GRADO EN INGENIERÍA INFORMÁTICA

TRABAJO FINAL DE GRADO

Red mallada de nodos Contiki para monitorización de temperaturas y envío de alertas por SMS

Realizado por:
Víctor POLO SAUCH

Supervisado por:
Adrián Rubén BORILLO NAVARRO

Tutorizado por:
Carlos Antonio HERNÁNDEZ ESPINOSA

Fecha de lectura: 14 de Septiembre de 2017
Curso académico 2016-2017

Resumen

En esta memoria se va a proceder a describir el diseño y proceso de desarrollo de una red de monitorización del entorno que envía periódicamente los datos recopilados y es capaz de enviar alarmas por SMS. Este documento recoge los detalles del proceso de diseño, la implementación y pruebas realizadas para la correcta implementación del proyecto. El diseño de la red de monitorización incluye el uso de nodos con el sistema operativo Contiki para recopilar los datos, de una Raspberry Pi para recibir los datos recopilados y enviar SMS en el caso que alguno de ellos sea una alarma, y de un nodo Border Router que permite la comunicación bidireccional entre los nodos Contiki y la Raspberry Pi. La comunicación entre los nodos Contiki y el Border Router se realiza mediante el protocolo RPL y la comunicación entre el Border Router y la Raspberry Pi se realiza sobre el protocolo Ethernet. Todos los mensajes de la red son codificados en JSON y enviados utilizando el protocolo CoAP. Los nodos Contiki son configurables.

Palabras clave

Nodos Contiki, 6LoWPAN, RPL, CoAP, MAX6675, Border Router, 802.15.4, Red mallada, monitorización, sensores, inalámbrico, alertas SMS

Keywords

Contiki nodes, 6LoWPAN, RPL, CoAP, MAX6675, Border Router, 802.15.4, Mesh network, monitoring, sensors, wireless, SMS alerts

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

1.1 Contexto y motivación del proyecto

El proyecto cuya memoria se presenta en este documento ha sido realizado en la empresa Thermesys. Ubicada en la Calle Lituania nº10 Edificio CIES-CS Ciudad del Transporte 12006 Castellón de la Plana.

Thermesys ha logrado crear un sistema de monitorización de entornos industriales inalámbrico, autosuficiente energéticamente, capaz de transmitir de forma periódica la información recopilada de los sensores que estén conectados al dispositivo AT100.

Al disponer de un sistema de monitorización activa, permite notificar una alerta en caso de detectar alguna anomalía en el sistema, aplicado a monitorizar redes de vapor, se notifica cuanto las trampas de vapor tienen un mal funcionamiento, acelerando el proceso de detección de fugas de vapor, reduciendo el gasto energético derivado de no resolver esas fugas de forma temprana, y por tanto reduciendo costes energéticos y medioambientales.

Al mismo tiempo el sistema que han diseñado permite su instalación “Plug and Play” sin necesidad de realizar ninguna modificación en las instalaciones donde se quiera saber la temperatura a monitorizar.

Este proyecto ha consistido en desarrollar un sistema de comunicaciones que monitorice las temperaturas en los sistemas de un entorno industrial y alerte de fluctuaciones de temperatura entre un intervalo predefinido, permitiendo detectar de forma inmediata posibles fugas/roturas, para así actuar inmediatamente reparando el elemento que funciona mal y evitando pérdidas energéticas durante meses que tienen un altísimo coste económico y medioambiental.

Los nodos cliente monitorizan las temperaturas de su entorno y notifican al nodo coordinador sus lecturas.

En este proyecto se van a utilizar los nodos para monitorizar temperaturas, pero no significa que se limiten a solo eso, los nodos se pueden adaptar a cualquier tipo de sensor para satisfacer las necesidades de un cliente en concreto.

1.2 Objetivos del proyecto

El principal objetivo de este proyecto es implementar una red malla utilizando el sistema operativo de código abierto Contiki en su versión 3.x y plataformas hardware basadas en el microcontrolador CC2538 de Texas Instruments para transmitir las lecturas de los sensores de temperatura MAX6675 desde la red de sensores hasta una Raspberry Pi (nodo sumidero) que almacenará/transmitirá los datos que reciba de los sensores.

En este proyecto se han utilizado sensores de temperatura MAX6675 pero se podrían utilizar otro tipo de sensores sin que afectara a la red de nodos diseñada.

La red de la parte de los sensores llamada WSN (Wireless Sensor Network), utiliza el protocolo de comunicación 6LoWPAN mientras que la red externa donde estará ubicada la Raspberry Pi utiliza el protocolo Ethernet IEEE 802.3.

Por lo tanto es necesario que el coordinador de la red WSN disponga de conectividad 6LoWPAN y Ethernet para actuar de Border Router entre ambas redes. El coordinador para este proyecto será una placa OpenMote que dispone de conectividad 6LoWPAN y una placa OpenBase que añade al OpenMote conectividad Ethernet.

Los nodos cliente envían las lecturas en un período determinado, este período debe ser configurable.

1.3 Estructura de la memoria

La documentación del desarrollo del proyecto comienza en el capítulo 2 con una descripción de las tecnologías empleadas y cómo se unen para formar los componentes del sistema. En el capítulo 3, se describe la idea de la arquitectura completa del sistema que tienen en Thermesys y la parte de esa arquitectura que ha sido desarrollada en el proyecto, la planificación seguida para la realización del proyecto y las modificaciones realizadas sobre la planificación inicial.

En el capítulo 4, se describen los requisitos técnicos definidos para el proyecto, tanto aquellos referentes a la infraestructura de red y de datos. El capítulo 5 ha sido divido en 2 partes. Se ha realizado una descripción técnica de la arquitectura de red diseñada y una descripción del diseño de la arquitectura lógica de los elementos que componen la red.

En el capítulo 6 se han especificado los detalles de la implementación de las funcionalidades más importantes del sistema y las pruebas realizadas para comprobar su correcto funcionamiento. Y, por último en el capítulo 7 se ha realizado una valoración del cumplimiento de los objetivos, se han propuesto mejoras y se ha incluido una opinión personal acerca del desarrollo del proyecto.

Capítulo 2

Descripción del proyecto

En este capítulo se describe el punto de partida del proyecto así como las tecnologías hardware y software utilizadas para su implementación.

2.1 Punto de partida

En la actualidad, Thermesys ya dispone de un sistema comercializado. En este proyecto se va a tratar de mejorarlo, suplir alguna de las carencias que tiene en la actualidad y añadir mayor funcionalidad al sistema gracias a Contiki 3.x.

La versión comercial consta de las siguientes características:

- Red Estrella entre nodos.
- Comunicación unidireccional, solamente los nodos clientes mandan información al coordinador pero no reciben.
- El nodo coordinador se comunica con una Raspberry Pi mediante un puerto serie TTL/USB.
- Hardware utilizado:
  - JENNIC JN5148 como microcontrolador.
  - Sensor de temperatura digital MAX6675 con un termopar tipo K.
  - JenniSense[1] como sistema operativo, es un port de Contiki OS que ya no tiene soporte del desarrollador y esta basado en una versión antigua (Contiki 2.5).
- Los nodos cliente envían la temperatura tomada cada 10 segundos al nodo coordinador.
- Los nodos se identifican con su dirección MAC y se comunican mediante su dirección IPV6.

En este proyecto se va a tratar de mejorar esa versión comercial añadiendo más funcionalidades:

- Red mallada (Mesh) entre nodos.
- Comunicación Bidireccional.
- El nodo coordinador se comunica con una Raspberry Pi mediante la interfaz Ethernet de ambos utilizando un switch.
- Hardware utilizado:
  - OpenMote CC2538 como microcontrolador.
  - OpenBase
  - Sensor de temperatura digital MAX6675 con un termopar tipo K.
- Contiki 3.x como sistema operativo de los nodos.
- Los nodos cliente pueden ser configurados remotamente para establecer el período entre mediciones de temperatura.
- Aprovechar que Contiki 3.x implementa la funcionalidad de IPv6 y utilizarla para los nodos de la red de sensores.

1 Enlace a wiki JenniSense: https://github.com/teco-kit/Jennisense/wiki

2.2 Definición del sistema

A continuación se detallan las tecnologías hardware y software utilizadas para desarrollar la red mallada de sensores, el nodo sumidero y el Border Router.

Para implementar el sistema de monitorización descrito, se han empleado las siguientes tecnologías hardware:

- Nodos con el microcontrolador CC2538 de Texas Instruments:
  - Nodos OpenMote
  - Nodos OpenChina
- Extensor de alcance CC2592:
  - Nodos OpenChina
- Placas OpenBase junto a los OpenMote
- Placas RHBDK1.1 junto a los OpenChina.
- Sensores de temperatura digital MAX6675 con termopar tipo K
- Programador JTAG J-Link SEGGER
- Olimex ARM-JTAG 20-10
- Raspberry Pi 1 Model B+
- Itead Raspberry PI GSM Add-on v2.0
  - Tarjetas SIM
- Cableado USB tipo A
- Cableado mini usb

Y las siguientes tecnologías software:

- Todo el sistema ha sido desarrollado en el lenguaje de programación C.
- Contiki OS como firmware de los Nodos.
  - Protocolos implementados:
    - IPv6
    - 6LoWPAN
    - RPL
    - CoAP
    - UDP
  - JSON para codificar los mensajes de los nodos.
  - Red mallada (mesh).
  - Servicios Implementados:
    - Servidor/Cliente UDP.
    - Servidor/Cliente CoAP.
- Raspbian Jessie Lite como sistema operativo de la Raspberry Pi:
  - Protocolos implementados:
    - CoAP (libcoap).
    - JSON para decodificar los mensajes de los nodos (jsmn).
  - Servicios Implementados:
    - Servidor CoAP.
    - Microservicio Alertas SMS.

Dichas tecnologías dan lugar a los tres componentes diferenciados del sistema:

1. **Nodo Cliente**: formados por un CC2538, una antena y un sensor MAX6675 junto a un termopar tipo K.
2. **Border Route**r: formado por un OpenMote conectado a una OpenBase y una antena de alta ganancia.
3. **Nodo Sumidero**: formado por una Raspberry Pi 1 model B+ y el módulo GSM Addon V2.0.

2.3 Descripción de las tecnologías hardware

En este apartado se describen las tecnologías hardware utilizadas para formar los componentes del sistema descritos antes.

2.3.1 Microcontrolador CC2538

El CC2538[2] es el microcontrolador inalámbrico ideal para aplicaciones WSN (Wireless Sensor Network) de alto rendimiento y bajo coste. Combina un potente procesador ARM Cortex-M3 con hasta 32 KB de RAM y hasta 512 KB de memoria flash con el robusto IEEE 802.15.4[3] estándar que define el nivel físico y el control de acceso al medio de redes inalámbricas de área personal con tasas bajas de transmisión de datos.

El microcontrolador, que está integrado en los nodos, está desarrollado por Texas Instruments, en concreto es el modelo CC2538SF53.

Figura 2.1: Microcontrolador CC2538

Tipo de dispositivo	Microcontrolador Inalámbrico
Frecuencia de reloj	24 MHz
Flash	512 KB
RAM	32 KB
Periféricos	USB I2C SSI SPI UART
Timers 16 bit	8
Timers 32 bit	4
GPIO	32
ADC	12 bit, 8 canales
Consumo en reposo	1.3 uA
Potencia de Salida	7 dBm
Sensibilidad de recepción (RX)	-97 dBm
Tasa de transmisión de datos	250 kbps
Temperaturas de funcionamiento	-40 ºC a 125 ºC
Aplicaciones clave	2.4-GHz IEEE 802.15.4
Tecnología	2.4-GHz IEEE 802.15.4 6LoWPAN ZigBee

Tabla 2.1: Especificaciones técnicas CC2538

2 Especificaciones del fabricante: http://www.ti.com/product/CC2538
3 Definición estándar IEEE 802.15.4: https://es.wikipedia.org/wiki/IEEE_802.15.4

2.3.2 Extensor de alcance CC2592

El CC2592[4] es un extensor de alcance compatible con toda la familia de microcontroladores CC25XX. Para amplificar el alcance de la radio, el CC2592 incorpora un amplificador de potencia para aumentar la potencia de salida y un LNA (Low Noise Amplifier) para mejorar la sensibilidad de la recepción. Utilizando el CC2592 junto al CC2538 se consigue una mejora en la sensibilidad de recepción de unos 3 dBm.

Figura 2.2: Extensor de alcance CC2592

Tipo de dispositivo	Extensor de alcance
Frecuencia	2400 MHz
Consumo en recepción (RX)	1.9 mA
Consumo en reposo	0.1 uA
Sensibilidad de recepción (RX)	Mejora de 3 dB
Potencia de salida	22 dBm
Tasa de transmisión de datos	500 2000
Temperaturas de funcionamiento	-40 ºC a 125 ºC
Tecnología	2.4-GHz IEEE 802.15.4 6LoWPAN ZigBee

Tabla 2.2: Especificaciones técnicas CC2592

2.3.3 OpenMote

OpenMote[5] es una placa de desarrollo que implementa el microcontrolador CC2538 previamente descrito, es una plataforma soportada oficialmente por Contiki OS.

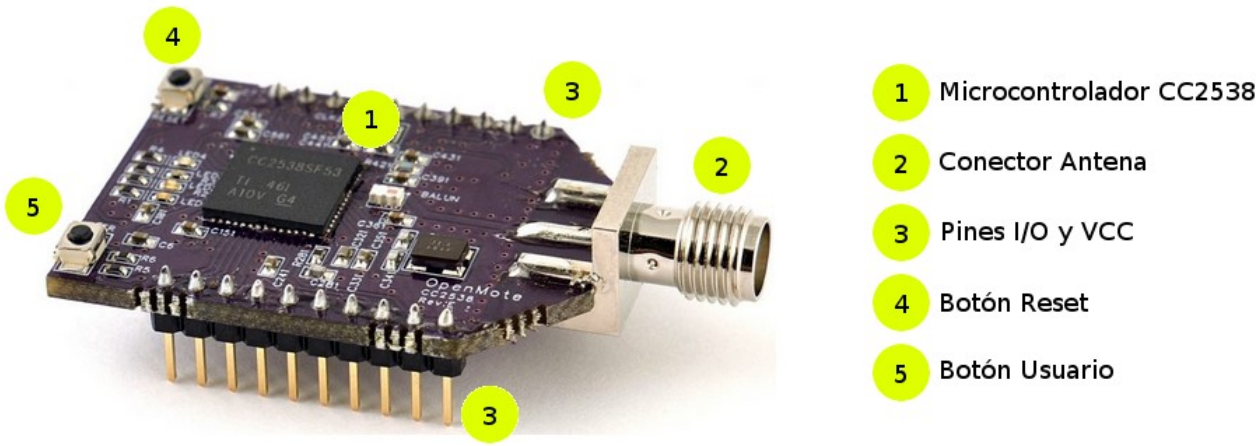


Figura 2.3: OpenMote

4 Especificaciones del fabricante: http://www.ti.com/product/CC2592
5 Enlace al fabricante: http://openmote.com/product/openmote-cc2538/

Las características a destacar de los OpenMote son:

- Microcontrolador CC2538.
- Factor forma Xbee.
- 20 Pines Entrada/salida accesibles
- 4 leds configurables.
- Botón configurable.
- Botón reset.
- Conector antena externa.

2.3.4 OpenBase

La placa de expansión OpenBase ha sido específicamente diseñada para la placa OpenMote. Esta placa añade conectividad USB al OpenMote para programación/debug y conectividad Ethernet que convierte al conjunto de OpenMote y OpenBase en el hardware ideal para ser utilizado como Border Router.

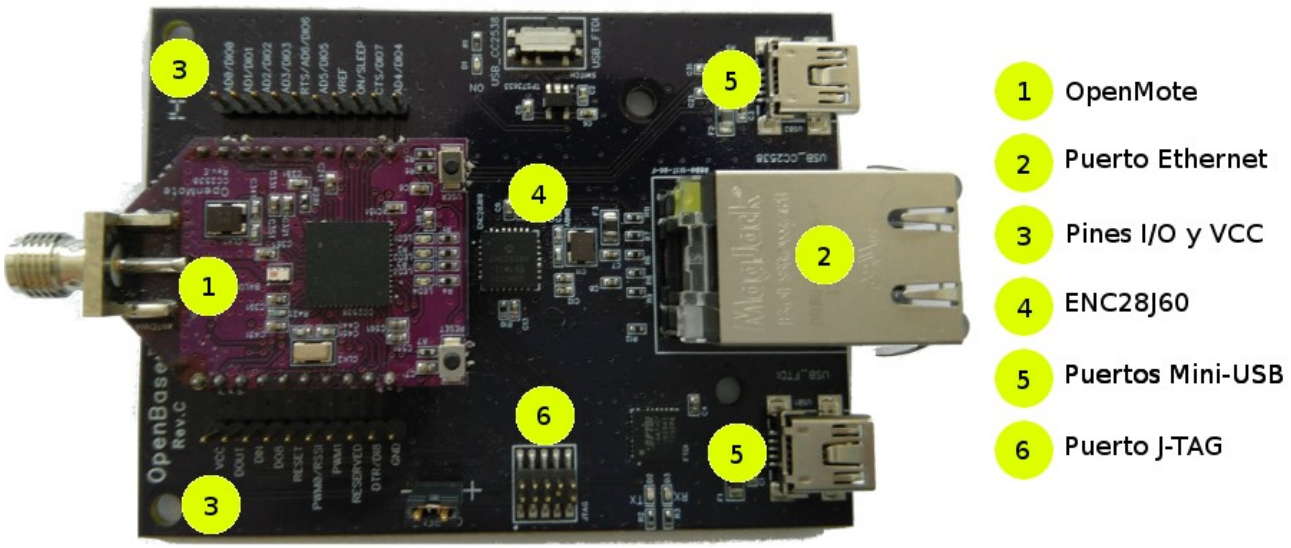


Figura 2.4: OpenBase con OpenMote

Añade las siguientes características a los OpenMote:

- Conectividad Ethernet.
  - Integrado ENC28J60 por puerto SPI.
- Puerto JTAG ARM 10.
- Puertos micro USB seleccionables mediante switch.
  - USB a UART (FTDI).
  - USB a CC2538.
- Los pines accesibles en los OpenMote están extendidos en la placa.
- Los pines del puerto JTAG ARM 10 están extendidos en la placa.

