Proyecto Fin de Máster
Máster en Ingeniería Electrónica Robótica y Automática

Despliegue de redes inalámbricas de sensores con Contiki

Autor: Pablo Montero García
Tutor: Juan José Murillo Fuentes

Dpto. Teoría de la Señal y Comunicaciones
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2020

Proyecto Fin de Máster
Máster Ingeniería Electrónica Robótica y Automática

Despliegue de redes inalámbricas de sensores con Contiki

Autor:
Pablo Montero García

Tutor:
Juan José Murillo Fuentes
Profesor titular

Dpto. de Teoría de la Señal y Comunicaciones
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2020

Proyecto Fin de Máster: Despliegue de redes inalámbricas de sensores con Contiki

Autor: Pablo Montero García

Tutor: Juan José Murillo Fuentes

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2020

El Secretario del Tribunal

A mi familia y amigos

Agradecimientos

Juan José Murillo Fuentes

Subdirección de Comunicaciones y Recursos Comunes

Sevilla, 2020

Resumen

Este documento expone el desarrollo de un sistema de adquisición de datos completo. Se propone el uso del sistema operativo para dispositivos de bajos recursos Contiki, en su variante Contiki-NG (Next Generation), para la implementación de una pequeña red de sensores a modo de demostración de las capacidades de los nuevos sistemas para Internet of Things. Esta red consiste en dispositivos basados en el módulo Zoul de Zolertia, plataforma soportada y optimizada para la ejecución de Contiki-NG. La interfaz que permitirá al usuario acceder a toda la información que recopilen los sensores de la red, consitirá en una aplicación web implementada mediante la pila estandar de software LAMP.

Abstract

This document exposes the development of a complete data acquisition system. It is proposed to use the operating system for low-income devices Contiki, in its Contiki-NG (Next Generation) variant, for the implementation of a small sensor network to demonstrate the capabilities of the new systems for Internet of Things. This network consists of devices based on the Zolertia Zoul module, a platform supported and optimized for the execution of Contiki-NG. The interface that will allow the user to access all the information collected by the sensors on the network will consist of a web application implemented in the standard LAMP software stack.

Índice

Agradecimientos **ix**

Resumen **xi**

Abstract **xiii**

Índice **xiv**

Índice de Figuras **xvi**

1 Introducción **1**
1.1 Redes Inalámbricas de Sensores *1*
1.2 Aplicaciones de las WSNs *2*

2 Tecnologías **5**
2.1 Contiki-NG *5*
2.1.1 Enrutamiento con RPL 6
2.1.2 Comunicaciones UDP 7
2.2 Plataformas Zolertia *9*
2.2.1 RE-Mote revisión b 9
2.2.2 Orion Ethernet Router 13
2.3 Servidor web *14*
2.3.1 Servidor Apache 15
2.3.2 Bases de datos 16
2.3.3 Gestores de datos 17
2.3.4 MySQL 17
2.3.5 SQL 19
2.3.6 PHP 20

3 Diseño **21**
3.1 Entorno de desarrollo *21*
3.1.1 Instalación de la máquina virtual 21
3.1.2 Descarga de Contiki-NG 22
3.1.3 Preparación de archivo Cooja 23
3.2 Comunicación entre motas *24*
3.2.1 Programación de motas 24
3.2.2 Compilación y carga en plataforma 26
3.2.3 Simulaciones en Cooja 28
3.2.4 Implementación de la red 32
3.3 Comunicación con el border-router *33*
3.3.1 Programa del border-router 33
3.3.2 Simulación con border-router 34
3.3.3 Implementación del border-router 36
3.4 Comunicación con el servidor *36*
3.4.1 Consulta HTTP desde el border-router 37
3.4.2 Envío de parámetros HTTP 37
3.4.3 Conexión entre el border-router y el servidor 38
3.4.4 Instalación del servidor LAMP 39
3.4.5 Diseño de la interfaz web 42
3.4.6 Implementación del servidor 46

4 Conclusiones **49**
4.1 Trabajos futuros *49*

5 Referencias **50**

6 Anexo **51**
6.1 Códigos de programas, comunicación entre motas *51*
6.1.1 Udp-server.c 51
6.1.2 Udp-client-temp.c 52
6.2 Códigos de programas, comunicación con router *54*
6.2.1 Border-router.c 54

Índice de Figuras

Figura 1. Redes inalámbricas atendiendo a la tasa de datos y a su cobertura geográfica. 1
Figura 2. Dominios de Aplicación de las Redes Inalámbricas de Sensores (WSNs). 3
Figura 3. Logotipo Contiki-NG. 6
Figura 4. Enrutado RPL. 6
Figura 5. Comparativa entre TCP y UDP. 8
Figura 6. Trama UDP. 9
Figura 7. Plataforma RE-Mote. 10
Figura 8. Pinout de la platafoma RE-Mote revisión b. 11
Figura 9. Conexiones y puertos externos de la RE-Mote rev b. 12
Figura 10. Pines de conexiónes Molex. 12
Figura 11. Dispositivo Orion Ethernet Router. 13
Figura 12. Pinout del Router Orion. 14
Figura 13. Estructura de una aplicación LAMP. 15
Figura 14. Servidor HTTP Apache. 15
Figura 15. Tipos de bases de datos. 17
Figura 16. Jerarquía en aplicaciones de MySQL. 18
Figura 17. Consulta de base de datos mediante sentencias. 19
Figura 18. Ventana de configuración de VirtualBox. 22
Figura 19. Descarga del compilador ARM. 22
Figura 20. Descargar de Contiki-NG. 23
Figura 21. Descarga de orden ant para Cooja. 23
Figura 22. Ubicación de carpeta "contiki-ng". 23
Figura 23. Inclusiones en el archivo "build.xml". 24
Figura 24 . Descarga de archivos MSPSim. 24
Figura 25. Instancias UDP. 24
Figura 26. Función de recepción UDP. 25
Figura 27. Envío de mensaje UDP. 25
Figura 28 Código de envio UDP y comprobación de ruta. 26
Figura 29. Compilación de programa desde el terminal. 26
Figura 30. Selección de conexión USB de la plataforma. 27
Figura 31. Carga de programa desde el terminal. 27
Figura 32. Conexión serial desde el terminal. 27
Figura 33. Modificación del programa "udp-client-temp.c". 28
Figura 34. Inicio de Cooja. 28
Figura 35. Creación de simulación. 29
Figura 36. Ventana principal de Cooja. 29

Figura 37. Creación de los nodos para simulación. 30
Figura 38. Ubicación de nodos. 30
Figura 39. Información de inicialización del nodo. 31
Figura 40. Transmisión de mensaje. 31
Figura 41. Reenvio de mensajes en la red. 32
Figura 42. Comprobación de batería. 32
Figura 43. Mensajes recibidos en la mota servidor. 33
Figura 44. Jerarquíca de llamadas a funciones httpd. 34
Figura 45. Definición de macros ADD y SEND. 34
Figura 46. Compilación para plataforma sky. 34
Figura 47. Comunicación y rutado de la red. 35
Figura 48. Conexión con el border-router en simulación. 35
Figura 49. Prueba de dirección IPv6. 35
Figura 50. Información de rutado para el usuario. 36
Figura 51. Conexión con el border-router. 36
Figura 52. Definición de funciones para http-socket. 37
Figura 53. Menú de selección de dispositivos. 38
Figura 54. Conexión router-ordenador. 38
Figura 55. Interfaces habilitadas. 39
Figura 56. Instalación del paquete Apache2. 40
Figura 57. Estado del servidor Apache. 40
Figura 58. Ejemplos de consultas SQL. 41
Figura 59. Modificación de archivos de configuración de Apache2. 42
Figura 60. Coexistencia de HTML y PHP. 43
Figura 61. Estructura de la aplicación web. 44
Figura 62. Ventana de acceso. 44
Figura 63. Ventana principal 45
Figura 64. Pestaña de mapa. 45
Figura 65. Pestaña de gráficas. 46
Figura 66. Pestaña de información. 46
Figura 67. Ubicación de la aplicación web. 46
Figura 68. Manipulación de la base de datos desde el terminal Linux. 47
Figura 69. Recepción de mensajes de las motas en el router. 47
Figura 70. Visualización de la red desde la aplicación web. 48
Figura 71. Visualización de datos. 48

1 INTRODUCCIÓN

Las redes de computadores han llegado a ser una parte esencial de la vida cotidiana: permiten tener comunicaciones personales, chatear con amigos, gestionar correo electrónico, transmitir fotos y videos y navegar por Internet. Esas mismas redes también permiten el desarrollo de aplicaciones de negocios, gestionar transferencias bancarias y monitorizar y operar plantas de manera remota. Facilitan las operacions logísticas y permiten comprar y vender objetos y servicios por Internet, que se reciben en casa u ordenadores personales en unos pocos días (en algunos casos horas o minutos). También permiten conectar objetos: sensores, electrodomésticos, consolas, etc., así como conectar periféricos y enlazar computadores entre sí , y conectarlos a Internet.

Muchas de estas redes son cableadas y otras inalámbricas. La conexión por cables proporciona una mayor fiabilidad en las comunicaciones y, en ocasiones, un mayor ancho de banda dedicado, con un elevado coste de instalación y de mantenimiento de las infraestructuras. Las comunicaciones inalámbricas son menos costosas de instalar y desplegar, pero su propia naturaleza ubicua las hace menos seguras y más sensibles a interferencia. A pesar de estas limitaciones, propias de las redes inalámbricas, éstas forman parte de nuestra vida diaria, y términos como Bluetooth, WiFi y 4G son familiares incluso para los que desconocen por completo el funcionamiento de las tecnologías que los soportan.

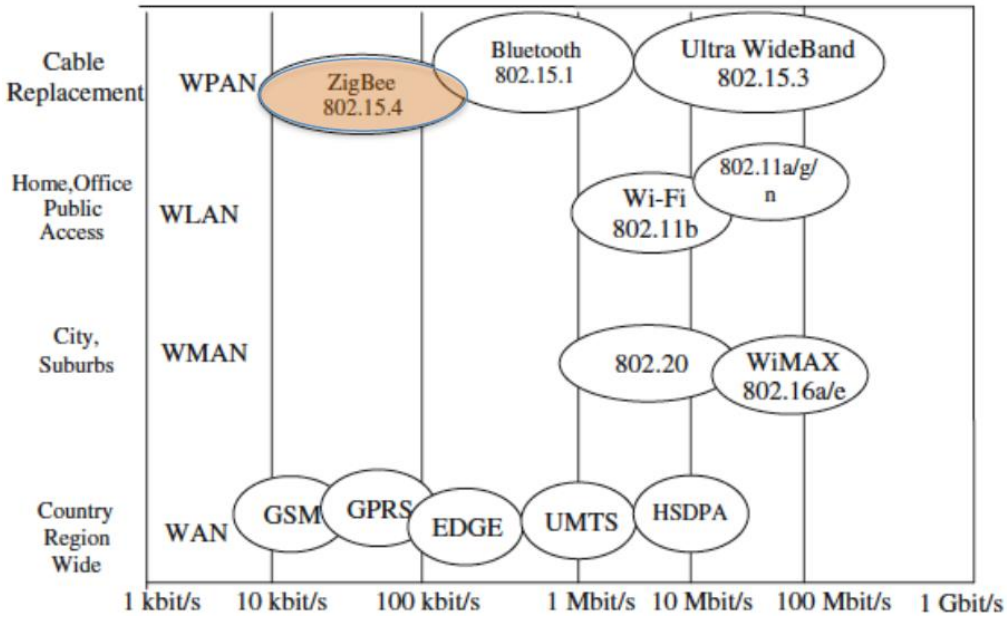

Figura 1. Redes inalámbricas atendiendo a la tasa de datos y a su cobertura geográfica.

La Figura 1 muestra de una manera esquemática las principales tecnologías que dan soporte a las comunicaciones inalámbricas más extendidas, distribuyéndolas en función de su cobertura geográfica y de la tasa de datos que soportan. En algunos casos se indica el nombre del estándar que las soporta, en otros casos se usan los nombres, comerciales o populares, con el que se las conoce.

1.1 Redes Inalámbricas de Sensores

Redes Inalámbricas Sensoriales (Wireless Sensor Networks o WSNs) se definen como un conjunto de sensores y actuadores autónomos que emplean una infraestructura de comunicación inalámbrica, destinados a monitorizar y controlar las condiciones físicas y ambientales en localizaciones diversas, y que de una manera cooperativa pasan sus datos a través de la red hasta una localización central, o pasan las órdenes a través de la red hasta un determinado actuador. A cada uno de los elementos que forman parte de la red de comunicaciones los llamaremos nodos de la red. Un nodo puede tener varios sensores y actuadores, aunque lo normal es que tengan uno sólo, o un número muy limitado de ellos.

Aunque no todas las WSNs tienen todas estas características a la vez, las WSNs generalmente precisan una baja tasa de datos y tienen alcances limitados, por lo que son redes LR-WPANs (Low-Rate Wireless Personal Area Network). Otras características de estas redes son:

- El número de nodos es de unas decenas o centenas; como mucho, algunos millares.
- Los nodos son autónomos, por lo que necesitan limitar su consumo de manera que puedan funcionar un tiempo suficiente alimentados por baterías, o por sistemas de extracción de energía del ambiente (energy harvesters).
- Los nodos no tienen una elevada capacidad de procesamiento, por lo que tanto las aplicaciones de control local de sensores y actuadores, como los algoritmos y protocolos de gestión de las comunicaciones deben ser tan simples como sea posible.

Con respecto a los nodos, éstos pueden desempeñar varias funciones:

- Función terminal gestionando los sensores y actuadores que tiene instalado y pasando la información de los sensores (recibiendo los comandos a los actuadores) a través de otros nodos de la red. A estos nodos se les denomina Nodos Terminales (Terminal Nodes) o Nodos Sensores (Sensor Nodes).
- Función de retransmisión de la información de un nodo a otro hasta alcanzar su destino final. A estos nodos se les denomina Nodos Enrutadores (Routers)
- Función de sumidero de información y/o de pasarela con un computador central o con otras redes. A estos nodos se les llama Nodos Fuente (Sink Nodes) o Estaciones Base (Base Stations), según el caso.

En la práctica, un mismo nodo, dependiendo de su programación o del estado de la red en un momento dado, puede actuar como un Terminal, un Router o una Estación Base.

Cada Nodo Terminal está equipado con un sensor o actuador (o un grupo de ellos), un procesador que gestiona la programación y operación de los sensores o actuadores, y un terminal radio que permite enlazar el nodo con el resto de la red inalámbrica. Los datos procedentes de los sensores (las órdenes destinadas a los actuadores) son recogidos en (enviados desde) el Nodo Sumidero o en la Estación Base, que se comunica con el sistema de supervisión y control, bien en modo local, bien de forma remota a través de una LAN, MAN o WAN.

Los Nodos Sensores no tienen, en la mayor parte de los casos, una localización específica. En algunos casos, incluso, son distribuidos de forma más o menos aleatoria para cubrir amplios espacios.

1.2 Aplicaciones de las WSNs

La Figura 2 muestra los principales dominios de aplicación de las WSNs, que podemos clasificar en: Aplicaciones de Seguimiento Remoto y Aplicaciones de Monitorización Remota, cada una de ellas con aplicaciones en interior (Indoor) y exterior (Outdoor).

Cuando se habla de Seguimiento se considera que el nodo se encuentra en movimiento y se desea seguir su trayectoria, posicionándolo en un entorno, que puede ser fijo o variable. El problema de Seguimiento tiene aplicaciones militares (seguimiento de blancos y del movimiento de tropas amigas o enemigas), de medio ambiente (seguimiento de animales en granjas o en grandes espacios naturales), o en transporte (seguimiento de contenedores en cadenas logísticas, seguimiento de vehículos en grandes instalaciones, etc.). En aplicaciones de interior se puede pensar en el seguimiento de pacientes o de ancianos en hospitales y residencias, seguimiento de grandes equipos en instalaciones singulares como almacenes y hospitales, o seguimiento de vehículos autónomos en cadenas de fabricación.

En cuanto a las aplicaciones de monitorización y control, destacan la detección temprana de incendios en bosques y parques naturales, la monitorización de las condiciones de las cargas en los procesos logísticos, o la monitorización de variables físicas (temperatura, humedad, texturas, dimensiones, etc.) en los procesos de fabricación. En entornos de interior, se puede monitorizar las condiciones de seguridad (detección de incendios, contaminación, etc.) en edificios e instalaciones o las variables fisiológicas de pacientes o deportistas. También cabe pensar en otras aplicaciones industriales y logísticas.

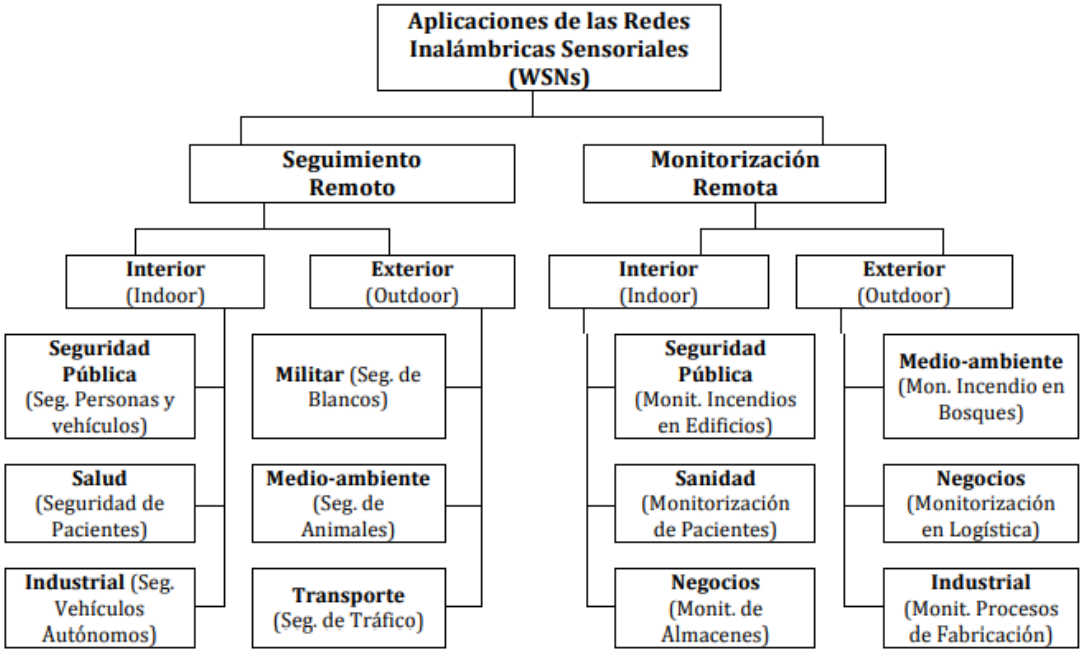

Figura 2. Dominios de Aplicación de las Redes Inalámbricas de Sensores (WSNs).

2 TECNOLOGÍAS

En este apartado se exponen las principales tecnologías, plataformas y herramientas utilizadas en el desarrollo del proyecto y que suponen los componentes principales y de mayor interés de este. En el proyecto se han empleado principalmente varios componentes, tanto software como hardware:

- **Contiki-NG:** sistema operativo para dispositivos con bajas prestaciones optimizado para redes de comunicación inalámbricas y aplicaciones de bajo consumo. Además, se exponen los puntos más importantes de comunicación en red que implementa este sistema operativo como son el protocolo de comunicación y el enrutado de dispositivos.
- **Plataformas Zolertia:** dispositivos hardware basados en microcontroladores de gran rendimiento con interfaces de comunicación inalámbricas, especialmente pensados para inplementación y gestión de redes de bajo consumo. Las plataformas a utilizar son RE-Mote revisión b y Router Ethernet Orion.
- **Servidor LAMP:** Una pila de software muy extendida en servicios web que combina las funciones de Apache, MySQL y PHP sobre un sistema operativo basado en Linux para cubrir todas las necesidades de un portal web básico.

2.1 Contiki-NG

Contiki-NG (1) es un sistema operativo para dispositivos con recursos limitados en Internet of Things. Contiki-NG contiene una pila de comunicación IPv6 de baja potencia que cumple con RFC, lo que permite la conectividad a Internet. El sistema se ejecuta en una variedad de plataformas basadas en arquitecturas energéticamente eficientes, como ARM Cortex-M3 / M4 y Texas Instruments MSP430. La huella del código es del orden de 100 kB, y el uso de la memoria se puede configurar para que sea tan bajo como 10 kB. El código fuente está disponible como código abierto con una licencia BSD de 3 cláusulas.

En 2017, Contiki-NG comenzó como una bifurcación del sistema operativo Contiki con los siguientes objetivos:

- Centrarse en una comunicación IPv6 confiable (confiable y segura) basada en estándares;
- Centrarse en las plataformas modernas de IoT, p. ARM Cortex M3 y otras MCU de 32 bits;
- Modernizar la estructura, configuración, registro y plataformas, para reflejar los objetivos anteriores;
- Mejore la documentación, tanto el código API, la descripción del módulo y los tutoriales;
- Implemente un proceso de desarrollo más ágil, con una inclusión más fácil de nuevas características y con lanzamientos periódicos.

Los aspectos más interesantes a profundizar en este proyecto son aquellos relacionados con las comunicaciones inalámbricas, las cuales se basan, entre otras capas del protocolo de comunicación, en comunicaciones UDP y enrutamiento con RPL.

Figura 3. Logotipo Contiki-NG (de (1)).

2.1.1 Enrutamiento con RPL

RPL (2) (Routing Protocol for Loosy and Low-power) es un protocolo de enrutamiento IPv6 para redes con pérdidas y de baja potencia especificado en RFC 6550, que se implementa en la pila de red IPv6 de Contiki-NG. RPL crea y mantiene una topología DODAG (Destinated Orientated Directed Acyclic Graph) que se origina en un nodo raíz RPL designado, que generalmente también sirve como un enrutador de borde a Internet. La información de enrutamiento se difunde a través de balizas de difusión de forma periódica. La topología se construye de acuerdo con un objetivo determinado mediante una función objetivo. Tal objetivo puede ser minimizar el recuento estimado de transmisión (Estimated Transmision Count, ETX) en la ruta desde el nodo a la raíz RPL, como lo es en las implementaciones de las dos Funciones Objetivos que admite, MRHOF y 0F0.

RPL admite diferentes sentidos de tráfico:

- Enrutamiento ascendente: de cualquier nodo a una raíz.
- Enrutamiento hacia abajo: desde la raíz a cualquier nodo.
- Enrutamiento de cualquiera a cualquier: donde el tráfico fluye entre pares arbitrarios de nodos en el DODAG al enrutar hacia arriba a su ancestro común más cercano (o la raíz en modo de no almacenamiento) en el DODAG, y luego hacia abajo al nodo de destino.

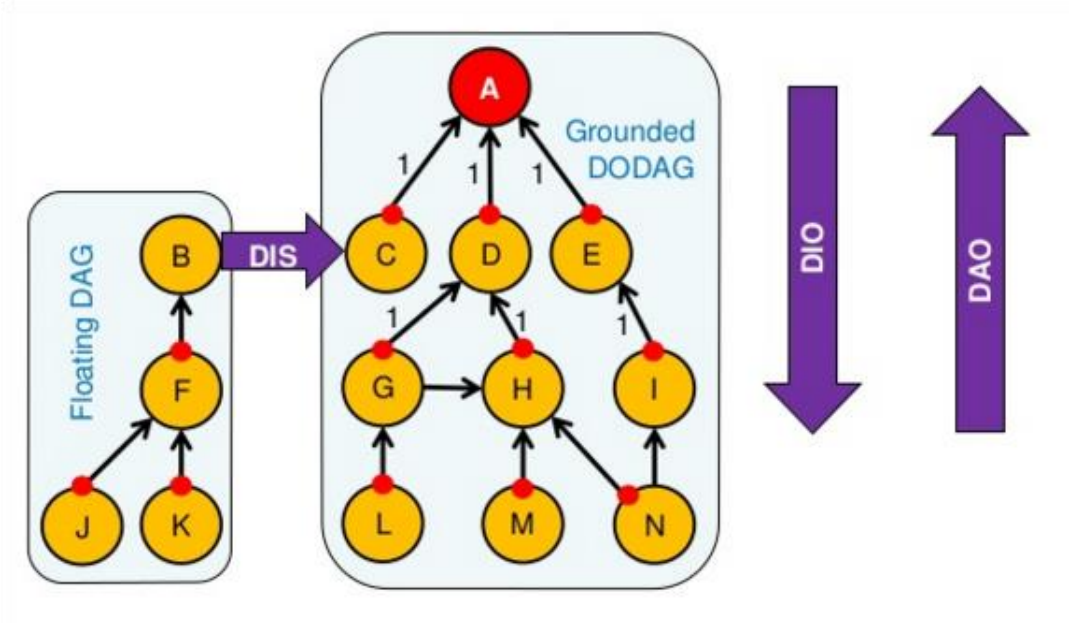

Figura 4. Enrutado RPL (de (2)).

Todo el enrutamiento ascendente se maneja haciendo que cada nodo en la ruta de tráfico reenvíe a través de un padre preferido, como en la Figura 4. El enrutamiento hacia abajo se puede manejar con dos tipos diferentes de modos: modo sin almacenamiento y modo de almacenamiento. Para el modo sin almacenamiento, se utiliza el enrutamiento de origen IPv6, lo que significa que los nodos no tienen que almacenar tablas de enrutamiento para los nodos debajo de ellos en el DODAG. Por otro lado, los paquetes pueden agrandarse si hay muchos saltos a lo largo de la ruta, cuyas direcciones tendrían que incrustarse en el encabezado de enrutamiento de origen. Para el modo de almacenamiento, las tablas de enrutamiento se almacenan en cada nodo, lo que puede imponer una huella de memoria significativa en redes grandes y ser difícil de mantener de manera consistente, pero por otro lado no hay necesidad de encabezados de enrutamiento de origen potencialmente grandes en los paquetes IPv6.

