }

        @Override
        protected void onProgressUpdate(Void... values) {
            super.onProgressUpdate(values);
        }

        @Override
        protected void onPostExecute(String s) {
            super.onPostExecute(s);

            // Application of the Array to the Spinner
```

```
            ArrayAdapter adapter = new
ArrayAdapter<>(getApplicationContext(), simple_spinner_item,
nodesArray);
            spinnercsv.setAdapter(adapter);

adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdo
wn_item); // The drop down view

        }

        @Override
        protected void onCancelled(String s) {
            super.onCancelled(s);
        }
    }
    class DownloadPastMeasures extends AsyncTask<String, Integer,
String> {

        ProgressDialog progressDialog;

        /**
         * Set up a ProgressDialog
         */
        @Override
        protected void onPreExecute() {
            progressDialog = new ProgressDialog(CSVActivity.this);
            progressDialog.setTitle("Download in progress...");

progressDialog.setProgressStyle(ProgressDialog.STYLE_HORIZONTAL);
            progressDialog.setMax(100);
            progressDialog.setProgress(0);
            progressDialog.show();

        }

        /**
         *  Background task
         */
        @Override
        protected String doInBackground(String... params) {

            HttpURLConnection connect = null;

            if ("1".equals("1")) {
                String path = params[0];
                String filename = params[1];
                int file_length;

                try {
                    URL url = new URL(path);
                    connect = (HttpURLConnection)
url.openConnection();
                    connect.connect();
                    //URLConnection urlConnection =
url.openConnection();
                    //urlConnection.connect();

                    // expect HTTP 200 OK, so we don't mistakenly save
error report
                    // instead of the file
```

```
                    if (connect.getResponseCode() !=
HttpURLConnection.HTTP_OK) {
                        //return "Server returned HTTP " +
connect.getResponseCode()
                                //+ " " +
connect.getResponseMessage();
                        return "No data found for node: " + node + "
on " + dia + "-" + mes + "-" + ano;
                    }

                    file_length = connect.getContentLength();

                    /**
                     * Create a folder
                     */
                    File new_folder = new
File(Environment.getExternalStorageDirectory().getAbsolutePath(),
"Sensor Monitoring Past Measures");
                    if (!new_folder.exists()) {
                        if (new_folder.mkdir()) {
                            Log.i("Info", "Folder succesfully
created");
                        } else {
                            Log.i("Info", "Failed to create folder");
                        }
                    } else {
                        Log.i("Info", "Folder already exists");
                    }

                    /**
                     * Create an output file to store the image for
download
                     */
                    File output_file = new File(new_folder, filename);
                    OutputStream outputStream = new
FileOutputStream(output_file);
                    InputStream inputStream = new
BufferedInputStream(url.openStream(), 8192);
                    byte[] data = new byte[1024];
                    int total = 0;
                    int count;
                    while ((count = inputStream.read(data)) != -1) {
                        total += count;
                        outputStream.write(data, 0, count);
                        int progress = 100 * total / file_length;
                        publishProgress(progress);

                        Log.i("Info", "Progress: " +
Integer.toString(progress));
                    }
                    inputStream.close();
                    outputStream.close();

                    Log.i("Info", "file_length: " +
Integer.toString(file_length));

                /*} catch (MalformedURLException e) {
                    e.printStackTrace();
                } catch (IOException e) {
                    e.printStackTrace();
                }*/
```

```
                } catch (Exception e) {
                    return e.toString();
                }
                    if (connect != null)
                        connect.disconnect();
                }
            return "Download complete";
        }

        @Override
        protected void onProgressUpdate(Integer... values) {
            progressDialog.setProgress(values[0]);
        }

        @Override
        protected void onPostExecute(String result) {
            progressDialog.hide();
            Toast.makeText(getApplicationContext(), result,
Toast.LENGTH_LONG).show();
            File folder = new
File(Environment.getExternalStorageDirectory().getAbsolutePath(),
"Sensor Monitoring Past Measures");
            File output_file = new File(folder, Globalfilename);
            String path = output_file.toString();

            Log.i("Info", "Path: " + path);
        }
    }
```

Anexo V. Código para la implementación WEB

I. Código Grafico Web

```
<?php
// Conectamos base de datos
$conexion = mysql_connect('localhost', 'root', 'admin')
or die('No se pudo conectar: ' . mysql_error());
mysql_select_db('TFG_Database') or die('No se pudo seleccionar la base
de datos');

//preparamos la consulta
$SQLDatos = "SELECT * FROM (SELECT * FROM Measures WHERE
node_id='270f' ORDER BY id DESC LIMIT 15) t ORDER BY id ASC";

//ejecutamos la consulta
$result = mysql_query($SQLDatos);


//obtenemos número filas
$numFilas = mysql_num_rows($result);