2.3.5 OpenChina

Conjunto del microcontrolador CC2538 más el extensor de alcance CC2592 definido por Texas Instruments, cada ensamblador varía los pines y puertos accesibles, en este caso el ensamblador la ha nombrado “SZ12 Zigbee PA module (CC2538 + CC2592)”. Ha sido renombrada a OpenChina por comodidad a la hora de añadir soporte para esta placa en Contiki 3.x (Ver *Anexo Nodos III: Añadir nuevo platform a Contiki*).

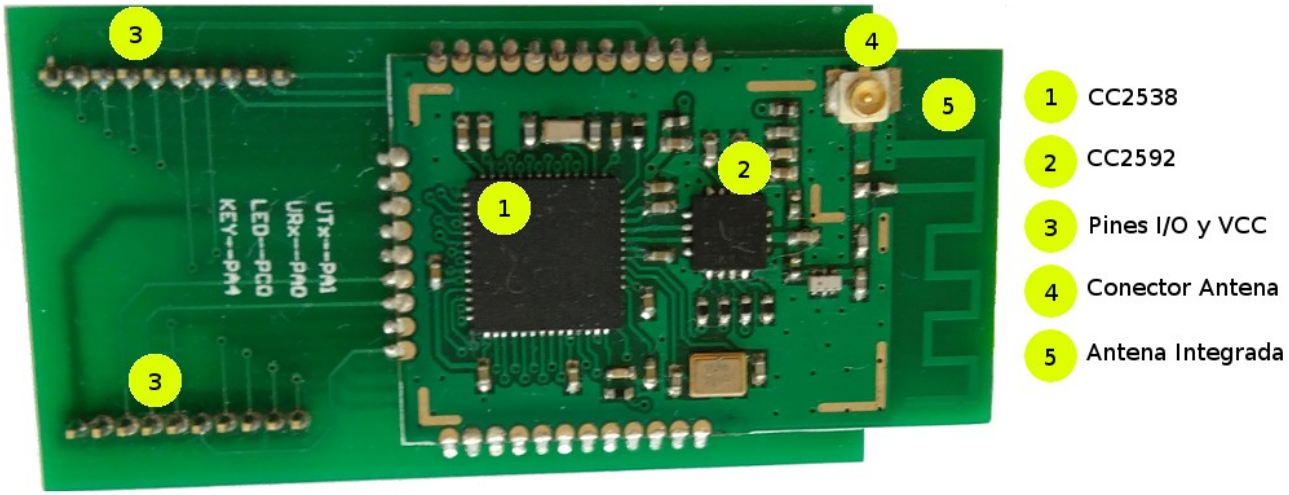


Figura 2.5: OpenChina

Las diferencias que tiene respecto los OpenMote son:

- Integrado CC2592 que permite extender el alcance de la radio.
- Posibilidad de utilizar una antena externa o utilizar la integrada en la misma placa PCB.
- No incorporan leds ni botones.

Parámetro	Valor	Detalles
Sensibilidad	-103.5dBm	
Potencia de salida	+ 20dBm	+19.5~+21dBm
Ganancia de entrada	12dB	
Ruido	4dB	
Consumo en emisión	<175mA	Máxima transmisión de potencia
Consumo en recepción	30mA	
Consumo en reposo	0.4uA	
Voltaje de funcionamiento	+3.3V	
Distancia de transmisión	600m	Con la antena integrada y visión directa
Tasa de transmisión de datos	250 Kbps	
Número de canales	16	
Antena	PCB o IPEX	
Temperaturas de funcionamiento	-40 °C a 80 °C	
Tamaño	21x32 mm	Espaciado entre pines 1.27mm

Tabla 2.3: Especificaciones técnicas OpenChina

2.3.6 RHBDK1.1

Placa de expansión para la OpenChina que permite la programación mediante el puerto JTAG ARM 10, conectividad USB tipo B, alimentación externa y acceso a algunos de los pines de la OpenChina.

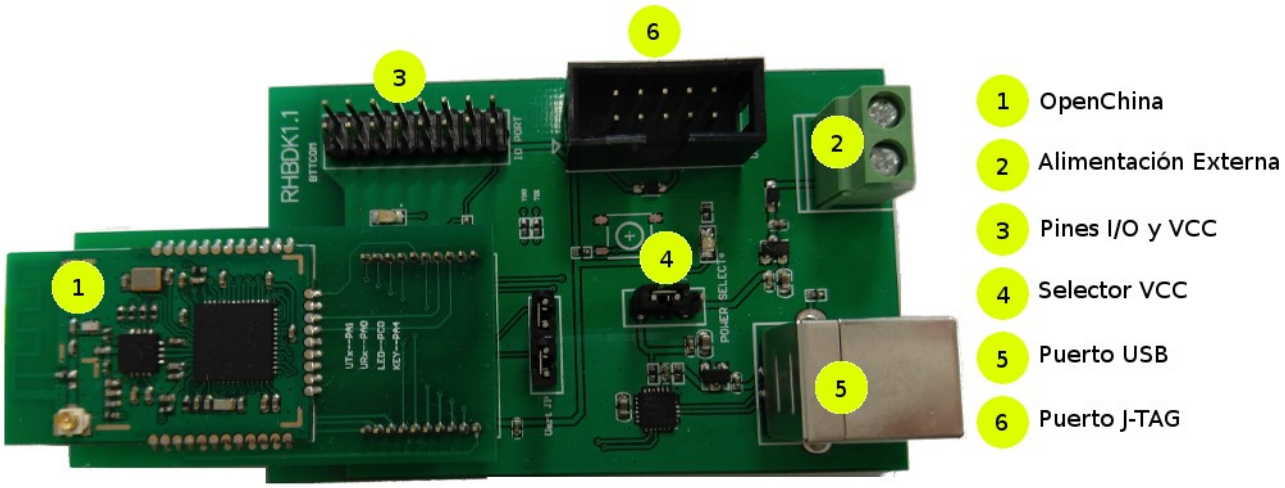


Figura 2.6: RHBDK con OpenChina

La placa de expansión RHBDK1.1 añade las siguientes características a los OpenChina:

- Puerto JTAG ARM 10.
- Puerto USB tipo B.
  - USB a UART (CP2104[6]).
- Alimentación externa.
- Led verde.
- Pulsador (opcional).
- Selección de entrada de alimentación mediante jumper.

Parámetro	Detalles
Puerto entrada/salida	16 pines
Puerto Debug	Arm 10 Jtag
Alimentación externa	+3.3V ~ 6V
Alimentación puerto USB	+5V
Consumo	<350 mA

Tabla 2.4: Especificaciones técnicas RHBDK 1.1

6 Datasheet CP2104: https://www.silabs.com/documents/public/data-sheets/cp2104.pdf

2.3.7 MAX6675

El sensor de temperatura MAX6675 permite conectar un termopar tipo K, convierte la señal analógica generada por la diferencia de potencial entre los dos terminales del termopar a digital con un rango de 12 bits, lo que permite tomar unas de mediciones positivas de 0º C hasta 1023º C. Utiliza el protocolo de comunicación SPI.

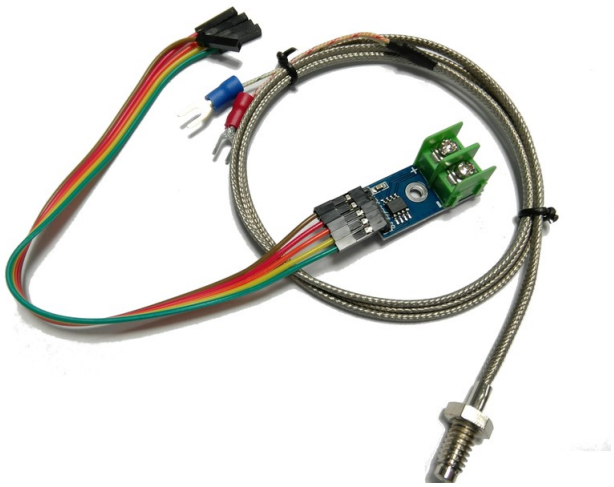


Figura 2.7: MAX6675 junto termopar tipo K

Parámetro	Detalles
Voltaje de operación	3.3V~5V
Corriente de trabajo	50mA
Rango de temperaturas termopar tipo K	-200ºC hasta 1300ºC
Rango de temperaturas del MAX6675	0º - 1023ºC
Resolución de temperatura	0.25º C
Protocolo de comunicación	SPI
Dimensiones	25mm*15mm*13mm
Número de pines	8

Tabla 2.5: Especificaciones técnicas MAX6675

2.3.8 Raspberry Pi 1 model B+

Raspberry Pi es una plataforma hardware del tamaño de una tarjeta de crédito que permite ejecutar sistemas operativos mediante una tarjeta SD/Micro SD (según modelo). Los sistemas operativos más comunes para esta placa son distribuciones Linux pero también existe Windows 10 IoT para la Raspberry Pi 2 en adelante.

La Raspberry Pi 1 model B+ es la última revisión del modelo original de Raspberry Pi. Es el antecesor de la Raspberry Pi 2.

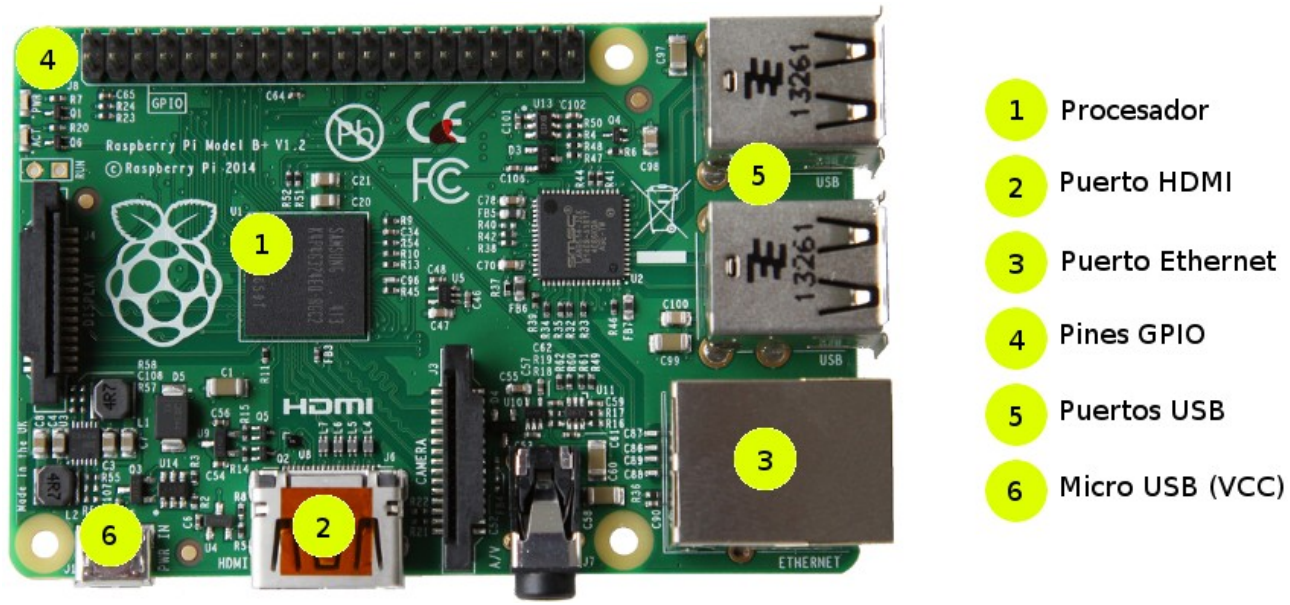


Figura 2.8: Raspberry Pi 1 model B+

Las características más destacables de modelo de placa son:

- Salida analógica de vídeo y audio mediante puerto jack.
- Salida digital mediante puerto HDMI.
- 4 puertos USB.
- Puerto Ethernet 10/100 Mbps.
- 40 pines GPIO (General Purpose Input/Output).
- Alimentación mediante puerto micro usb o pines GPIO.
- SOC Broadcom BCM2835 con un core funcionando a 700Mhz.
- 512 Mb de memoria RAM.
- Consumo: 0.5 W a 1 W

2.3.9 Itead Raspberry PI GSM Add-on v2.0

Raspberry Pi SIM800 GSM/GPRS Add-on V2.0[7] es una placa basada en el módulo GSM/GPRS/BT SIM800. Específicamente diseñado para la Raspberry Pi model B+ y modelos B posteriores. Formato apilable que se conecta directamente a los pines GPIO de la Raspberry Pi.

El módulo SIM800 funciona mediante comandos AT utilizando el puerto serie (UART) de la Raspberry Pi a una velocidad por defecto de 9600 baudios. Permite enviar/recibir llamadas, enviar/recibir SMS y utilizar la conectividad GSM para acceder a Internet (Ver *Anexo Raspberry Pi: Configuración GSM Add-on v2.0 como acceso a Internet*).

Figura 2.9: GSM Add-on conectado a Raspberry Pi

2.4 Descripción de las tecnologías software

En este apartado se describen las tecnologías software utilizadas para formar los componentes del sistema descritos en el capítulo anterior.

2.4.1 Lenguaje de programación C

Se ha elegido este lenguaje de programación porque es el necesario para desarrollar en Contiki, las librerías de CoAP utilizadas también están desarrolladas en C. Por lo que todos los programas escritos en el proyecto han sido en C.

7 Especificaciones del fabricante: https://www.itead.cc/wiki/RPI_SIM800_GSM/GPRS_ADD-ON_V2.0

2.4.2 Contiki OS

Contiki es un sistema operativo de código abierto, multiplataforma y multitarea en tiempo real para el Internet de las cosas (IOT), conecta microcontroladores de bajo coste y bajo consumo a Internet. Está escrito en lenguaje C y utiliza el concepto de Protothread[8], lo que le permite un uso muy reducido de la memoria del microprocesador en comparación con otras arquitecturas multitarea. Enfocado para usarse en sistemas sencillos y empotrados sobre microcontroladores, como pueden ser los nodos de una red de sensores.

Soporta multitud de plataformas entre las que se incluye la plataforma hardware OpenMote anteriormente descrita. Además incluye diversas utilidades e implementaciones de protocolos (CoAP, UDP, FTP, etc) para operar a nivel de aplicación en su código fuente.

2.4.3 Red mallada (mesh)

Una red mallada es aquella red en la se que mezclan dos topologías de red inalámbricas, la topología Ad-hoc[9] y la topología infraestructura[10]. Una red mallada permite que se unan a ella dispositivos que no tienen alcance con el punto de acceso aunque si que lo tienen a un nodo vecino, el cual tiene alcance directo o indirecto al punto de acceso.

Por lo tanto, un nodo de la red puede estar transmitiendo paquetes suyos o de un vecino que no tiene alcance con el punto de acceso. Esto permite que la red se pueda distribuir a mayores distancias solamente ubicando físicamente nodos en los puntos ciegos para permitir alcanzar al punto de acceso a los nodos lejanos. No es necesario que todos los nodos tengan visión directa del nodo coordinador, pueden alcanzarlo utilizando a sus vecinos (otros nodos cliente). Un cliente que no ve a su coordinador, preguntará a sus vecinos si ven a ese nodo. Por lo tanto, se va a establecer una ruta mediante saltos que le permita a ese nodo llegar a su coordinador.

Gracias a que se establece una comunicación indirecta con el nodo coordinador mediante saltos, se pueden ubicar nodos en lugares más alejados sin la obligación de estar cerca del nodo coordinador y se pueden superar obstáculos infranqueables.

Una red mallada precisa además de un protocolo de enrutamiento para transmitir la información hasta su destino con el mínimo número de saltos.

2.4.4 RPL

En el año 2008 se fundó el grupo “Routing Over Low-power and Lossy networks (ROLL)” para crear un nuevo protocolo que encajara con el estándar 6LoWPAN ya que este no define el protocolo de enrutamiento. El nuevo protocolo de enrutamiento creado fue Routing Protocol for Low power and Lossy Networks (RPL). El objetivo principal de este protocolo es proporcionar diferentes caminos de enrutamiento para tres tipos de patrones de tráfico: multi-punto a punto (MP2P), punto a multipunto (P2MP) y punto a punto (P2P).[1]

8 Protothread: Metodología de programación basada en macros que se utiliza en microcontroladores con bajos recursos de forma similar a un sistema de tareas pero sin utilizar memoria RAM para cada uno de los hilos que se ejecutan
9 Ad-hoc: no requieren un punto de acceso central para iniciar una comunicación, se realizan conexiones punto a punto.
10 Topología Infraestructura: se requiere de un punto de acceso central para que todos los nodos se conecten.

2.4.5 CoAP

CoAP (Constrained Application Protocol) es un protocolo software REST (Representational State Transfer) a nivel de aplicación pensado para ser utilizado en dispositivos electrónicos de bajo consumo. Es un protocolo que funciona sobre una capa de transporte no orientada a la conexión (UDP), hereda directamente del protocolo HTTP (orientado a conexión TCP) los métodos CRUD (Create Read Update Delete) para la gestión de los recursos: GET, POST, PUT y DELETE.

CoAP utiliza dos tipos de mensajes:

- Con confirmación (CON): El receptor de un mensaje CON debe responder al emisor.