Contiki-NG dispone de dos implementaciones diferentes de RPL (2):

- RPL Classic: RPL Classic es la continuación de la implementación original de RPL de Contiki: ContikiRPL. Esta implementación se creó ya en 2009, mientras que el estándar RPL todavía estaba en desarrollo. A lo largo de los años, se ha agregado una gran cantidad de funcionalidades al RPL, como soporte para múltiples instancias y DODAG, modo de almacenamiento y no almacenamiento, multidifusión y más. Por lo tanto, al precio de soportar una gran cantidad de funcionalidades de estándares y borradores de Internet, la implementación se ha vuelto compleja y ha logrado una gran huella de ROM.

- RPL Lite: RPL Lite es la implementación de RPL predeterminada de Contiki-NG. Comenzó como una reescritura importante de la versión 2017 de Contiki RPL, con un enfoque en la funcionalidad más importante y estable, como la comunidad ha experimentado en muchas implementaciones y experimentos de investigación. RPL Lite elimina la compatibilidad con el modo de almacenamiento en favor del modo sin almacenamiento y la complejidad de manejar múltiples instancias y DODAG. A través de estos cambios, RPL Lite generalmente exhibe un mejor rendimiento y tiene una huella ROM considerablemente menor. Por otro lado, estas optimizaciones tienen un nivel de interoperabilidad más bajo con otras implementaciones, que pueden usar el modo de almacenamiento, por ejemplo.

2.1.2 Comunicaciones UDP

El protocolo de datagramas de usuario, abreviado como UDP (3), es un protocolo que permite la transmisión sin conexión de datagramas en redes basadas en IP. Para obtener los servicios deseados en los hosts de destino, los puertos están listados como uno de los campos principales en la cabecera UDP. Como muchos otros protocolos de red, UDP pertenece a la familia de protocolos de Internet, por lo que debe clasificarse en el nivel de transporte y, entre la capa de red y la capa de aplicación.

Mediante el protocolo de datagramas de usuario, una aplicación puede enviar información muy rápidamente, ya que no es necesario establecer una conexión con el receptor ni esperar una respuesta. Sin embargo, no hay garantía de que los paquetes vayan a llegar completos y respetando el orden en el que fueron enviados, a diferencia de TCP como se muestra en la Figura 5. Además, este protocolo no ofrece ninguna protección frente a la alteración o acceso por parte de terceros. Sin embargo, el UDP puede añadir opcionalmente una suma de verificación (que es obligatoria en IPv6) que permite detectar los paquetes defectuosos.

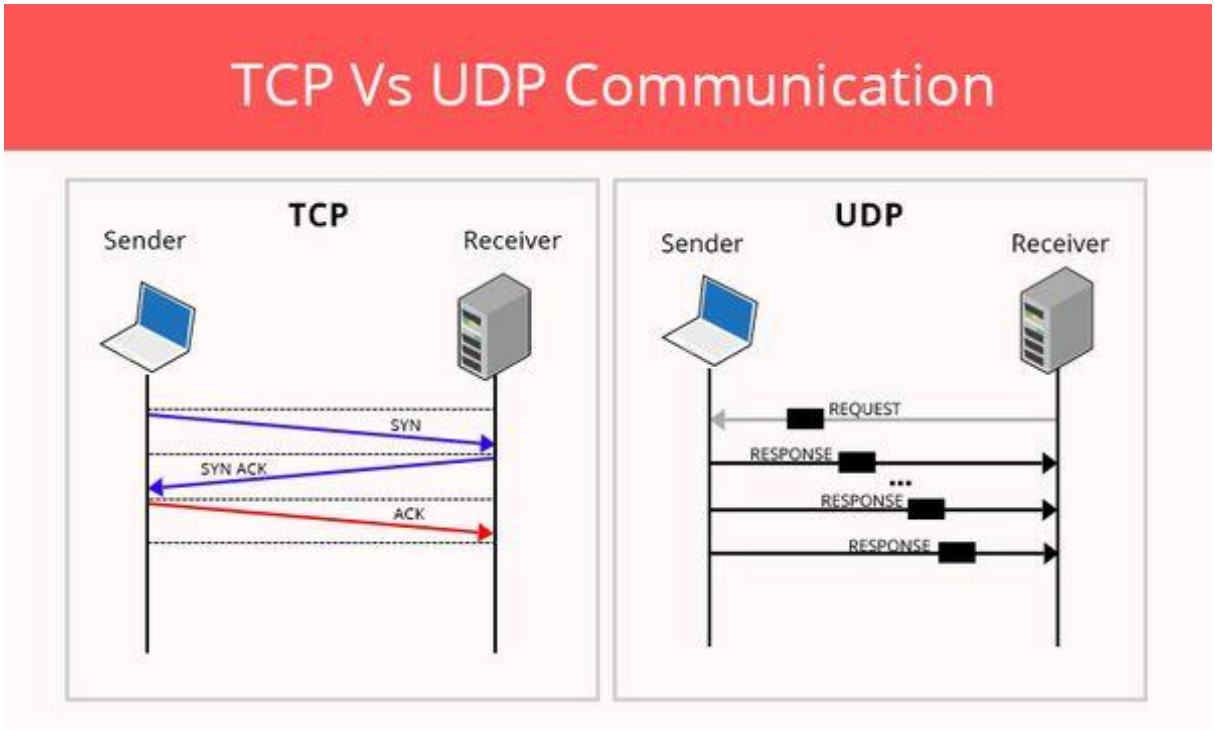

Figura 5. Comparativa entre TCP y UDP (de (3)).

Para comprender a fondo cómo funciona la transmisión de datagramas en este protocolo, resulta útil analizar detenidamente cuáles son las propiedades del protocolo de datagramas de usuario (3).

1. El protocolo UDP **funciona sin conexión**: el protocolo UDP se caracteriza porque permite el envío de datagramas a través de la red sin que se haya establecido previamente una conexión entre el emisor y el receptor. Los datagramas respectivos se envían a la dirección IP preferida de la secuencia especificando el puerto de destino, sin que sea necesario que el ordenador asociado al mismo tenga que dar una respuesta. No obstante, cuando los paquetes tienen que ser devueltos al emisor, existe la posibilidad de incluir en la cabecera UDP información sobre el puerto de origen.
2. UDP **utiliza puertos**: al igual que el TCP, el protocolo UDP utiliza puertos para permitir que los datagramas se transfieran a los protocolos correctos, es decir, a las aplicaciones elegidas del sistema de destino. Los puertos quedan definidos mediante un número conforme a un rango de valores válidos, estando reservado el rango de 0 a 1023 para los servicios fijos.
3. El protocolo UDP permite una **comunicación rápida y sin retardos**: el protocolo de transporte es el adecuado para una transmisión de datos rápida debido a que no hay que llevar a cabo una configuración de la conexión. Esto resulta también del hecho de que la pérdida de un paquete individual afecta exclusivamente a la calidad de la transmisión. En el caso de conexiones TCP, en cambio, se intenta reenviar de nuevo los paquetes perdidos de forma automática, lo que provoca que todo el proceso de transmisión se detenga.
4. El protocolo UDP **no ofrece ninguna garantía de seguridad** e integridad de los datos: la ausencia de acuse de recibo mutuo entre el emisor y el receptor garantiza que la velocidad de transmisión en el protocolo UDP sea excelente; no obstante, el protocolo no puede garantizar la seguridad ni la integridad de los datagramas. Tampoco puede garantizar el orden de los paquetes enviados. Por ello, los servicios que utilizan UDP deben aplicar sus propias medidas de corrección y protección.

Como es típico en todos los protocolos, los paquetes UDP consisten en una cabecera (header) y los datos reales del usuario. La cabecera UDP (Figura 6) contiene toda la información necesaria para la transmisión de datos utilizando el protocolo de transporte y hace que un paquete UDP se pueda identificar como tal. La cabecera UDP consta de 4 campos y está dividida en 2 bloques de 32 bits con la siguiente estructura:

+	Bits 0 - 15	16 - 31
0	Puerto origen	Puerto destino
32	Longitud del Mensaje	Suma de verificación
64	Datos	

Figura 6. Trama UDP (de (3)).

Los primeros 16 bits de la cabecera identifican el puerto de origen desde el que se ha enviado un datagrama concreto. El receptor necesita esta información para poder responder al paquete. Ya que UDP funciona sin conexión y básicamente no requiere ninguna comunicación entre el emisor y el receptor, el campo de puerto de origen es opcional. En caso de no ser utilizado, el puerto de origen debe ser puesto a cero.

En el siguiente campo se especifica el puerto de destino, es decir, se indica el servicio solicitado. Esta información es obligatoria, al contrario que el puerto de origen, porque si no, no sería posible asignar correctamente el datagrama.

El campo longitud define la longitud del datagrama. Se compone de la longitud de la cabecera (8 bytes) y el tamaño de los datos de usuario (máximo teórico: 65.535 bytes). Cuando se utiliza Ipv4, el límite real para los datos de usuario es de 65 507 bytes, tras deducir las cabeceras IP y UDP. En Ipv6 se aceptan paquetes (llamados jumbogramas) que superan ese límite. Según la RFC 2675 , el valor del campo de longitud se pone a cero en esos casos.

La cabecera UDP se completa con la checksum o suma de comprobación, que se utiliza para detectar errores durante la transmisión. De esta manera, se puede detectar si los datos han sufrido alguna alteración en el camino. No obstante, los paquetes detectados se descartan y no se cursa una nueva solicitud. Para generar la suma, se utilizan partes de la cabecera UDP, de los datos del usuario y de la conocida como pseudocabecera (que contiene información sobre la cabecera IP).

La suma de comprobación es opcional en IPv4, pero la mayoría de las aplicaciones la utilizan por defecto. Si no se realiza la suma, se le da a este campo el valor cero. Si se utiliza UDP con IPv6, la suma de comprobación es obligatoria.

2.2 Plataformas Zolertia

2.2.1 RE-Mote revisión b

RE-Mote (4) es una plataforma de desarrollo de hardware diseñada conjuntamente con universidades y socios industriales, en el marco del proyecto de investigación europeo RERUM (REliable, Resilient y secUre IoT for sMart city applications). El objetivo de RE-Mote es llenar el vacío de las plataformas IoT existentes que carecen de un diseño de grado industrial y un consumo de energía ultra bajo, permitiendo a los fabricantes e investigadores desarrollar aplicaciones IoT y productos conectados.

La plataforma se basa en el System on Chip (SoC) CC2538 ARM Cortex-M3 de Texas Instruments, con una interfaz RF IEEE 802.15.4 integrada de 2.4 GHz, que funciona a hasta 32 MHz con 512 KB de flash programable y 32 KB de RAM, incluida con un transceptor RF Texas Instruments CC1200 868/915 MHz para permitir la operación de doble banda.

Figura 7. Plataforma RE-Mote (de (4)).

Las principales características de este dispositivo son (4):

- ISM 2.4-GHz IEEE 802.15.4 y radio compatible con Zigbee.
- ISM 863-950-MHz Banda ISM / SRD Radio compatible con IEEE 802.15.4.
- ARM Cortex-M3 Velocidad de reloj de 32 MHz, 512 KB flash y 32 KB RAM (retención de 16 KB)
- AES-128/256, motor de cifrado de hardware SHA2
- ECC-128/256, motor de aceleración de hardware RSA para intercambio seguro de claves
- Usuario y botón de reinicio
- Consumo hasta 150 nA utilizando el modo de apagado.
- Programación sobre BSL sin necesidad de presionar ningún botón para ingresar al modo del gestor de arranque.
- Cargador de batería incorporado (500 mA), que facilita la recolección de energía y la conexión directa a los paneles solares y a las baterías estándar de LiPo.
- Amplio rango de entrada de alimentación de CC: 3.3-16 V.
- Pequeño factor de forma (73 x 40 mm).
- MicroSD (sobre SPI).
- RTCC a bordo (Real Time Clock Calendar programable) y watchdog timer externo (WDT).
- Interruptor de RF programable para conectar una antena externa a la interfaz de RF de 2.4 GHz o Sub 1 GHz a través del conector RP-SMA.
- Compatible con sistemas operativos de código abierto como Contiki, RIOT y OpenWSN (en progreso).

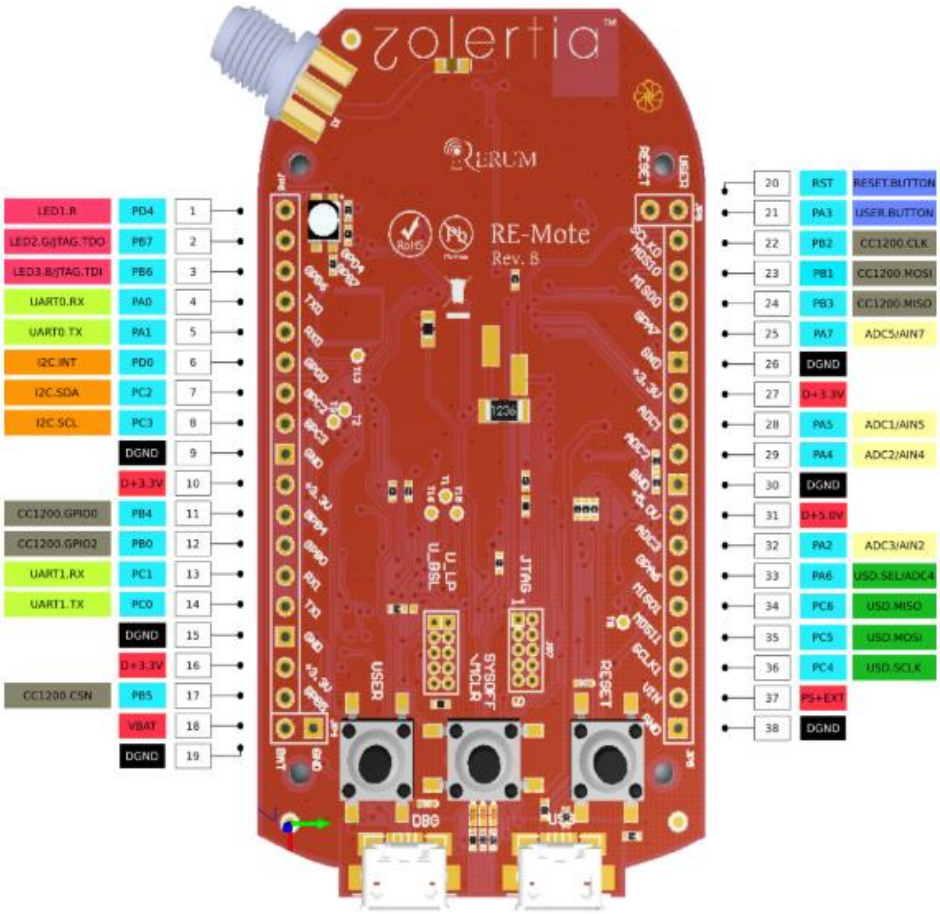

Figura 8. Pinout de la platafoma RE-Mote revisión b (de (4)).

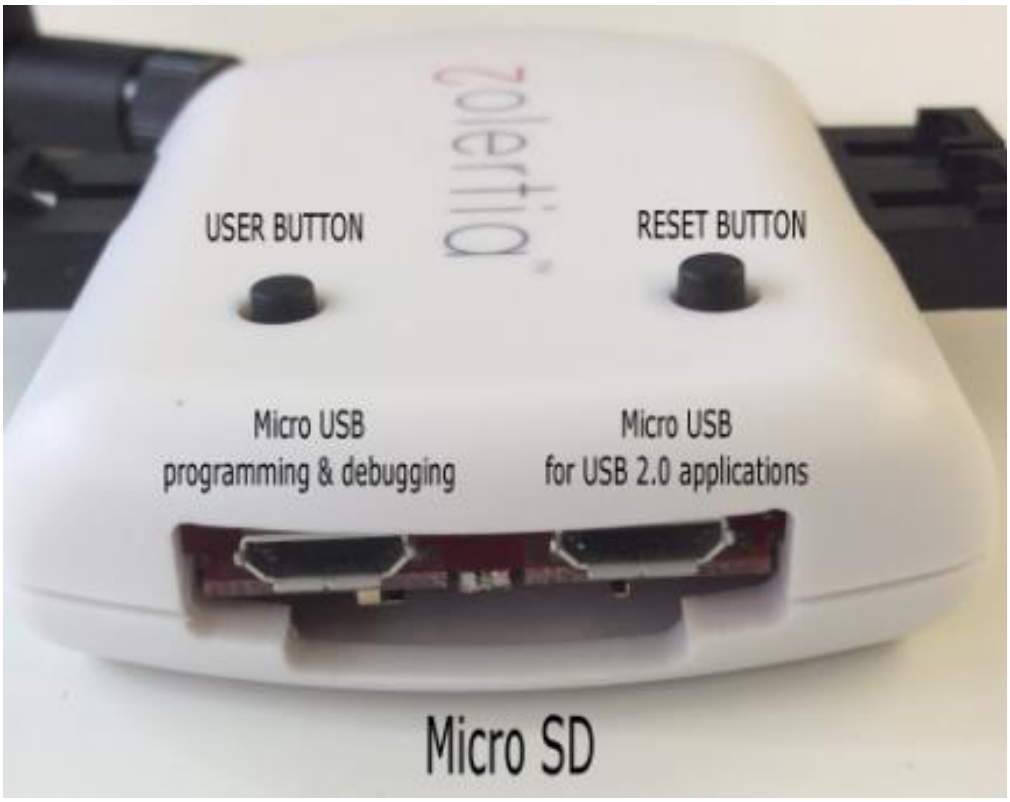

Figura 9. Conexiones y puertos externos de la RE-Mote rev b (de (4)).

RE-Mote utiliza el conector de cabecera macho WM4903-ND (Figura 10) de 5 pines de Molex para conectar sensores digitales basados en protocolos I2C y SPI, así como otros sensores o actuadores que pueda necesitar conectar. También tiene 2 puertos ADC disponibles que se pueden usar con el conector de cabecera macho WM4901-ND de 3 pines de Molex, que proporciona los pines GND y VCC normalmente utilizados para conectar sensores analógicos. Dependiendo del requisito de operación de potencia del sensor, puede usar el ADC1 (3.3V) o el ADC3 (5V) (Teniendo en cuenta que ADC2 no está expuesto a un conector, pero siempre se puede soldar un cable).

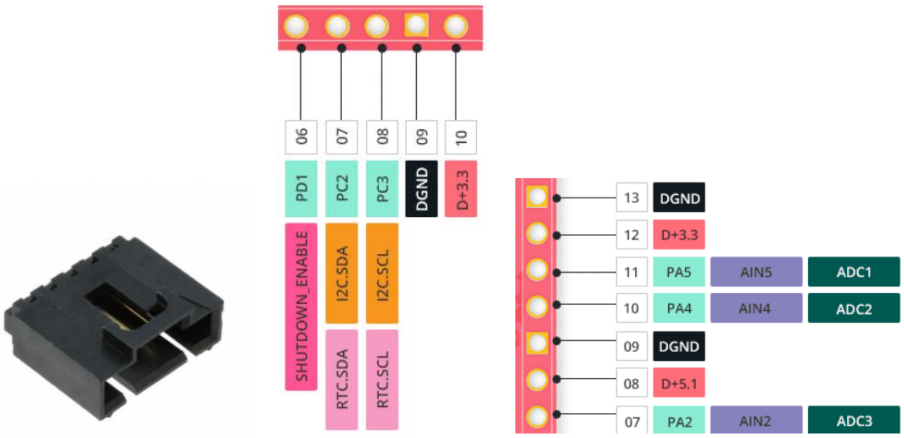

Figura 10. Pines de conexiónes Molex (de (4)).

2.2.2 Orion Ethernet Router

El Router Orion (5) es un dispositivo de enrutamiento IPv4 / IPv6, con una interfaz Ethernet y radio inalámbrica dual, alimentado a través de micro-USB o Power Over Ethernet (POE). El dispositivo integra el módulo Ethernet ENC28J60 y un módulo POE externo, que admite hasta 48 V CC.

El enrutador Zolertia Ethernet expone un botón de alimentación DPDT, un botón RESET para reiniciar el SoC CC2538 y un botón USR programable. El conector RP-SMA para una antena externa de 2.4GHz está ubicado al lado del conector RJ Ethernet, mientras que el conector SMA para antena sub-GHz está ubicado en el lado opuesto. El conector micro-USB se usa tanto para alimentar el dispositivo a través de USB (5VDC) para programar o depurar.

La conexión RJ-45 Ethernet admite Power Over Ethernet (POE) 802.3af activo, hasta 48VDC. El uso de POE permite transportar datos y alimentación a través del mismo cableado, lo que reduce la cantidad de elementos necesarios para conectar y alimentar el enrutador Zolertia Ethernet.

Un conector JTAG no poblado también está disponible para depurar aún más el dispositivo utilizando una herramienta JTAG externa. Dos LED integrados en la parte fronta como indicadores de luz. El LED verde al lado del interruptor de encendido muestra el estado de alimentación del dispositivo (encendido cuando está encendido, de lo contrario apagado).

Figura 11. Dispositivo Orion Ethernet Router (de (5)).

Las principales características de este dispositivo son (5):

- ARM Cortex-M3 con flash de 512 KB y 32 KB de RAM (retención de 16 KB), 32 MHz.
- ISM 2.4-GHz IEEE 802.15.4 y compatible con Zigbee.
- Banda ISM 868-, 915-, 920-, 950-MHz ISM / SRD.
- Conector RP-SMA para una antena externa de 2.4GHz.
- Conector SMA para una antena externa de 868 / 915MHz.
- Conector ethernet RJ45.
- Ethernet 10BASE-T IPv4 / IP64.
- AES-128/256, motor de cifrado de hardware SHA2.
- ECC-128/256, motor de aceleración de hardware RSA para el intercambio seguro de claves.

- CP2104 / PIC incorporado para flashear sobre su conector micro-USB.
- botones de usuario y reinicio.
- Botón de encendido / apagado y LED para mostrar el estado de encendido
- LED RGB para permitir más de 7 combinaciones de colores.
- Diseño 40.29 x 73.75 mm

Hay conectores adicionales (no incluidos de forma predeterminada) con un espaciado de paso de 2,54 mm, exponiendo los pines CC2538. Los conectores JP4 y JP6 exponen los pines principalmente a sensores de interfaz, actuadores y buses de comunicación. Los pines pares JP6 son compatibles con los pines predeterminados I2C utilizados por otras plataformas como Firefly o RE-Mote). El bloque JP5 expone los pines del CC2538 conectado al transceptor CC1200 sub-GHz y al módulo Ethernet ENC28J60.

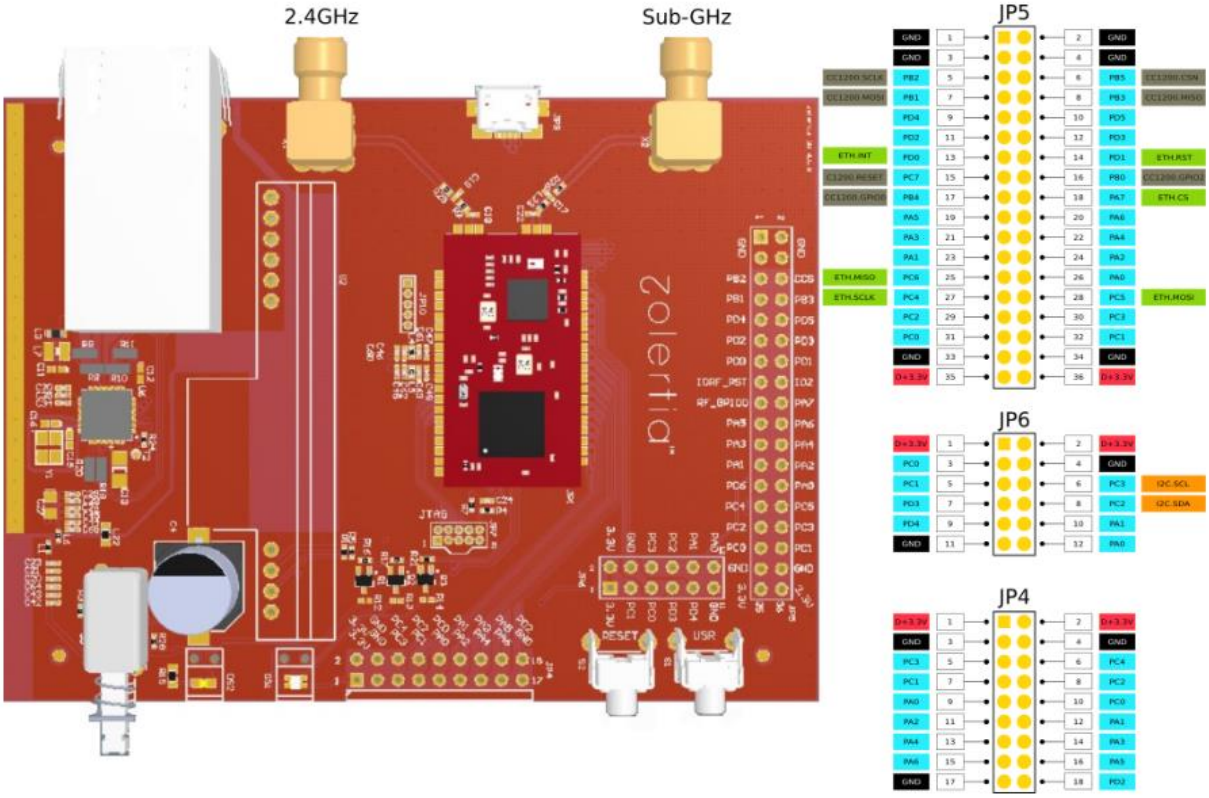

Figura 12. Pinout del Router Orion (de (5)).