//cargamos array con los nombres de las métricas a visualizar
$datos[0] = array('time','temperature');
$datos1[0] = array('time','battery');
$datos2[0] = array('time','external1');
$datos3[0] = array('time','external2');
//print json_encode($datos[2]);
//recorremos filas
for ($i=1; $i<($numFilas+1); $i++)
{
    $datos[$i] = array(mysql_result($result, $i-1, "time"),
    (double) mysql_result($result, $i-1, "temperature"));
    //print json_encode($datos[$i]);
    $datos1[$i] = array(mysql_result($result, $i-1, "time"),
    (int) mysql_result($result, $i-1, "battery"));
    $datos2[$i] = array(mysql_result($result, $i-1, "time"),
    (int) mysql_result($result, $i-1, "external1"));
    $datos3[$i] = array(mysql_result($result, $i-1, "time"),
    (int) mysql_result($result, $i-1, "external2"));
}

mysql_close($conexion);
?>
<html>
  <head>
    <meta http-equiv="refresh" content="30">
    <title>Graph node 270f</title>

    <link
href="//fonts.googleapis.com/css?family=Roboto:300,400,400italic,500,5
00italic,700,700italic|Roboto+Mono:400,700|Material+Icons"
          rel="stylesheet">
    <link rel="stylesheet"
          href="https://google-
developers.appspot.com/_static/6db4302793/css/devsite-cyan.css?hl=es">
    <script src="https://google-
developers.appspot.com/_static/6db4302793/js/prettify-
bundle.js?hl=es"></script>
```

```
    <meta name="robots" content="noindex">
    <script src="https://google-
developers.appspot.com/_static/6db4302793/js/jquery-
bundle.js?hl=es"></script>
    <script src="https://google-
developers.appspot.com/_static/6db4302793/js/jquery_ui-
bundle.js?hl=es"></script>
    <script src="//www.google.com/jsapi?key=AIzaSyCZfHRnq7tigC-
COeQRmoa9Cxr0vbrK6xw&hl=es"></script>

    <script src="https://google-
developers.appspot.com/_static/6db4302793/js/framebox.js?hl=es"></scri
pt>
  </head>

  <body class="devsite-layout-docs devsite-framebox
                ">
<p style="display: none; height: 0; width: 0">This is a snippet from
developers.google.com</p>
  <html>

<head>

  <script type="text/javascript"
src="https://www.gstatic.com/charts/loader.js"></script>
    <script type="text/javascript">
      // [START script_body]
      google.charts.load('current', {'packages':['line',
'corechart']});
      google.charts.setOnLoadCallback(drawChart);

drawToolbar();
    function drawChart() {

      var button = document.getElementById('change-chart');
      var button1 = document.getElementById('change-external1chart');
      var button2 = document.getElementById('change-external2chart');
      var chartDiv = document.getElementById('chart_div');
      var click = document.getElementById('click');

      //var refreshChart = document.getElementById('refresh-chart');
        var cargaDatos = <?php echo json_encode($datos); ?>;
        var cargaDatos1 = <?php echo json_encode($datos1); ?>;
        var cargaDatos2 = <?php echo json_encode($datos2); ?>;
        var cargaDatos3 = <?php echo json_encode($datos3); ?>;

        var datosFinales =
google.visualization.arrayToDataTable(cargaDatos); //Datos temperatura
        var datosFinales1 =
google.visualization.arrayToDataTable(cargaDatos1); // Datos Bateria
        var datosFinales2 =
google.visualization.arrayToDataTable(cargaDatos2); //Datos Sensor
Externo 1
          var datosFinales3 =
google.visualization.arrayToDataTable(cargaDatos3); //Datos sensor
Externo 2


      // [START material_opts]
      var batteryOptions = {
```

```
        chart: {
          title: 'Datos Batería'
        },
        width: 2000,
        height: 700,
        series: {
          // Gives each series an axis name that matches the Y-axis
below.
          0: {axis: 'Batería'},
        },
        axes: {
          // Adds labels to each axis; they don't have to match the
axis names.
          y: {
            Batería: {label: 'Valor Batería'},
          }
        }
      };
      // [END material_opts]

      // [START classic_opts]
      var tempOptions = {
        chart: {
          title: 'Datos Temperatura'
        },
        width: 2000,
        height: 700,
        series: {
          // Gives each series an axis name that matches the Y-axis
below.
          0: {axis: 'Temperatura'},
        },
        axes: {
          // Adds labels to each axis; they don't have to match the
axis names.
          y: {
            Temperatura: {label: 'Temperatura (Celsius)'},
          }
        }
      };
     var external1Options = {
        chart: {
          title: 'Datos Sensor Externo 1 '
        },
        width: 2000,
        height: 700,
        series: {
          // Gives each series an axis name that matches the Y-axis
below.
          0: {axis: 'Datos 1'},
        },
        axes: {
          // Adds labels to each axis; they don't have to match the
axis names.
          y: {
            Temperatura: {label: 'Datos 1'},
          }
        }
      };
    var external2Options = {
        chart: {
```

```
          title: 'Datos Sensor Externo 2'
        },
        width: 2000,
        height: 700,
        series: {
          // Gives each series an axis name that matches the Y-axis
below.
          0: {axis: 'Datos 2'},
        },
        axes: {
          // Adds labels to each axis; they don't have to match the
axis names.
          y: {
            Temperatura: {label: 'Datos 2'},
          }
        }
      };