- Sin confirmación (NON): El receptor de un mensaje NON no debe responder al emisor. El uso de este tipo de mensajes evita la saturación de la red.

CoAP codifica los mensajes en formato binario en lugar de texto plano como hace HTTP, lo que supone una reducción significativa de la longitud de las cabeceras y del propio mensaje, permitiendo un mejor aprovechamiento del ancho de banda disponible.

2.4.6 Copper (Cu)

Copper (Cu)[11] es un plugin para el navegador Mozilla Firefox que proporciona a éste de un cliente CoAP, permitiéndole acceder a los recursos de un servidor CoAP.

2.4.7 JSON

JSON (JavaScript Object Notation), es un formato de texto ligero para el intercambio de datos.

Entre sus características destacan las siguientes:

- Es fácil para las personas de leer y de escribir.
- Es fácil para las máquinas para analizar y generar.
- JSON es un formato de texto que es completamente independiente del lenguaje de programación que se utilice.

2.4.8 Librería: jsmn[12]

Librería escrita en C que permite codificar y decodificar mensajes escritos en formato JSON. La ventaja de esta librería respecto a otras más complejas es que se adapta perfectamente al propósito de este proyecto y es muy simple de utilizar.

2.4.9 Librería: libcoap[13]

Librería CoAP escrita en C multiplataforma, diseñada específicamente para dispositivos con limitación de recursos: como la potencia de computación, alcance de la radio, memoria, ancho de banda o tamaño de paquete.

libcoap está diseñado para ejecutarse en dispositivos embebidos así como sistemas informáticos de alto rendimiento con POSIX OS, lo que permite desarrollar y probar aplicaciones CoAP en un equipo Linux y más tarde portar esa implementación a la plataforma de destino con facilidad.

11 Enlace descarga plugin Copper: https://addons.mozilla.org/es/firefox/addon/copper-270430/
12 Enlace código fuente librería jsmn: https://github.com/zserge/jsmn
13 Enlace página web libcoap: https://libcoap.net/

2.4.10 CETIC 6LBR

CETIC 6LBR[14] es una solución Border Router 6LoWPAN/RPL. 6LBR puede funcionar como un router autónomo en un sistema empotrado o funcionar en un sistema Linux. Está diseñado para ser flexible, se puede configurar para soportar varias topologías de red mientras interconecta las redes de sensores (WSN) con el mundo IP. Listo para funcionar en sistemas Linux y las plataformas soportadas por Contiki sin modificaciones. Proporciona de una interfaz web que permite realizar los ajustes de igual forma que en un router convencional y además permite ver que nodos tiene conectados, como están conectados a él y cuantos saltos realizan para alcanzarlo.

2.4.11 Comandos AT

Los comandos AT, también conocidos como "conjunto de comandos Hayes[15]", desarrollados por la compañía Hayes Communications se convirtieron prácticamente en estándar abierto de comandos para configurar y parametrizar módems. Actualmente sirve para multitud de dispositivos como módems GSM, dispositivos GPS y dispositivos Bluetooth.

2.5 Descripción de las herramientas de desarrollo

En este apartado se van a detallar las herramientas hardware y software utilizadas para realizar el proyecto.

2.5.1 Herramientas de desarrollo hardware

SEGGER J-Link

SEGGER J-Link[16] es un emulador JTAG por USB diseñado específicamente para procesadores ARM. Está basado en un procesador RISC de 32-bits, se puede comunicar a alta velocidad con un gran número de procesadores ARM entre los que se incluye el utilizado en este proyecto. Dispone del puerto JTAG ARM 20. Permite programar el firmware a una gran variedad de dispositivos con procesador ARM.

Figura 2.10: SEGGER J-Link

14 Enlace wiki código fuente CETIC 6LBR: https://github.com/cetic/6lbr/wiki
15 Definición comandos AT: https://es.wikipedia.org/wiki/Conjunto_de_comandos_Hayes
16 Enlace especificaciones J-link: http://www.mouser.es/new/segger/seggerjlink/

Olimex ARM-JTAG 20-10

Adaptador para poder programar los OpenMote junto a su OpenBase utilizando el J-Link SEGGER.

Figura 2.11: Olimex ARM-JTAG 20-10

Otras herramientas utilizadas

- Osciloscopio SIGLENT modelo SDS 1102CNL.
- Multímetro digital.
- Cableado mini usb y usb tipo B.
- Cableado para protoboard.

2.5.2 Herramientas de desarrollo software

- Para el desarrollo del software en C se han utilizado los editores de texto **Gedit** y **Nano**.

- **picocom:** Programa por línea de comandos que permite establecer conexión por puerto serie.

- **GDB:** Depurador GNU para la plataforma que lo ejecuta.

- **Valgrind:** Permite depurar código en tiempo de ejecución, muy útil para detectar fallos en la asignación de memoria o accesos incorrectos.

- **JlinkGBDServer:** Con el SEGGER J-Link conectado al PC y al puerto JTAG del microcontrolador a programar o realizar debug, JlinkGBDServer crea un servidor TCP accesible mediante la utilidad GDB.

- **arm-none-eabi-gdb:** Depurador GNU para dispositivos ARM, permite establecer una conexión TCP al servidor creado por el JlinkGBDServer. Una vez conectados al servidor, podemos realizar tareas de debug sobre la plataforma ARM a la que nos hemos conectado e incluso cargar otros binarios compilados para esa plataforma.

- **cc2538-bsl:** Script escrito en python que permite la programación de los nodos con el microcontrolador CC2538 con el flag bootloader backdoor activo. De esta forma se pueden programar los nodos simplemente conectándolos por el puerto usb al pc con un puente a GND o VCC de un pin de la placa.

Capítulo 3

Planificación del proyecto

La planificación se realizó a partir de la descripción del proyecto y de varias reuniones con los integrantes de Thermesys donde se detalló el diseño de arquitectura completo que querían implementar. Y qué parte se iba a implementar en el proyecto.

3.1 Descripción de la arquitectura completa del sistema

La idea de la arquitectura completa del sistema que quiere implementar Thermesys consistiría en varias redes de nodos, cada una de ellas con los sensores convenientes y un nodo sumidero encargado de subir los datos recopilados a un servidor en la nube.

La conectividad de cada una de las redes de nodos dependerá del medio dónde se ubiquen las redes, pero todas se conectarán a un servidor VPN (Virtual Private Network) para enviar sus datos recopilados y ser fácilmente identificables y accesibles al pertenecer todas al rango de red de la VPN sin importar que dirección IP pública tienen en cada momento.

Muy útil, puesto que muchas de las redes de sensores dispondrán solamente de conectividad 2G/3G por lo que su acceso desde el exterior puede ser complicado al depender del proveedor de red de la SIM utilizada y si ese proveedor da una IP pública y accesible o comparte la misma dirección IP entre varios clientes.

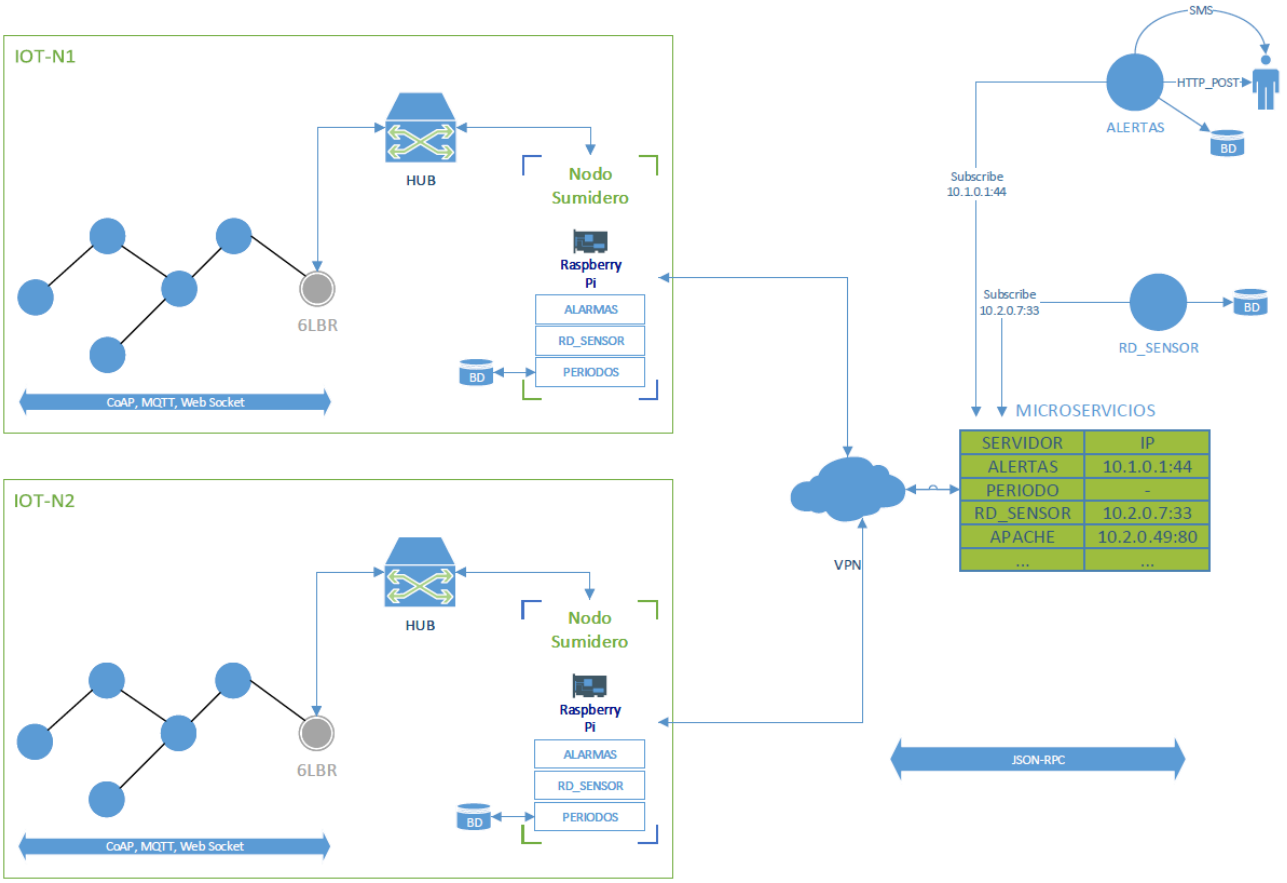


Figura 3.12: Esquema arquitectura completa del sistema

Como se puede observar en la figura anterior, en esa red VPN existirán varios microservicios que no tienen porque estar implementados en un mismo servidor como alertas por SMS, servidores web para acceder a las redes de sensores, etc. Toda comunicación en el sistema seria codificada en JSON-RPC. Y los mensajes de las redes de sensores se enviarían utilizando algún protocolo de comunicación por determinar como CoAP, MQTT o Web Socket.

3.1.1 Requisitos del proyecto

En este proyecto se va a realizar la implementación de las redes de nodos y del nodo sumidero que se puede ver en la siguiente figura.

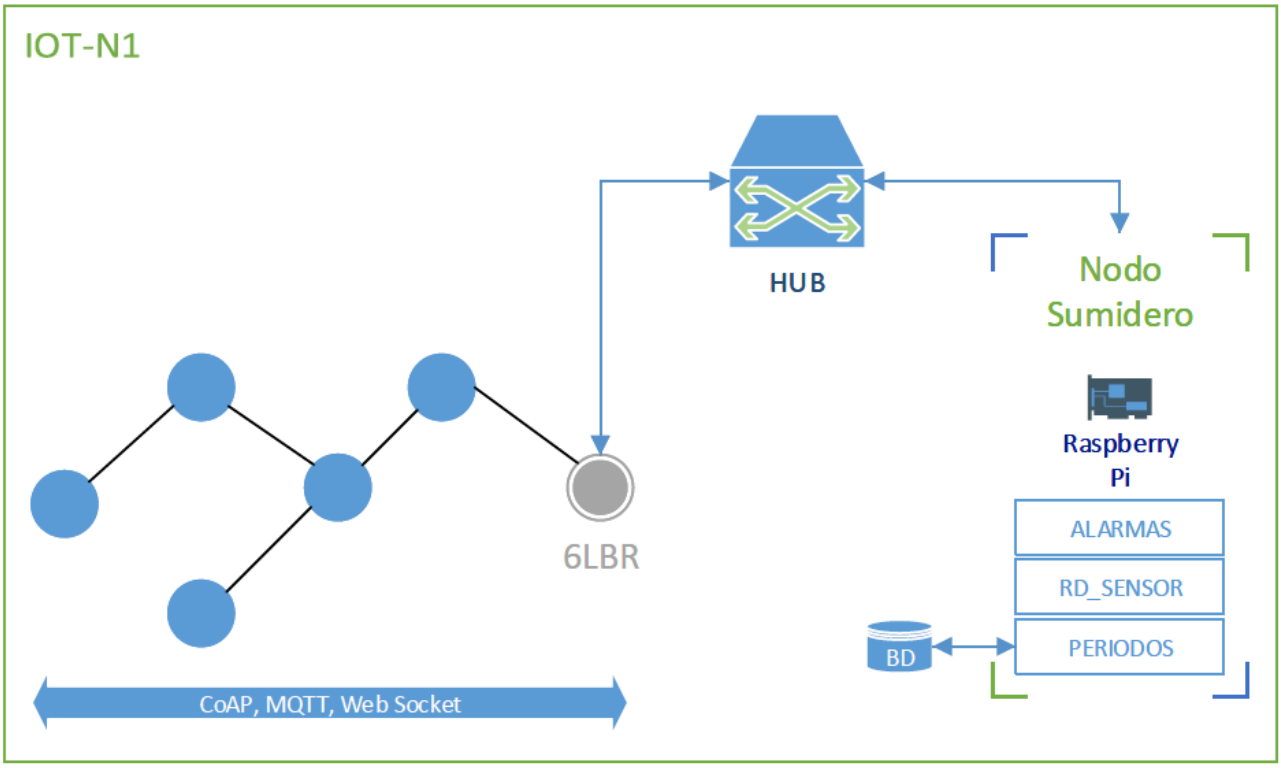


Figura 3.13: Esquema arquitectura requisitos del proyecto inicial

La descripción del proyecto a desarrollar en Thermesys fue la siguiente:

"El proyecto consiste en implementar una red mesh usando Contiki 3.x y la plataforma hardware OpenMote, para transmitir las lecturas de los sensores de temperatura (MAX6675) desde la red de sensores hasta un nodo coordinador de la red (Raspberry Pi).

Entrando un poco en detalle, las tareas serán las siguientes:

0) Familiarizarse con el entorno de desarrollo

1) Implementar driver SPI para el sensor de temperatura MAX6675 usando Contiki 3.x.

2) Implementar comunicaciones en mesh usando el protocolo RPL implementado por Contiki.

3) Puesta en marcha de la Raspberry PI para actuar como nodo sumidero.

4) Puesta en marcha del Border Router 6LBR para interconectar nodos de la red con nodo sumidero.

5) Determinar el protocolo a utilizar para enviar datos desde los sensores al nodo sumidero (CoAP, MQTT o Web Socket).

6) Pequeña pagina web que permita al usuario listar todos los sensores disponibles en la red (sus IPv6), configurar un periodo de envió para todos ellos, y actualizar esa información a todos los sensores de la red."

3.2 Planificación

A partir de los requisitos iniciales del proyecto se especificaron las tareas del mismo y se estimó la duración de las mismas.