2.3 Servidor web

Para poder dar acceso al usuario la red de sensores desde una interfaz cómoda e intuitiva, almacenar los datos y mensajes enviados por los sensores, es necesario el uso de un software específico que soporte estas funcionalidades. Una de las opciones es el uso de una aplicación tipo LAMP (Figura 13). Las aplicaciones LAMP combinan:

- **Apache**: es el componente de servidor web más popular en este tipo de plataformas. Gestionan la conexión de los clientes al servidor y emite las respuestas.
- **MySQL**: es la base de datos más popularizada. Almacena toda la información de los elementos del sistema, mensajes y códigos enviados y los datos de gestión internos de la aplicación.
- Los lenguajes de programación **PHP**/Perl/Python (y ahora también Ruby). Desarrollan todo el proceso de gestión interna del servidor para procesar las peticiones de los clientes.

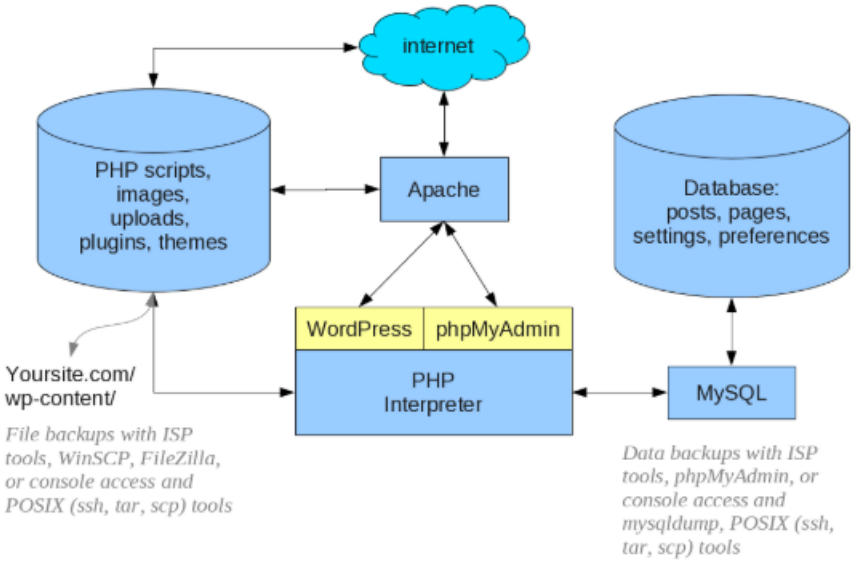

Figura 13. Estructura de una aplicación LAMP.

2.3.1 Servidor Apache

El servidor HTTP Apache (6) es un servidor web HTTP de código abierto, para plataformas Unix (BSD, GNU/Linux, etc.), Microsoft Windows, Macintosh y otras, que implementa el protocolo HTTP/1.1 y la noción de sitio virtual según la normativa RFC 2616. Cuando comenzó su desarrollo en 1995 se basó inicialmente en código del popular NCSA HTTPd 1.3, pero más tarde fue reescrito por completo, y consistía solamente en un conjunto de parches a aplicar al servidor de NCSA. Es desarrollado y mantenido por una comunidad de usuarios bajo la supervisión de la Apache Software Foundation dentro del proyecto HTTP Server (httpd).

Apache presenta entre otras características altamente configurables, bases de datos de autenticación y negociado de contenido, pero fue criticado por la falta de una interfaz gráfica que ayude en su configuración.

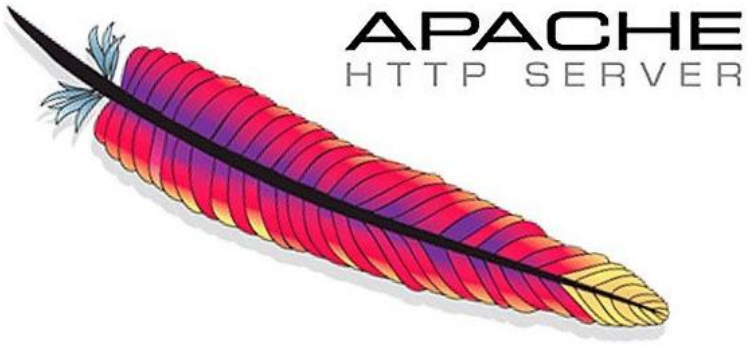

Figura 14. Servidor HTTP Apache (de (6)).

Apache tiene amplia aceptación en la red: desde 1996, Apache es el servidor HTTP más usado. Jugó un papel fundamental en el desarrollo de la World Wide Web y alcanzó su máxima cuota de mercado en 2005, siendo el servidor empleado en el 70% de los sitios web en el mundo. Sin embargo, ha sufrido un descenso en su cuota de mercado en los últimos años (estadísticas históricas y de uso diario proporcionadas por Netcraft). En 2009, se convirtió en el primer servidor web que alojó más de 100 millones de sitios web.

Apache es usado principalmente para enviar páginas web estáticas y dinámicas en la World Wide Web. Muchas aplicaciones web están diseñadas asumiendo como ambiente de implantación a Apache, o que utilizarán características propias de este servidor web.

Este servidor web es redistribuido como parte de varios paquetes propietarios de software, incluyendo la

base de datos Oracle y el IBM WebSphere application server. MacOS integra apache como parte de su propio servidor web y como soporte de su servidor de aplicaciones WebObjects. Es soportado de alguna manera por Borland en las herramientas de desarrollo Kylix y Delphi. Apache es incluido con Novell NetWare 6.5, donde es el servidor web por defecto, y en muchas distribuciones Linux

Un servidor web Apache puede ser una excelente opción para ejecutar tu sitio web en una plataforma estable y versátil. Sin embargo, también presenta algunas desventajas a las que debes prestarle atención:

Ventajas:

1. De código abierto y gratuito, incluso para uso comercial.
2. Software confiable y estable.
3. Parches de seguridad regulares y actualizados con frecuencia.
4. Flexible debido a su estructura basada en módulos.
5. Fácil de configurar para principiantes.
6. Multiplataforma (funciona tanto en servidores Unix como en Windows).
7. Viene listo para trabajar con sitios de WordPress.
8. Enorme comunidad y soporte fácilmente disponible en caso de cualquier problema.

Desventajas:

1. Problemas de rendimiento en sitios web con demasiado tráfico.
2. Demasiadas opciones de configuración pueden generar vulnerabilidades de seguridad.

2.3.2 Bases de datos

Una base de datos (7) es un conjunto de datos pertenecientes a un mismo contexto y almacenados sistemáticamente para su posterior uso. En este sentido; una biblioteca puede considerarse una base de datos compuesta en su mayoría por documentos y textos impresos en papel e indexados para su consulta. Actualmente, y debido al desarrollo tecnológico de campos como la informática y la electrónica, la mayoría de las bases de datos están en formato digital, siendo este un componente electrónico, por tanto, se ha desarrollado y se ofrece un amplio rango de soluciones al problema del almacenamiento de datos.

Existen diferentes clasificaciones de las bases de datos (Figura 15), atendiendo a características puntuales:

Según su variabilidad. Conforme a los procesos de recuperación y preservación de los datos, podemos hablar de:

- **Bases de datos estáticas**. Típicas de la inteligencia empresarial y otras áreas de análisis histórico, son bases de datos de sólo lectura, de las cuales se puede extraer información, pero no modificar la ya existente.
- **Bases de datos dinámicas**. Aparte de las operaciones básicas de consulta, estas bases de datos manejan procesos de actualización, reorganización, añadidura y borrado de información.

Según su contenido. De acuerdo a la naturaleza de la información contenida, pueden ser:

- **Bibliográficas**. Contienen diverso material de lectura (libros, revistas, etc.) ordenado a partir de información clave como son los datos del autor, del editor, del año de aparición, del área temática o del título del libro, entre otras muchas posibilidades.
- **De texto completo**. Se manejan con textos históricos o documentales, cuya preservación debe ser a todo nivel y se consideran fuentes primarias.
- **Directorios**. Listados enormes de datos personalizados o de direcciones de correo electrónico, números telefónicos, etc. Las empresas de servicios manejan enormes directorios clientelares, por ejemplo.
- **Especializadas**. Bases de datos de información hiperespecializada o técnica, pensadas a partir de las necesidades puntuales de un público determinado que consume dicha información.

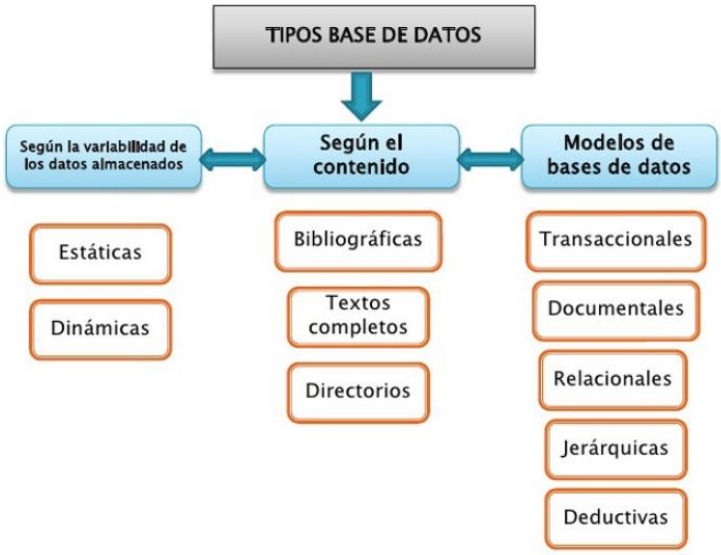

Figura 15. Tipos de bases de datos.

2.3.2.1 Bases de datos relacionales

Además de la clasificación por la función de las bases de datos, estas también se pueden clasificar de acuerdo a su modelo de administración de datos.

Un modelo de datos es básicamente una "descripción" de algo conocido como contenedor de datos (algo en donde se guardan los datos), así como de los métodos para almacenar y recuperar datos de esos contenedores. Los modelos de datos no son cosas físicas: son abstracciones que permiten la implementación de un sistema eficiente de base de datos; por lo general se refieren a algoritmos, y conceptos matemáticos.

El modelo relacional es el modelo utilizado en la actualidad para representar problemas reales y administrar datos dinámicamente. En este modelo, el lugar y la forma en que se almacenen los datos no tienen relevancia (a diferencia de otros modelos como el jerárquico y el de red). Esto tiene la considerable ventaja de que es más fácil de entender y de utilizar para un usuario esporádico de la base de datos. La información puede ser recuperada o almacenada mediante "consultas" que ofrecen una amplia flexibilidad y poder para administrar la información.

2.3.3 Gestores de datos

Existen programas denominados sistemas gestores de bases de datos, abreviado SGBD (del inglés *Database Management System* o DBMS), que permiten almacenar y posteriormente acceder a los datos de forma rápida y estructurada. Las propiedades de estos DBMS, así como su utilización y administración, se estudian dentro del ámbito de la informática.

Las aplicaciones más usuales son para la gestión de empresas e instituciones públicas; También son ampliamente utilizadas en entornos científicos con el objeto de almacenar la información experimental.

La base de datos y el software SGBD pueden estar distribuidos en múltiples sitios conectados por una red. Hay de dos tipos:

- **Distribuidos homogéneos**: utilizan el mismo SGBD en múltiples sitios.
- **Distribuidos heterogéneos**: Da lugar a los SGBD federados o sistemas multibase de datos en los que los SGBD participantes tienen cierto grado de autonomía local y tienen acceso a varias bases de datos autónomas preexistentes almacenados en los SGBD, muchos de estos emplean una arquitectura cliente-servidor.

2.3.4 MySQL

MySQL es un sistema de gestión de bases de datos relacional (Figura 16) desarrollado bajo licencia

dual: Licencia pública general/Licencia comercial por Oracle Corporation y está considerada como la base datos de código abierto más popular del mundo, y una de las más populares en general junto a Oracle y Microsoft SQL Server, sobre todo para entornos de desarrollo web. Está desarrollado en su mayor parte en ANSI C y C++. Tradicionalmente se considera uno de los cuatro componentes de la pila de desarrollo LAMP y WAMP. MySQL funciona sobre múltiples plataformas, incluyendo: AIX, BSD, FreeBSD, HP-UX, Kurisu OS, GNU/Linux, Mac OS X, NetBSD, OpenBSD, OS/2 Warp, QNX, SGI IRIX, Solaris, SunOS, SCO OpenServer, SCO UnixWare, Tru64, eBD, Windows (multiples versiones), OpenVMS.

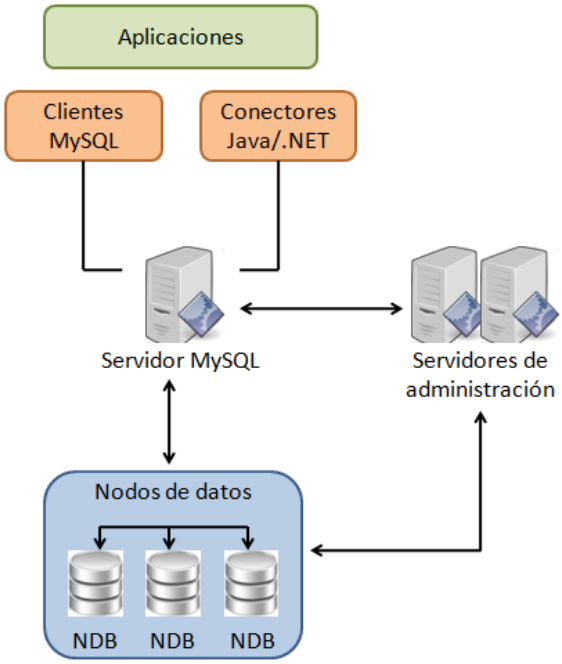

Figura 16. Jerarquía en aplicaciones de MySQL.

Inicialmente, MySQL carecía de elementos considerados esenciales en las bases de datos relacionales, tales como integridad referencial y transacciones. A pesar de ello, atrajo a los desarrolladores de páginas web con contenido dinámico, justamente por su simplicidad.

Poco a poco los elementos de los que carecía MySQL están siendo incorporados tanto por desarrollos internos, como por desarrolladores de software libre. Entre las características disponibles en las últimas versiones se puede destacar:

- Amplio subconjunto del lenguaje SQL. Algunas extensiones son incluidas igualmente.
- Disponibilidad en gran cantidad de plataformas y sistemas.
- Posibilidad de selección de mecanismos de almacenamiento que ofrecen diferentes velocidades de operación, soporte físico, capacidad, distribución geográfica, transacciones...
- Transacciones y claves foráneas.
- Conectividad segura.
- Replicación.
- Búsqueda e indexación de campos de texto.

MySQL es un sistema de administración relacional de bases de datos. Una base de datos relacional archiva datos en tablas separadas en vez de colocar todos los datos en un gran archivo. Esto permite velocidad y flexibilidad. Las tablas están conectadas por relaciones definidas que hacen posible combinar datos de diferentes tablas sobre pedido.

2.3.5 SQL

SQL (Structured Query Language) (8) es un lenguaje estándar e interactivo de acceso a bases de datos relacionales que permite especificar diversos tipos de operaciones en ellas, gracias a la utilización del álgebra y de cálculos relacionales, el SQL brinda la posibilidad de realizar consultas con el objetivo de recuperar información de las bases de datos de manera sencilla (Figura 17). Las consultas toman la forma de un lenguaje de comandos que permite seleccionar, insertar, actualizar, averiguar la ubicación de los datos, y más.

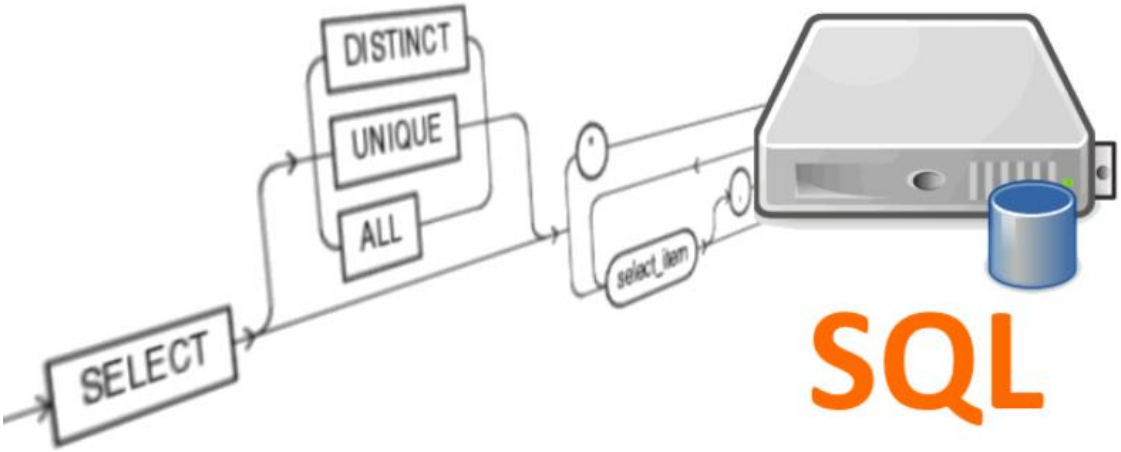

Figura 17. Consulta de base de datos mediante sentencias.

SQL es un lenguaje de acceso a bases de datos que explota la flexibilidad y potencia de los sistemas relacionales y permite así gran variedad de operaciones.

Es un lenguaje declarativo de "alto nivel" o "de no procedimiento" que, gracias a su fuerte base teórica y su orientación al manejo de conjuntos de registros —y no a registros individuales— permite una alta productividad en codificación y la orientación a objetos. De esta forma, una sola sentencia puede equivaler a uno o más programas que se utilizarían en un lenguaje de bajo nivel orientado a registros. SQL también tiene las siguientes características (8):

- **Lenguaje de definición de datos**: El LDD de SQL proporciona comandos para la definición de esquemas de relación, borrado de relaciones y modificaciones de los esquemas de relación.
- **Lenguaje interactivo de manipulación de datos**: El LMD de SQL incluye lenguajes de consultas basado tanto en álgebra relacional como en cálculo relacional de tuplas.
- **Integridad**: El LDD de SQL incluye comandos para especificar las restricciones de integridad que deben cumplir los datos almacenados en la base de datos.
- **Definición de vistas**: El LDD incluye comandos para definir las vistas.
- **Control de transacciones**: SQL tiene comandos para especificar el comienzo y el final de una transacción.
- **SQL incorporado y dinámico**: Esto quiere decir que se pueden incorporar instrucciones de SQL en lenguajes de programación como: C++, C, Java, PHP, Cobol, Pascal y Fortran.
- **Autorización**: El LDD incluye comandos para especificar los derechos de acceso a las relaciones y a las vistas.

SQL es un lenguaje declarativo. O sea, que especifica qué es lo que se quiere y no cómo conseguirlo, por lo que una sentencia no establece explícitamente un orden de ejecución.

Los sistemas de bases de datos modernos poseen un componente llamado optimizador de consultas. Este realiza un detallado análisis de los posibles planes de ejecución de una consulta SQL y elige aquel que sea más eficiente para llevar adelante la misma.

Existe una ampliación de SQL conocida como FSQL (Fuzzy SQL, SQL difuso) que permite el acceso a bases de datos difusas, usando la lógica difusa. Este lenguaje ha sido implementado a nivel experimental y está evolucionando rápidamente.

2.3.6 PHP

PHP es un lenguaje de código abierto muy popular, adecuado para desarrollo web y que puede ser incrustado en HTML. Es popular porque un gran número de páginas y portales web están creadas con PHP. Incrustado en HTML significa que en un mismo archivo vamos a poder combinar código PHP con código HTML, siguiendo unas reglas. Esta característica permite generar páginas webs dinámicas, es decir, páginas cuyos contenidos pueden cambiar en base a los cambios que haya en una base de datos, de búsquedas o aportaciones de los usuarios, etc.

Lo que distingue a PHP de JavaScript es que el código es ejecutado en el servidor, generando HTML y enviándolo al cliente como si fuera una página web estática. El cliente recibirá los resultados que el servidor devuelve después de interpretar el código PHP, sin ninguna posibilidad de determinar qué código ha producido el resultado recibido. Es decir, a través de nuestro navegador podríamos ver el código HTML, pero nunca el código PHP que dio lugar al resultado HTML. El servidor web puede ser incluso configurado para que los usuarios no puedan saber si estás o no utilizando PHP.

2.3.6.1 PhpMyAdmin

PhpMyAdmin es una herramienta escrita en PHP con la intención de manejar la administración de MySQL a través de páginas web, utilizando un navegador web. Actualmente puede crear y eliminar Bases de Datos, crear, eliminar y alterar tablas, borrar, editar y añadir campos, ejecutar cualquier sentencia SQL, administrar claves en campos, administrar privilegios, exportar datos en varios formatos y está disponible en 72 idiomas.

3 DISEÑO

En este apartado se exponen las distintas etapas en las que se ha desarrollado cronológicamente el proyecto expuesto en este documento. Para reducir la complejidad y asegurar la evolución adecuada del sistema objetivo, se ha conseguido dividir el proyecto en tres fases, partiendo de los fundamentos de la red de comunicaciones hasta el tratamiento y presentación de la información recabada.

La primera fase consiste en lograr establecer una comunicación entre las diferentes motas que conformarán la red final. Estas comunicaciones son del tipo UDP y el enrutado de la red es RPL. La labor de armar y organizar la red recaerá sobre una mota que actuará de servidor, a la cual enviarán el resto de motas la información del sensor de temperatura que tienen integrados.

En la segunda fase, se sustituye la mota servidor por un border-router. Este border-router, además de realizar las mismas funciones que la mota servidora, incorpora la capacidad de establece una conexión TCP/IP con un ordenador. Gracias a esta conexión es posible acceder mediante consultas HTTP a la información de enrutado, y conocer la relación que el border-router ha creado entre las motas de la red.

Para finalizar, se implementa el servidor LAMP en el ordenador que mantiene la conexión con el border-router. Este servidor gestiona la recepción y almacenamiento de los datos recibidos en el border-router, así como dotar al sistema de un método de acceso para el usuario de esos datos.

3.1 Entorno de desarrollo

Para desarrollar todo el proyecto, se ha utilizado un entorno Linux. Este será ejecutado usando una imagen de la maquina virtual de Linux ejecutada en Virtual Box, no obstante, también podría utilizarse VMware. En este entorno, es posible hacer uso del programa Code Comporser Studio para la programación de los dispositivos en un entorno cómodo y la compilación se realizará desde el terminal, en el cuál también se cargarán los programas en los diferentes dispositivos. Por último, esta máquina virtual incluye diferentes simuladores, tanto de dispositivos como de redes de comunicación, para ejecutar y comprobar el funcionamiento de los códigos desarrollados antes de ser probados en los equipos físicos. El simulador a utilizar y por tanto instalar es Cooja, que permite crear y monitorizar redes de sensores y las comunicaciones entre motas.

3.1.1 Instalación de la máquina virtual

La preparación del entorno Linux comienza por la instalación del software en el que se ejecutará la imagen de la máquina virtual. Desde "https://www.virtualbox.org/", es posible acceder al distribuidor de VirtualBox, desde el cual se descarga la versión 6.1 a fecha de este proyecto. Por otro lado, en "https://www.osboxes.org/lubuntu/" está disponible la descarga de una imagen de la maquina virtual de Lubuntu para ser ejecutada en VirtualBox. Lubuntu es un sistema operativo rápido y ligero, cuyo núcleo esta basado en Linux y Ubuntu, focalizado en la velocidad y la eficiencia, lo que permite ser ejecutado en sistemas con recursos limitados.

Una vez instalado, en la ventana principal se debe importar la imagen de Lubuntu desde la pestaña de archivos. Antes de importarla se puede ver la información de sistema de la maquina virtual. Antes de ejecutarla y con la máquina virtual apagada, es necesario abrir la ventana de configuración y ajustar varios aspectos como se indica en la Figura 18:

- **Sistema:** ajustar la memoria RAM, mínimo 2GB, y el número de núcleos al gusto.
- **Pantalla:** elegir la memoria de vídeo, mínimo 32 MB.
- **Red:** verifivar que el adaptador 1 se encuentra en modo NAT
- **USB:** comprobar que el controlador de USB es al menos 2.0 (EHCI).