      // [END classic_opts]

      function drawBatteryChart() {
        var batteryChart = new google.charts.Line(chartDiv);
        batteryChart.draw(datosFinales1, batteryOptions);
        button.innerText = 'Datos Temperatura';
      button1.innerText = 'Datos Sensor 1';
      button2.innerText = 'Datos Sensor 2';
        button.onclick = drawTempChart;
      button1.onclick = drawExternal1Chart;
      button2.onclick = drawExternal2Chart;
      click.onclick= setTimeout(EnableButton,100);
      //button1.disable= true;
      //button2.disable= true;

      //refreshChart.innerText = 'Refresh Bateria';
      //refreshChart.onclick= drawMaterialChart;
      }

      function drawTempChart() {
        var tempChart = new google.charts.Line(chartDiv);
        tempChart.draw(datosFinales, tempOptions);
      //refreshChart.innerText = 'Refresh Temperatura';
      //refreshChart.onclick= drawClassicChart;
        button.innerText = 'Datos Batería';
      button1.innerText = 'Datos Sensor 1';
      button2.innerText = 'Datos Sensor 2';
        button.onclick = drawBatteryChart;
      button1.onclick = drawExternal1Chart;
      button2.onclick = drawExternal2Chart;
      click.onclick= setTimeout(EnableButton,100);



      }
      function drawExternal1Chart() {
        var external1Chart = new google.charts.Line(chartDiv);
        external1Chart.draw(datosFinales2, external1Options);
        button.innerText = 'Datos Temperatura';
      button2.innerText = 'Datos Sensor 1';
      button2.innerText = 'Datos Sensor 2';
        button.onclick = drawTempChart;
      button2.onclick = drawExternal2Chart;
```

```
      button1.onclick =document.getElementById("change-
external1chart").disabled = true;
      click.onclick= setTimeout(EnableButton2,100);

      //refreshChart.innerText = 'Refresh Bateria';
      //refreshChart.onclick= drawMaterialChart;
      }
      function drawExternal2Chart() {
        var external2Chart = new google.charts.Line(chartDiv);
        external2Chart.draw(datosFinales3, external2Options);
        button.innerText = 'Datos Temperatura';
      button1.innerText = 'Datos Sensor 1';
      button2.innerText = 'Datos Sensor 2';
        button.onclick = drawTempChart;
      button2.onclick =document.getElementById("change-
external2chart").disabled = true;
      button1.onclick = drawExternal1Chart;
      click.onclick= setTimeout(EnableButton1,100);
      //refreshChart.innerText = 'Refresh Bateria';
      //refreshChart.onclick= drawMaterialChart;
      }

      drawTempChart();

      function EnableButton(){
      document.getElementById("change-external1chart").disabled =
false;
      document.getElementById("change-external2chart").disabled =
false;
       }
      function EnableButton1() {
      document.getElementById("change-external1chart").disabled =
false;

      }
      function EnableButton2() {
      document.getElementById("change-external2chart").disabled =
false;
      }

    }

    //[END script_body]
  </script>
</head>
<body>
  <!-- [START html_body] -->
  <button id="change-chart">Change to Classic</button>
  <br><br>
  <button id="change-external1chart"></button>
  <br><br>
  <button id="change-external2chart"></button>
  <br><br>
  <select name="forma" onchange="location = this.value;">
  <option selected disabled>Export Data</option>
  <option value="exportcsv.php">Export in CSV</option>
  <option value="exportjson.php">Export in JSON</option>
  </select>
  <br><br>
  <div id="chart_div"></div>
  <!-- [END html_body] -->
```

```
<form action="nodelist.html">
    <input type="submit" value="Go Back" />
</form>
</body>
```

II. Script PHP export de datos CSV

```
<?php

$conexion = mysql_connect('localhost', 'root', 'admin')
or die('No se pudo conectar: ' . mysql_error());
mysql_select_db('TFG_Database') or die('No se pudo seleccionar la base
de datos');

//preparamos la consulta
$SQLDatos = "SELECT * FROM (SELECT * FROM Measures WHERE
node_id='270f' ORDER BY id DESC LIMIT 15) t ORDER BY id ASC";

//ejecutamos la consulta
$result = mysql_query($SQLDatos);


//obtenemos número filas
$fields = mysql_num_rows($result);
$export = mysql_query ( $SQLDatos ) or die ( "Sql error : " .
mysql_error( ) );



for ( $i = 0; $i < $fields; $i++ )
{
    $header .= mysql_field_name( $export , $i ) . "\t";
}

while( $row = mysql_fetch_row( $export ) )
{
    $line = '';
    foreach( $row as $value )
    {
        if ( ( !isset( $value ) ) || ( $value == "" ) )
        {
            $value = "\t";
        }
        else
        {
            $value = str_replace( '"' , '""' , $value );
            $value = '"' . $value . '"' . "\t";
        }
        $line .= $value;
    }
    $data .= trim( $line ) . "\n";
}
$data = str_replace( "\r" , "" , $data );

if ( $data == "" )
{
    $data = "\n(0) Records Found!\n";
}

header("Content-type: application/octet-stream");
header("Content-Disposition: attachment; filename=datoscsv.csv");
```