3.2.1 Planificación temporal inicial

Nº	Tareas	Tiempo (h)	Dependencias
	Duración total de proyecto	300	
1	Formación	15	
1.1	Familiarizarse con el entorno de desarrollo	15	
2	Tareas	285	
2.1	Implementar driver SPI	21	
2.1.1	Analizar Datasheet de los OpenMote, OpenChina y del sensor MAX6675	5	1
2.1.2	Programar driver para OpenMote y OpenChina con Contiki 3.x	15	2.1.1
2.1.3	Verificar su correcto funcionamiento mediante un programa de prueba	1	2.1.2
2.2	Implementar comunicaciones mesh con el protocolo RPL	30	
2.2.1	Documentarse sobre el funcionamiento de las redes Mesh y el protocolo RPL	8	2.1
2.2.2	Documentarse sobre su uso en Contiki 3.x	10	2.2.1
2.2.3	Implementación en nodos OpenMote y OpenChina con Contiki 3.x	10	2.2.2
2.2.4	Verificación de su funcionamiento enviando la temperatura leída por un nodo a otro	2	2.2.3
2.3	Puesta en marcha del nodo sumidero	9	
2.3.1	Instalación de Raspbian Jessie Lite	1	2.2
2.3.2	Configuración de Raspbian	7	2.3.1
2.3.3	Verificar conectividad IPv6	1	2.3.2
2.4	Puesta en marcha del Border Router 6LBR	65	
2.4.1	Documentarse sobre el funcionamiento de Cetic 6LBR	25	2.3
2.4.2	Descargar los fuentes y compilarlos para OpenMote	10	2.4.1

2.4.3	Programar un OpenMote junto a su OpenBase con los binarios generados	1	2.4.2
2.4.4.	Configurar Border Router 6LBR y Raspberry Pi para tener conectividad IPv6	15	2.4.3
2.4.5	Programar otros nodos como clientes del Border Router	11	2.4.4
2.4.6	Realizar pruebas de conectividad de la red Ethernet y la red WSN mediante el envío de paquetes ICMP.	3	2.4.5
2.5	Determinar qué protocolo se adapta mejor para enviar datos desde los sensores al nodo sumidero	20	
2.5.1	Documentarse sobre el protocolo CoAP	5	2.4
2.5.2	Documentarse sobro el protocolo MQTT	5	2.5.1
2.5.3	Documentarse sobre los Web Sockets	5	2.5.2
2.5.4	Decidir qué protocolo se adapta mejor a las necesidades del proyecto.	5	2.5.3
2.6	Utilizar el protocolo elegido para enviar datos desde los sensores al nodo sumidero	90	
2.6.1	Realizar pruebas con el protocolo elegido enviando las mediciones de los nodos a la Raspberry Pi	20	2.5
2.6.2	Realizar pruebas con el protocolo elegido enviando ajustes de la Raspberry Pi hacia un nodo y hacia varios	20	2.6.1
2.6.3	Implementar Cliente/Servidor en los nodos	20	2.6.2
2.6.4	Implementar Cliente/Servidor en la Raspberry Pi	28	2.6.3
2.6.5	Verificar su correcto funcionamiento	2	2.6.2
2.7	Página web para gestionar los nodos y visualizar la información	50	
2.7.1	Implementación y configuración de servidor web en Raspberry Pi	5	2.6
2.7.2	Programación de la página web que permita el uso del protocolo elegido en el punto 2.6	35	2.7.1
2.7.3	Verificar su correcto funcionamiento, tanto de envío de ajustes como de recepción de datos	10	2.7.2

Tabla 3.6: Planificación temporal inicial

3.2.2 Planificación temporal definitiva

Nº	Tareas	Tiempo (h)	Dependencias
	Duración total de proyecto	300	
1	Formación	15	
1.1	Familiarizarse con el entorno de desarrollo	15	
2	Tareas	285	
2.1	Implementar driver SPI	21	
2.1.1	Analizar Datasheet de los OpenMote, OpenChina y del sensor MAX6675	5	1
2.1.2	Programar driver para OpenMote y OpenChina con Contiki 3.x	15	2.1.1
2.1.3	Verificar su correcto funcionamiento mediante un programa de prueba	1	2.1.2
2.2	Implementar comunicaciones mesh con el protocolo RPL	30	
2.2.1	Documentarse sobre el funcionamiento de las redes Mesh y el protocolo RPL	8	2.1
2.2.2	Documentarse sobre su uso en Contiki 3.x	10	2.2.1
2.2.3	Implementación en nodos OpenMote y OpenChina con Contiki 3.x	10	2.2.2
2.2.4	Verificación de su funcionamiento enviando la temperatura leída por un nodo a otro	2	2.2.3
2.3	Puesta en marcha del nodo sumidero	9	
2.3.1	Instalación de Raspbian Jessie Lite	1	2.2
2.3.2	Configuración de Raspbian	7	2.3.1
2.3.3	Verificar conectividad IPv6	1	2.3.2
2.4	Puesta en marcha del Border Router 6LBR	65	
2.4.1	Documentarse sobre el funcionamiento de Cetic 6LBR	25	2.3
2.4.2	Descargar los fuentes y compilarlos para OpenMote	10	2.4.1
2.4.3	Programar un OpenMote junto a su OpenBase con los binarios generados	1	2.4.2
2.4.4.	Configurar Border Router 6LBR y Raspberry Pi para tener conectividad IPv6	15	2.4.3
2.4.5	Programar otros nodos como clientes del Border Router	11	2.4.4
2.4.6	Realizar pruebas de conectividad de la red Ethernet y la red WSN mediante el envío de paquetes ICMP.	3	2.4.5
2.5	Determinar qué protocolo se adapta mejor para enviar datos desde los sensores al nodo sumidero	20	
2.5.1	Documentarse sobre el protocolo CoAP	5	2.4

2.5.2	Documentarse sobro el protocolo MQTT	5	2.5.1
2.5.3	Documentarse sobre los Web Sockets	5	2.5.2
2.5.4	Decidir qué protocolo se adapta mejor a las necesidades del proyecto.	5	2.5.3
2.6	Utilizar el protocolo CoAP para enviar datos desde los sensores al nodo sumidero	90	
2.6.1	Realizar pruebas con el protocolo CoAP enviando las mediciones de los nodos cliente al nodo sumidero	20	2.5
2.6.2	Realizar pruebas con el protocolo CoAP enviando ajustes de la Raspberry Pi hacia un nodo y hacia varios nodos	20	2.6.1
2.6.3	Implementar Cliente y Servidor en los nodos cliente	20	2.6.2
2.6.4	Implementar Cliente y Servidor en el nodo sumidero	28	2.6.3
2.6.5	Verificar su correcto funcionamiento	2	2.6.2
2.7	Implementar Microservicio que permita enviar notificaciones de los nodos por SMS	50	
2.7.1	Documentarse sobre los comandos AT GSM	2	2.6
2.7.2	Puesta en marcha del GSM Addon Raspberry Pi y primeras pruebas	5	2.7.1
2.7.3	Implementar Microservicio alertas SMS	30	2.7.2
2.7.4	Implementar conectividad al Microservicio en Servidor CoAP del nodo sumidero	10	2.7.3
2.7.5	Verificar su correcto funcionamiento	3	2.7.4

Tabla 3.7: Planificación temporal definitiva

Id	Modo de tarea	Nombre de tarea	Duración	Comienzo	Fin
1		**Proyecto**	**37,38 días**	**lun 27/03/17**	**mar 30/05/17**
2		**Formación**	**1,88 días**	**lun 27/03/17**	**mié 29/03/17**
3		Familiarizarse con el entorno de desarrollo	15 horas	lun 27/03/17	mié 29/03/17
4		**Tareas**	**35,63 días**	**mié 29/03/17**	**mar 30/05/17**
5		**Implementar driver SPI**	**2,63 días**	**mié 29/03/17**	**lun 03/04/17**
6		Analizar Datasheet de los OpenMote, OpenChina y sensor MAX6675	5 horas	mié 29/03/17	mié 29/03/17
7		Programar driver para OpenMote y OpenChina con Contiki 3.x	15 horas	mié 29/03/17	vie 31/03/17
8		Verificar su correcto funcionamiento mediante un programa de prueba	1 hora	lun 03/04/17	lun 03/04/17
9		**Implementar comunicaciones mesh con el protocolo RPL**	**3,75 días**	**lun 03/04/17**	**vie 07/04/17**
10		Documentarse sobre el funcionamiento de las redes Mesh y el protocolo RPL	8 horas	lun 03/04/17	mar 04/04/17
11		Documentarse sobre su uso en Contiki 3.x	10 horas	mar 04/04/17	mié 05/04/17
12		Implementación en nodos OpenMote y OpenChina con Contiki 3.x	10 horas	mié 05/04/17	jue 06/04/17
13		Verificar su correcto funcionamiento enviando la temperatura leída de un nodo a	2 horas	jue 06/04/17	vie 07/04/17
14		**Puesta en marcha de la Raspberry Pi**	**1,13 días**	**vie 07/04/17**	**lun 10/04/17**
15		Instalación de Raspbian Jessie Lite	1 hora	vie 07/04/17	vie 07/04/17
16		Configuración de Raspbian	7 horas	vie 07/04/17	lun 10/04/17
17		Verificar conectividad IPv6	1 hora	lun 10/04/17	lun 10/04/17
18		**Puesta en marcha del Border Router 6LBR**	**8,13 días**	**lun 10/04/17**	**mié 26/04/17**
19		Documentarse sobre el funcionamiento de Cetic 6LBR	25 horas	lun 10/04/17	mié 19/04/17
20		Descargar los fuentes y compilarlos para Oper	10 horas	mié 19/04/17	jue 20/04/17
21		Programar un OpenMote junto a su OpenBase con los binarios generados	1 hora	jue 20/04/17	jue 20/04/17
22		Configurar Border Router 6LBR y Raspberry Pi para tener conectividad IPv6	15 horas	jue 20/04/17	lun 24/04/17
23		Programar otros nodos como clientes del Border Router	11 horas	lun 24/04/17	mié 26/04/17
24		Realizar pruebas de cnectividad de la red Ethernet y la red WSN mediante el envío de	3 horas	mié 26/04/17	mié 26/04/17
25		**Determinar que protocolo se adapta mejor para enviar datos desde la red de sensores al nodo sumidero**	**2,5 días**	**mié 26/04/17**	**mar 02/05/17**
26		Documentarse sobre el protocolo CoAP	5 horas	mié 26/04/17	jue 27/04/17
27		Documentarse sobre el protocolo MQTT	5 horas	jue 27/04/17	vie 28/04/17
28		Documentarse sobre el uso de Web Sockets	5 horas	vie 28/04/17	mar 02/05/17
29		Decidir que protocolo se adapta mejor a las necesidades del proyecto	5 horas	mar 02/05/17	mar 02/05/17
30		**Utilizar el protocolo CoAP para enviar datos desde la red de sensores al nodo sumidero**	**11,25 días**	**mié 03/05/17**	**vie 19/05/17**
31		Realizar pruebas con el protocolo CoAP enviando las mediciones de los nodos cliente	20 horas	mié 03/05/17	vie 05/05/17
32		Realizar pruebas con el protocolo CoAP enviando ajustes de la Raspberry Pi hacia un nodo y hacia varios nodos	20 horas	vie 05/05/17	mié 10/05/17
33		Implementar Cliente y Servidor en los nodos c	20 horas	mié 10/05/17	lun 15/05/17
34		Implementar Cliente y Servidor en el nodo sumidero	28 horas	lun 15/05/17	vie 19/05/17
35		Verificar su correcto funcionamiento	2 horas	vie 19/05/17	vie 19/05/17
36		**Implementar microservicio que permita enviar notificaciones de los nodos por SMS**	**6,25 días**	**vie 19/05/17**	**mar 30/05/17**
37		Documentarse sobre los comandos AT GSM	2 horas	vie 19/05/17	lun 22/05/17
38		Puesta en marcha del GSM Addon Raspberry Pi y primeras pruebas	5 horas	lun 22/05/17	lun 22/05/17
39		Implementar Microservicio alertas SMS	30 horas	lun 22/05/17	lun 29/05/17
40		Implementar conectividad al Microservicio en Servidor CoAP del nodo sumidero	10 horas	lun 29/05/17	mar 30/05/17
41		Verificar su correcto funcionamiento	3 horas	mar 30/05/17	mar 30/05/17
42		Entrega del proyecto	0 días	mar 30/05/17	mar 30/05/17

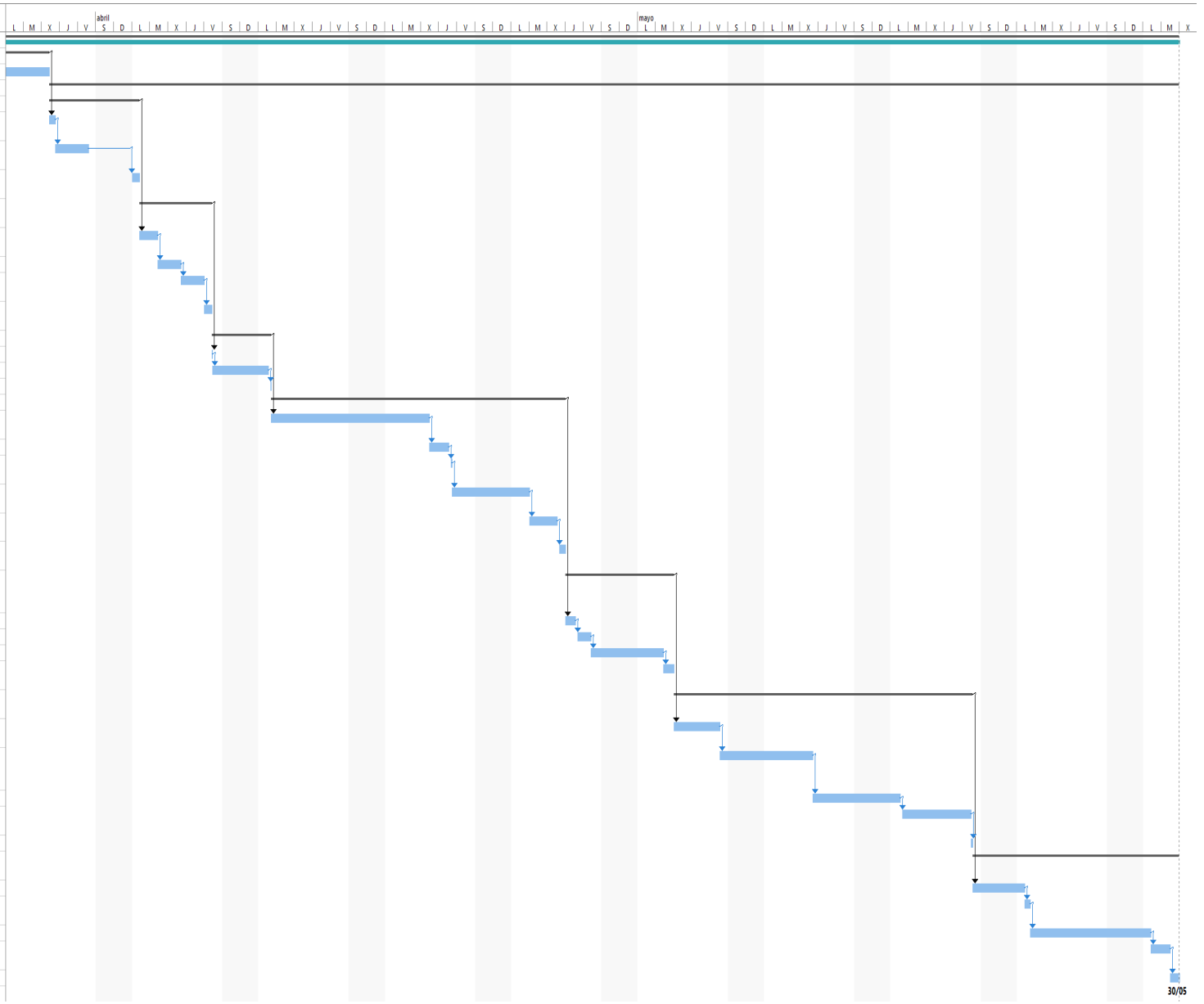


Figura 3.14: Diagrama de Gantt de la planificación definitiva del proyecto

3.2.3 Diferencias entre la planificación temporal inicial y definitiva

Uno de los cambios realizados a la planificación temporal inicial es la sustitución de la tarea "*2.7 – Página web para gestionar los nodos y visualizar la información*" de la planificación inicial por la tarea "*2.7 – Implementar microservicio que permite enviar notificaciones de los nodos por SMS*" de la planificación definitiva.

El motivo de este cambio ha sido que Thermesys adquirió un módem 3G apilable al nodo sumidero y a todos nos pareció mucho más interesante implementar un microservicio para las alertas por SMS que la página web ya que no daba tiempo para realizar ambas tareas.

Cuando se adquirió el módem 3G ya se había implementado el cliente/servidor CoAP de los nodos por lo que al utilizar el cliente CoAP Copper para Firefox, la funcionalidad del sistema para establecer períodos y alertas a los nodos ya estaba implementada sin ser necesaria la tarea de la página web para su implementación. También la funcionalidad de listar los nodos y ver sus IPv6 especificadas en esa tarea estaba implementada en el Border Router.

Otro cambio realizado ha sido especificar en la planificación definitiva el protocolo CoAP en la tarea "2.6 – Utilizar el protocolo elegido para enviar datos desde los sensores al nodo sumidero".

Después de confirmar los cambios comentados, el esquema del apartado "*3.1.1 – Requisitos del proyecto*" queda de la siguiente forma:

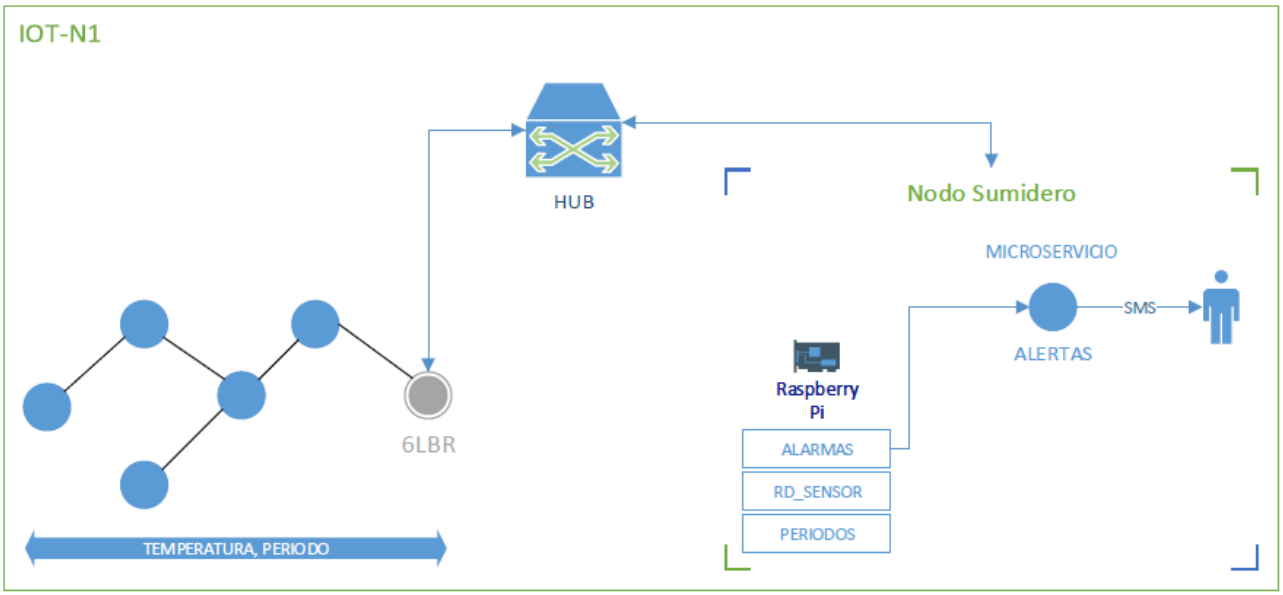


Figura 3.15: Esquema arquitectura planificación definitiva

Como se puede observar en la figura anterior el microservicio de alertas por SMS será implementado en el nodo sumidero.