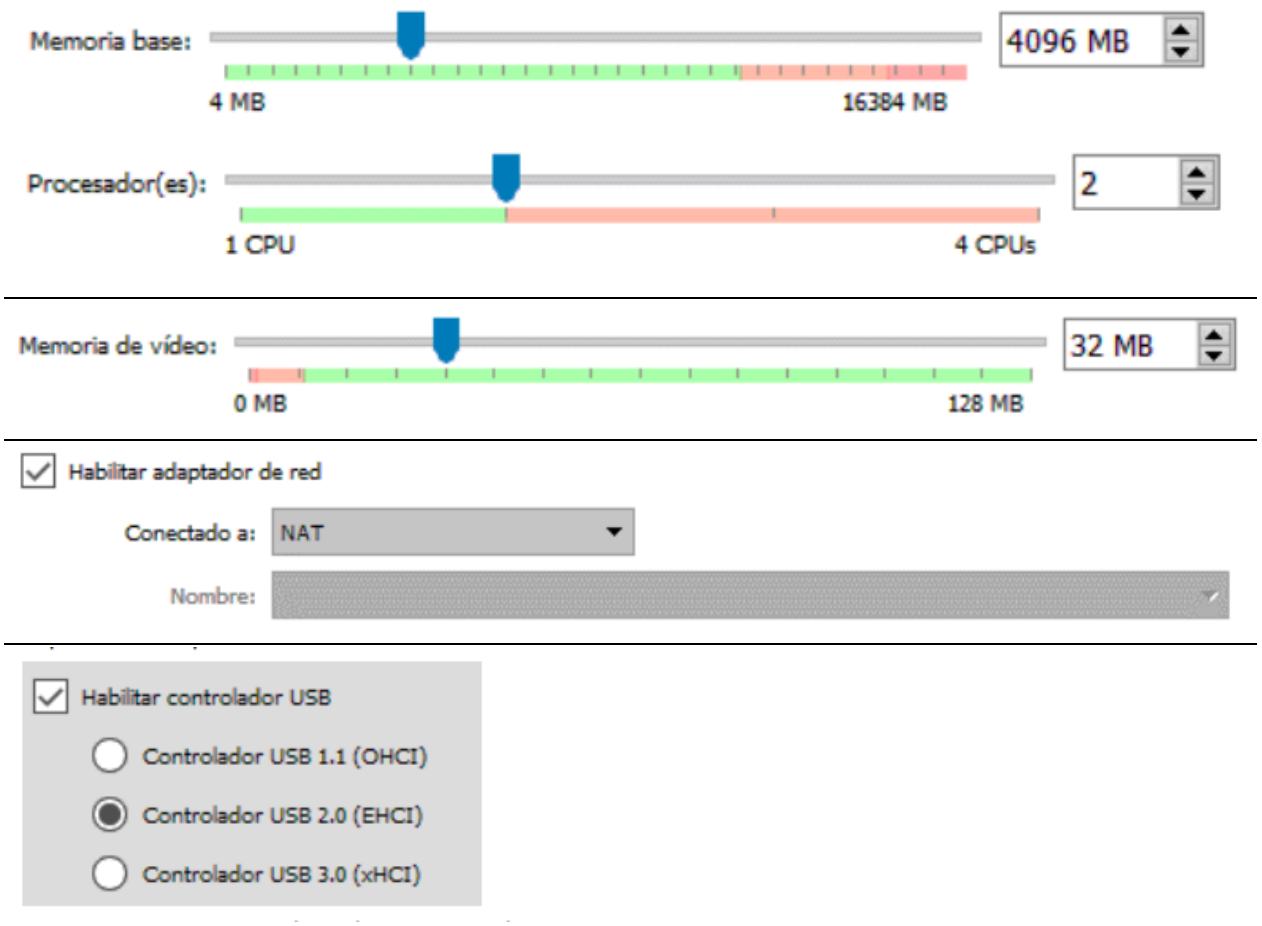

Figura 18. Ventana de configuración de VirtualBox.

En esta situación, ya está preparado el entorno de desarrollo para su uso.

3.1.2 Descarga de Contiki-NG

Desde el entorno de Lubuntu, el siguiente paso es descargar el toolchains (8) desde la aplicación terminal:

1. Instalar varias herramientas de desarrolo: primero se instalan varios paquetes mediante el comando "`sudo apt install build-essential doxygen git curl python-serial srecord rlwrap`". También es conveniente instalar "net-tools" con el comando "`sudo apt-get install net-tools`".
2. Instalar el compilador ARM: necesario para cargar los archivos binarios en plataformas CC2538DK o Zoul (que es el caso de este proyecto). Para instalar el compiador es necesario ejecutar el comando "`sudo apt install gcc-arm-none-eabi`" (Figura 19. Descarga del compilador ARM.).

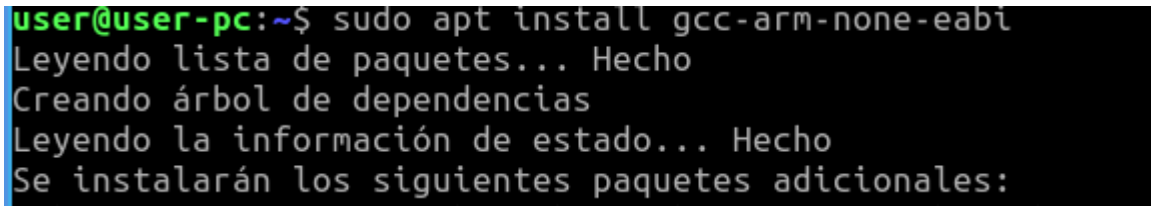

Figura 19. Descarga del compilador ARM.

3. Instalar el compilador MSP430: utilizado para las simulaciones. Ejecutando el comando "`sudo apt install gcc-msp430`" se descargará el compilador.
4. Descargar Contiki-NG desde su repositorio de Github (https://github.com/contiki-ng/contiki-ng). Además, es necesario actualizar los submodulos de la carpeta descargada de "contiki-ng" como se

hace en la Figura 20.

```
user@user-pc:~$ sudo git clone https://github.com/contiki-ng/contiki-ng.git
user@user-pc:~$ cd contiki-ng
user@user-pc:~/contiki-ng$ sudo git submodule update --init
```

Figura 20. Descargar de Contiki-NG.

5. Instalar Java for Cooja: imprescindible para poder ejecutar el simulador Cooja (Figura 21. Descarga de orden ant para Cooja.).

```
user@user-pc:~$ sudo apt install default-jdk ant
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias
Leyendo la información de estado... Hecho
Se instalarán los siguientes paquetes adicionales:
```

Figura 21. Descarga de orden ant para Cooja.

En caso de necesitar más información, es posibe consultar la guía de Contiki-NG de los desarrolladores "https://github.com/contiki-ng/contiki-ng/wiki/Toolchain-installation-on-Linux".

Con estos pasos, se habrá creado la carpeta "contiki-ng" (Figura 22), en la que están guardados los archivos de arquitecturas de plataformas soportadas, los archivos del SO, herramientas de simulación y varios ejemplos de Contiki-NG. Desde esta carpeta se desarrollará el diseño de los programas, la ejecución de simulaciones y la compilación y carga en las plataformas.

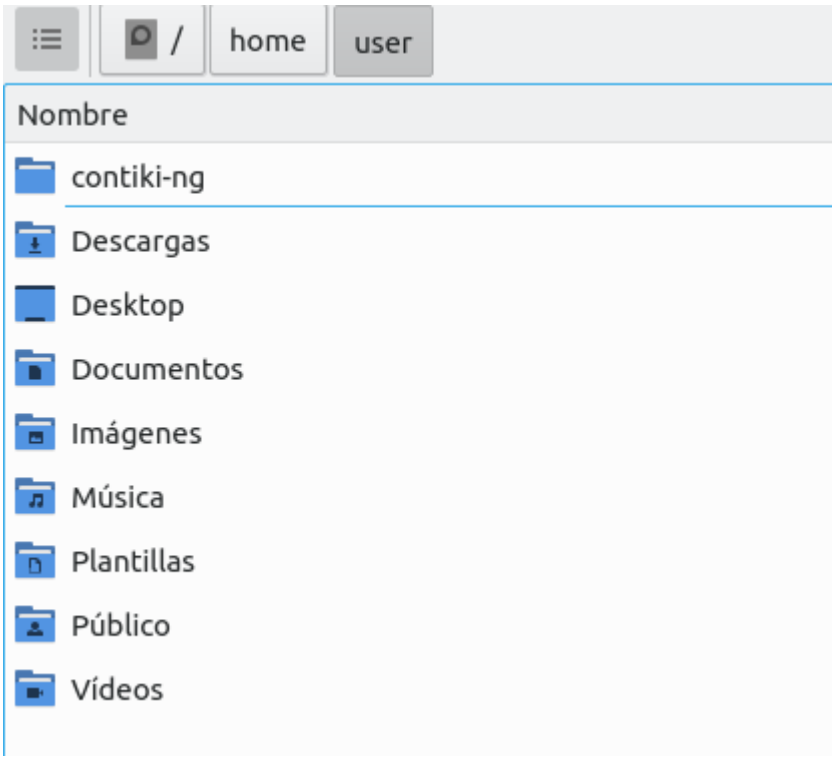

Figura 22. Ubicación de carpeta "contiki-ng".

3.1.3 Preparación de archivo Cooja

Para terminar, es necesario realizar una modificación en los archivos y carpeta del simulador Cooja para poder ejecutarlo. Se debe descargar el archivo de la dirección "https://search.maven.org/artifact/javax.xml.bind/jaxb-api/2.4.0-b180830.0359/jar" y guardarlo en "-/contiki-ng/tools/cooja/lib". Luego, hay que incluir el archivo .jar descargado en el archivo "-/contiki-ng/tools/cooja/build.xml", para lo cual hay que incluir en todos los apartados de <classpath> la localización del archivo descargado como se muestra en la Figura 23.

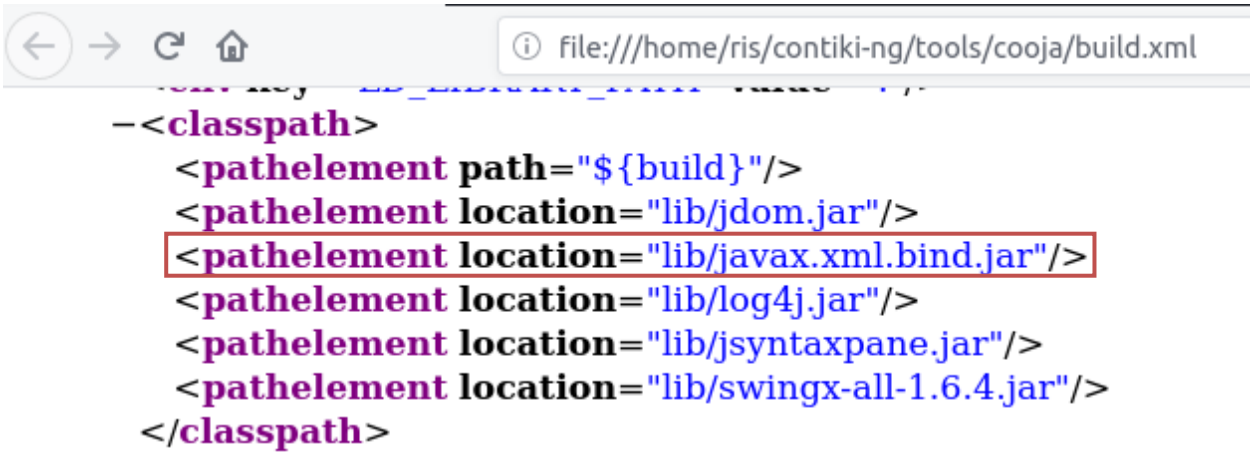

Figura 23. Inclusiones en el archivo "build.xml".

Tras añadir el archivo .jar, es posible que aun se produzca un error en relación al simulador MSPSim. En tal caso, el propio programa emite una propuesta de ejecutar una actualización de los módulos del directorio con lal directiva "--recusive" (Figura 24). Después del actualizar, ya será posible abrir Cooja.

```
user@user-pc:~/contiki-ng/tools/cooja$ sudo git submodule update --init --recursive
Submódulo 'mspsim' (https://github.com/contiki-ng/mspsim.git) registrado para ruta 'mspsim'
Clonando en '/home/user/contiki-ng/tools/cooja/mspsim'...
Ruta de submódulo 'mspsim': check out realizado a '4e48946494a2e28cb438a9b3b2fddec4545565c5'
user@user-pc:~/contiki-ng/tools/cooja$ sudo ant run
Buildfile: /home/user/contiki-ng/tools/cooja/build.xml
```

Figura 24 . Descarga de archivos MSPSim.

3.2 Comunicación entre motas

Como ya se ha explicado, la primera fase del proyecto establece la base principal del sistema que es la comunicación y configuración de la red de sensores. Para ello, se armará una red básica RPL con comunicación UDP, creando una mota que actue como servidor para enrutar todas las demás motas de la red hacia el. Los códigos de los programas requeridos se recogen en el apartado 6.1, correspondiente a los códigos fuente "udp-server.c" y "udp-client-temp.c". En estos códigos se establecen la comunicación UDP, transmisión o recepción según corresponda, y la lectura de los sensores para las motas cliente.

3.2.1 Programación de motas

Desde el repositorio de Contiki-NG, se propone un ejemplo básico de comunicación UDP entre nodos. Primero se incluye la libreria "simple-udp.h", se define una estuctura "simple_udp_connection" y de define el registro de UDP con la información de puertos y la función de recepción (Figura 25).

```
#include "simple-udp.h"
...
static struct simple_udp_connection udp_conn;
simple_udp_register(&udp_conn, UDP_PORT, NULL, UDP_PORT, udp_rx_callback);
```

Figura 25. Instancias UDP.

La defición de la función de recepción se realiza como se indica en la Figura 26. Para cada mensaje que se reciba, se ejecutará el código de la función para poder procesar tanto los datos del mensaje, como las

direcciones y los puertos de la comunicación si se desea.

```
static void
udp_rx_callback(struct simple_udp_connection *c,
         const uip_ipaddr_t *sender_addr,
         uint16_t sender_port,
         const uip_ipaddr_t *receiver_addr,
         uint16_t receiver_port,
         const uint8_t *data,
         uint16_t datalen)
{
...
}
```

Figura 26. Función de recepción UDP.

Por último, para enviar un mensaje se utiliza la función "simple_udp_sendto", indicada en la Figura 27, para enviar un mensaje a partir de una conexión, unos datos y su tamaño y una dirección destino indicadas.

```
uint8_t payload[64] = { 0 };
simple_udp_sendto(&udp_conn, payload, 2, &destination_ipaddr);
```

Figura 27. Envío de mensaje UDP.

Con estas pautas ya es posible la transmisión, recepción y procesamiento de mensajes entre nodos. Sin embargo esto no es suficiente para establecer un red de comunicaciones autoconfigurabe. Para ello es necesario el uso de algoritmos de enrutamiento para configurar las direcciones de todos los nodos y las asociaciones entre ellos. Con ese fin, se hace uso del enrutamiento de la pila de red integrada en Contiki-NG.

El algoritmo de enrutamiento que integra la pila de red es RPL, en sus versiones Classic y Lite. Para usarlo, basta con incluir la ruta "net/routing/routing.h" en el código del programa y utilizar los siguientes métodos del objeto NETSTACK_ROUTING:

- root_start(): arranca el enrutado de la red tomando el nodo que ejecuta esta función como nodo raíz.
- node_is_reachable(): activa una reparación de topología global para comprobar si el nodo se encuentra dentro del enrutado de la red.
- get_root_ipaddr(uip_ipaddr_t *ipaddr): devuelve la IPv6 global de un nodo de enrutamiento raíz.

Con estas funciones es posible crear un nodo que genere un enrutamiento de cualquier otro nodo hasta el y varios nodos que puedan enviar mensajes UDP al nodo raíz. En la Figura 28 se muestra un fragmento del apartado 6.1.2 del Anexo, en el cual un nodo comprueba su estado en el enrutamiento de la red y envia un mensaje UDP al nodo raíz.

```
        if(NETSTACK_ROUTING.node_is_reachable() &&
NETSTACK_ROUTING.get_root_ipaddr(&dest_ipaddr)) {
            /* Send to DAG root */
            snprintf(str, sizeof(str), "%1x",LEDS_RED);

            LOG_INFO("Sending: '%.*s' to ", strlen(str), (char *) str);
            LOG_INFO_6ADDR(&dest_ipaddr);
            LOG_INFO_("\n");

            simple_udp_sendto(&udp_conn, str, strlen(str), &dest_ipaddr);

        } else {
            LOG_INFO("Not reachable yet\n");
        }
```

Figura 28. Código de envio UDP y comprobación de ruta.

3.2.2 Compilación y carga en plataforma

Tras comprender el funcionamiento que se desea en las motas, se debe asegurar que la programación es correcta mediante la compilación y carga en las plataformas finales. El proceso se realiza desde el terminal de Lubuntu. Como ejemplo para este apartado, se va a compilar el código fuente "udp-server.c", ya que para las motas cliente ("udp-client-temp.c") se procede de manera similar. Los pasos a seguir para compilar y cargar un programa son los siguientes:

1. **Compilación:** Tras ubicar el terminal en la carpeta que contenga el archivo fuente que se desee compilar, ejecutar el comando: `make TARGET="nombre de la familia" BOARD="nombre de la placa" "archivo".` Para conocer la palabra clave BOARD de cada placa se puede consultar el archivo "contiki-ng/arch/platform/zoul/Makefiel.zoul", donde se encuentran las palabras clave BOARD de todas las placas de la familia zoul. En la Figura 29 se pone un ejemplo del programa udp-server.c para la tarjeta RE-Mote revisión b de la familia Zoul.

```
ris@ris-pc:~/contiki-ng/TFM/rpl-udp$ make TARGET=zoul BOARD=remote-revb udp-server
  CC        ../../arch/cpu/cc2538/./ieee-addr.c
  CC        ../../arch/cpu/cc2538/cc2538.lds
  CC        ../../arch/cpu/cc2538/./startup-gcc.c
  CC        udp-server.c
  LD        build/zoul/remote-revb/udp-server.elf
  OBJCOPY   build/zoul/remote-revb/udp-server.elf --> build/zoul/remote-revb/udp-se
  SREC_CAT  build/zoul/remote-revb/udp-server.i16hex --> build/zoul/remote-revb/udp
  OBJCOPY   build/zoul/remote-revb/udp-server.elf --> build/zoul/remote-revb/udp-se
  CP        build/zoul/remote-revb/udp-server.elf --> build/zoul/remote-revb/udp-se
  CP        build/zoul/remote-revb/udp-server.zoul --> udp-server.zoul
rm build/zoul/remote-revb/udp-server.i16hex build/zoul/remote-revb/obj/startup-gcc.
```

Figura 29. Compilación de programa desde el terminal.

   Tras la compilación, se habrá generado un archivo ."familia" (udp-server.zoul en el ejemplo) en el directorio del código fuente. Este será el archivo que se carge en la plataforma.

2. **Carga:** Para compilar el programa, se debe conectar la plataforma al ordenador mediante un puerto USB. Es fundamental que el puerto USB al que se conecte sea USB 2.0 para poder ser reconocido por a máquina virtual y esta pueda gestionarlo. Para reconocer el dispositivo conectado hay que pulsar con el botón derecho del ratón sobre el icono USB de la barra de herramientas. Se desplegará un menú similar al de la Figura 30, en el cual habrá que seleccionar el dispositivo que deseemos programas. Una vez seleccionado correctamente, aparecerá un check azul en el dispositivo indicando que se ha conectado correctamente.

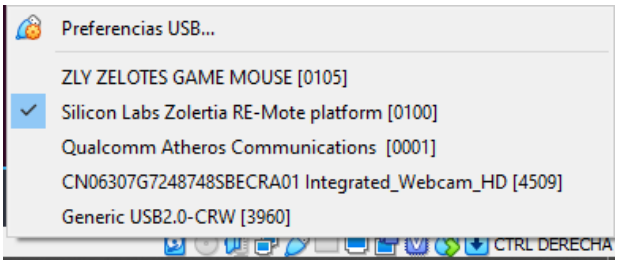

Figura 30. Selección de conexión USB de la plataforma.

Una vez conectado el dispositivo y manteniendo desde el terminal la carpeta en la que se compiló el programa, se ejecuta el comando: `make TARGET="nombre de la familia" BOARD="nombre de la placa" "archivo".upload.` En la Figura 31 se muestra la información de carga del ejemplo. En el caso de que no aparezca ninguna información, es necesario rehacer la conexión como en el apartado anterior hasta que se carge correctamente.

```
ris@ris-pc:~/contiki-ng/TFM/rpl-udp$ make TARGET=zoul BOARD=remote-revb udp-server.upload
  CC        ../../arch/cpu/cc2538/./ieee-addr.c
  CC        ../../arch/cpu/cc2538/cc2538.lds
  CC        ../../arch/cpu/cc2538/./startup-gcc.c
  CC        udp-server.c
  LD        build/zoul/remote-revb/udp-server.elf
  OBJCOPY   build/zoul/remote-revb/udp-server.elf --> build/zoul/remote-revb/udp-server.bi
Flashing /dev/ttyUSB0
Opening port /dev/ttyUSB0, baud 460800
Reading data from build/zoul/remote-revb/udp-server.bin
Cannot auto-detect firmware filetype: Assuming .bin
Connecting to target...
CC2538 PG2.0: 512KB Flash, 32KB SRAM, CCFG at 0x0027FFD4
Primary IEEE Address: 00:12:4B:00:09:DF:8E:B1
Erasing 524288 bytes starting at address 0x00200000
    Erase done
Writing 516096 bytes starting at address 0x00202000
Write 8 bytes at 0x0027FFF8F00
    Write done
Verifying by comparing CRC32 calculations.
    Verified (match: 0x37f4f0ee)
rm build/zoul/remote-revb/obj/startup-gcc.o udp-server.o
```

Figura 31. Carga de programa desde el terminal.

3. **Conexión Serie:** Si se desea observar información durante la ejecución del programa en las sentencias "LOG()", al igual que aparecen en las simulaciones, se puede ejecutar el comando `make TARGET="nombre de la familia" BOARD="nombre de la placa" login.` En la Figura 32 se puede ver la información de contiki que se está ejecutando al igual que aparece en las simulaciones al ejecutar el comando.

```
ris@ris-pc:~/contiki-ng/TFM/rpl-udp$ make TARGET=zoul BOARD=remote-revb login
rlwrap ../../tools/serial-io/serialdump -b115200 /dev/ttyUSB0
connecting to /dev/ttyUSB0 [OK]
[INFO: Main      ] Starting Contiki-NG-release/v4.4-33-gf87846cfa-dirty
[INFO: Main      ] - Routing: RPL Lite
[INFO: Main      ] - Net: sicslowpan
[INFO: Main      ] - MAC: CSMA
[INFO: Main      ] - 802.15.4 PANID: 0xabcd
[INFO: Main      ] - 802.15.4 Default channel: 26
[INFO: Main      ] Node ID: 36529
[INFO: Main      ] Link-layer address: 0012.4b00.09df.8eb1
[INFO: Main      ] Tentative link-local IPv6 address: fe80::212:4b00:9df:8eb1
[INFO: Zoul      ] Zolertia RE-Mote revision B platform
[INFO: App       ] DAG root started
```

Figura 32. Conexión serial desde el terminal.

3.2.3 Simulaciones en Cooja

Una vez los programas de los microcontroladores están listos, se desea comprobar que el funcionamiento es el deseado. El principal interés es observar el funcionamiento de los microcontroladores a nivel de red y poder visualizar las transmisiones que se realicen en ella. Como es habitual en proyectos de esta índole, se procede a realizar una simulación de la red que permitirá mejorar el entendimiento de los equipos.

Sin embargo, para poder simularlo correctamente es necesario comentar y/o modificar ciertas lineas del programa "udp-client-temp.c". Esto es debido a que la simulación se realizará sobre una plataforma diferente a la implementación física final, lo que obliga a modificar los puntos referentes a los sensores integrados en cada placa (9). Las lineas a modificar se muestran en la Figura 33. En el caso de la linea 80, se debe sustituir la lectura del sensor por una generación aleatoria para la simulación.

```
 7 #include "net/ipv6/simple-udp.h"
 8
 9 //#include "dev/zoul-sensors.h"
10 #include "dev/leds.h"
11
60
61 PROCESS_THREAD(udp_client_process, ev, data)
62 {
63   //SENSORS_ACTIVATE(cc2538_temp_sensor);
64
65   static struct etimer periodic_timer;
78     /* Send to DAG root */
79     int temp = random_rand()%15+10;
80 //cc2538_temp_sensor.value(CC2538_SENSORS_VALUE_TYPE_CONVERTED);
81
82     LOG_INFO("Sending Temp %d ( %d : %d ) to ", temp,x_coor,y_coor);
```

Figura 33. Modificación del programa "udp-client-temp.c".

Hechas las modificaciones, se pasa a compilar ambos programas, como se indica en el apartado anterior. Ambos deben compilarse con la palabra clave "TARGERT=z1", correspondiente a la plataforma en la que se simularán. Tras obtener los archivos binarios ".z1", se puede ejecutar la aplicación Cooja.

Cooja es un simulador de redes inalámbricas que permite incorporar motas de varias plataforma y ejecutar un programa en cada una de ellas. Si en estos programas se incluyen transmisiones RF, también es posible seguirlas cronológicamente.

Para ejecutar la aplicación (Figura 34), hay que acceder a la carpeta "tools/cooja" de Contiki-NG y ejecutar la aplicación desde un terminal con el comando "ant run", tras lo cual se abrirá una ventana vacía. Para crear una nueva simulación hay que acceder a "Archivo > Crear simulación".

```
ris@ris-pc:~$ cd contiki-ng/tools/cooja
ris@ris-pc:~/contiki-ng/tools/cooja$ ant run
Buildfile: /home/ris/contiki-ng/tools/cooja/build.xml
```

Figura 34. Inicio de Cooja.

Al pulsar en la pestaña, se abrirá la ventana de la Figura 35. En esta ventana, además de un nombre para la simulación, es posible elegir un modelo para las transmisiones de RF, el retardo de inicio de los nodos y elegir una semilla para la simulación.