```
header("Pragma: no-cache");
header("Expires: 0");
print "$header\n$data";
mysql_close($conexion);

?>
```

III. Script PHP export de datos en JSON

```
<?php
$conexion = mysql_connect('localhost', 'root', 'admin')
or die('No se pudo conectar: ' . mysql_error());

mysql_select_db('TFG_Database') or die('No se pudo seleccionar la base
de datos');

//preparamos la consulta
$SQLDatos = "SELECT * FROM (SELECT * FROM Measures WHERE
node_id='270f' ORDER BY id DESC LIMIT 15) t ORDER BY id ASC";



$result=mysql_query($SQLDatos);

 $emparray = array();
    while($row =mysql_fetch_assoc($result))
    {
        $emparray[] = $row;
    }
    echo json_encode($emparray);

    //close the db connection
header("Content-type: application/octet-stream");
header("Content-Disposition: attachment; filename=datosjson.json");
header("Pragma: no-cache");
header("Expires: 0");
print "$header\n$data";
mysql_close($conexion);
?>
```

Anexo VI. Tablas Comparativas

A continuación, se podrá ver una comparativa de SBC teniendo en cuenta el nivel, donde el nivel menor tiene la mayor frecuencia CPU y contra más alto sea el nivel, menor será la frecuencia de procesado.

I. Tabla comparativa nivel 1

	NIVEL 1									
Fabricante	**INTEL**	**UDOO**		**UDOO**		**ODROID**	**UDOO**		**ODROID**	**BANANA PI**
Modelo	**Joule**	**x86 Advanced Plus**		**x86 Advanced**		**XU4**	**x86 Basic**		**C2**	**BPI-M2+**
Fabricante CPU	Intel	Intel	Intel	Intel	Intel	Samsung	Intel	Intel	Amlogic	Allwinner
Chip CPU		Braswell	Curie	Braswell	Curie	Exynos 5422	Braswell	Curie	S905	H3
Proceso CPU		14 nm		14 nm		28 nm	14 nm		28 nm	
Núcleos CPU	4 (Quad Core 64 bits)	4	1	4	1	4 + 4	4	1	4	4
Diseño CPU	Atom	Celeron N3160	Quark SE	Celeron N3160	Quark SE	Cortex A15/A7	Atom X5-E8000	Quark SE	Cortex A53	Cortex A7
Frecuencia CPU	1,5 GHz o 1,7 GHz con un burst de hasta 2,4 GHz	2,24 GHz	32 MHz	2,24 GHz	32 MHz	2,1 GHz / 1,5 GHz	2 GHz	32 MHz	2 GHz	2 GHz
Instrucción CPU						ARMv7			ARMv8	ARM
Fabricante GPU	Intel	Intel		Intel		ARM	Intel		ARM	ARM
Diseño GPU	HD Graphics	HD Graphics 400		HD Graphics 400		Mali T628 MP6	HD Graphics		3 x Mali 450	Mali 400MP2
Frecuencia GPU	450 MHz con burst hasta 650 MHz	640 MHz		640 MHz		600MHz	320 MHz		700 MHz	600 MHz
Decodificador H264		Compatible		Compatible		Compatible (1080P60)	Compatible		Compatible (1080P60)	Compatible (2160P)
Codificador H264		Compatible		Compatible		Compatible (1080P60)	Compatible		Compatible (1080P60)	Compatible (1080P30)
Decodificador H265		Compatible		Compatible		No	Compatible		Compatible (4K60)	
Memoria RAM	3 o 4 GB RAM LPDDR4	4 GB DDR3L Canal dual		4 GB DDR3L Dual Channel		2 GB DDR3	2 GB DDR3L		2 GB DDR3	1 GB DDR3
Velocidad de Reloj	32,768 KHz y 19,2 MHz					750 MHz			912 MHz	
GPIO (pins)	Hasta 48 GPIO (con 4 PWM)	20 GPIOs (digital)	14 (PWM) I/O Pins Digitales	20 GPIOs (digital)	14 (PWM) I/O Pins Digitales		20 GPIOs (digital)	14 (PWM) I/O Pins Digitales		
			6 pins I/O analógicos (10 Bits de resolución)		6 pins I/O analógicos (10 Bits de resolución)			6 pins I/O analógicos (10 Bits de resolución)		
Soporte de almacenamiento	Tarjeta SD con interfaz 3.0	Conector SATA/ SSD / MicroSD		Conector SATA/ SSD / MicroSD		MicroSD / eMMC	Conector SATA/ SSD / MicroSD		MicroSD / eMMC	Micro SD / eMMC