3.3 Estimación de recursos y costes del proyecto

Los recursos necesarios para desarrollar el proyecto incluyen las herramientas de trabajo (H) y los componentes (C) necesarios para implementar el sistema, los cuales han sido desglosados en la siguiente tabla:

Tipo	Recurso	Unidades	Coste	Total
H	Estación de trabajo	1	-	-
H	Software desarrollo	-	-	-
H	J-Link + Olimex	1	60,36€	60,36€
H	Osciloscopio SIGLENT SDS 1102CNL	1	-	-
C	OpenMote y OpenBase	3	200€	600€
C	OpenChina y RHBDK1.1	5	26€	130€
C	Sensor MAX6675	10	3,05€	30,50€
C	Raspberry Pi 1 B+	1	36,80€	36,80€
C	GSM Add-on v2.0	1	34,95€	34,95€

Tabla 3.8: Desglose de costes del proyecto

El coste de los recursos materiales empleados suma un total de 892,61€.

El coste estimado de las horas dedicadas al desarrollo del proyecto por parte de un programador junior durante unos 2 meses y medio a razón de 35 horas semanales (~7 horas diarias), es decir 300 horas a 7,7€/hora[17] es de 2310€.

Una vez definidos todos lo recursos necesarios es posible estimar el coste del proyecto. Si al coste de los recursos materiales se les suman las horas invertidas en el desarrollo, el coste total del proyecto asciende a un total de 3202,61€.

3.4 Seguimiento del proyecto

Todas las fases del proyecto han sido supervisadas por la empresa y el tutor mediante informes quincenales en los que se han descrito las tareas realizadas durante cada quincena así como los objetivos a cumplir en la siguiente quincena.

Los problemas encontrados durante el desarrollo del proyecto están explicados en el Capítulo 6 junto a una descripción de las acciones llevadas a cabo hasta encontrar una solución al problema.

17 Monto calculado partiendo de la base de que a fecha del 19 de julio de 2017 la media de un sueldo de programador junior, sin tener en cuenta especializaciones, es de unos 17.219€ al año, haciendo un total de 1230€ al mes o 7,7€/ hora para una jornada completa de 8 horas.

Capítulo 4

Análisis del sistema

En este capítulo se realiza un análisis de la red de monitorización del entorno describiendo los requisitos del sistema y los casos de uso, y cómo se combinan entre sí para proporcionar la funcionalidad requerida.

4.1 Introducción

El proyecto llevado a cabo ha sido propuesto por la empresa Thermesys que quiere mejorar su sistema de monitorización actual añadiéndole más funcionalidades y ver si es factible llevar a cabo todas la tareas del proyecto utilizando el hardware descrito en el capítulo 2 en la red de sensores.

Los requisitos se pueden clasificar en tres tipos:

- Requisitos técnicos: se trata de requisitos no funcionales que recogen aspectos básicos de la operatividad del sistema.
- Requisitos funcionales: expresan las funciones de software y de gestión de los datos del sistema y sus componentes.
- Requisitos de datos: recogen aquellas cuestiones relacionadas con los datos que recopilará y almacenará el sistema.

4.2 Requisitos técnicos

La red de sensores debe poder desplegarse de forma autónoma una vez los nodos sean programados, deben poder comunicarse a grandes distancias de forma inalámbrica y los nodos cliente deben ser capaces se encontrar rutas alternativas en caso de perder conectividad con el Border Router sin intervención externa.

El Border Router debe ser capaz de comunicarse con el nodo sumidero para reenviar los datos de la red de sensores. Los nodos cliente deben recopilar y enviar periódicamente datos al nodo sumidero, ser capaces de recibir nuevos ajustes y atender peticiones de datos concretos, por lo tanto la comunicación de la red debe ser bidireccional permitiendo que un equipo ajeno a la red de sensores pero que pertenezca a la red del nodo sumidero pueda comunicarse con los nodos. El nodo sumidero debe ser capaz de identificar alarmas en los mensajes periódicos de los nodos cliente y enviar un mensaje SMS.

4.3 Requisitos Funcionales

En primer lugar se describen los requisitos funcionales de la infraestructura de red:

- **Solicitar acceso a la red:** los nodos deben ser capaces de conectarse al Border Router automáticamente al inicializarse.
- **Enviar datos al solicitante:** los nodos deben atender las peticiones de datos que reciban, recopilar los datos y responder al solicitante.
- **Recibir parámetros de configuración:** los nodos deben ser capaces de recibir parámetros de configuración y aplicarlos.
- **Enviar datos al nodo sumidero:** los nodos deben recopilar datos del sensor MAX6675 y enviarlos al nodo sumidero.
- **Recibir datos de los nodos:** el nodo sumidero debe ser capaz de recibir datos de los nodos y mostrarlos.
- **Recibir alertas de los nodos:** el nodo sumidero debe ser capaz de recibir alertas de los nodos y notificar mediante el envío de SMS.
- **Enrutar los mensajes:** el sistema debe ser capaz de enrutar mensajes, desde un nodo al sumidero, de un elemento cualquiera de la red del nodo sumidero a un nodo y viceversa.

En primer lugar se describen los requisitos funcionales de la gestión de los nodos cliente:

- **Consultar datos:** el usuario deberá se capaz de consultar datos directamente a los nodos cliente utilizando un simple navegador web.
- **Administrar nodos:** el usuario deberá se capaz de consultar y establecer ajustes directamente a los nodos cliente utilizando un simple navegador web.

4.3.1 Diagrama de casos de uso

De los anteriores requisitos funcionales se deriva el conjunto de casos de uso de la siguiente figura.

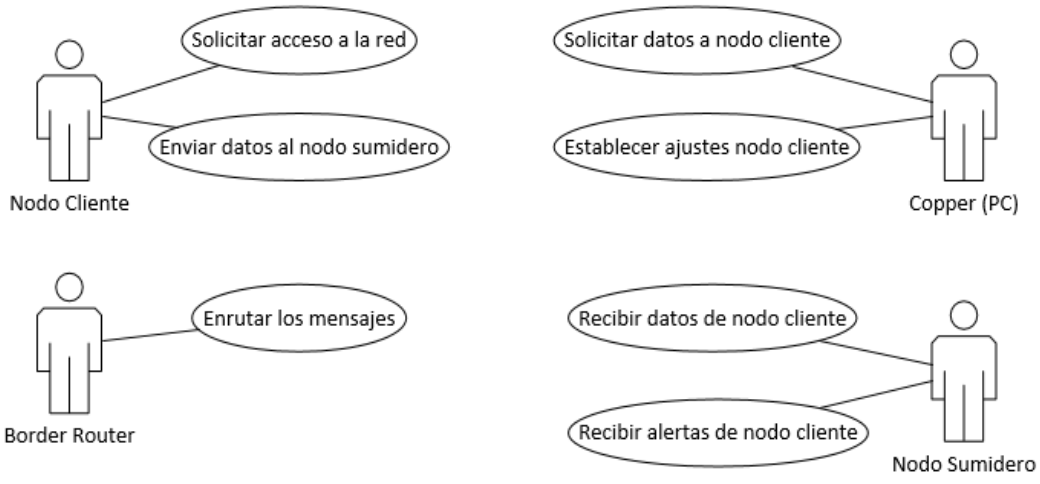


Figura 4.16: Diagrama de casos de uso de los requisitos funcionales

A continuación, se procede a describir los casos de uso para los requisitos funcionales de la infraestructura de red:

Solicitar acceso a la red

Origen de la acción:
Nodo cliente
Destino de la acción:
Nodos vecinos / Border Router
Descripción:
El nodo solicita unirse a la red WSN para transmitir datos
Escenario:
- El nodo solicita al Border Router los datos de configuración.
- El Border Router responde al nodo con los datos necesarios para poder conectarse a la red WSN.
- El nodo se auto-configura y pasa a formar parte de la red.

Enviar datos al solicitante

Origen de la acción:
Nodo cliente
Destino de la acción:
Estación de trabajo (red Ethernet)
Descripción:
El nodo recibe la petición de algún dato
Escenario:
- Un equipo que pertenece a la red del nodo sumidero establece conexión con el servidor CoAP del nodo.
- El equipo solicita (GET) de alguno de los recursos disponibles por el nodo.
- El nodo recibe esa petición y responde al equipo solicitante con el dato.

Recibir parámetros de configuración

Origen de la acción:
Nodo cliente
Destino de la acción:
Estación de trabajo (red Ethernet)
Descripción:
El nodo recibe un nuevo ajuste para alguno de sus parámetros configurables.
Escenario:
- Un equipo que pertenece a la red del nodo sumidero establece conexión con el servidor CoAP del nodo.
- El equipo ajusta (POST) alguno de los parámetros configurablespor el nodo.
- El nodo recibe el nuevo ajuste y lo aplica.

Enviar datos al nodo sumidero

Origen de la acción:
Nodo cliente
Destino de la acción:
Nodo sumidero
Descripción:
El nodo transmite los datos recopilados cuando se cumple su periodo al nodo sumidero.
Escenario:
- El temporizador del nodo para enviar datos al nodo sumidero llega a 0.
- El nodo recopila los datos de los sensores y los envía al nodo sumidero.
- El nodo reinicia su temporizador.

Recibir datos de los nodos

Origen de la acción:
Nodo sumidero
Destino de la acción:
Nodo sumidero
Descripción:
El nodo sumidero recibe los datos recopilados por los nodos y los muestra por consola.
Escenario:
- El nodo sumidero recibe mensajes de los nodos cliente.
- El nodo sumidero los decodifica y los muestra por consola.

Recibir alertas de los nodos

Origen de la acción:
Nodo sumidero
Destino de la acción:
Nodo sumidero
Descripción:
El nodo sumidero recibe los datos recopilados por los nodos y los muestra por consola.
Escenario:
- El nodo sumidero recibe mensajes de los nodos cliente.
- El nodo sumidero los decodifica y los muestra por consola.
- El nodo sumidero decodifica que uno de los mensajes recibidos es una alarma y reenvía el mensaje al microservicio de SMS
- El microservicio de SMS construye un SMS con el contenido de la alarma y lo envía.

Enrutar los mensajes

Origen de la acción:
Red WSN / Red Ethernet
Destino de la acción:
Red Ethernet / Red WSN
Descripción:
El Border Router se encarga de enrutar los paquetes recibidos entre sus dos interfaces de red.
Escenario:
- Un nodo de la red de sensores envía un paquete con destino bbbb::
- El Border Router envía ese paquete por su interfaz Ethernet

- Un equipo/nodo de la red Ethernet envía un paquete con destino aaaa::
- El Border Router envía ese paquete por su interfaz WSN

4.4 Requisitos de datos

Los datos recopilados por los nodos cliente deben tener formato estructurado que permita la diferenciación de cada nodo y dato recopilado. En la siguiente tabla se muestra la estructura definida para cada mensaje:

Estructura: Mensaje	
Datos recogidos	• Dirección MAC nodo origen como identificador • Identificador dato recogido • Dato recogido

Tabla 4.9: Estructura mensaje nodo cliente

Capítulo 5

Diseño de la arquitectura del sistema

En este capítulo se describe el diseño de la arquitectura del sistema, que se divide en:

- Diseño de la arquitectura de red.
- Diseño de la arquitectura lógica del Border Router, nodos cliente y nodo sumidero.

5.1 Diseño de la arquitectura de red

En la siguiente figura se puede observar como se ha diseñado la red del sistema de monitorización y cómo se comunican entre sí los diferentes elementos de la red.

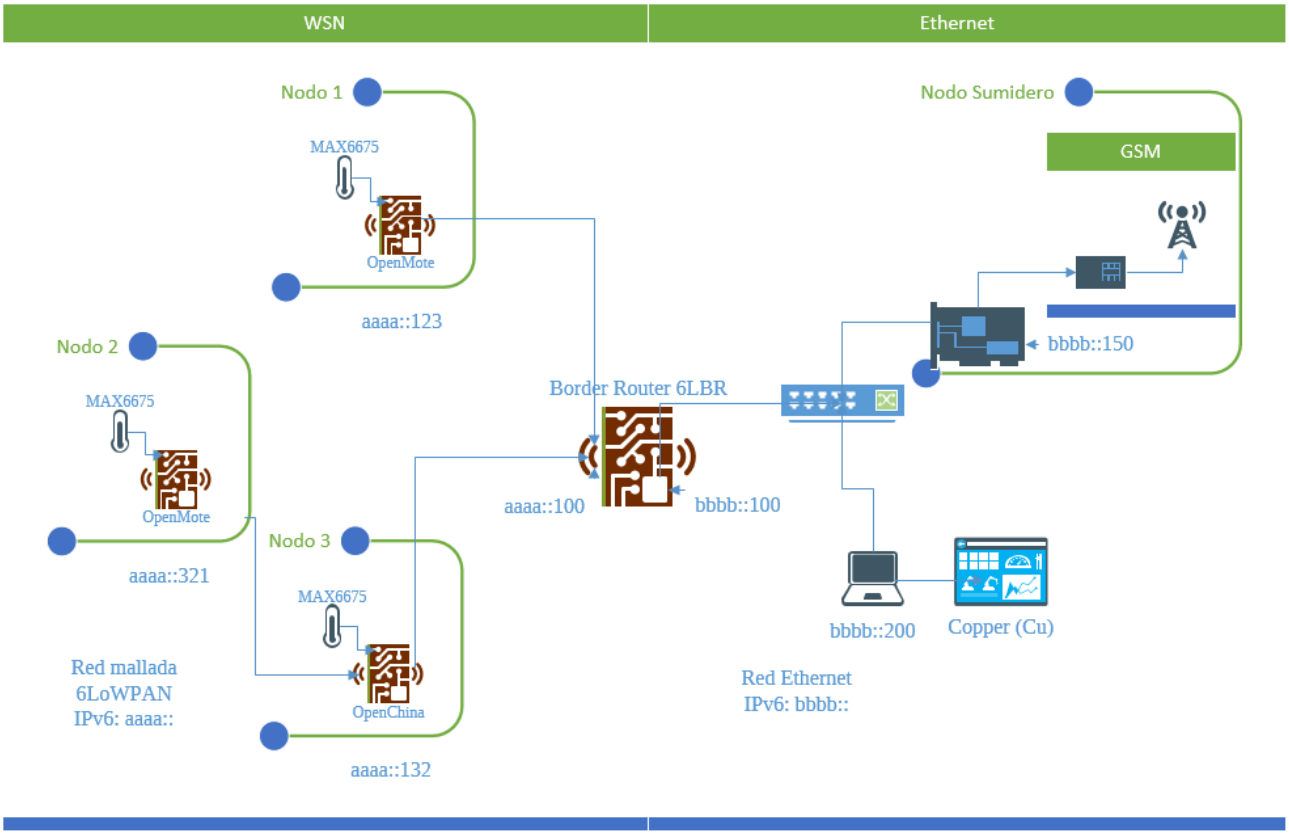


Figura 5.17: Diseño de la arquitectura de red

Los nodos cliente (lado izquierdo) se comunican con el nodo sumidero (lado derecho) a través del Border Router (centro) y viceversa.

5.1.1 Topología de red

Uno de los objetivos de este proyecto es implementar una red tipo malla para mejorar la red existente implementada en una red tipo estrella. El motivo principal por implementar una red tipo malla es que a diferencia de una red tipo estrella (implementación del sistema comercial actual), no existe la limitación de distancias debida a que todos los nodos cliente de la red deben tener el nodo central a su alcance directo (visibilidad directa) ya que el sistema funciona con conexiones punto a punto.

Es una gran ventaja disponer de una red mallada ya que si existe algún punto ciego en la distribución de los sensores, se puede utilizar un nodo cliente como salto para alcanzar al nodo central. Como se puede observar en el diagrama del apartado anterior, el diseño de la arquitectura de la red se va a dividir en dos redes que se van a unir utilizando un Border Router. La red de la izquierda es la WSN todos los nodos cliente que pertenezcan a esta red tendrán el prefijo IPv6 aaaa::, el resto de dirección IPv6 se genera de forma automática cuando el nodo se conecta a la red puesto que están configurados en modo auto-configuración (AUTOCONF).

"El proceso de auto-configuración consiste en lo siguiente: creación de una dirección de enlace local (link-local) y comprobar su unicidad en el enlace, determinar qué tipo de información se debe auto-configurar (direcciones, otros datos o los dos) y, en el caso de direcciones, determinar qué mecanismo se debe usar para obtenerlas (con control de estado, sin él o con los dos). La obtención automática de configuración sólo se aplica a hosts y nunca a routers (si bien éstos pueden generar su propia dirección de enlace local), ya que los hosts necesitan información anunciada por éstos y alguien tendrá que configurarlos."[2]

La red Ethernet tiene el prefijo IPv6 bbbb:: y el nodo sumidero tiene la IPv6 estática bbbb::150. El Border Router tiene ambas interfaces de red configuradas de forma estática, en su interfaz de red WSN tiene la IPv6 aaaa::100 y en su interfaz Ethernet la IPv6 bbbb::100.

En el esquema podemos observar una de las ventajas de utilizar una red mallada, podemos ver como el Nodo 2 no está directamente conectado al Border Router si no que está utilizando al Nodo 3 como salto para alcanzarlo.

El diseño de esta arquitectura de red permite la comunicación bidireccional de ambas redes de forma transparente al usuario.

5.1.2 Direccionamiento y enrutado

Para el direccionamiento y el enrutado de los paquetes se ha utilizado el protocolo RPL sobre la red de sensores 6LoWPAN. Los nodos se comunican con el nodo sumidero, utilizando el Border Router como puente.