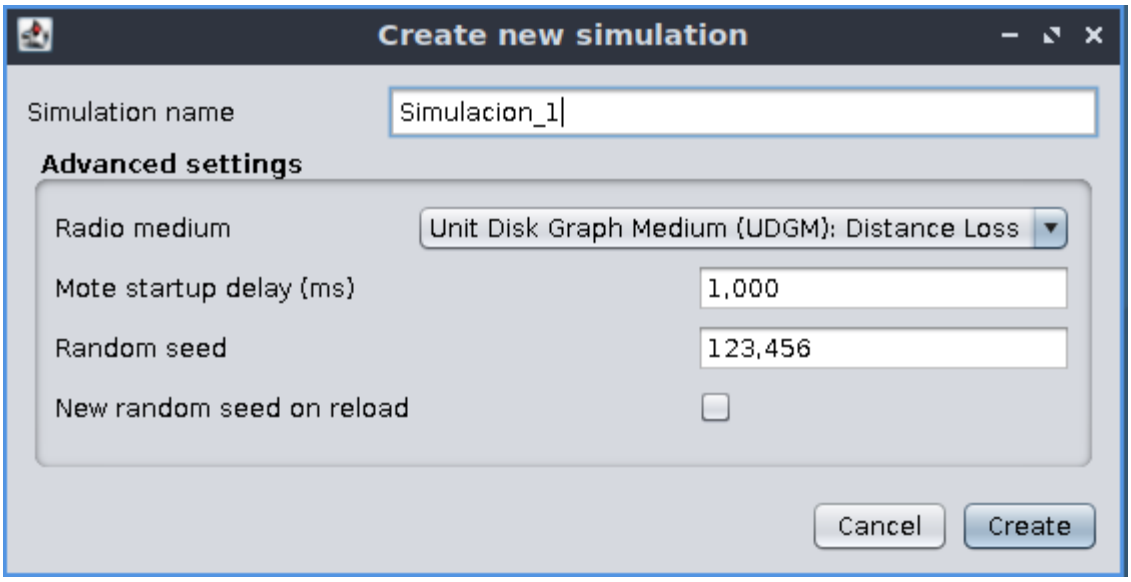

Figura 35. Creación de simulación.

Una vez creada, se abrirá la ventana de simulación de la Figura 36. Esta ventana se compone de diferentes apartados que permiten controlar, configurar y observar varios aspectos de la simulación:

1. **Red**: En esta ventana se pueden ubicar los nodos en las coordenadas que se deseen colocar y ajustar el alcance de sus transmisiones.
2. **Control de Simulación**: Permite iniciar, pausar, ejecutar un paso y reiniciar la simulación por completo.
3. **Salidas de nodos**: Muestras los mensajes que emiten los nodos en las sentencias "LOG" de los programas y el instante de tiempo en el que se producen.
4. **Cronograma**: Indica los momentos en los que se realiza una transmisión y que nodos están al alcance de esa transmisión cronológicamente.

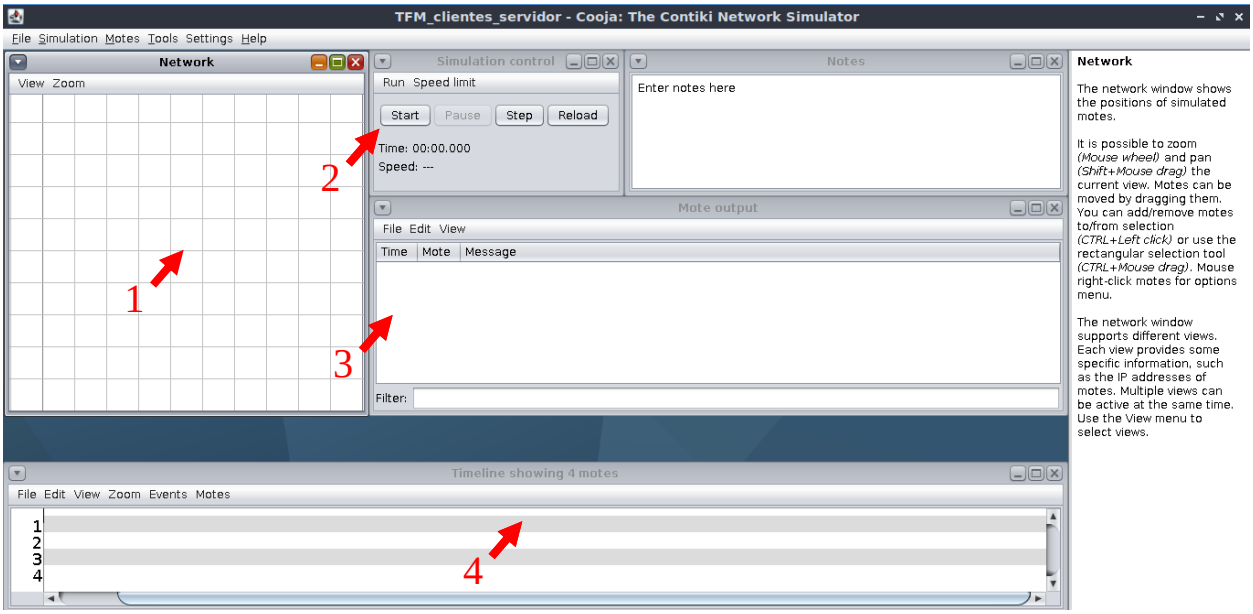

Figura 36. Ventana principal de Cooja.

A continuación se incluiyen las motas que va a intervenir en la simulación. Para este proyecto se va elegir la plataforma Z1 para las simulación de la red., que es una plataforma también de Zolertia que viene incluida por defecto en el simulador. Para incluir un nodo Z1 hay que seleccionar la siguiente pestaña "Motes > add motes > Create new mote type > Z1 mote". Al hacerlo se abrirá una ventana como en la Figura 37, en la que se puede dar un nombre a la mota y elegir el programa que se desee. Es posible elegir un archivo binario para cargar en

la placa o un archivo fuente para ser compilado directamente en la aplicación.

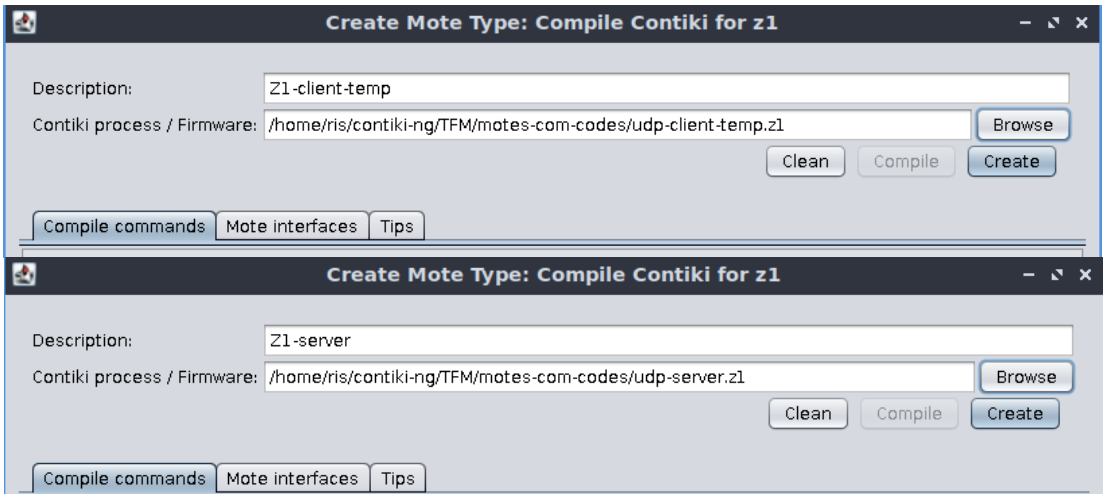

Figura 37. Creación de las motas para simulación.

Tras crear cada mota, se pregunta cuantas se desea incluir en la simulación y su ubicación.Se colocan una mota "Z1-server" y tantas mota "Z1-client-temp" como se desee. También se podrán incluir más motas en otros momentos de la simulación si se desea ampliar la red, sin necesidad de repetir el proceso de creación.

Para el caso de este documento se colocan tres motas cliente y una mota servidora (número 1). Su disposición se puede ver en la Figura 38, en la que la mota servidora tiene al alcance los nodos 2 y 3. La mota 4 no tiene alcance suficiente para tener transmisión directa con la mota servidor (número 1). El algoritmo RPL asignará como padré de la mota 4 la mota 3, quién reenviará todas las transmisiones a la mota servidor.

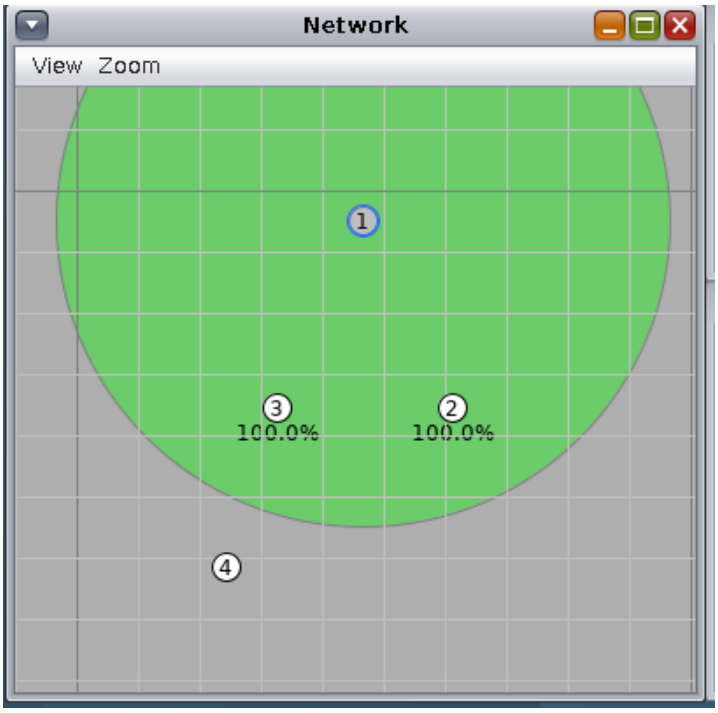

Figura 38. Ubicación de motas.

Una vez dispuesta la red, se puede ejecutar la simulación desde la ventana de control. Al iniciar, en la ventana de salida de las motas aparecerá la información del programa que se está ejecutando: versión del SO, capas de la pila de red y direcciones de enlace. En el caso del servidor, aparece el mensaje adicional del rutado DAG (Figura 39).

```
00:01.250  ID:1   [INFO: Main      ] Starting Contiki-NG-release/v4.4-33-gf87846cfa-dirty
00:01.253  ID:1   [INFO: Main      ] - Routing: RPL Lite
00:01.256  ID:1   [INFO: Main      ] - Net: sicslowpan
00:01.258  ID:1   [INFO: Main      ] - MAC: CSMA
00:01.261  ID:1   [INFO: Main      ] - 802.15.4 PANID: 0xabcd
00:01.265  ID:1   [INFO: Main      ] - 802.15.4 Default channel: 26
00:01.267  ID:1   [INFO: Main      ] Node ID: 1
00:01.272  ID:1   [INFO: Main      ] Link-layer address: c10c.0000.0000.0001
00:01.279  ID:1   [INFO: Main      ] Tentative link-local IPv6 address: fe80::c30c:0:0:1
00:01.283  ID:1   [INFO: Z1        ] CC2420 CCA threshold -45
00:01.286  ID:1   [INFO: App       ] DAG root started
```

Figura 39. Información de inicialización del nodo.

Tras establecer el rutado, las motas clientes mandarán periódicamente un número aleatório simulando medir la temperatura. En el ejemplo de la , la mota 2 manda sus datos a la mota 1, la cual lo recibe y le responde para confirmar la recepción. En el mapa de la red se puede ver como los mensajes llegan también a las motas 4 y 3, pero dado que ninguno de los dos mensajes va dirigido a ellas (como se puede ver por las direcciones IPv6 del destino en el terminal) estas los ignoran.

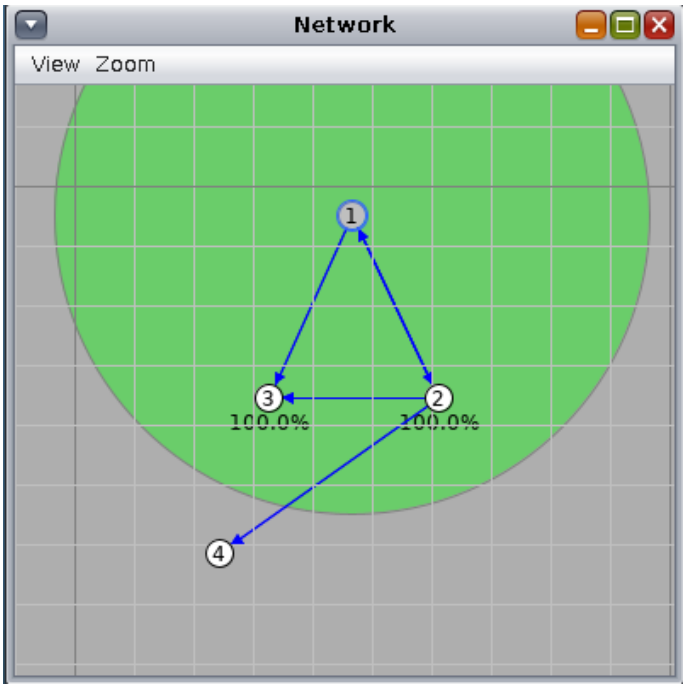

Figura 40. Transmisión de mensaje.

Para los casos especiales como la mota 4 que no tienen alcance directo con el servidor, es interesante ver en el cronograma el momento en el que la mota 4 manda un mensaje (Figura 41, marca azul). Esta manda el mensaje, que solo llega a las motas 2 y 3 (marcas verdes), y tras procesarlo la mota 3 lo reenvía a la mota 1. De manera similar se produce la respuesta.

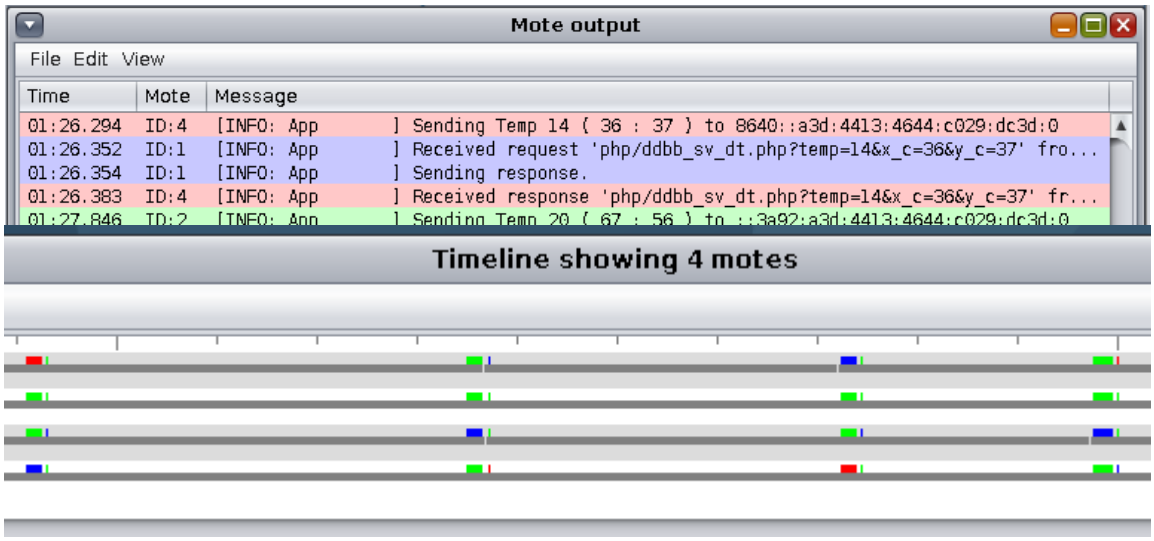

Figura 41. Reenvío de mensajes en la red.

Con esta simulación se puede afirmar el correcto funcionamiento de la red, con lo cual ya se pueden compilar los códigos fuente y cargarlos en las plataformas físicas para verificarlo.

3.2.4 Implementación de la red

Tras haber comprobado en simulación que los programas funcionan correctamente, se procede a compilar los códigos en las plataformas finales. Primero hay que descomentar las lineas anteriormente modificadas del programa "udp-client-temp.c", ya que ahora se va a utilizar el sensor de temperatura de la plataforma "Zoul". Además, se deben indicar las coordenadas en las que se va a ubicar la motes, modificando el valor de las variables "x_coor" e "y_coor" (linea 68 del código) para ubicar el dispositivo en la interfaz de usuario.

El segundo punto a considerar es el estado de la batería. Para asegurar la carga de estas, hay que extraer la batería del interior de la carcasa, desconectarla de la placa y medir el voltaje de sus terminales. En la Figura 42 se indica el montaje a realizar. Dado que el voltaje nominal viene especificado a 3.7V, se puede esperar un voltaje de hasta 4V a máxima carga. Por debajo del voltaje nominal, será necesario cargala, utilizando una fuente de alimentación regulable. En caso de no disponer de una, puede utilizarse un cargador de 5V estandar.

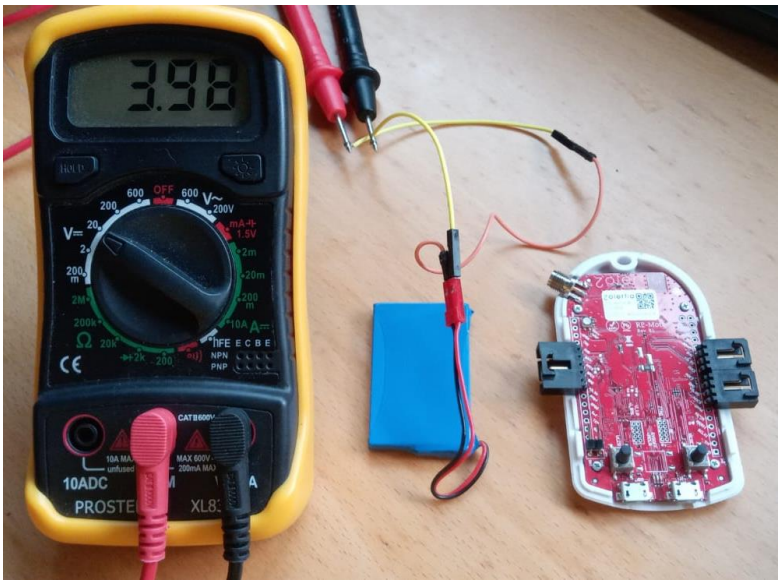

Figura 42. Comprobación de batería.

Tras asegurar a carga de la batería, se compila el programa correspondiente a cada mota, una con el código "udp-server.c" y tantas motas con el programa "udp-client-temp.c" como se deseen.

A la hora de desplegar la red, se recomienda ubicar primero la mota servidor y posteriormente las motas

cliente de una en una, y comprobar que cada nueva mota es capaz de comunicarse desde las ubicaciones escogidas. También deben colocarse de las más cercanas al servidor a las más lejanas, para que se pueda crear el enrutamiento adecuado con las motas intermediarias. En el caso de replicar el despliege de la Figura 38, las motas se desplegarían acorde al número que se la ha asignado a cada una.

Para comprobar que una mota se ha desplegado correctamente, se debe conectar a través de un terminal serie a la mota servidor (apartado Figura 32). Con la mota servidor conectada al ordenador y la mota cliente que se desea comprobar desplegada, se reinician ambos equipos y se espera a que se reciba en el servidor un mensaje como el de la Figura 43. Mensajes recibidos en la mota servidor. Para confirmar que se trata de la baliza adecuada, se deben comprobar que las coordenadas recibidas se corresponden con las coordenadas esperadas según la ubicación, ya que a medida que se despliegue toda la red se recibirán mensajes de múltiples motas. Además se puede comprobar visualmente si una mota cliente está transmitiendo o no correctamente al servidor. Cada vez que una mota envía un mensaje, se invierte el estado de led rojo de la placa, y dado que el periodo de envío para esta prueba es de 10 segundos, es muy sencillo para un usuario realizar la comprobación.

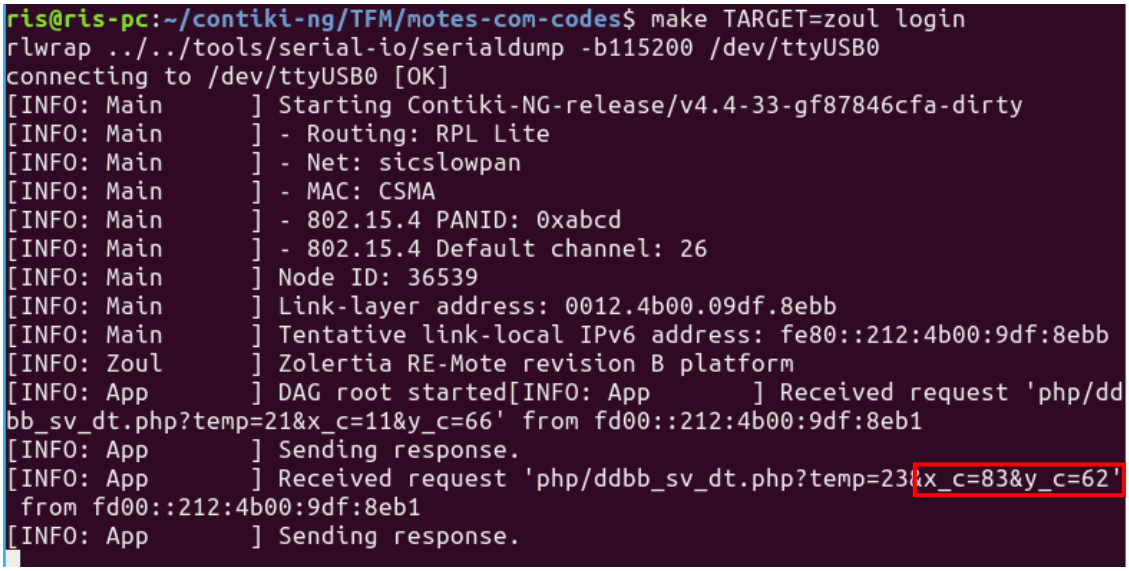

Figura 43. Mensajes recibidos en la mota servidor.

Por último, cabe destacar que es recomendable, pero no imprescindible, cada vez que se despliega una nueva mota y comprobado que las anteriores tiene una comunicación estable, reiniciar las motas de la red. Esto se debe a que, si bien las nuevas motas se acabaría integrando eventualmente en el rutado de la red, este proceso es mucho más rápido si se reinicia el enrutamiento.

3.3 Comunicación con el border-router

Al concluir la primera fase satisfactoriamente, se ha conseguido establecer una red de sensores centralizada y autoconfigurable. De esta manera ya es posible monitorizar la temperatura de una zona y recibir dicha información en un dispositivo terminal. La segunda fase del proyecto busca sustituir la mota servidor por otro dispositivo con mayores prestaciones y capacidades de comunicación: el border router Orion de Zolertia.

Al incorporar este nuevo dispositivo, se busca dotar a la red de mayor conectividad con otros equipo. Se pretende encontrar la manera de conectar la red con un dispositivo final, con el objetivo de transmitir toda la información de la red a un almacenamiento final.

3.3.1 Programa del border-router

La programación del border-router parte de las misma funciones de la mota servidora de la primera fase. A estas, se le suma la capacidad de comunicación de un ordenador con el border router Orion de Zolertia. Se usa el mismo funcionamiento de la red y de rutado de la primera fase, pero sustituyendo la mota servidor por el border-router, además de un acceso TCP/IP para acceder a una pagina web con la información del rutado de la

red.

La configuración de la comunicación UDP se hace de manera similar a la primera fase. Para utilizar el servidor se debe incluir la librería "httpd-simple.h" y se puede consultar el ejemplo "rpl-border-route" como guía de utilización de la misma. En la carpeta de este ejemplo se encuentran los archivos fuente que configuran la conexión y página web. En el archivo "border-router.c" (apartado 6.2.1) se arranca un proto-hilo en el que se gestiona el servidor httpd.

El objetivo de este protohilo es responder ante una solicitud HTTP que se realice desde fuera del border-router mostrando la información de rutado de la red. En la Figura 44, se recoge la jerarquía de funciones definidas para responder a una conexión TCP/IP con la información de la red. Al recibir una conexión, solo se necesita llamar a la función httpd_appcall() y definir el nombre de la función que genere las etiquetas HTML para el cliente TCP/IP.

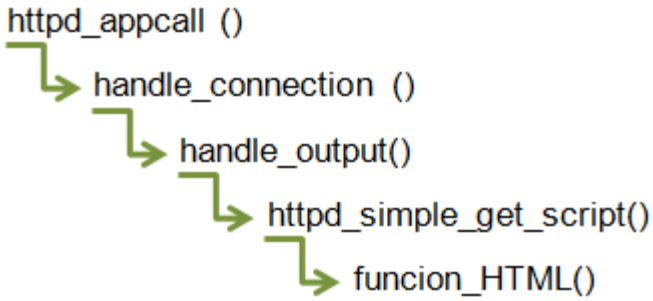

Figura 44. Jerarquía de llamadas a funciones httpd.