Almacenamiento en la placa	8 o 16 GB de eMMC flash	32GB eMMC								
Almacenamiento						SD 3.0 / eMMC 5.0			SD 3.0 / eMMC 5.0	
USB 2.0	x2	x0		x0		x1	x0		x4 + 1 OTG	x2
USB 3.0	x1 o PCIe	x3		x3		x2	x3		x0	x0
Ethernet	No	10/100/1000		10/100/1000		10/100/1000	10/100/1000		10/100/1000	10/100/1000
WIFI	802.11 ac dual band (2,4 GHz y 5 GHz)	No		No		No	No		No	WIFI b/g/n
Bluetooth	Bluetooth 4.2	No	Bluetooth Low Energy	No	Bluetooth Low Energy	No	No	Bluetooth Low Energy	No	Bluetooth 4.0
IrDA						No			Sí	
HDMI	Sí (1080p)	Sí		Sí		Sí (1060P60)	Sí		Sí(4K60)	Sí (1200P60)
RTC						Sí			No	
Fuente de energía						5V 4A			5V 2A	5V 2A
Consumo		5-6 W		5-6 W			5-6 W			
Función de servidor?	Sí	Sí		Sí		Sí	Sí		Sí	Sí
Tamaño		120 mm x 85 mm		120 mm x85 mm			120 mm x85 mm			65 mm x 65 mm
Precio	346 $	165 E		149 E		135 E	125 E		84 E	50 E
Funciones o características extra		Compatible con Arduino 101 y con la mayoría de Arduino Shields		Compatible con Arduino 101 y con la mayoría de Arduino Shields			Compatible con Arduino 101 y con la mayoría de Arduino Shields			
		Sensor de 6 ejes con acelerómetro y giroscopio		Sensor de 6 ejes con acelerómetro y giroscopio			Sensor de 6 ejes con acelerómetro y giroscopio			
		Ranura E para modulos WIFI		Ranura E para modulos WIFI			Ranura E para modulos WIFI			

II. Tabla comparativa nivel 2

NIVEL 2							
Lattepanda		BANANA PI	ORANGE PI	ODROID	ORANGE PI		
Lattepanda 4GB/64GB	Lattepanda 2GB/32GB	BPI-M3	Plus 2	C1+	Plus	PC	One
Intel	Intel	Allwinner	Allwinner	Amlogic	Allwinner	Allwinner	Allwinner
Cherry Trail Z8300 / Built-in Arduino Co-processor: ATmega32u4	Cherry Trail Z8300 / Built-in Arduino Co-processor: ATmega32u4	A83T	H3	S805	H3	H3	H3
		28 nm	28 nm	28 nm	28 nm	28 nm	28 nm
Quad Core	Quad Core	8 (Octa Core)	4	4	4	4	4
Atom X5	Atom X5	Cortex A7	Cortex A7	Cortex A5	Cortex A7	Cortex A7	Cortex A7
1,87 GHz	1,87 GHz	1,8 GHz	1,5 GHz	1,5 GHz	1,5 GHz	1,5 GHz	1,5 GHz
		ARMv7	ARMv7	ARMv7	ARMv7	ARMv7	ARMv7
Intel	Intel	PowerVR	ARM	ARM	ARM	ARM	ARM
HD Graphics	HD Graphics	SGX544MP1	Mali 400MP2	2 x Mali 450	Mali 400MP2	Mali 400MP2	Mali 400MP2
12 EUs 200-500 MHz	12 EUs 200-500 MHz	700 MHz	600 MHz	600 MHz	600 MHz	600 MHz	600 MHz
		Compatible (1080P60)	Compatible (1080P60)	Compatible (1080P60)	Compatible (1080P60)	Compatible (1080P60)	Compatible (1080P60)
		Compatible (1080P60)	Compatible (1080P60)	Compatible (1080P30)	Compatible (1080P30)	Compatible (1080P30)	Compatible (1080P30)
		Compatible (1080P30)	Compatible (4KP30)	Compatible (1080P60)	Compatible (4KP30)	Compatible (4KP30)	Compatible (4KP30)
4 GB DDR3L	2 GB DDR3L	2 GB DDR3	2 GB DDR3	1 GB DDR3	1 GB DDR3	1 GB DDR3	512 MB DDR3
		672 MHz	672 MHz	792 MHz	672 MHz	672 MHz	672MHz
6 GPIOs del procesador Cherry Trail	6 GPIOs del procesador Cherry Trail						
20 GPIOs del Arduino Leonardo	20 GPIOs del Arduino Leonardo						
6 conectores para sensores de gravedad plug and play	6 conectores para sensores de gravedad plug and play						
		MicroSD / USB / SATA 2.0	MicroSD / USB / SATA 2.0	MicroSD / eMMC	MicroSD / USB / SATA 2.0	MicroSD	MicroSD
		8GB eMMC	16GB eMMC		8GB eMMC		
64 GB	32 GB	SD2.0 / eMMC 4.4	SD3.0 / eMMC 4.5	SD3.0 / eMMC 4.5	SD3.0 / eMMC 4.5	SD3.0	SD3.0