Si un nodo no tiene visibilidad directa con el Border Router, enviará su mensaje utilizando alguno de sus vecinos que si tenga visibilidad directa o indirecta con el Border Router como salto o saltos hasta alcanzarlo.

5.2 Diseño de la arquitectura lógica

En este apartado se va a explicar la arquitectura lógica de:

- Border Router.
- Nodo Cliente.
- Nodo sumidero.
- Formato de los mensajes entre Nodos cliente y Nodo sumidero.
- Mensajes periódicos de los Nodos cliente al Nodo sumidero.
- Comportamiento del Nodo sumidero al recibir una alarma de un Nodo cliente.

5.2.1 Border Router

En la siguiente figura se puede observar como el Border Router dispone de dos interfaces de red, una Ethernet que conecta con la red bbbb:: del nodo sumidero y otra 6LoWPAN que conecta con la red aaaa:: de los nodos cliente.

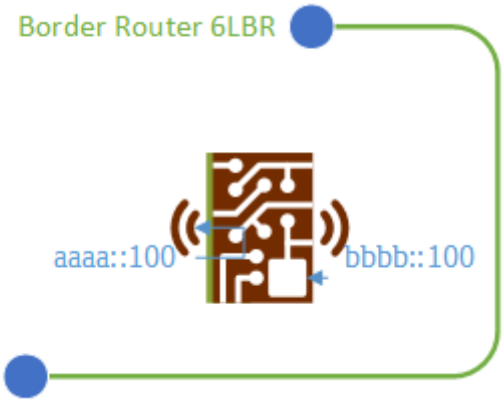


Figura 5.18: Interfaces de red Border Router

La tarea del Border Router es la de interconectar dos topologías de red distintas y permitir la comunicación bidireccional entre las redes. En el caso de este proyecto se ha utilizado el Border Router implementado por Cetic, éste dispone de varios modos de funcionamiento para el desarrollo de este proyecto se ha utilizado el modo "Router mode[18]".

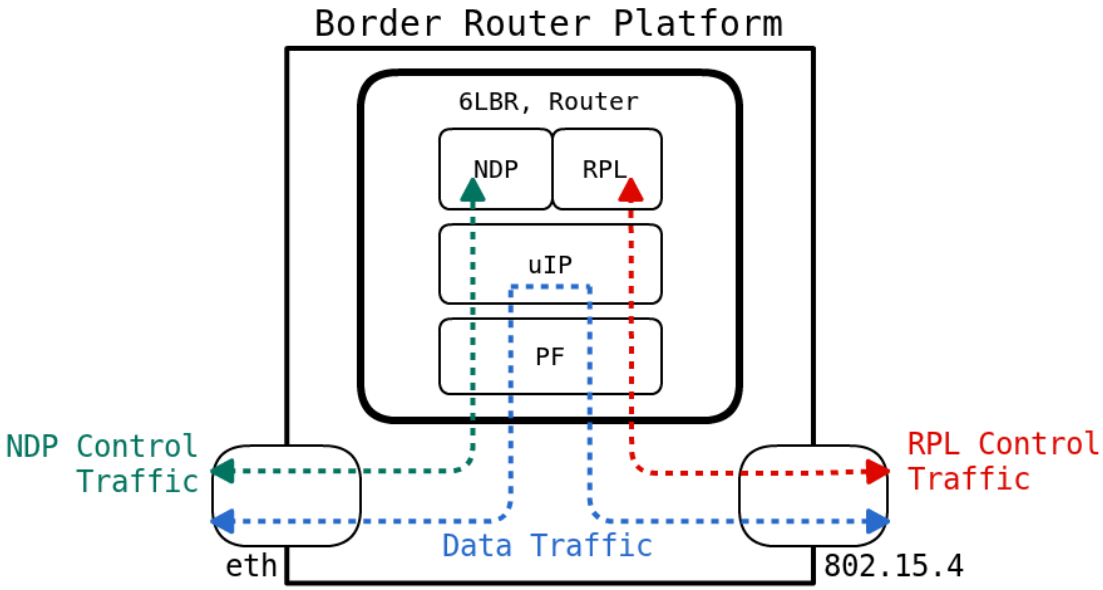


Figura 5.19: Funcionamiento Border Router modo "Router mode" [La imagen pertenece a la wiki de Cetic en github]

Este modo de funcionamiento permite a un OpenMote junto a su OpenBase a funcionar como un completo router IPv6 que interconecta dos subredes IPv6. La red de sensores 802.15.4 con el prefijo aaaa:: y la red Ethernet donde está ubicado el nodo sumidero con el prefijo bbbb::. Como se puede observar en la figura anterior, el Border Router es capaz de enrutar los paquetes de origen aaaa:: hacia el destino bbbb:: y viceversa.

18 Enlace a los modos de funcionamiento del Border Router: https://github.com/cetic/6lbr/wiki/6LBR-Modes

5.2.2 Nodo Cliente

En la siguiente figura se puede observar la estructura lógica de los nodos cliente, mostrando los servicios implementados en ellos.

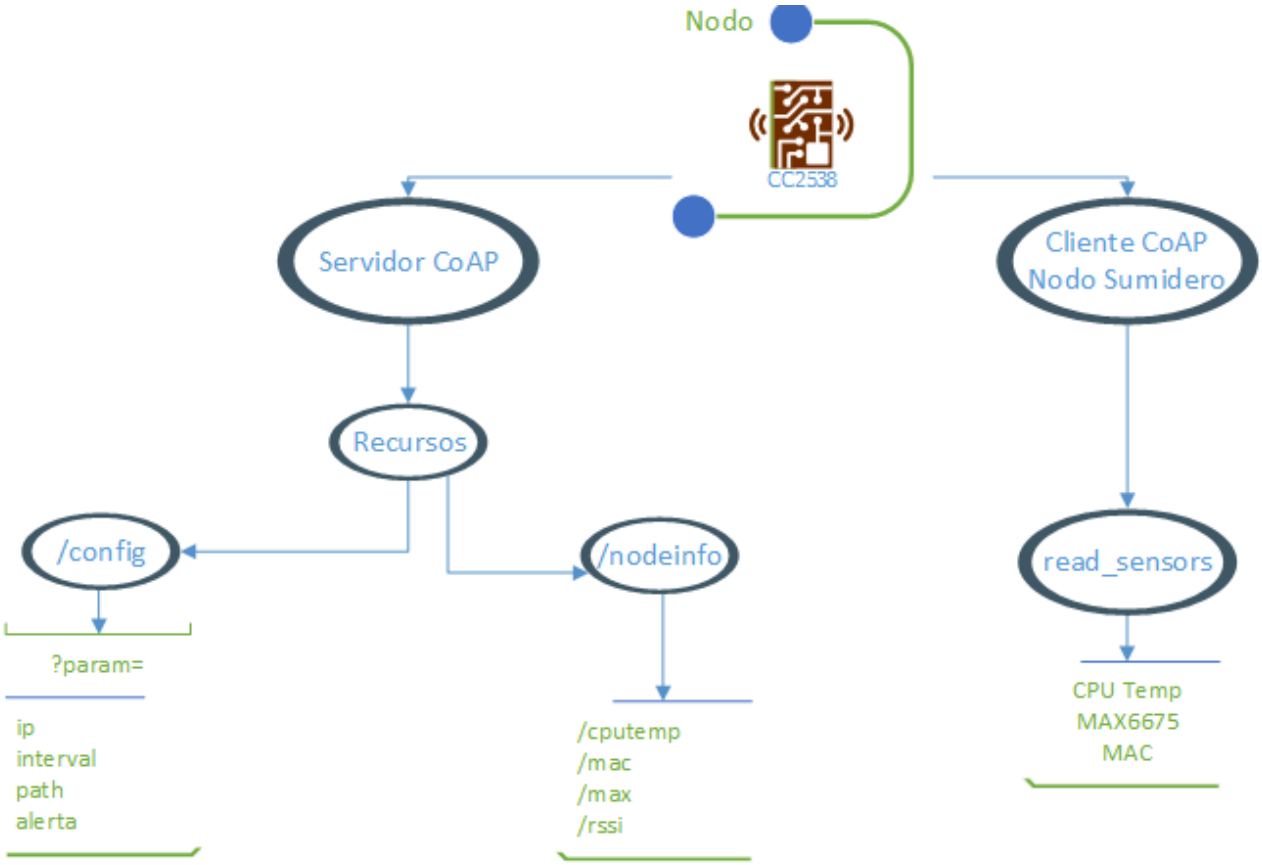


Figura 5.20: Servicios y recursos implementados en Nodo Cliente

Un nodo cliente es el encargado de leer la temperatura del sensor MAX6675 y enviarla al nodo sumidero cada vez que se cumple su intervalo de envíos. También implementa un servidor CoAP que permite establecer ajustes y tomar lecturas, proporcionando de esta forma conectividad bidireccional.

Un Nodo cliente es cliente CoAP del servidor CoAP del nodo sumidero para el envío de datos periódicos lanzando el proceso **read_sensor** cada vez que se cumple el período de envío configurado. Este proceso es el encargado de recopilar los datos del nodo, generar el mensaje a enviar en formato JSON y enviar ese dato (POST) al recurso **/sink** del nodo sumidero.

Para acceder al servidor CoAP utilizando el cliente Copper debemos abrir el navegador Firefox y acceder a: **coap://[IPv6_nodo_cliente]:5683**

Un Nodo cliente es servidor CoAP para permitir consultar (GET) los recursos disponibles en **/nodeinfo** desde un cliente CoAP:

- **/cputemp**: consultar la temperatura de la CPU del nodo.
- **/mac**: Consultar la dirección MAC del nodo
- **/max**: Consultar la temperatura del sensor MAX6675
- **/rssi**: Consultar los dbm del Nodo.

Y para permitir la consulta (GET) y ajuste (POST) de los parámetros de configuración de los nodos en **/config?param=**:

- **ip**: parámetro que especifica la dirección IPv6 del nodo sumidero.
- **interval**: parámetro que especifica el tiempo en segundos entre notificaciones al nodo sumidero.
- **path**: parámetro para establecer el recurso remoto al que enviar los datos cada vez que se cumpla el periodo definido.
- **alerta**: parámetro para establecer la temperatura máxima a la que enviar alarmas al nodo sumidero.

5.2.3 Formato de los mensajes

Los mensajes de los nodos a el nodo sumidero, están codificados en JSON y encapsulados en CoAP dando como resultado la siguiente estructura de mensaje:

- Dirección MAC del nodo.
- Dato o Datos.

Para identificar a los nodos, cada mensaje contiene la dirección MAC del nodo que envía el mensaje y después el dato o los datos que forman el resto del mensaje.

Un ejemplo de mensaje JSON es:

{"MAC":"4B:00:04:32:9A:C0","alertaMAX":"75"}

En este caso, el nodo cliente se identifica por la dirección MAC y alertaMAX indica que es una temperatura que supera el máximo configurado en el nodo. En el punto "*5.2.6 – Alarma Nodo Cliente a Nodo sumidero*" se explica que acciones desencadena un mensaje codificado en JSON que contenga la etiqueta alertaMAX.

5.2.4 Nodo Sumidero

El nodo sumidero es el encargado de recibir los datos que son recopilados por los nodos cliente, decodificarlos y en el caso de que alguno de ellos sea una alarma, iniciar los procedimientos necesarios para realizar un envío de SMS utilizando el módem conectado a él.

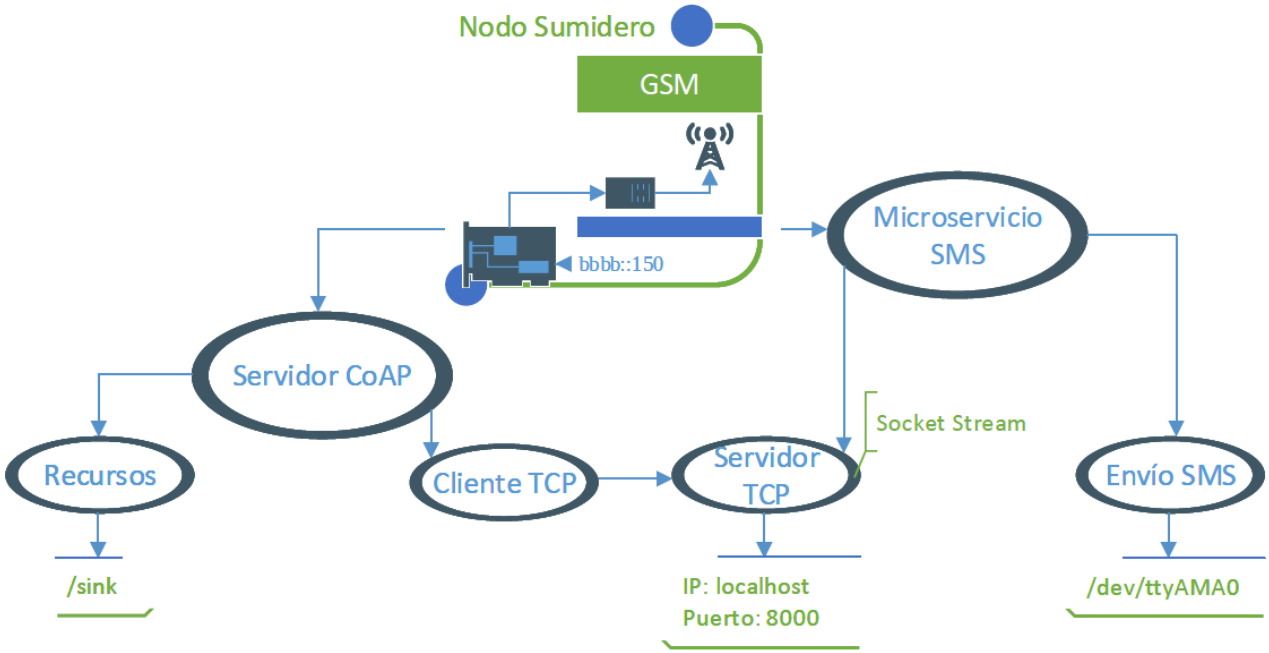


Figura 5.21: Servicios y recursos implementados en Nodo Sumidero

Como se puede observar en la figura anterior, el nodo sumidero es una Raspberry Pi 1 model B+ con un módem 2g/3g conectado a sus pines GPIO. El nodo sumidero dispone de los siguientes servicios en funcionamiento:

- **Servidor CoAP**: encargado de recibir los mensajes periódico de los nodos cliente en su recurso **/sink** y mostrar los datos recibidos por pantalla. Si uno de esos mensajes es una **alarma** etiquetada como “alertaMAX” se conectará con el Microservicio de SMS mediante un socket stream y le reenviará el mensaje JSON de la alarma.

- **Microservicio SMS**: servidor TCP que escucha conexiones por el puerto 8000, si se establece una conexión con él, se inicia el proceso de generación del SMS, configuración del módem y envío del SMS. Una vez termina el proceso de envío de SMS, el servidor TCP vuelve a estar disponible para atender más alarmas enviadas por el servidor CoAP de otros nodos.

5.2.5 Mensaje Nodo Cliente a Nodo Sumidero

Cada vez que se cumple el período de tiempo establecido para notificar al nodo sumidero, el nodo cliente lee los datos de los sensores, genera el mensaje codificado en JSON y lo envía al servidor CoAP del nodo sumidero.

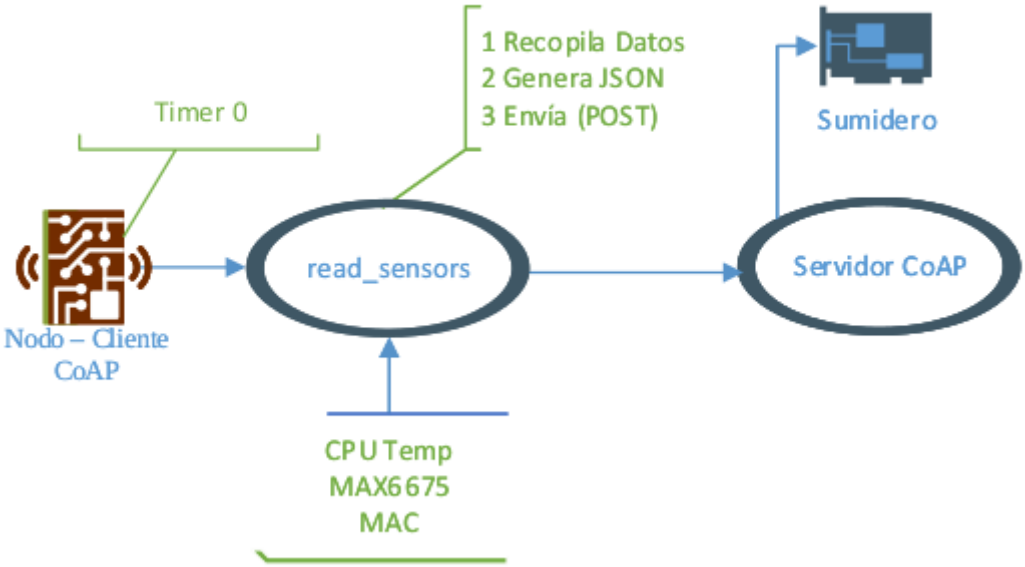


Figura 5.22: Esquema funcionamiento envío periódico de mensajes a nodo sumidero

Como se puede observar en la figura anterior, cuando un periodo de tiempo finaliza en un nodo cliente, en este caso un timer, se lanza un proceso llamado **read_sensors** encargado de:

1. Recopilar los datos del nodo.
2. Codificarlos en formato JSON.
3. Enviar el mensaje JSON al nodo sumidero (POST) al recurso **/sink** de este.