El diseño de la página se realiza en la función especificada en el apartado "httpd_simple_get_script()". En este apartado se utilizan las macros "ADD" y "SEND" definidas en el propio archivo (Figura 45). ADD se utiliza para añadir etiquetas HTML al string que se envía al cliente al usar la macro SEND(s). Basandose en esto, se puede mostrar cualquier información contenida en el router, como es el caso del rutado en esta prueba.

```
#define ADD(...) do {
    blen += snprintf(&buf[blen], sizeof(buf) - blen, __VA_ARGS__);
  } while(0)
#define SEND(s) do { \
  SEND_STRING(s, buf); \
  blen = 0; \
} while(0);
```

Figura 45. Definición de macros ADD y SEND.

3.3.2 Simulación con border-router

Para simular el comportamiento de la red con el border-router, se partirá de la simulacion de la fase uno, sustituyendo la mota servidor por el router. Para simularlo, se utilizará la plataforma "sky" que viene incluida entre las plataformas de Cooja. Antes de incluirlo, es necesario la compilación previa del programa con la particularidad de incluir la sentencia SMALL=1 (Figura 46. Compilación para plataforma sky.) a la hora de compilar debido a las restrisciones de memoria de la plataforma.

```
ris@ris-pc:~/contiki-ng/TFM/rpl-border-router$ make TARGET=sky SMALL=1
/bin/sh: 1: test: -lt: unexpected operator
  CC        border-router.c
  LD        build/sky/border-router.sky
  CP        build/sky/border-router.sky --> border-router.sky
rm border-router.o
```

Figura 46. Compilación para plataforma sky.

Una vez incluido el border-router de la misma manera que se realiza en la Figura 37, salvo que en este caso se

trata de la plataforma Sky, se sustituye la mota servidor por el router. Opcionalmente se pueden redistrubir la red de acuerdo a la Figura 47 para que el rutado de la red sea más complejo. Antes de ejecutar la simulación es necesario hacer click derecho sobre el border-router en la ventana de red y acceder al menu "Mote tools > Serial Socket {SERVER}" para iniciar el puerto en la dirección 60001 y poder acceder a el.

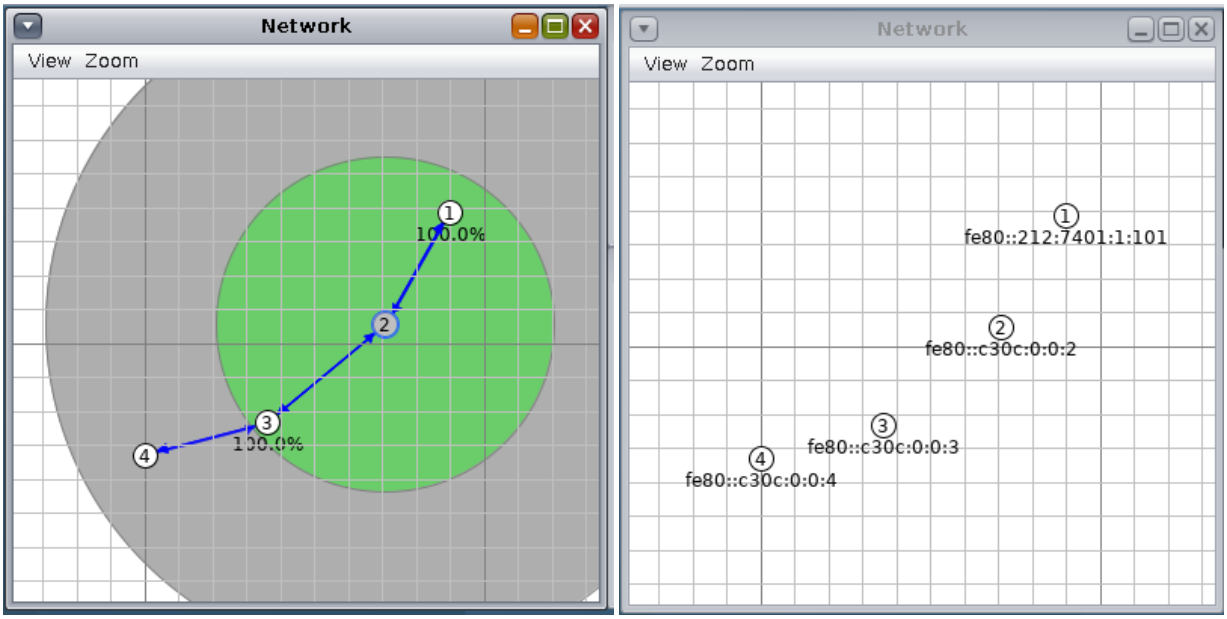

Figura 47. Comunicación y rutado de la red.

Tras iniciar la simulación, hay que utilizar el comando "`make TARGET=sky connect-router-cooja`" y conectar con el border-router. Al hacer esto, se mostrará la información de inicio, de la cual hay que destacar la dirección IPv6 del servidor (en el caso de la Figura 48 es fd00::212:7401:1:101). Con esta dirección se puede comprobar si efectivamente el servidor es accesible, para lo cual es posible utilizar el comando "`ping6`" con la dirección a la que se desee pingear. En la Figura 49 se ve el tiempo que tarda cada paquete en enviarse y el tamaño de los paquetes enviados y recibidos.

```
ris@ris-pc:~/contiki-ng/TFM/rpl-border-router$ make TARGET=sky connect-router-cooja
/bin/sh: 1: test: -lt: unexpected operator
sudo ../../tools/serial-io/tunslip6 -a 127.0.0.1 fd00::1/64
slip connected to ``127.0.0.1:60001''
opened tun device ``/dev/tun0''
ifconfig tun0 inet `hostname` mtu 1500 up
ifconfig tun0 add fd00::1/64
ifconfig tun0 add fe80::0:0:0:1/64
ifconfig tun0

[INFO: BR        ] Waiting for prefix
*** Address:fd00::1 => fd00:0000:0000:0000
[INFO: BR        ] Waiting for prefix
[INFO: BR        ] Server IPv6 addresses:
[INFO: BR        ]   fd00::212:7401:1:101
[INFO: BR        ]   fe80::212:7401:1:101
```

Figura 48. Conexión con el border-router en simulación.

```
ris@ris-pc:~/contiki-ng/examples/rpl-border-router$ ping fd00::212:7401:1:101
PING fd00::212:7401:1:101(fd00::212:7401:1:101) 56 data bytes
64 bytes from fd00::212:7401:1:101: icmp_seq=11 ttl=64 time=17.0 ms
64 bytes from fd00::212:7401:1:101: icmp_seq=12 ttl=64 time=23.9 ms
64 bytes from fd00::212:7401:1:101: icmp_seq=13 ttl=64 time=13.9 ms
64 bytes from fd00::212:7401:1:101: icmp_seq=14 ttl=64 time=12.5 ms
64 bytes from fd00::212:7401:1:101: icmp_seq=15 ttl=64 time=38.4 ms
64 bytes from fd00::212:7401:1:101: icmp_seq=16 ttl=64 time=15.7 ms
```

Figura 49. Prueba de dirección IPv6.

Por último, si buscamos la dirección IPv6 del servidor en el navegador, se accede a la información de la red. Comparando la Figura 50 con la Figura 47 se puede comprobar que la información, tanto de vecinos al alcance del boreder-router como las relaciones jerárquicas entre las direcciones de las motas, se corresponde con las ubicaciones de las motas, su distancia al border-router y las motas intermedias.

Figura 50. Información de rutado para el usuario.

3.3.3 Implementación del border-router

Asegurado el acceso al border-router en simulación, se pasa a compilar y cargar el programa en el border-router con la red desplegada de la primera fase. En este caso, las palabras clave de la plataforma son "TARGET=zoul BOARD=orion". Tras la carga del programa, hay que conectarse al router. El comando para conectarse al border-router real es muy similar al empleado para el border-router de la simulación, modificando en este caso también TARGET y BOARD: "`make TARGET=zoul BOARD=orion connect-router`".

```
ris@ris-pc:~/contiki-ng/TFM/border-router-com-codes$ make TARGET
=zoul BOARD=orion connect-router
sudo ../../tools/serial-io/tunslip6 -s /dev/ttyUSB0 fd00::1/64
[INFO: BR        ] Waiting for prefix
*** Address:fd00::1 => fd00:0000:0000:0000
[INFO: BR        ] Waiting for prefix
[INFO: BR        ] Server IPv6 addresses:
[INFO: BR        ]   fd00::212:7401:1:101
[INFO: BR        ]   fe80::212:7401:1:101
```

Figura 51. Conexión con el border-router.

En la consola aparecerá de nuevo la información del border-router y, si se desea, se puede comprobar la conexión utilizando el comando "`ping6`" como se hace en el apartado anterior. Para corroborar finalmente que el border-router consigue correctamente enrutar y comunicar a toda la red, he accede a su dirección IPv6 desde un navegador y se visualizar el enrutado y las motas conectadas. Se podrá observa un enrutado y una jerarquía similar al de la Figura 50, dependiendo de la distribucion física de las motas.

Así pues, se confirma la correcta integracion del border-router como maestro en las comunicación y gestión de la red y se procede a implementar la última fase del proyecto: la integración del servidor LAMP.

3.4 Comunicación con el servidor

La última fase de este proyecto se divide en dos partes: dotar al border-router de la capacidad para comunicarse con el border-router y diseñar e implementar una aplicación que reciba, almacene, trate y muestre al usuario la información recogida por la red. La primera parte pretende ahondar en los mecanimos de

comunicación que Contiki-NG proporciona para programar las comunicaciones del border-router, de entre los cuales se ha optado por las consultas HTTP.

3.4.1 Consulta HTTP desde el border-router

Los recursos que proporciona Contiki-NG para comunicaciones HTTP se recogen en la carpeta "/os/net/app-layer/http-socket" del repositorio "contiki-ng". En esta librería se definen funciones de HTTP, socket´s y servicios web. Para este proyecto solo son necesarias las funciones definicas en los archivos "http-socket", de cabecera (Figura 52) y fuente, y en concreto las que se explican a continuación:

- **http_socket_init**(): función para inicializar un la estructura de datos utilizada para gestionar las siguientes conexiones del programa.
- **http_socket_get**(): se envía una petición http utilizando el método get de transferencia de parámetros a la dirección url indicada. Para este proyecto, la dirección url debe comenzar por "http://[fd00::1]", indicando el tipo de petición y la direccion del servidor, seguido del directorio y el archivo al que queremos acceder.

El objetivo de utilizar estas funciones, es la de reenviar los datos recibidos de los sensores al servidor. Para ello, cada vez que se recibe un mensaje UDP, se deben extraer los datos en la trama, construir una url apropiada y enviarla al archivo PHP del servidor que guardará los datos en el servidor. La extracción y construcción de la url es un tratamiento básico de cadenas de caracteres, sin embargo es importante diferenciar las dos fomas de paso de parámetros en HTTP.

```
void http_socket_init(struct http_socket *s);

int http_socket_get(struct http_socket *s, const char *url,
                    int64_t pos, uint64_t length,
                    http_socket_callback_t callback,
                    void *callbackptr);

int http_socket_post(struct http_socket *s, const char *url,
                     const void *postdata,
                     uint16_t postdatalen,
                     const char *content_type,
                     http_socket_callback_t callback,
                     void *callbackptr);

int http_socket_close(struct http_socket *socket);

void http_socket_set_proxy(struct http_socket *s,
                           const uip_ipaddr_t *addr, uint16_t port);
```

Figura 52. Definición de funciones para http-socket.

3.4.2 Envío de parámetros HTTP

En los servicios web, internet y aplicaciones de servidores LAMP, existen dos formas básicas de envio de parámetros:

- **GET**: la información enviada es visible para el usuario. Se realiza incluyendo al final de la url el nombre de cada dato y su valor, separando cada dato entre si mediante el caracter '&'. Es una manera muy sencilla reenviar parámetros entre peticiones, ya que es muy sencillo para cualquier programa secuencial concatenar los parámetros con la url. Es un método muy utilizado en aplicaciones que no requieran de privacidad o no sea crítico si el usuario los modifica (busquedas en Google, visualización de videos en Youtube, indicación de ubicaciones con Google Maps, etc).
- **POST**: los datos no son accesibles para el usuario durante la transmisión. Muy adecuado para información de formularios o contraseñas, que deba ser privada y transmitida de forma segura. Es un método más complejo que el anterior debido a que conlleva un procesamiento mayor en las peticiones.

Ya que la información de este proyecto son medidas de sensores e índices propios de cada mota, y a que la conexión se realiza de manera local y privada, el método get es más que suficiente para este proyecto y es más sencillo de procesar. Por esto se utiliza la función "http_socket_get()" en lugar de la función "http_socket_post()".

3.4.3 Conexión entre el border-router y el servidor

Expuestas las herramientas de HTTP, ya es posible entender completamente el código "border-router.c". Pero antes de programar el router para que pueda realizar estas consultas, se necesita conocer la ubicación que el servidor tendrá para e lborder- router cuando este se conecte. Esta ubicación es necesaria ya que todas las referencias a los archivos del servidor que guardaran los datos de los sensores parte de la dirección IP habilitada en la conexión. Para averiguar cual es esta dirección, se procede de la siguiente forma:

1. En primer lugar, hay que conectar el dispositivo Orion Ethernet Router (no es necesario que esté encendido) al ordenador en el que se este ejecutando la máquina virtual a través de un puerto USB 2.0 (es fundamental que el puerto sea 2.0 para que la máquina virtual pueda reconocer el dispositivo). Un vez conectado, hay que dirigirse a la parte inferior derecha de la interfaz y pulsar en el icono del conector USB. Se abrirá un menú desplegable como el de la Figura 53. En este menú se debe seleccionar el dispositivo "Silicon Labs Zolertia Orion Ethernet router" has que aparezca un check azul a la izquierda de este, indicando que se ha seleccionado correctamente.

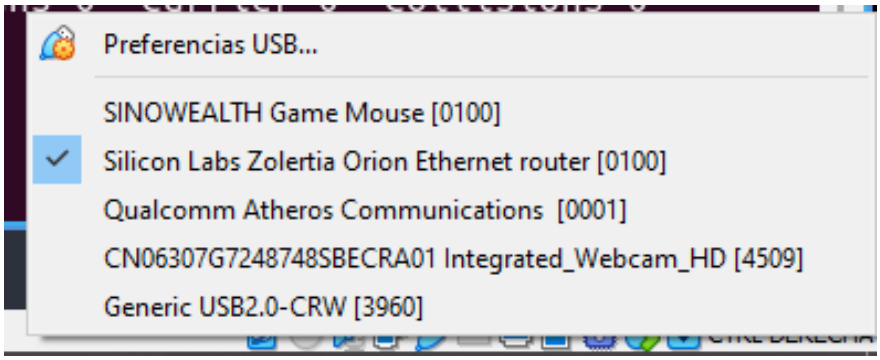

Figura 53. Menú de selección de dispositivos.

2. A continuación desde un terminal, se accede al directorio en el que se encuentre el programa que más adelante se cargará en el router dentro de "contiki-ng" (en el caso de este proyecto "contiki-ng/TFM/rpl-border-router-server). Desde este directorio se ejecuta el comando "`$make TARGET=zoul BOARD=orion connect-router`" para abrir una conexión entre el router y el ordenador. En la Figura 54 se ve que la interfaz de conexión de este ejemplo es la "tun0".

```
ris@ris-pc:~/contiki-ng/TFM/rpl-border-router-server$ make TARGET=zoul BOARD=orion connect-router
sudo ../../tools/serial-io/tunslip6 -s /dev/ttyUSB0 fd00::1/64
********SLIP started on ``/dev/ttyUSB0''
opened tun device ``/dev/tun0''
ifconfig tun0 inet `hostname` mtu 1500 up
ifconfig tun0 add fd00::1/64
ifconfig tun0 add fe80::0:0:0:1/64
ifconfig tun0

tun0: flags=4305<UP,POINTOPOINT,RUNNING,NOARP,MULTICAST>  mtu 1500
        inet 127.0.1.1  netmask 255.255.255.255  destination 127.0.1.1
        inet6 fe80::cbae:abd2:4520:41a9  prefixlen 64  scopeid 0x20<link>
        inet6 fd00::1  prefixlen 64  scopeid 0x0<global>
        inet6 fe80::1  prefixlen 64  scopeid 0x20<link>
        unspec 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00  txqueuelen 500  (UNSPEC)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Figura 54. Conexión router-ordenador.

3. Tras identificar cual es la interfaz para el router, se necesita buscar la dirección ip que se ha abierto para el "host" de esta conexión. Abriendo una nueva consola (para mantener activa la conexión router-ordenador), se ejecuta el comando "$ `ifconfig`", para motrar todas las interfaces que existen en el

ordenador y es importante fijarse en el orden en el que estas aparecen. Para terminar, se ejecuta "`hostname -I`" para conocer todos las IP´s de las interfaces. Estas direcciones se mostrarán de manera ordenada de acuerdo al orden de las interfaces. Para este proyecto (Figura 55) la interfaz "tun0" ocupa la tercera posición, y para esta interfaz la dirección del host el "[fd00::1]".

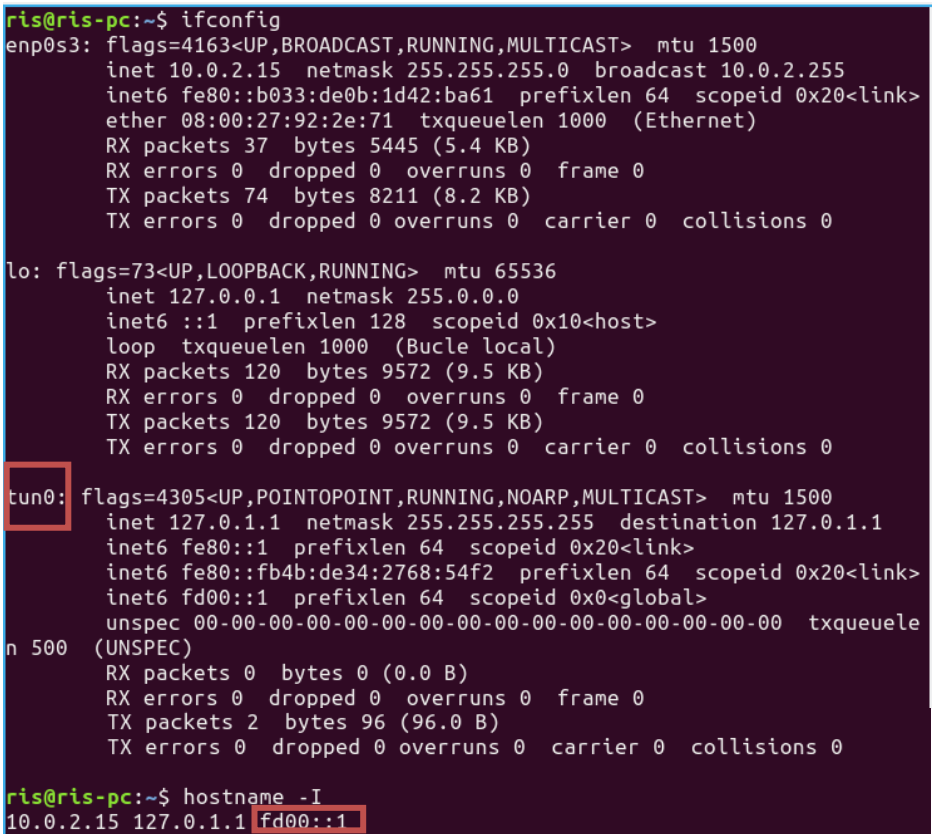

Figura 55. Interfaces habilitadas.

Conociendo la dirección IP del host, ya se conoce cuál es la raíz de la URL a la que se tendrá que consultar desde el router, y por tanto ya es posible enviar mensajes desde el router al servidor. El siguiente paso es instalar la pila de software LAMP que permiter implementar el servidor en el ordenador final.

3.4.4 Instalación del servidor LAMP

Como ya se ha expuesto en el apartado 2.3 de este documento, la preparación de toda la pila LAMP pasa por la instalación (10) de sus tres componentes principales: Apache2, Mysql-server y PHP.

3.4.4.1 Apache2

Para instalar el servidor Apache, se accederá al terminal de Linux y se utilizarán los comandos "`$sudo apt update`" y "`$sudo apt install apache2`". Tras la instalación, se puede utilizar el comando "`$ apache2 -version` " para comprobar que se ha instalado correctamente (Figura 56).

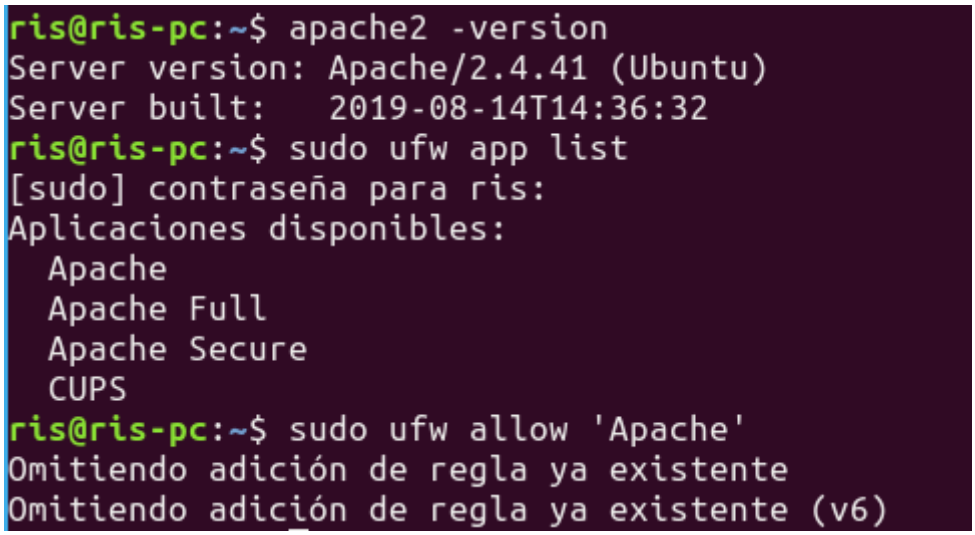

Figura 56. Instalación del paquete Apache2.

Una vez confirmada la instalación, hay que habilitar el tráfico HTTP y HTTPS a través del cortafuegos UFW. Para hacerlo, hay que verifica que el UFW tiene un perfil de aplicación para Apache mediante el comando " `sudo ufw app list` ". Se mostrará, entre otros, la aplicación Apache, y se hará uso de "`$sudo afw allow 'Apache'`" para habilitarlo con el cortafuegos como se indica en la Figura 56.

Para finalizar, desde un navegador se puede acceder a la url "http://localhost" para obtener la información del servidor Apache y comprobar que el servidor está funcionando. Además, desde el terminal se pueden introducir varios comandos para manejar el estado del servidor:

- `$sudo systemctl status apache2:` permite ver el estado del servidor y su información (Figura 56).
- `$sudo systemctl start apache2:` permite iniciar del servidor.
- `$sudo systemctl stop apache2:` permite parar servidor.
- `$sudo systemctl restart apache2:` permite reiniciar el servidor.

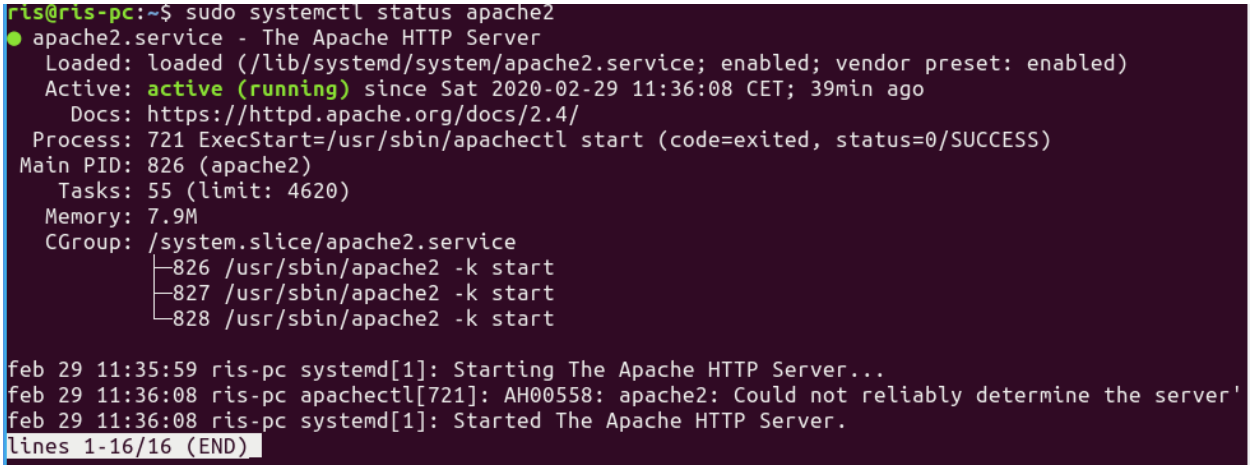

Figura 57. Estado del servidor Apache.

3.4.4.2 Mysql-server

Haciendo uso del comando "`$sudo apt install mysql-server`". Opcionalmente, se puede ejecutar el archivo del comandos "`$sudo mysql_secure_installation`" para configurar manualmente el nivel de

seguridad para el acceso a las bases de datos y las contraseñas que se deben utilizar.

Para comprobar que la instalación es correcta, ejecutaremos y accederemos a MySQL con el comando "`mysql -u root`". Si se pide una contrañesa y durante la instalación no se definió ninguna para el usuario root (usuario por defecto), visitar la referencia (11) para solucionarlo.

Una vez en MySQL, es aconsejable confirmar que se tiene permiso tanto para crear bases de datos y tablas, insertar datos y borrar datos y estructuras. Los comandos más habitulaes (12) empleados para ello, cuyos ejemplos se muestran en la Figura 58, son:

- **Crear base de datos**: Es la primera separación dentro de todos los datos que se almacenen en el equipo para todas las aplicaciones que se esten ejecutando y requieran bases de datos.
- **Usar base de datos**: Especifica cual de todas las bases de datos almacenada vamos a gestionar con los comandos.
- **Crear tabla**: Utilizada para organizar en diferentes apartados toda la información de nuestra aplicación de manera ordenada. Por cada tabla se podrá especificar el número de tablas que la componen, y por cada tabla el tipo de dato, su tamaño si es necesario y otros parametros especiales de configuración.
- **Insertar dato**: Permite guardar datos en la tabla especificada, indicando el valor que toma cada columna para esa entrada.
- **Seleccionar dato**: Indica de que columna de la tabla queremos recuperar la información que tiene. Opcionalmente se puede indicar una condición que deben cumplir los datos para ser seleccionados mediante la sentencia "WHERE <condición>"