x2	x2	x2 + 1 OTG	x4 + 1 OTG	x4 + 1 OTG	x4 + 1 OTG	x3 + 1 OTG	x1 + 1 OTG
x1	x1	x0	x0	x0	x0	x0	x0
10/100	10/100	10/100/1000	10/100/1000	10/100/1000	10/100/1000	10/100	10/100
Sí	Sí	WIFI b/g/n	WIFI b/g/n	No	WIFI b/g/n	No	WIFI b/g/n
Bluetooth 4.0	Bluetooth 4.0	Bluetooth 4.0	No	No	No	No	No
		Sí	Sí	Sí	Sí	Sí	Sí
Sí	Sí	Sí (1200P60)	Sí (4KP30)	Sí (1200P60)	Sí (4KP30)	Sí (4KP30)	Sí (4KP30)
		No	No	Sí	No	No	No
5V 2A	5V 2A	5V 2A	5V 2A	5V 2A	5V 2A	5V 2A	5V 2A
Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí
		92 mm x 60 mm					
149 $	119$	71-91$	54 E	42 E	33 E	26 E	30 E

III. Tabla comparativa nivel 3

NIVEL 3						
Banana Pi	**Raspberry Pi**	**Banana Pi**			**Raspberry Pi**	
BPI-G1	**3 Model B**	**BPI-M2**	**BPI-M1+**	**BPI-M1**	**Zero**	**2 Model B**
Allwinner	Broadcom	Allwinner	Allwinner	Allwinner	Broadcom	Broadcom
A31S	BCM2837	A31S	A20	A20	BCM2835	BCM2836
40 nm	40 nm	40 nm			40 nm	40 nm
4	4	4	2	2	1	4
Cortex A7	Cortex A53	Cortex A7	Cortex A7	Cortex A7	ARM1176JZF-S	Cortex A7
1,2 GHz	1,2 GHz	1 GHz	1 GHz	1 GHz	1 GHz	0,9 GHz
ARM	ARMv8	ARMv7	ARM	ARM	ARMv6	ARMv7
PowerVR	Broadcom	PowerVR	ARM	ARM	Broadcom	Broadcom
SGX544MP2	VideoCore IV	SGX544MP2	Mali 400MP2	Mali 400MP2	VideoCore IV	VideoCore IV
350 MHz	400 MHz	355 MHz			250 MHz	250 MHz
Compatible (4K)	Compatible (1080P30)	Compatible (1080P60)	Compatible (2160P)	Compatible (2160P)	Compatible (1080P30)	Compatible (1080P30)
Compatible (1080P30)	Compatible (1080P30)	Compatible (1080P30)	Compatible (1080P30)	Compatible (1080P30)	Compatible (1080P30)	Compatible (1080P30)
1 GB DDR3	1 GB DDR2	1 GB DDR3	1 GB DDR3	1 GB DDR3	512 MB DDR2	1 GB DDR2
	400 MHz	432 MHz			400 MHz	400 MHz
Tarjeta TF / Ranura para tarjetas MMC	MicroSD	MicroSD	Tarjeta SD / SATA	Tarjeta SD / SATA	MicroSD	MicroSD
	SD2.0	SD3.0			SD2.0	SD2.0
x4	x4	x4 + 1 OTG	x2	x2	1 OTG	x4
x0	x0	x0	x0	x0	x0	x0
10/100/1000	10/100	10/100/1000	10/100/1000	10/100/1000	No	10/100
WIFI b/g/n	WIFI b/g/n	WIFI b/g/n	WIFI b/g/n	Opcional	No	No
Bluetooth 4.0	Bluetooth 4.1	No	No	No	No	No
	No	Sí			No	No
Sí (1200P60)	Sí (1200P60)	Sí (1200P60)	Sí (1200P60)	Sí (1200P60)	Mini HDMI	Sí (1200P60)
	No	No			No	No
	5V 2,5A	5V 2A	5V DC (Micro USB)	5V DC (Micro USB)	5V 1A	5V 2A
	2,5 A / 12,5 W / 5 V				160 mA / 0,8 W / 5 V	800 mA / 4 W / 5 V
Sí	Sí	Sí	Sí	Sí	No	No
92 mm x 60 mm	86 mm x 56 mm	92 mm x 60 mm	92 mm x 60 mm	92 mm x 60 mm	65 mm x 30 mm	85 mm x 56 mm
56 $	35 $	42 E	38 E	36 E	5 $	35 $

IV. Tabla comparativa nivel 4.1

							NIVEL 4
Raspberry Pi				Tessel	ARDUINO (IoT)		Intel
Model A	Model A+	Model B	Model B+	Tessel 2	TIAN		Intel Edison
Broadcom	Broadcom	Broadcom	Broadcom	Mediatek	Atmel (microcontrolador)	Qualcomm (microprocesador)	Intel
BCM2835	BCM2835	BCM2835	BCM2835	ARM Cortex M0 microcontroller Atmel SAMD21 (48MHz)	SAMD21G18	Atheros AR9342	Intel Quark
							22 nm Atom SoC
1	1	1	1	1			2
ARM1176JZF-S	ARM1176JZF-S	ARM1176JZF-S	ARM1176JZF-S	MT7620n	Cortex M0+		
0,7 GHz	0,7 GHz	0,7 GHz	0,7 GHz	580 MHz	48 MHz	560 MHz	500 MHz
					ARM		
Broadcom	Broadcom	Broadcom	Broadcom	No			No
VideoCore IV	VideoCore IV	VideoCore IV	VideoCore IV				
256 MB	256 MB	512 MB	512 MB	64 MB DDR2	32 KB SRAM	64 MB DDR2	1 GB LPDDR3 POP