Una vez enviado el mensaje al nodo sumidero, el timer es reiniciado al tiempo en segundos configurado en el parámetro **interval** del nodo.

Cuando el nodo sumidero recibe el mensaje del nodo, el servidor CoAP es el encargado de decodificar el mensaje JSON recibido y de mostrarlo por pantalla. Si el mensaje contiene la etiqueta “alertaMAX” se iniciará el procedimiento para el envío de un mensaje SMS. En el siguiente punto se explica el caso de la generación de una alarma.

5.2.6 Alarma Nodo Cliente a Nodo Sumidero

Una alarma en el sistema que se va desarrollar es una temperatura superior a la temperatura máxima definida en el parámetro **alerta** de un nodo cliente. Si cuando se lee la temperatura del sensor MAX6675 para realizar el envío periódico de mensajes es superior a la definida, y se genera un mensaje en formato JSON.

Cuando un mensaje recibido de un nodo cliente es decodificado como una alarma (contiene la etiqueta “alertaMAX”), aparte de mostrarse por pantalla, se desencadenan varios procedimientos que dan como resultado el envío de un mensaje SMS que contiene la alarma recibida. En la siguiente figura se describen los procedimientos que se desencadenan

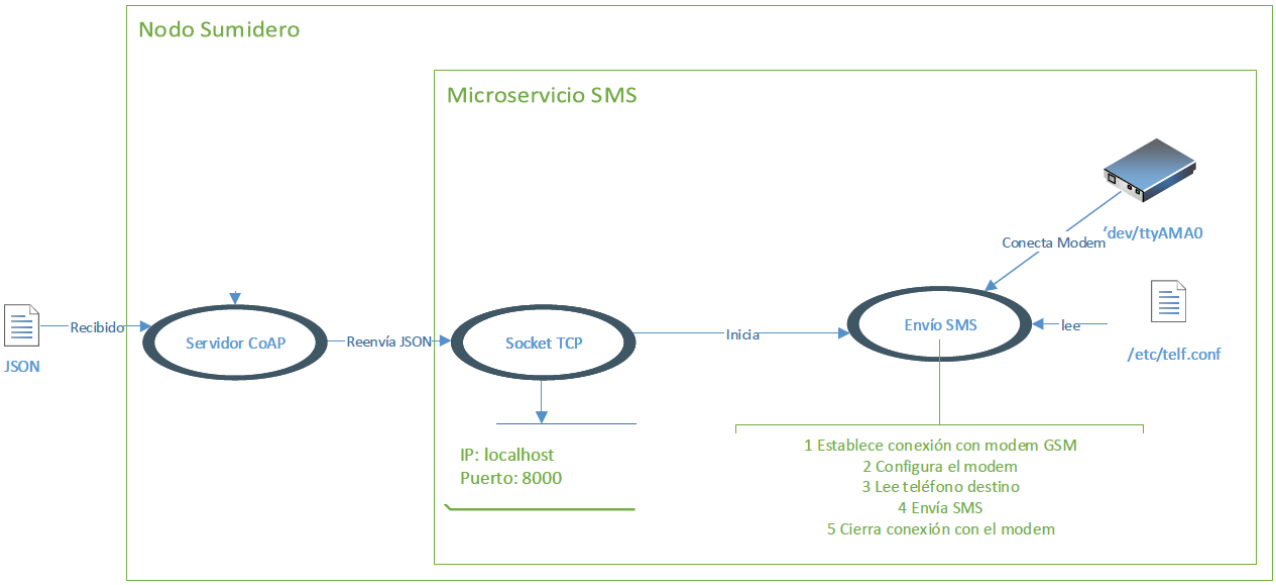


Figura 5.23: Esquema funcionamiento envío de alarma por SMS

El procedimiento paso a paso cuando se recibe un mensaje JSON que contiene la etiqueta alertaMAX es el siguiente:

Servidor CoAP:

1. Decodifica el mensaje JSON recibido como “alertaMAX”.
2. Establece una conexión TCP con el Microservicio SMS, localhost:8000.
3. Reenvía el mensaje JSON recibido al Microservicio SMS.

Microservicio SMS:

4. Decodifica el mensaje JSON enviado por el servidor CoAP.
5. Genera el contenido del SMS con la siguiente estructura:
   “ALERTA
   NODO: 4B:00:06:0D:9F:18
   Temperatura alcanzada: 70 grados centígrados”
6. Inicia el proceso de envío de SMS.

Envío SMS:

7. Establece conexión con módem GSM por puerto serie en /dev/ttyAMA0.
8. Configura el módem utilizando comandos AT.
9. Lee teléfono destino en **/etc/telf.conf**
   1. El contenido de este fichero es el siguiente:
      TELF=600000000
10. Envío SMS:
    1. Establece el destinatario del SMS.
    2. Añade el contenido del SMS generado en el Microservicio.
    3. Envía el SMS.
11. Cierra conexión con el módem.

Capítulo 6

Implementación y pruebas

En este capítulo se describe la implementación de los diferentes componentes del sistema detallados en el capítulo anterior y de las pruebas realizadas para comprobar el correcto funcionamiento de los componentes.

6.1 Detalles de la implementación

En este apartado se va detallar la implementación del driver MAX6675 para Contiki, la implementación del servidor/cliente CoAP para los nodos cliente, el servidor CoAP y microservicio del nodo sumidero. Durante la realización de las distintas implementaciones se han encontrado problemas, en los apartados donde han ocurrido, se detallan los pasos realizados para encontrar una solución.

La plataforma hardware OpenChina ha sido soportada gracias a que su base es el microcontrolador CC2538, se ha utilizado como base la plataforma OpenMote y ha sido adaptada para añadirle soporte en Contiki (Ver *Anexo Nodos III: Añadir nuevo platform a Contiki*).

6.1.1 Nodos Cliente: Implementación driver MAX6675 para Contiki OS

Para la implementación del driver MAX6675 para nodos con el microcontrolador CC2538, se partió del driver que Thermesys tenía diseñado para JenniSense. La funciones y macros utilizadas por ese driver han sido reescritas en Contiki 3.x y algunas de ellas reubicadas en otros ficheros fuente.

Para implementar el driver en Contiki 3.x, se ha consultado el código fuente de Contiki (Ver *Anexo Nodos II: Estructura de directorios Contiki*):

- En **contiki/cpu/cc2538/spi-arch.h** están definidas la nuevas funciones y macros para el protocolo SPI.

- En la versión de JenniSense se utiliza el puerto SPI SSI0, pero a partir de la versión Contiki 3.0 el puerto SPI SSI0 se utiliza en los OpenMote para el puerto Ethernet de la OpenBase. Se ha comparado el fichero **contiki/platform/openmote-cc2538/board.h** de ambas versiones.

La tareas llevadas a cabo para adaptar el driver a Contiki 3.x fueron las siguientes:

- Adaptar el código a las nuevas funciones y macros.
- Eliminar includes innecesarios por la reubicación de las funciones y macros en Contiki 3.x.
- Implementar macro “SPIX_READ()” que permite leer la temperatura del MAX6675.
- Utilizar el puerto SPI SSI1.
- Modificar el fichero “board.h” para establecer los pines SPI SSI1 ya que por defecto los pines SPI SSI1 están asignados a los led de la OpenMote.
- El pin CS (Chip Select) de cada MAX6675 se especifica como parámetro en la función “max6675_read()”, de esta forma se permite el uso de N sensores por nodo. Tantos como pines disponibles para asignarlos como CS en la placa.

El driver se ubica en el directorio **contiki/platform/openmote/dev** y se tiene que editar el fichero Makefile.openmote y añadir el fichero max6675.c en CONTIKI_TARGET_SOURCEFILES para poder utilizar el driver en los programas necesarios.

Problemas encontrados durante el desarrollo del driver:

Problema: Los OpenMote programados entraban en bootloop (reinicio constante)

Solución: Cambiar el puerto SPI utilizado, en Contiki 3.x el puerto SPI 0 está siendo utilizado por el integrado ENC28J60 ubicado en la OpenBase que proporciona conectividad Ethernet al OpenMote. Al pasar a utilizar el puerto SPI 1, el problema se solucionó.

Problema: No se leían los datos del MAX6675

Pasos seguidos:

- Programar un OpenMote con la versión del driver de Thermesys y Contki 2.7 para analizar su comportamiento. Comparándolo con el Datasheet del MAX6675 que indica los valores correctos e incorrectos de las lecturas.
    - Quitando el CS, el resultado leído por el MAX6675 es:
        - Temperatura: 0º C
        - State: 0x0
    - Quitando el GND, el resultado leído por el MAX6675 es:
        - Temperatura: 1023º C
        - State: 0x1 (termopar abierto)
- Analizar los pines CS, SO y CLK del MAX6675 en el OpenMote con el driver nuevo y con el anterior con un Osciloscopio SIGLENT SDS 1102CNL.
    - Pin CS, mismo funcionamiento en ambos driver. Se podía ver el flanco de bajada como en la figura 6.24.
    - Pin SO, mismo funcionamiento en ambos. Al menos eso parecía. Se podía ver la trama enviada por el MAX6675 como en la figura 6.25.
    - Pin CLK:
        - En el driver anterior se podía ver la señal del reloj.
        - En el nuevo driver no había ningún reloj.

Solución: Consultar con detalle el Datasheet del CC2538 y ver que solamente hay dos pines del microcontrolador que permiten ser utilizados como señal de reloj (CLK).

Los pines utilizables como CLK son:

- Puerto A, Pin 2 (PA2).
- Puerto C, Pin 4 (PC4).

En los OpenBase está accesible el pin PA2 llamado en la placa AD2/DIO2 y en los OpenChina está accesible el pin PC4. Cambiando el pin CLK por el accesible en el fichero project-conf.h del código que utiliza el driver, se consiguió solucionar el problema. Ver *Anexo Nodos II: Estructura de directorios Contiki* para más detalles.

El mismo driver desarrollado para el OpenMote sirve para las placas OpenChina, se deben añadir los ficheros del driver dentro de: **contiki/platform/openchina/dev**, editar el fichero Makefile.openchina y añadir el fichero max6675.c de la misma forma que para el OpenMote.

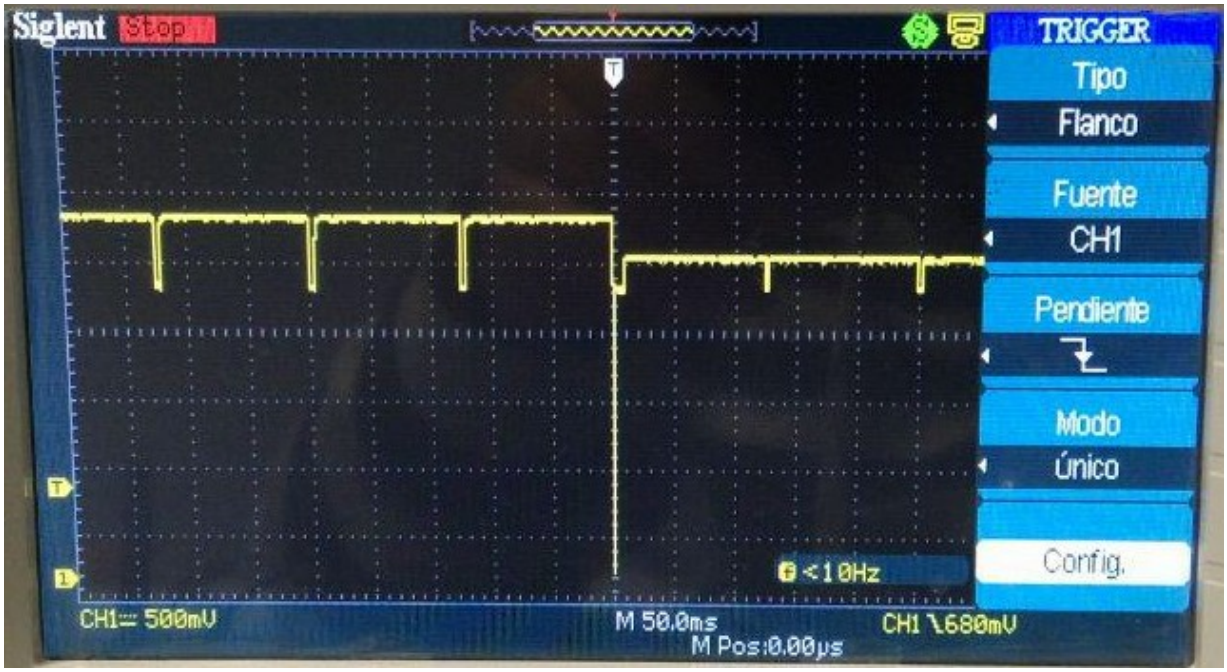


Figura 6.24: Flanco de bajada en CS al leer la temperatura del MAX6675

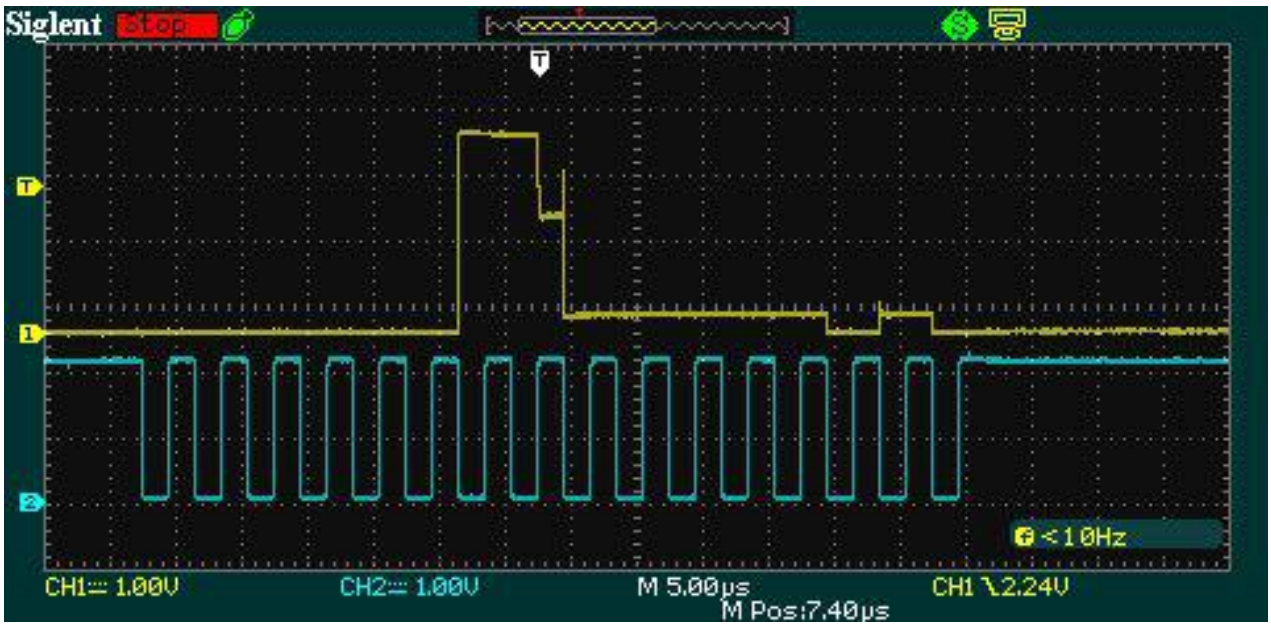


Figura 6.25: Trama lectura pin SO (azul) del MAX6675 después de un flanco de bajada en el pin CS

6.1.2 Nodos Cliente: Implementación red mesh con el protocolo RPL

En este punto se hicieron unas implementaciones de prueba utilizando el protocolo RPL. Uno de los OpenMote disponía de un sensor MAX6675 conectado y otro OpenMote solamente estaba conectado al PC por el puerto mini usb.

El PC tenía establecida una conexión al puerto serie del OpenMote sin MAX6675.

La comunicación que se realizó fue la siguiente:

- El OpenMote con el sensor de temperatura MAX6675 lee la temperatura y la envía al OpenMote conectado al PC.
- El OpenMote conectado al PC muestra por su puerto serie la temperatura tomada por el OpenMote con MAX6675.