```
CREATE DATABASE databasename;

CREATE TABLE table_name (
    column1 datatype,
    column2 datatype,
    column3 datatype,
   ....
);

INSERT INTO table_name (column1, column2, column3, ...)
VALUES (value1, value2, value3, ...);

SELECT column1, column2, ...
FROM table_name;

DELETE FROM table_name WHERE condition;
```

Figura 58. Ejemplos de consultas SQL.

Estos comandos básicos y otros muchos más específicos de SQL permiten tener un manejo total y flexible sobre la información de nuestra aplicación que estará perfectamente alamacenada y a disposición del usuario.

3.4.4.3 PHP

Para terminar de instalar la pila LAMP, se ejecuta el comando "`$sudo apt install php libapache2-php php-mysql`". En este punto, al realizar una consulta en una capeta del servidor, Apache mostrará en primer lugar el archivo llamado "index.html" si no se le especifica un archivo concreto en la consulta. Tras haber instalado PHP es deseable que el servidor de prioridad a los archivos PHP que permiten generar páginas dinámicas. Con ese objetivo, se puede modificar el archivo "/etc/apache2/mods-enabled/dir.conf" reordenando la proridad de los tipos de archivos que buscar Apache para priorizar los archivos ".php" como se ve en la Figura 59.

/etc/apache2/mods-enabled/dir.conf

```
<IfModule mod_dir.c>
    DirectoryIndex index.html index.cgi index.pl index.php index.xhtml index.htm
</IfModule>
```

/etc/apache2/mods-enabled/dir.conf

```
<IfModule mod_dir.c>
    DirectoryIndex index.php index.html index.cgi index.pl index.xhtml index.htm
</IfModule>
```

Figura 59. Modificación de archivos de configuración de Apache2.

Si se desea comprobar que versión de PHP se ha instalado, es posible crear un archivo en el servidor "/var/www/htm/info.php" con las sentencias "<?php phpinfo() ?>", y abrirlo desde el navegador web para mostrar toda la información de la versión.

3.4.5 Diseño de la interfaz web

La parte final del desarrollo de este proyecto consiste en construir una interfaz para el usuario, con la que sea posible tener acceso a toda la información recopilada por los sensores y a la ubicación de cada uno de los nodos de la red de manera cómoda y segura. Por otro lado, la información de la red debe esta almacenada de manera ordenada, segura y accesible para la aplicación que deba motrarla.

3.4.5.1 Organización de la base de datos

La forma en la que se organizarán las medidas que tomen los sensores, los enlaces y las posiciones de los dispositivos, los usuarios que tengan acceso a la aplicación web y los parámetros de gestión de la aplicación web será en tablas en la base de datos.

Con estos requisitos, la base de datos (nombrada como "tfm_ddbb") queda organizada de la siguiente forma:

- **Motes_list (lista de motas)**. Guarda la información relevante a los dispositivos de la red relevante para el usuario:
  - id_mote: llave de identificación del dispositivo en la base de datos.
  - id_parent: llave de identificación del dispositivo en la base de datos que está asignado como dispositivo superior en el rutado de la red.
  - x_coor: coordenada X de la ubicación del dispositivo.
  - y_coor: coordenada Y de la ubicación del dispositivo.
- **Data_list (lista de datos)**. Almacena las medidas tomadas por los sensores y su contexto:
  - id_data: llave de identificación del dato en la base de datos.
  - id_mote: llave de identificación del dispositivo en la base de datos que envió el dato.
  - name: nombre de la magnitud que se ha medido ("temp" de temperatura para este proyecto siempre).
  - value: valor de la magnitud.
  - date: fecha en la que se ha guardado el dato en la base de datos.
- **Users_list (lista de usuarios)**. Contiene todos los usuarios que tienen acceso a la aplicación y sus credenciales:
  - id_user: llave de identificación del usuario en la base de datos.

- name: nombre de usuario
- password: contraseña de acceso para el usuario.
- upload_date: fecha en la que se registró al usuario en la aplicación.

- **Variables_list (lista de variables)**. Gestiona una serie de datos enteros para gestionar la apariencia de la interfaz:
  - id_variable: llave de identificación de la variables en la base de datos.
  - name: nombre de la variable.
  - value: valor que tiene la variable.

3.4.5.2 Integración de PHP y HTML en archivos

Con la base de datos creada, falta diseñar un método para que esa información sea mostrada al usuario. Dado que la interfaz consiste en un aplicación web, es posible trabajar conjuntamente en un mismo archivo con el lenguaje basado en etiquetas, HTML, intercalado con las sentencias PHP. Con PHP es posible crear códigos HTML dinámicos, es decir, páginas web cuyo diseño se modifique con la información de la base de datos. Esto es posible hacerlo con las sentencias "echo" (Figura 60), las cuales permiten crear codigo HTML desde los archivos de .php. Además, PHP integra funciones para conectar y gestionar bases de datos con consultas SQL.

```
    <div class="form_line_cls"></div><br>
    <input class="button_cls" type="submit" value="Log in"><br>
    <?php
    if(isset($_SESSION["Err_msg"]))
        echo $_SESSION["Err_msg"];
    ?>
</form>
```

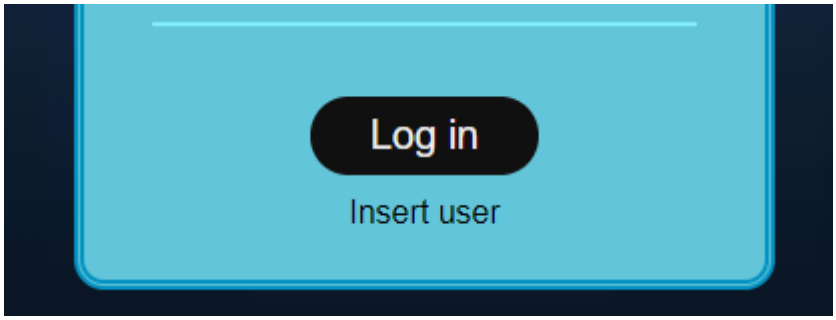

Figura 60. Coexistencia de HTML y PHP.

Utilizando estas herramientas, es posible mostrar todas las medidas que tomen los sensores en gráficas, controlar el acceso solo para usuarios autorizados, o crear el mapa de red de dispositivos y que el usuario pueda visualizarlo.

Por otro lado, la conexión del router al servidor es un caso particular. En su caso, el router no necesita de una visulización de usuario, solo guardar los datos que reciba del resto de dispositivos. No es necesario integrar etiquetas HTML en los archivos creados para guardar la información de la red de sensores, solo sentencias PHP que gestionen el acceso a las listas de datos y dispositivos. Para transferir estos datos los archivos se utiliza el metodo GET, ya explicado en el apartado 3.4.2.

3.4.5.3 Estructura de la aplicación web

Con todo lo expuesto en los apartados anteriores, se ha diseñado una aplicación web que satisface las necesitades de visualización y seguridad del proyecto. La estructura de la aplicación se expone en la Figura 61 y a continuación se desarrollan las partes de la interfaz:

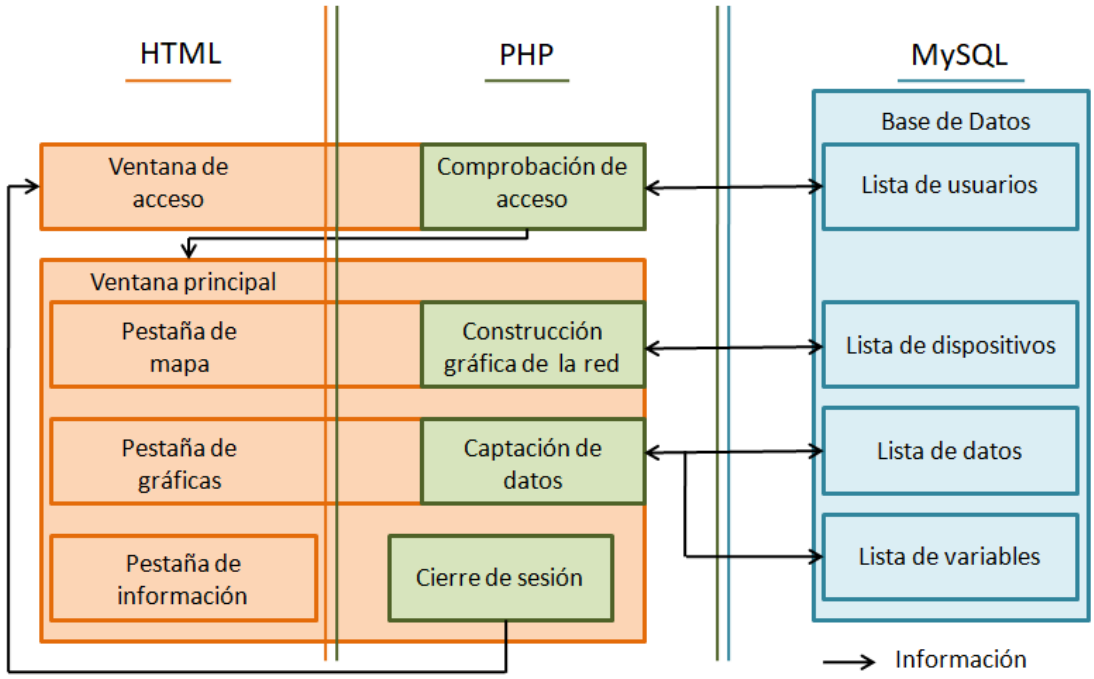

Figura 61. Estructura de la aplicación web.

- **Ventana de acceso:** Se muestra un formulario para ingresar el nombre y la contraseña con la que se quiere acceder y un botón para enviar la solicitud de acceso. En caso de error, se muestra debajo de el botón de envío el mensaje de error específico para cada situación.

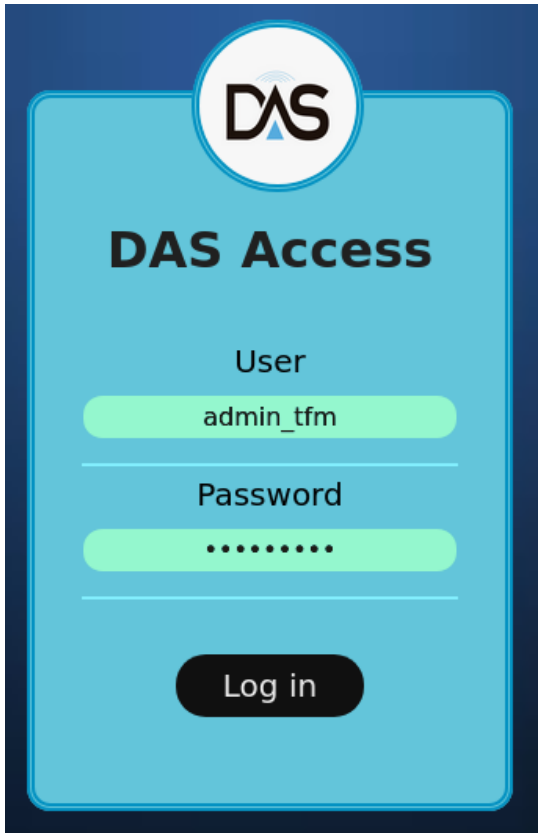

Figura 62. Ventana de acceso.

- **Ventana principal:** Es el cuerpo de la aplicación. En la parte izquierda se muestra el usuario con el que se ha accedido y el menú principal para poder acceder a las diferentes pestañas. En último lugar está el botón para salir de la aplicación y borrar las credenciales de acceso de la sesión de manera segura.

Figura 63. Ventana principal

- **Pestaña de mapa:** Se ubican en un mapa proporcionado por el usuario, la posición de todos los dispositivos que pertenecen a la red y los enlaces de comunicación de toda la red, para que el usuario tenga conocimiento de la arquitectura de la red.

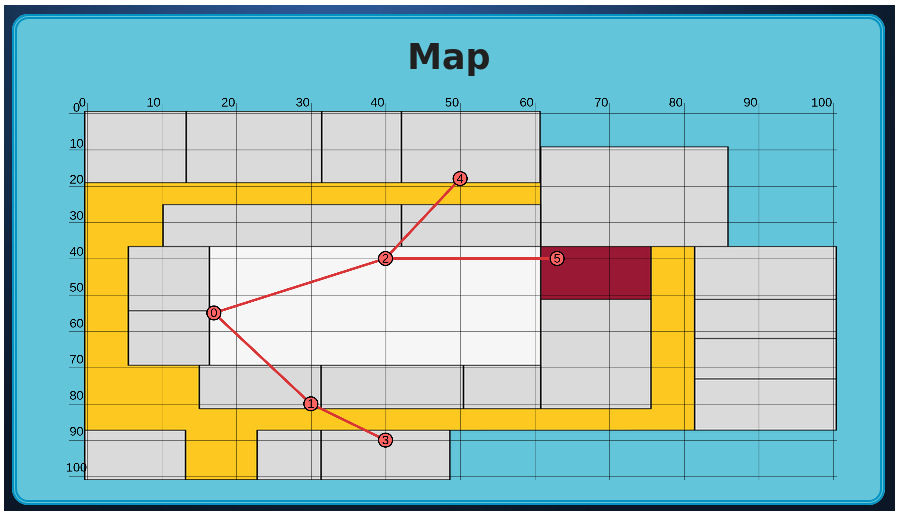

Figura 64. Pestaña de mapa.

- **Pestaña de gráficas:** Un formulario permite seleccionar individualmente los nodos de la red, el sensor y el número de medidas de temperatura que se desee mostrar en la gráfica de ese nodo. Tras actualizar la gráfica, es posible navegar por las medidas cronológicamente de acuerdo al número de datos por gráfica especificado.

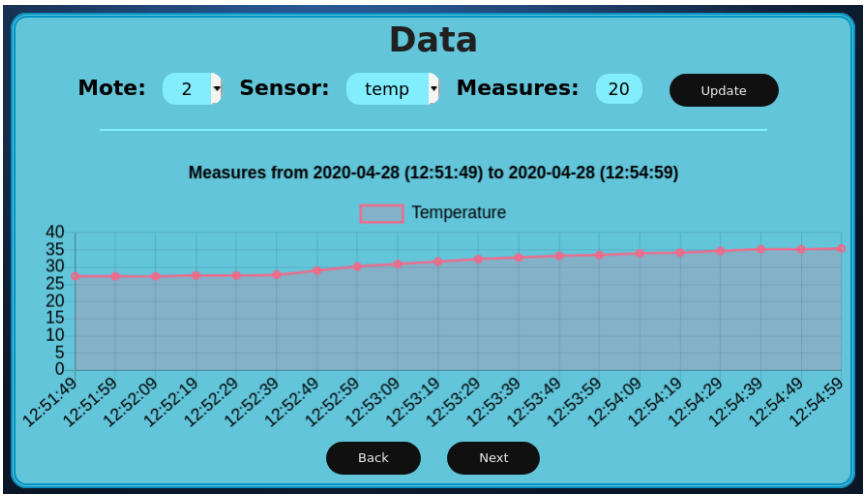

Figura 65. Pestaña de gráficas.

- **Pestaña de información:** se explica el contexto y motivo por el que se ha desarrollado la aplicación y el proyecto.

Figura 66. Pestaña de información.

3.4.6 Implementación del servidor

Una vez comprendida la estructura del servidor y de la base de datos, se pasa a implementar los archivos que sostienen su funcionamiento. Junto a este documento se adjunta la carpeta "Web_TFM", donde se recogen todos los códigos HTML, PHP, JavaScript y CSS así como librerías adicionales que se han desarrollado para el funcionamiento de la aplicación web. Esta carpeta se debe ubicar en la carpeta "/var/www/html" (Figura 67), pues a esta se dirigen todas las conexiones TCP/IP que se realicen.

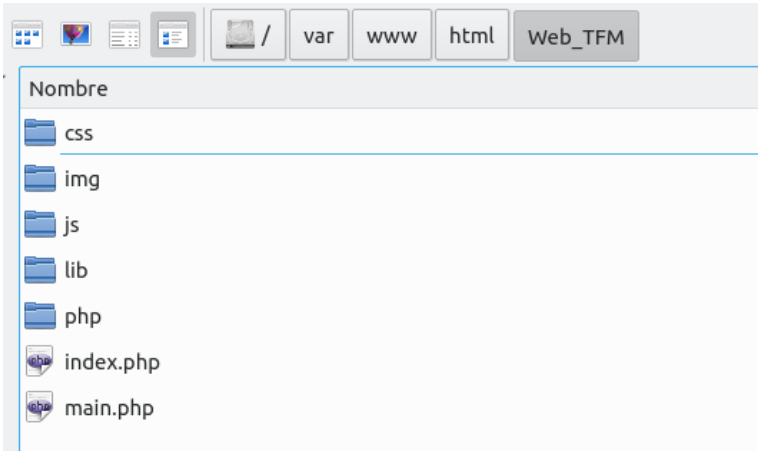

Figura 67. Ubicación de la aplicación web.

La segunda tarea para implementar el servidor es crear la base de datos y todas las tablas que la forman. Haciendo uso del terminar Linux y los comandos SQL indicados en el apartado 3.4.4.2 se debe crear una base de datos con la estructura que se especifica en el apartado 3.4.5.1. Es fundamental que todos los nombres de la base de datos, tablas y columnas se ajusten a los nombres indicados en inglés para que el resto de archivos de la aplicación puedan acceder a ellos.

```
ris@ris-pc:~$ mysql -u root
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 67
Server version: 10.3.22-MariaDB-0ubuntu0.19.10.1 Ubuntu 19.10

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> USE tfm_ddbb
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
MariaDB [tfm_ddbb]> SELECT * FROM motes_list;
+---------+-----------+--------+--------+
| id_mote | id_parent | x_coor | y_coor |
+---------+-----------+--------+--------+
|       0 |         0 |     17 |     55 |
|       1 |         0 |     30 |     80 |
|       2 |         0 |     40 |     40 |
|       3 |         1 |     40 |     90 |
|       4 |         2 |     50 |     18 |
|       5 |         2 |     63 |     40 |
+---------+-----------+--------+--------+
6 rows in set (0.000 sec)
```

Figura 68. Manipulación de la base de datos desde el terminal Linux.

La Figura 68 expone como acceder a MySQL desde un terminal con el usuario -root e introducir ejemplos de comandos SQL. Tras crear toda la base de datos completa, ya es posible comprobar el funcionamiento al completo del sistema conectando el border-router como en la segunda fase del proyecto.

Una vez conectado el border-router, inmediatamente se verá un mensaje correspondiente a la primera conexión al servidor LAMP (Figura 69). Esta conexión tiene dos objetivos. El primero, realizar una comprobación del estado de la conexión (en caso de ser erróneo se verá un mensaje de "Socket Aborted" o similar, con lo cual será necesario rehacer la conexión). El segundo guardar la información de la ubicación de router en la base de datos. El router se verá como el nodo 0 en la interfaz, y desde el se irá ramificando toda la red.

```
ris@ris-pc:~/contiki-ng/TFM/border-router-com-codes$ make TARGET
=zoul BOARD=orion connect-router
sudo ../../tools/serial-io/tunslip6 -s /dev/ttyUSB0 fd00::1/64
[INFO: BR        ] Waiting for prefix
*** Address:fd00::1 => fd00:0000:0000:0000
[INFO: BR        ] Waiting for prefix
[INFO: BR        ] Server IPv6 addresses:
[INFO: BR        ]   fd00::212:4b00:60d:680a
[INFO: BR        ]   fe80::212:4b00:60d:680a
Connected
HTTP socket received 10 bytes of data expects:10
-----------------------
Mote saved
-----------------------
HTTP socket closed, 10 bytes received
Closed
[INFO: RPL BR    ] Received message 'php/ddbb_sv_dt.php?temp=252
38&x_c=31&y_c=79' from fd00::212:4b00:9df:8eb1
url http:///[fd00::1]/Web_TFM/php/ddbb_sv_mt.php?x_c=31&y_c=79&id
_mote=1&id_parent=0 host fd00::1 port 80 path /Web_TFM/php/ddbb_
sv_mt.php?x_c=31&y_c=79&id_mote=1&id_parent=0
Connected
HTTP socket received 10 bytes of data expects:10
-----------------------
Mote saved
-----------------------
HTTP socket closed, 20 bytes received
```

Figura 69. Recepción de mensajes de las motas en el router.

A medida que se vayan recibiendo mensajes de las motas, estas se irán incluyendo en la interfaz, asignándoles un número y su información de enrutado (Figura 70). No es necesario apagar ni reiniciar las motas ya desplegadas con la prueba anterior, pues se irán enrutando y uniendo a la red a medida que vaya enviando las

medidas periodicamente (Figura 69).

Para acceder a la aplicación web desde un navegador web, se utiliza la URL "`http://localhost/Web_TFM/`". Desde ahí accederemos a la ventana principal de la aplicación tras proporcionar los credenciales de usuario.

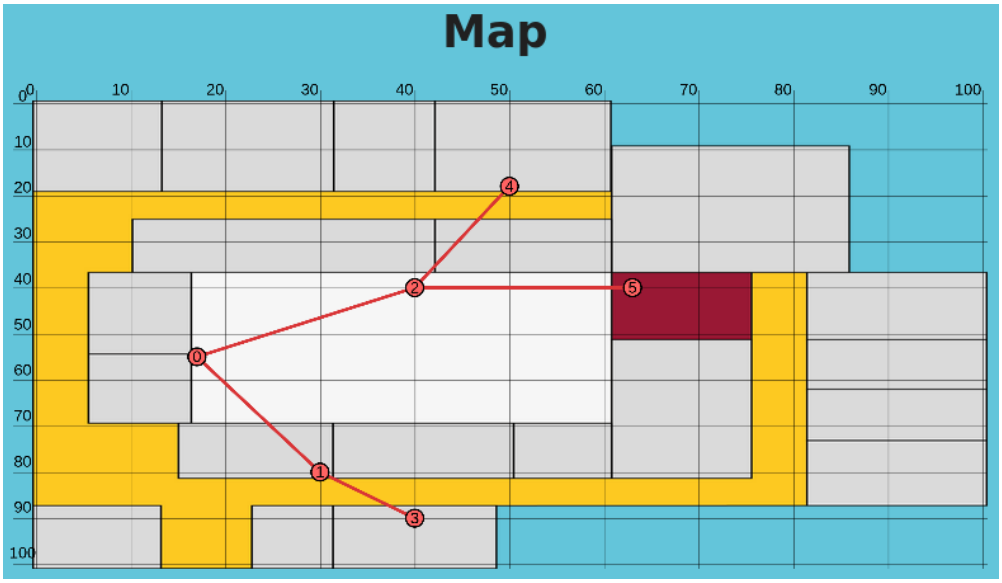

Figura 70. Visualización de la red desde la aplicación web.

Desde la pestaña de mapa de la aplicación se pude comprobar el número de motas y las coordenadas en las que están ubicadas, de manera que se podrá comprobar de manera muy sencilla si alguna de las motas desplegadas no se ha conectado a la red.

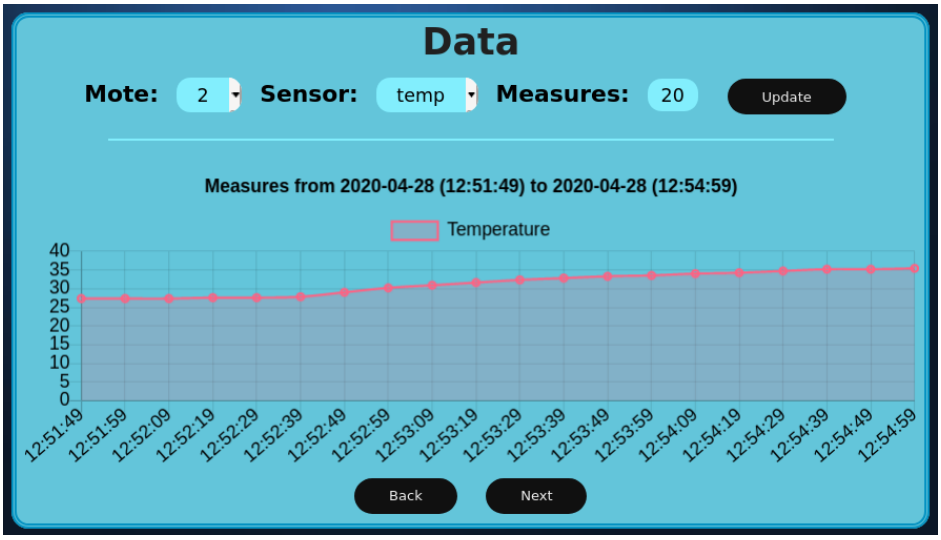

Figura 71. Visualización de datos.

Para finalizar, en la pestaña de datos se pueden comprobar las información de los sensores que se ha recopilado hasta el momento. Se puede seleccionar la mota que queremos mostrar, el sensor (unicamente de temperatura para este proyecto) y el número de datos que queremos mostrar por gráfica. Actualizando la gráfica, aparecerán los datos empezando por los más antiguos que existen en la base de datos. Se muestran las fechas de la primera y la última medida mostrada y la hora de cada una de las que se visulizan. También es posible navegar por los datos haciendo uso de los botones "Back" y "Next", haciendo que se muestren los valores consecutivos según el número de datos por gráfica especificados.

4 CONCLUSIONES

Este documento expone las herramientas, plataformas, métodos y fases necesarios en el diseño de una red inalámbrica de sensores basada en Contiki-NG y los dispositivos Zolertia. Se han detallado los pasos que se han seguido en el estudio desde cero para comprobar la viabilidad y consistencia de estás redes, y el máximo rendimiento que es posible alcanzar con estos sistemas.

No hay duda de que las WAN serán un punto de extremo interés en el futuro próximo de las tecnologías de la información y las telecomunicaciones, concretamente en el campo cada vez más extendido del Internet of Things. Sin embargo, es importante tener en cuenta las limitaciones que estas redes y las aplicaciones que son factibles de implementarse con estos sistemas. El progreso debe ser seguro y considerando siempre cuales con los aspectos más importantes que se deben cuidar de cada proyecto.

Personalmente, este proyecto me ha permitido completar mis conocimientos sobre telecomunicaciones, redes de comunicaciones, compiladores y librerías de programas, simulación de sistemas y gestión de conexiones enrutadas.

4.1 Trabajos futuros

A partir de este proyecto, se abren multiples posibilidades para aumentar las capacidades y funciones de la red y la interfaz. Las ampliaciones más inmediatas del proyecto son:

- Amplia el tamaño de la red, incorporando nuevos dispositivos con sensores diferentes de mayor interés.
- Implementar dispositivos actuadores y emitir ordenes desde el servidor a los efectores finales de la red.
- Extender las funcionalidades de interfaz de usuario y el servidor para realizar un procesamiento más intensivo de los datos de la red para extraer conclusiones de los datos.
- Profundizar en la sincronización y la gestión de energías de toda la red, para extender el tiempo de trabajo de todos los dispositivos. Contiki-NG dispone de diferentes herramientas de sincronización de red y, dependiento de plataformas, diferentes herramientas de gestión de energía. Sin embargo, no ha sido posible profundizar en estos puntos en este proyecto, quedando como una de las siguientes continuaciones del mismo.

5 REFERENCIAS

1. Contiki-NG. [En línea] https://github.com/contiki-ng/contiki-ng/wiki.

2. Enrutado RPL. [En línea] https://github.com/contiki-ng/contiki-ng/wiki/Documentation:-RPL.

3. Comunicaciones UDP. [En línea] https://www.ionos.es/digitalguide/servidores/know-how/udp-user-datagram-protocol/.

4. Zolertia RE-Moter platform. [En línea] https://github.com/Zolertia/Resources/wiki/RE-Mote.

5. Zolertia Orion Ethernet IP64 Router. [En línea] https://github.com/Zolertia/Resources/wiki/Orion. .

6. Servidor Apache. [En línea] https://blog.infranetworking.com/que-es-apache-servidor/ , https://es.wikipedia.org/wiki/Servidor_HTTP_Apache.

7. Bases de datos. [En línea] https://concepto.de/base-de-datos/ , https://es.wikipedia.org/wiki/Base_de_datos.

8. Instalación en Linux del ToolChain. [En línea] https://github.com/contiki-ng/contiki-ng/wiki/Toolchain-installation-on-Linux.

9. Compatibilidades Z1 - ReMote-revb. [En línea] https://github.com/Zolertia/Resources/wiki/Migrate-From-Z1-to-RE-Mote.

10. Instalación de la pila LAMP. [En línea] https://www.digitalocean.com/community/tutorials/como-instalar-en-ubuntu-18-04-la-pila-lamp-linux-apache-mysql-y-php-es.

11. Error de acceso denegado a usuario 'root'. [En línea] https://www.zeppelinux.es/corregir-error-1698-28000-access-denied-for-user-rootlocalhost-en-mariadb-mysql/.

12. Consultas SQL. [En línea] https://www.w3schools.com/sql/default.asp.

6 ANEXO

6.1 Códigos de programas, comunicación entre motas

6.1.1 Udp-server.c