				2 x módulos Tessel (10 pins)	6 pins I/O analógicos		70 pins sobre cabecera intermedia de 0,4 mm
SD	MicroSD	SD	MicroSD	Micro USB			Tarjeta SD
				32 MB Flash	256 KB flash	16 MB + 4 GB eMMC	4 GB eMMc
x1	x1	x2	x4	x2	x0	x1	x1
x0	x0	x0	x0	x0	x0	x0	x0
No	No	10/100	10/100	10/100	No	10/100/1000	No
No	No	No	No	WIFI b/g/n	No	WIFI b/g/n (dual band-5GHz)	WIFI a/b/g/n dual band
No	No	No	No	No	Bluetooth Low Energy (4.0)	No	Bluetooth 2.1 / 4.0
Sí (1200P60)	Sí (1200P60)	Sí (1200P60)	Sí (1200P60)		No	No	No
					5 V		3,3 V - 4,5 V
300 mA / 1,5 W / 5 V	400 mA / 2 W / 5 V	700 mA / 3,5 W / 5 V	500 mA / 2,5 W / 5 V		470 mA		1 W
No	No	No	No	No	No		No
85.6 mm x 53.98 mm	65 mm x 56 mm	85.6 mm x 53.98 mm	85 mm x 56 mm				
25 $	20 $	35 $	35 $	45 $	85 E		100 $

V. Tabla comparativa nivel 4.2

Olimex	Intel	ARDUINO (IoT)				Banana Pi	Olimex
iMX233-OLinuXino-MAXI	Intel Galileo	INDUSTRIAL 101		YÚN		BPI-D1	ESP32-EVB
	Intel	Atmel (microcontrolador)	Qualcomm (microprocesador)	Atmel (microcontrolador)	Qualcomm (microprocesador)		
iMX233 ARM926J	Quark Soc x1000	ATmega32U4	Atheros AR9331	ATmega32U4	Atheros AR9331	ARM926EJ	2x low-power Xtensa® 32-bit LX6
	1					1	
ARM9		AVR		AVR		ARM	
454 MHz	400 MHz	16 MHz	400 MHz	16 MHz	400 MHz	400 MHz	240 MHz
	No						
						Mali 400MP2	
						Compatible (2160P)	
						Compatible (1080P30)	
64 MB SDRAM	256 MB DDR3	2,5 KB SRAM	64 MB DDR2 / 2,5KB SRAM	2,5 KB SRAM	64 MB DDR2 / 2,5 KB SRAM	64 MB DDR2	520 Kbytes SRAM
40 pins GPIO	20 pins I/O digitales	3 GPIOs		12 pins I/O analógicos			40 pins GPIO
	6 entradas analógicas	4 entradas analógicas		20 pins I/O digitales			
	6 PWMs (12-bits de resolución)	20 pins I/O digitales					
Tarjeta SD		MicroSD	MicroSD	MicroSD	MicroSD	SD card	MicroSD
512 KB	8 MB Flash / Hasta 32 GB (opcional)	32KB flash / 1 KB EEPROM	16 MB / 1 KB EEPROM	32 KB flash / 1 KB EEPROM	16 MB / 1 KB EEPROM		448 KB
x2	x1 (Tipo A) x1 (Tipo B)	x1	x1	x1	x1	1 OTG	x2
x0	x0	x0	x0	x0	x0	x0	x0
10/100	10/100	No	10/100	No	10/100	No	10/100
Módulo RTL8188CU opcional	No	No	WIFI b/g/n	No	WIFI b/g/n	USB WIFI	WIFI b/g/n
No	No	No	No	No	No	No	Bluetooth 4.2 BR/EDR y Bluetooth Low Energy
No	No	No	No	No	No	No	
						RTC Circuit	
5 V	3,3V - 5V	5V		5V		5 V DC (Micro USB)	2,2 - 3,6 V
		130 mA		130 mA		5 V - 350 mA	80 mA (procesador)
No	No	No	No	No	No	No	No
94 mm x 67,3 mm						38 mm x 38 mm	75 mm x 75 mm
45 E	89 E	35 E		62 E		38 E	26 E
				Conexión MicroUSB			
				USB-A			

VI. Tabla comparativa nivel 5

NIVEL 5						
ARDUINO (IoT)	**Zolertia**		**ARDUINO (IoT)**			**Zolertia**
MKRFOX1200	**RE-Mote**	**Firefly**	**LEONARDO ETH**		**ETHERNET**	**Z1**
Atmel	Texas Instruments		Atmel (microcontrolador)	Microprocesador Arduino	Atmel	Texas Instruments
SAMD21	CC2538		ATmega32U4		ATmega328	MSP430F2617
Cortex M0+ (ARM)	ARM Cortex-M3	ARM Cortex-M3				RISC CPU
48 MHz	32 MHz	32 MHz	16 MHz		16 MHz	16 MHz
	No	No				No
32 KB SRAM	32 KB (16KB retención)	32 KB (16KB retención)	2,5 KB SRAM		2 KB SRAM	8 KB
8 pins I/O digitales			12 pins I/O analógicos		14 pins I/O digitales	
7 pins de entrada analógicos (ADC 8/10/12 bits)			20 pins I/O digitales		6 entradas analógicas	
1 pin de salida analógico (DAC 10 bits)						
	Micro SD	Micro SD	MicroSD		MicroSD	