```
/* INCLUDES */

#include "contiki.h"
#include "net/routing/routing.h"
#include "net/netstack.h"
#include "net/ipv6/simple-udp.h"

#include "sys/log.h"

/* DEFINES */

#define LOG_MODULE "App"
#define LOG_LEVEL LOG_LEVEL_INFO

#define WITH_SERVER_REPLY  1
#define UDP_CLIENT_PORT    8765
#define UDP_SERVER_PORT    5678

static struct simple_udp_connection udp_conn;

PROCESS(udp_server_process, "UDP server");
AUTOSTART_PROCESSES(&udp_server_process);
/*---------------------------------------------------------------------*/
static void
udp_rx_callback(struct simple_udp_connection *c,
         const uip_ipaddr_t *sender_addr,
         uint16_t sender_port,
         const uip_ipaddr_t *receiver_addr,
         uint16_t receiver_port,
         const uint8_t *data,
         uint16_t datalen)
{
  LOG_INFO("Received request '%.*s' from ", datalen, (char *) data);
  LOG_INFO_6ADDR(sender_addr);
  LOG_INFO_("\n");
#if WITH_SERVER_REPLY
  LOG_INFO("Sending response.\n");
  simple_udp_sendto(&udp_conn, data, datalen, sender_addr);
#endif
}
/*---------------------------------------------------------------------*/
PROCESS_THREAD(udp_server_process, ev, data)
{
  PROCESS_BEGIN();

  /* Initialize DAG root */
  NETSTACK_ROUTING.root_start();
  LOG_INFO("DAG root started");
```

```
  /* Initialize UDP connection */
  simple_udp_register(&udp_conn, UDP_SERVER_PORT, NULL,
                      UDP_CLIENT_PORT, udp_rx_callback);

  PROCESS_END();
}
/*---------------------------------------------------------------------*/
```

6.1.2 Udp-client-temp.c

```
/* INCLUDES */

#include "contiki.h"
#include "net/routing/routing.h"
#include "random.h"
#include "net/netstack.h"
#include "net/ipv6/simple-udp.h"

#include "dev/zoul-sensors.h"
#include "dev/leds.h"

#include "sys/log.h"

/* DEFINES */

#define LOG_MODULE "App"
#define LOG_LEVEL LOG_LEVEL_INFO

#define WITH_SERVER_REPLY  1
#define UDP_CLIENT_PORT    8765
#define UDP_SERVER_PORT    5678

#define SEND_INTERVAL        (10 * CLOCK_SECOND)

#ifndef MOTE_X_COOR
#define MOTE_X_COOR        100
#endif
#ifndef MOTE_Y_COOR
#define MOTE_Y_COOR        100
#endif

static struct simple_udp_connection udp_conn;

/*---------------------------------------------------------------------*/

PROCESS(udp_client_process, "UDP client");
AUTOSTART_PROCESSES(&udp_client_process);

/*---------------------------------------------------------------------*/

static void
udp_rx_callback(struct simple_udp_connection *c,
         const uip_ipaddr_t *sender_addr,
         uint16_t sender_port,
         const uip_ipaddr_t *receiver_addr,
```

```
         uint16_t receiver_port,
         const uint8_t *data,
         uint16_t datalen)
{

  LOG_INFO("Received response '%.*s' from ", datalen, (char *) data);
  LOG_INFO_6ADDR(sender_addr);
#if LLSEC802154_CONF_ENABLED
  LOG_INFO_(" LLSEC LV:%d", uipbuf_get_attr(UIPBUF_ATTR_LLSEC_LEVEL));
#endif
  LOG_INFO_("\n");
}

/*---------------------------------------------------------------------*/

PROCESS_THREAD(udp_client_process, ev, data)
{
  SENSORS_ACTIVATE(cc2538_temp_sensor);

  static struct etimer periodic_timer;
  static char str[64];
  uip_ipaddr_t dest_ipaddr;
  uint16_t x_coor = random_rand() % MOTE_X_COOR,
           y_coor = random_rand() % MOTE_Y_COOR;

  PROCESS_BEGIN();

  /* Initialize UDP connection */
  simple_udp_register(&udp_conn, UDP_CLIENT_PORT, NULL,
                      UDP_SERVER_PORT, udp_rx_callback);

  while(1) {

     /* Send to DAG root */
     int temp
=cc2538_temp_sensor.value(CC2538_SENSORS_VALUE_TYPE_CONVERTED);

    LOG_INFO("Sending Temp %d ( %d : %d ) to ", temp,x_coor,y_coor);
    LOG_INFO_6ADDR(&dest_ipaddr);
    LOG_INFO_("\n");

    if(NETSTACK_ROUTING.node_is_reachable() &&
NETSTACK_ROUTING.get_root_ipaddr(&dest_ipaddr)) {

      leds_toggle(LEDS_RED);

      snprintf(str, sizeof(str),
"php/ddbb_sv_dt.php?temp=%d&x_c=%d&y_c=%d",temp,x_coor,y_coor);
      simple_udp_sendto(&udp_conn, str, strlen(str), &dest_ipaddr);

      etimer_set(&periodic_timer,(SEND_INTERVAL));
      PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&periodic_timer));

    } else {

      LOG_INFO("Not reachable yet\n");
```

```
      etimer_set(&periodic_timer,(CLOCK_SECOND + random_rand()%(2*
CLOCK_SECOND)));
      PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&periodic_timer));

    }
  }

  PROCESS_END();
}

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

6.2 Códigos de programas, comunicación con router

6.2.1 Border-router.c

```
/* INCLUDES */

#include "contiki.h"
#include "net/routing/routing.h"
#include "random.h"
#include "net/netstack.h"
#include "net/ipv6/simple-udp.h"
/* Log configuration */

#include "sys/log.h"
#include "net/ipv6/uip-ds6-nbr.h"
#include "net/ipv6/uip-ds6-route.h"
#include "net/ipv6/uip-sr.h"
#include "httpd-simple.h"

#include "contiki-net.h"
#include "net/app-layer/http-socket/http-socket.h"
#include "net/ipv6/uip-ds6.h"
#include "ipv6/ip64-addr.h"

#include <stdio.h>
#include <string.h>

/*---------------------------------------------------------------------
-------*/
static const char *TOP = "<html>\n  <head>\n    <title>Contiki-
NG</title>\n  </head>\n<body>\n";
static const char *BOTTOM = "\n</body>\n</html>\n";

const uint8_t x_router=20,y_router=30;

static struct http_socket s;
static int bytes_received = 0;

static char buf[256];
static int blen;
#define ADD(...) do {
\
```

```
    blen += snprintf(&buf[blen], sizeof(buf) - blen, __VA_ARGS__);
\
  } while(0)
#define SEND(s) do { \
  SEND_STRING(s, buf); \
  blen = 0; \
} while(0);

/* Use simple webserver with only one page for minimum footprint.
 * Multiple connections can result in interleaved tcp segments since
 * a single static buffer is used for all segments.
 */


#define LOG_MODULE "RPL BR"
#define LOG_LEVEL LOG_LEVEL_INFO

#define UDP_CLIENT_PORT    5678
#define UDP_SERVER_PORT    8765

#define NUM_OF_NODES 5+1

struct{
     uip_ipaddr_t ip_mt;
     uint8_t id_pt;
} motes_list[NUM_OF_NODES];
/* Declare and auto-start this file's process */
PROCESS(contiki_ng_br, "Contiki-NG Border Router");
AUTOSTART_PROCESSES(&contiki_ng_br);

static struct simple_udp_connection udp_conn;

/*-------------------------------------------------------------------
-------*/
int get_substring_index(const uint8_t *S,uint16_t sizeS, char subS[],
uint16_t sizeSubS){
     int index=-1,j=0;
     for(int i=0;i<sizeS;i++){
          //printf("S(%c)\n",S[i]);
          if(S[i] == subS[0]){
               for(j=0;j<sizeSubS;j++){
                    //printf("- S(%c) = subS(%c)\n",S[j+i],subS[j]);
                    if(S[j+i] != subS[j])
                         break;
               }
               //printf("j=%d\n",j);
               if(j == (sizeSubS)){
                    index=i;
                    //printf("ind=%d\n",index);
               }
          }
     }
     //printf("final ind=%d\n",index);
     return index;
}
```

```
/*-------------------------------------------------------------------
-------*/
int get_index_mote_list(const uip_ipaddr_t *sender_addr){
      int ind=0;
      uint8_t i;
      //uint8_t i,last_mote_ind=0;
      char ip_mote[UIPLIB_IPV6_MAX_STR_LEN],
ip_aux[UIPLIB_IPV6_MAX_STR_LEN];

      uiplib_ipaddr_snprint(ip_mote, sizeof(ip_mote), sender_addr);

      for(i=0;i<NUM_OF_NODES;i++){
            uiplib_ipaddr_snprint(ip_aux, sizeof(ip_aux),
&motes_list[i].ip_mt);
            /*-- Compare IPs --*/
            bool check_ips=1;
            for(int i=0;i<UIPLIB_IPV6_MAX_STR_LEN;i++){
                  if((int)ip_mote[i] != (int)ip_aux[i]){
                        check_ips=0;
                        break;
                  }
                  if(((int)ip_mote[i] == 0) | ((int)ip_aux[i] == 0))
                        break;
            }

            if(check_ips){
                  ind=i;
                  break;
            }
            /*if(!((ip_aux[0] == ':') & (ip_aux[1] == ':'))){
                  last_mote_ind=i;
            }*/
      }

      return ind;
}
/*-------------------------------------------------------------------
-------*/
static void
http_callback(struct http_socket *s, void *ptr,
         http_socket_event_t e,
         const uint8_t *data, uint16_t datalen)
{
  if(e == HTTP_SOCKET_ERR) {
    printf("HTTP socket error\n");
  } else if(e == HTTP_SOCKET_TIMEDOUT) {
    printf("HTTP socket error: timed out\n");
  } else if(e == HTTP_SOCKET_ABORTED) {
    printf("HTTP socket error: aborted\n");
  } else if(e == HTTP_SOCKET_HOSTNAME_NOT_FOUND) {
    printf("HTTP socket error: hostname not found\n");
  } else if(e == HTTP_SOCKET_CLOSED) {
    printf("HTTP socket closed, %d bytes received\n", bytes_received);
  } else if(e == HTTP_SOCKET_DATA) {
    int i;
    bytes_received += datalen;
```

```
    printf("HTTP socket received %d bytes of data expects:%d\n",
datalen,
           (int) s->header.content_length);
    printf("-----------------------\n");
    for(i = 0; i < datalen; i++) {
      printf("%c", data[i]);
    }
    printf("\n-----------------------\n");
  }
}

/*---------------------------------------------------------------------
-------*/
static void
udp_rx_callback(struct simple_udp_connection *c,
         const uip_ipaddr_t *sender_addr,
         uint16_t sender_port,
         const uip_ipaddr_t *receiver_addr,
         uint16_t receiver_port,
         const uint8_t *data,
         uint16_t datalen)
{

  LOG_INFO("Received message '%.*s' from ", datalen, (char *) data);
  LOG_INFO_6ADDR(sender_addr);
#if LLSEC802154_CONF_ENABLED
  LOG_INFO_(" LLSEC LV:%d",
uipbuf_get_attrsender_addr(UIPBUF_ATTR_LLSEC_LEVEL));
#endif
  LOG_INFO_("\n");

  static char str[128],subS[128],str_mote[128];
  char ip_mote[UIPLIB_IPV6_MAX_STR_LEN],
ip_aux[UIPLIB_IPV6_MAX_STR_LEN], auxS[10];

  snprintf(subS, sizeof(subS), "x_c");
  int ind_x_coor=get_substring_index(data, datalen, subS, 3);

  /*-- Check mote on motes_list --*/
  bool check_mote=0;
  uint8_t i,last_mote_ind=0;
  uiplib_ipaddr_snprint(ip_mote, sizeof(ip_mote), sender_addr);

  for(i=0;i<NUM_OF_NODES;i++){
     uiplib_ipaddr_snprint(ip_aux, sizeof(ip_aux),
&motes_list[i].ip_mt);

     /*-- Compare IPs --*/
     bool check_ips=1;
     for(int i=0;i<UIPLIB_IPV6_MAX_STR_LEN;i++){
          if((int)ip_mote[i] != (int)ip_aux[i]){
               check_ips=0;
               break;
          }
          if(((int)ip_mote[i] == 0) | ((int)ip_aux[i] == 0))
               break;
     }
```

```
      if(check_ips){
            check_mote=1;
            break;
      }
      if(!((ip_aux[0] == ':') & (ip_aux[1] == ':'))){
            last_mote_ind=i;
      }
  }
  if(!check_mote){
     motes_list[last_mote_ind+1].ip_mt = *sender_addr;
     uiplib_ipaddr_snprint(ip_aux, sizeof(ip_aux),
&motes_list[last_mote_ind+1].ip_mt);

     /*-- Mote update --*/

     (char *) str_mote);
     snprintf(str_mote, sizeof(str_mote), " ");
     snprintf(subS, sizeof(subS), " ");
     snprintf(str_mote, sizeof(str_mote),
"http://[fd00::1]/Web_TFM/php/ddbb_sv_mt.php?");

     for(int i=ind_x_coor ;i < datalen;i++){
           subS[i-ind_x_coor]=(char) data[i];
     }

     for(int i=datalen ;i < sizeof(subS);i++){
           subS[i-ind_x_coor]=0;
     }

     strcat(str_mote, subS);
     strcat(str_mote, "&id_mote=");
     snprintf(auxS, sizeof(auxS), "%d",(last_mote_ind+1));
     strcat(str_mote,auxS);
     strcat(str_mote, "&id_parent=");

     int index_parent;
     uip_sr_node_t *link;

     uiplib_ipaddr_snprint(ip_mote, sizeof(ip_mote), sender_addr);
     for(link = uip_sr_node_head(); link != NULL; link =
uip_sr_node_next(link)) {
      if(link->parent != NULL) {
        uip_ipaddr_t child_ipaddr,parent_ipaddr;
           char buf_parent[UIPLIB_IPV6_MAX_STR_LEN];

        NETSTACK_ROUTING.get_sr_node_ipaddr(&child_ipaddr, link);
        NETSTACK_ROUTING.get_sr_node_ipaddr(&parent_ipaddr, link-
>parent);

           uiplib_ipaddr_snprint(buf_parent, sizeof(buf_parent),
&child_ipaddr);

           bool check=1;
           for(int i=0;i<sizeof(buf_parent);i++){
                if((int)buf_parent[i] != (int)ip_mote[i]){
                     //printf("diferencia\n");
```

```
                    check=0;
                    break;
                }
                if(((int)buf_parent[i] == 0) | ((int)ip_mote[i] == 0))
                    break;
            }
        if(check){
                index_parent=get_index_mote_list(&parent_ipaddr);
                uiplib_ipaddr_snprint(buf_parent, sizeof(buf_parent),
&parent_ipaddr);

                motes_list[last_mote_ind+1].id_pt=0;
                snprintf(auxS, sizeof(auxS), "%d",index_parent);
                strcat(str_mote,auxS);
            }
        }
     }
     http_socket_get(&s, str_mote, 0, 0,
                  http_callback, NULL);
    snprintf(str_mote, sizeof(str_mote), " ");

  }else{

  /*-- Mote temp save */

  snprintf(str, sizeof(str), "http://[fd00::1]/Web_TFM/");


  for(int i=0 ;i < (ind_x_coor-1);i++)
     subS[i]=(char)data[i];
  strcat(str, subS);

  int index=get_index_mote_list(sender_addr);
  printf("index:%d\n",index);

  snprintf(auxS, sizeof(auxS), "&id_mote=");
  strcat(str, auxS);
  snprintf(auxS, sizeof(auxS), "%d",index);
  strcat(str,auxS);

  http_socket_get(&s, str, 0, 0,
                  http_callback, NULL);
  }

  simple_udp_sendto(c, data, datalen, sender_addr);
}

/*-------------------------------------------------------------------
-------*/
PROCESS_THREAD(contiki_ng_br, ev, data)
{

  PROCESS_BEGIN();

  LOG_INFO("Starting Udp register\n");
  simple_udp_register(&udp_conn, UDP_CLIENT_PORT, NULL,
                      UDP_SERVER_PORT, udp_rx_callback);
```

```
//#if BORDER_ROUTER_CONF_WEBSERVER
  PROCESS_NAME(webserver_nogui_process);
  process_start(&webserver_nogui_process, NULL);
//#endif /* BORDER_ROUTER_CONF_WEBSERVER */
  LOG_INFO("Contiki-NG Border Router started\n");


  PROCESS_END();
}
/*---------------------------------------------------------------------
-------*/
static void
ipaddr_add(const uip_ipaddr_t *addr)
{
  uint16_t a;
  int i, f;
  for(i = 0, f = 0; i < sizeof(uip_ipaddr_t); i += 2) {
    a = (addr->u8[i] << 8) + addr->u8[i + 1];
    if(a == 0 && f >= 0) {
      if(f++ == 0) {
        ADD("::");
      }
    } else {
      if(f > 0) {
        f = -1;
      } else if(i > 0) {
        ADD(":");
      }
      ADD("%x", a);
    }
  }
}
/*---------------------------------------------------------------------
-------*/
static
PT_THREAD(generate_routes(struct httpd_state *s))
{
  static uip_ds6_nbr_t *nbr;

  PSOCK_BEGIN(&s->sout);
  SEND_STRING(&s->sout, TOP);
  ADD("<table align=center><tr><td><div>");
  SEND(&s->sout);
  ADD("<h2 align=center>DODAG Routing</h2>");
  SEND(&s->sout);
  ADD("<div style='border:2px solid #000;padding:10px'><b>Neighbors\n
</b><ul>\n");
  SEND(&s->sout);
  for(nbr = uip_ds6_nbr_head();
      nbr != NULL;
      nbr = uip_ds6_nbr_next(nbr)) {
    ADD("    <li>");
    ipaddr_add(&nbr->ipaddr);
    ADD("</li>\n");
    SEND(&s->sout);
```

```
  }
  ADD("  </ul>\n");
  SEND(&s->sout);

#if (UIP_MAX_ROUTES != 0)
  {
    static uip_ds6_route_t *r;
    ADD("  Routes\n  <ul>\n");
    SEND(&s->sout);
    for(r = uip_ds6_route_head(); r != NULL; r =
uip_ds6_route_next(r)) {
      ADD("    <li>");
      ipaddr_add(&r->ipaddr);
      ADD("/%u (via ", r->length);
      ipaddr_add(uip_ds6_route_nexthop(r));
      ADD(") %lus", (unsigned long)r->state.lifetime);
      ADD("</li>\n");
      SEND(&s->sout);
    }
    ADD("  </ul>\n");
    SEND(&s->sout);
  }
#endif /* UIP_MAX_ROUTES != 0 */

#if (UIP_SR_LINK_NUM != 0)
  if(uip_sr_num_nodes() > 0) {
    static uip_sr_node_t *link;
    ADD(" <b> Routing links\n  <b><ul>\n");
    SEND(&s->sout);
    for(link = uip_sr_node_head(); link != NULL; link =
uip_sr_node_next(link)) {
      if(link->parent != NULL) {
        uip_ipaddr_t child_ipaddr;
        uip_ipaddr_t parent_ipaddr;

        NETSTACK_ROUTING.get_sr_node_ipaddr(&child_ipaddr, link);
        NETSTACK_ROUTING.get_sr_node_ipaddr(&parent_ipaddr, link-
>parent);

        ADD("    <li>");
        ipaddr_add(&child_ipaddr);

        ADD(" (parent: ");
        ipaddr_add(&parent_ipaddr);
        ADD(") %us", (unsigned int)link->lifetime);

        ADD("</li>\n");
        SEND(&s->sout);
      }
    }
    ADD("  </ul>");
    SEND(&s->sout);
  }
#endif /* UIP_SR_LINK_NUM != 0 */

  SEND_STRING(&s->sout, BOTTOM);
```

```
  PSOCK_END(&s->sout);
}
/*---------------------------------------------------------------------
-------*/
PROCESS(webserver_nogui_process, "Web server");
PROCESS_THREAD(webserver_nogui_process, ev, data)
{

  uip_ip4addr_t ip4addr;
  uip_ip6addr_t ip6addr;

  PROCESS_BEGIN();

  uip_ipaddr(&ip4addr, 8,8,8,8);
  ip64_addr_4to6(&ip4addr, &ip6addr);
  uip_nameserver_update(&ip6addr, UIP_NAMESERVER_INFINITE_LIFETIME);

  httpd_init();
  http_socket_init(&s);

  char str[128];
  snprintf(str, sizeof(str),
"http://[fd00::1]/Web_TFM/php/ddbb_sv_mt.php?id_mote=0&id_parent=0&x_c
=%d&y_c=%d",x_router,y_router);
  http_socket_get(&s, str, 0, 0,
                  http_callback, NULL);

  static struct etimer http_timer;
  etimer_set(&http_timer, 10 * CLOCK_SECOND);
  PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&http_timer));

  while(1) {
    PROCESS_WAIT_EVENT_UNTIL(ev == tcpip_event);
    httpd_appcall(data);
  }

  PROCESS_END();
}
/*---------------------------------------------------------------------
-------*/
httpd_simple_script_t
httpd_simple_get_script(const char *name)
{
  return generate_routes;
}
```