256 KB flash	512 KB flash	512KB flash	32KB flash / 1 KB EEPROM		32 KB flash / 1 KB EEPROM	92 KB flash / M25P16 (2MB)
x1			x1	x0	x0	
x0			x0	x0	x0	
No			No	10/100	10/100	
No			No	No	No	
No			No	No	No	
						Yes
No			No	No	No	
5 V			7-12 V		5 V	1,8 - 3,6 V
	150 nA (modo sleep) - 500 mA		82 mA			365 uA (activo) / 0,5 uA (stand by) / 0,1 uA (apagado)
No	No	No	No		No	No
	73 mm x 40 mm	68 mm x 25 mm				
35 E	75 E	42 E	45 E		25 E	
Frecuencia portadora: 868 MHz	Bandas compatibles: 2,4 GHz, 868/915 MHz Estándar: IEEE 802.15.4	Bandas compatibles: 2,4 GHz Estándar: IEEE 802.15.4				Bandas compatibles: 2,4 GHz Estándar: IEEE 802.15.4
	Switch de RF programable para cambiar de antena					Rango de 100 m (antena 0dBi, LOS)
	Rango de 20 Km (868/915 MHz, LOS)					

VII. Tabla comparativa de software nivel 1-5

	NIVEL 1							NIVEL 2								NIVEL 3				
	INTEL	UDOO	UDOO	ODROID	UDOO	ODROID	BANANA PI	Lattepanda		BANANA PI	ORANGE PI	ODROID	ORANGE PI			Banana Pi	Raspberry Pi	Banana Pi		
	Joule	x86 Advanced Plus	x86 Advanced	XU4	x86 Basic	C2	BPI-M2+	Lattepanda 4GB/64GB	Lattepanda 2GB/32GB	BPI-M3	Plus 2	C1+	Plus	PC	One	BPI-G1	3 Model B	BPI-M2	BPI-M1+	BPI-M1
Linux	Kernel 4.4 Ubuntu 16.04			Ubuntu 16.04		Ubuntu 16.04	Ubuntu				Ubuntu	Ubuntu	Ubuntu	Ubuntu	Ubuntu		Ubuntu Mate Snappy Ubuntu Core		Ubuntu	Ubuntu
Android				4.4 - 7.1		5.1														
Windows	W10 IoT Core	W7; W8.1; W10	W7; W8.1; W10		W7; W8.1; W10			W10	W10								W10 IoT Core			
Raspbian																				
Debian																				
Bananian																				
OpenSUSE																				
Raspberry Pi Image																				
Free-RTOS																				
Ti-OS																				
Coustom-OS																				
OSMC																				
LIBREELEC																				
RISC OS																				
PINET																				
Contiki OS																				
RIOT OS																				
Open WSN																				
Tiny OS																				
Mans Os																				

			NIVEL 4													NIVEL 5					
	Raspberry Pi		Raspberry Pi				Tessel	ARDUINO (IoT)	Intel	Olimex	Intel	ARDUINO (IoT)		Banana Pi	Olimex	ARDUINO (IoT)	Zolertia		ARDUINO (IoT)		Zolertia
	Zero	2 Model B	Model A	Model A+	Model B	Model B+	Tessel 2	TIAN	Intel Edison	iMX233-OLinuXino-MAXI	Intel Galileo	INDUSTRIAL 101	YÚN	BPI-D1	ESP32-EVB	MKRFOX1200	RE-Mote	Firefly	LEONARDO ETH	ETHERNET	Z1
Linux	Ubuntu Mate Snappy Ubuntu Core	Ubuntu Mate Snappy Ubuntu Core	Ubuntu Mate Snappy Ubuntu Core	Ubuntu Mate Snappy Ubuntu Core	Ubuntu Mate Snappy Ubuntu Core	Ubuntu Mate Snappy Ubuntu Core		Linino OS				Linino OS	Linino OS	3.4.35					Linino OS		
Android																					
Windows	W10 IoT Core	W10 IoT Core	W10 IoT Core	W10 IoT Core	W10 IoT Core	W10 IoT Core															
Raspbian																					
Debian																					
Bananian																					
OpenSUSE																					
Raspberry Pi Image																					
Free-RTOS																					
TI-OS																					
Coustom-OS																					
OSMC																					
LIBREELEC																					
RISC OS																					
PINET																					
Contiki OS																					
RIOT OS																					
Open WSN																					
Tiny OS																					
Mans Os																					