6.1.3 Border Router: Puesta en marcha

Con el repositorio del Border Router 6LBR de Cetic[19] clonado en la branch "dev-1.4.x" se compila el ejemplo "6lbr" para generar los binarios del Border Router con sus diferentes modos de funcionamiento para el OpenMote utilizando el siguiente comando:

```
make TARGET=openmote all
```

Figura 6.26: Comando compilación Border Router

Este comando compila el Border Router de Cetic y genera un binario por cada modo de funcionamiento.

```
victor@victor-VPCEB1S8E:~/Thermesys/6lbr/examples/6lbr$ ls -l
total 2160
drwxrwxr-x  2 victor victor   4096 jun  8 19:47 6lbr
drwxrwxr-x  2 victor victor   4096 jun  8 19:47 angstrom
drwxrwxr-x  9 victor victor   4096 jun  8 19:31 apps
drwxrwxr-x  2 victor victor   4096 jun  8 19:32 bin
drwxrwxr-x  2 victor victor   4096 jun 13 11:12 bin_openchina
drwxrwxr-x  2 victor victor   4096 jul 20 20:00 bin_openmote
```

Figura 6.27: Directorios generados tras compilación

En la figura anterior se puede observar el directorio generado al finalizar la compilación que contiene los binarios de cada uno de los modos de funcionamiento del Border Router llamado "bin_openmote".

```
victor@victor-VPCEB1S8E:~/Thermesys/6lbr/examples/6lbr$ ls -l bin_openmote/
total 3072
-rwxrwxr-x 1 victor victor 524288 jul 20 20:00 cetic_6lbr_6lr.bin
-rwxrwxr-x 1 victor victor 524288 jul 20 19:59 cetic_6lbr_full_transparent_bridge.bin
-rwxrwxr-x 1 victor victor 524288 jul 20 19:59 cetic_6lbr_ndp_router.bin
-rwxrwxr-x 1 victor victor 524288 jul 20 19:59 cetic_6lbr_router.bin
-rwxrwxr-x 1 victor victor 524288 jul 20 19:59 cetic_6lbr_rpl_relay.bin
-rwxrwxr-x 1 victor victor 524288 jul 20 19:58 cetic_6lbr_smart_bridge.bin
```

Figura 6.28: Binarios generados cada uno de ellos es un modo de funcionamiento

19 Enlace al código fuente del Border Router de Cetic: https://github.com/cetic/6lbr/tree/dev-1.4.x

El binario con el nombre “cetic_6lbr_router.bin” es el utilizado para el proyecto. Para programar un OpenMote junto a su OpenBase se utilizó el programa escrito en python “cc2538-bsl”. Para más detalles sobre la programación por USB de los OpenMote y OpenChina consultar el *Anexo Nodos V: Programación de nodos por USB*.

```
victor@victor-VPCEB1S8E:~/Thermesys/6lbr/examples/6lbr/bin_openmote$ sudo ~/Ther
mesys/cc2538-bsl/cc2538-bsl.py -p /dev/ttyUSB0 -ewv cetic_6lbr_router.bin
Opening port /dev/ttyUSB0, baud 500000
Reading data from cetic_6lbr_router.bin
Connecting to target...
    Target id 0xb964, CC2538
Erasing 524288 bytes starting at address 0x200000
    Erase done
Writing 524288 bytes starting at address 0x200000
    Write done
Verifying by comparing CRC32 calculations.
    Verified (match: 0xa1f96c8b)
```

Figura 6.29: Programación de OpenMote con cc2538-bsl.py

Una vez programado uno de los OpenMote con el binario “cetic_6lbr_router.bin” se puede acceder al mismo desde cualquier navegador utilizando la siguiente dirección IPv6: [bbbb::100] gracias a que dispone de un servidor Web que permite su configuración y visualización de nodos conectados a él. Para comprobar que se ha programado correctamente, podemos utilizar una conexión serie con picocom.

```
picocom --imap lfcrlf --b 115200 /dev/ttyUSB0
```

Figura 6.30: Comando conexión puerto serie

Como se puede observar en la figura anterior, se establece una conexión serie a /dev/ttyUSB0 con un baud rate de 115200.

```
OpenMote-CC2538
 Net: sicslowpan
 MAC: CSMA
 RDC: nullrdc
 Channel: 25
 PAN-ID: abcd
Rime configured with address 00:12:4b:00:06:0d:9e:fa
 Antenna: external
NOTICE: 6LBR: Starting 6LBR version 1.4.x (Contiki-develop-1.4.1-32-ga074405)
```

Figura 6.31: OpenMote iniciando como Border Router

La figura anterior muestra una conexión establecida por el puerto serie del OpenMote programado y como este inicia correctamente. Por lo que podemos pasar a configurarlo utilizando su página web.

Los ajustes del Border Router se pueden realizar también antes de compilar el binario modificando el fichero “project-conf.h” para establecer por ejemplo el canal a utilizar en la red de sensores. No se configuró nada modificando este fichero porque todos los ajustes se pueden realizar una vez el nodo esta programado. También se puede establecer seguridad a la red de sensores pero no se llegó a activar para poder avanzar en el proyecto. En la siguiente figura se pueden observar los ajustes establecidos. Para más detalles sobre la página web del Border Router consultar el *Anexo Nodos VIII: Ajustes del Border Router*.

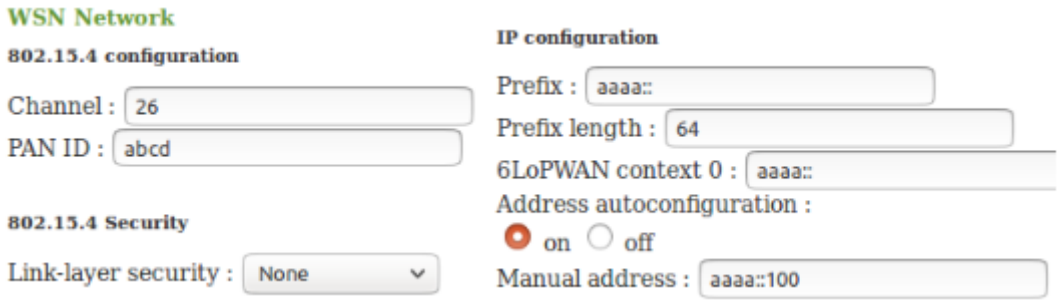


Figura 6.32: Ajustes interfaz WSN

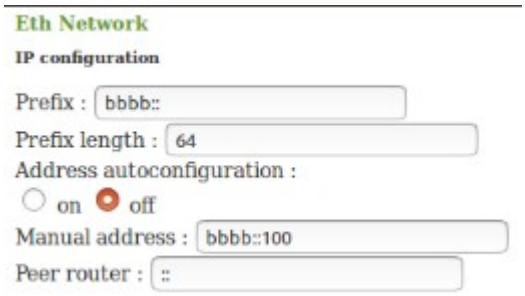


Figura 6.33: Ajustes interfaz Ethernet

Problemas encontrados:

Errores de compilación:

Al clonar el repositorio a su última versión estable a fecha del 21/01/2017 “1.5.x”, no se podían compilar los binarios para ningún nodo solamente se podía compilar para el TARGET “NATIVE”, el sistema Linux que ejecuta el make. Se descargó la última versión a fecha del 10/04/2016 y con esta versión se podían generar los binarios correctamente.

Se analizó el código fuente comparándolo con la última versión estable y en esa versión estable faltaba un “AND” en el fichero “**dev-1.5.x/core/dev/slip.c**” en la línea 46. Lo que provocaba que no se lograran compilar los binarios para la OpenMote ya que al faltar ese “AND” se trataban de compilar los binarios como si fueran “NATIVE” en lugar del TARGET especificado al realizar el MAKE.

Contenido original que provocaba error al compilar para la OpenMote:

```
#if CETIC_6LBR

#include "native-config.h"

#endif
```

Contenido modificado para arreglar el error:

```
#if CETIC_6LBR && TARGET_CONTIKI_NATIVE

#include "native-config.h"

#endif
```

Una vez solucionado, se consiguió generar los binarios pero a la hora de programarse los nodos no llegaban a arrancar y se quedaban con los leds encendidos con poca intensidad.

Como la versión estable “1.4.x” respecto la estable “1.5.x” tiene una diferencia de menos de un mes en su salida. No se investigó más para poder seguir con el proyecto se decidió utilizar la versión “1.4.x”.

6.1.4 Nodos Cliente: Clientes del Border Router

Para que los nodos cliente se conecten al Border Router, estos deben tener los mismos ajustes durante el arranque que los que tiene el Border Router configurados. Para establecer los ajustes de Radio, RPL y 6LoWPAN acorde a como están en el Border Router, se han establecido en el fichero “project-conf.h”. Este fichero se utiliza en Contiki para establecer ajustes, parámetros y sobrescribir puertos y pines a la hora de compilar un binario. Para mas información consultar el *Anexo Nodos XI: Fichero project-conf.h*.

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

#define IEEE802154_CONF_PANID 0xABCD
#define RF_CHANNEL 26
#define CC2538_RF_CONF_CHANNEL RF_CHANNEL

#define NETSTACK_CONF_MAC               csma_driver
#define NETSTACK_CONF_RDC               nullrdc_driver

/*---------------------------------------------------------
/* 6LoWPAN
/*---------------------------------------------------------

#define CETIC_6LBR_6LOWPAN_CONTEXT_0
        { 0xAA, 0xAA, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 }

/*---------------------------------------------------------
/* RPL & Network
/*---------------------------------------------------------

#define RPL_CONF_INIT_LINK_METRIC                       2
#define RPL_MAX_DAG_PER_INSTANCE        2
#define RPL_MAX_INSTANCES               1
```

Figura 6.34: Ajustes de red nodo cliente

Una vez establecidos los mismos ajustes de red que el Border Router en su interfaz WSN, se generan los binarios del nodo cliente, los nodos son programados y se comprueba que los nodos cliente se conectan al Border Router.

Sensors configuration

Node
0:12:4b:0:4:1b:5f:5f
0:12:4b:0:4:1b:e3:c4
0:12:4b:0:4:32:9a:c0
0:12:4b:0:6:d:9f:18
0:12:4b:0:4:32:9a:a2
0:12:4b:0:4:1b:e3:9d

Figura 6.35: Nodos cliente que están conectados al Border Router

6.1.5 Nodos Cliente: Implementación de Servidor/Cliente CoAP

Para realizar esta parte se pensó que forma era la mejor para tener unos nodos que informaran de forma periódica al nodo sumidero y que fueran al mismo tiempo accesibles y configurables de forma externa en cualquier momento.

Los nodos cliente como servidor CoAP:

- Permiten consultar (GET) los siguientes recursos utilizando un cliente CoAP que se conecte a ellos:
    - Temperatura actual del sensor MAX6675.
    - Temperatura actual de la CPU del nodo.
    - Dirección MAC del nodo.
    - RSSI del nodo.
- Permiten consultar (GET) y configurar (POST) los siguientes parámetros:
    - Dirección IPv6 del nodo sumidero.
    - Período de notificación.
    - Temperatura de envío de alerta.
    - Recurso de destino del nodo sumidero.

Los nodos cliente como cliente CoAP del servidor CoAP en el nodo sumidero:

- Informan cada vez que su timer interno llega a cero (POST) entre otros datos de:
    - La temperatura leída por el sensor MAX6675.
    - Envían una alerta si la temperatura leída es superior a un máximo establecido (configurable).
    - Su dirección MAC (identificador único).
    - Temperatura de la CPU.

Esta sección ha sido dividida en cuatro puntos para detallar las distintas partes que se implementan en un nodo cliente:

- Codificación de los mensajes CoAP en JSON.
- Cliente CoAP.
- Servidor CoAP.
- Implementación de un recurso genérico en el servidor CoAP del nodo.

6.1.5.1 Codificación de los mensajes CoAP en JSON

Existe una limitación en el tamaño de los mensajes por utilizar CoAP que funciona sobre la capa de transporte UDP (64 bytes), por lo que se ha decidido en lugar de mandar 1 mensaje con varios datos, enviar un mensaje por dato.

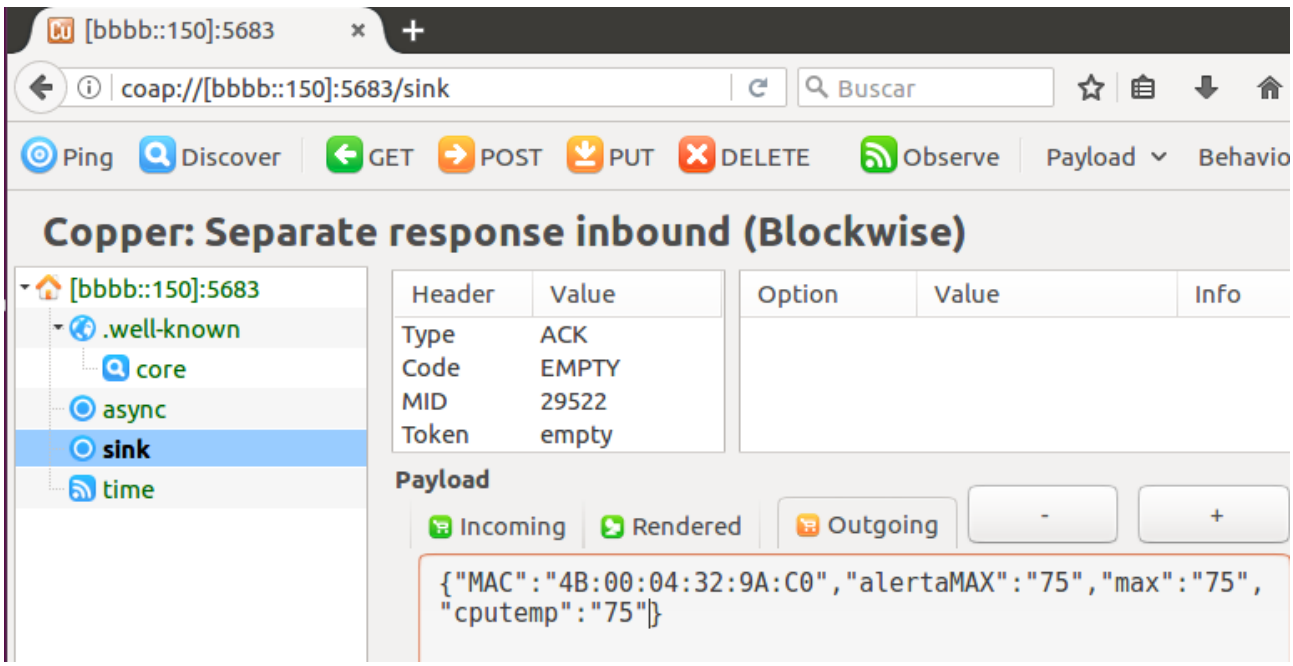


Figura 6.36: Envío de JSON largo

En la siguiente figura se puede observar qué ocurría cuando un mensaje codificado en JSON excedía los 65 caracteres como el de la figura anterior:

```
vpolo@zenderone:~$ ./projects/COAP/example/coap-srv -A bbbb::150 -v 3
ENTRO EN DATA POST HANDLER
RES = 1
SIZE: 64

DATOS:
{"MAC":"4B:00:04:32:9A:C0","alertaMAX":"75","max":"75","cputemp"

Failed to parse JSON: -3
```

Figura 6.37: JSON largo recibido en servidor CoAP

El mensaje al alcanzar una talla de 65 caracteres es truncado por lo que deja de ser un mensaje con formato JSON válido y por lo tanto no puede ser decodificado por la librería jsmn.

Por este motivo se decidió que en lugar de mandar un solo mensaje, se mandarían varios, uno por cada dato leído:

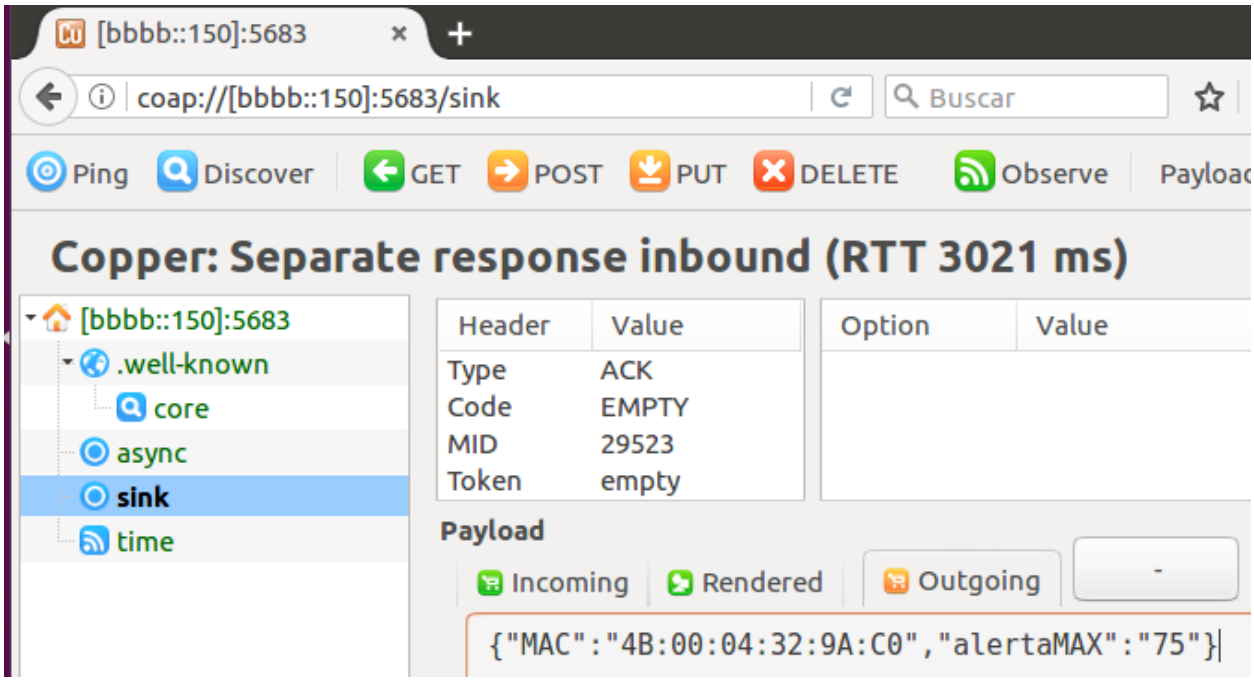


Figura 6.38: Envío de JSON corto

En la siguiente figura se puede observar que el mensaje llega correctamente al servidor CoAP y decodificado sin problemas al no perder su estructura.

```
ENTRO EN DATA POST HANDLER
RES = 1
SIZE: 44

DATOS:
{"MAC":"4B:00:04:32:9A:C0","alertaMAX":"75"}

MAC: 4B:00:04:32:9A:C0
ALERTA!!! MAX TEMP: 75º
Creando conexión con microservicio de alertas SMS
```

Figura 6.39: Mensaje JSON corto recibido en servidor CoAP

Para construir un mensaje codificado en JSON en el cliente CoAP del nodo cliente se utiliza un buffer y la función sprintf para llenarlo.

```
/*Mensaje MAX6675*/

mac_address_read(buf_node_id, 17); //Se lee la dirección MAC de nodo
n += sprintf(&(buf[n]),"{\"MAC\":\"%s\"",buf_node_id);
temperature = max6675_read(SPI_SEL_PORT,SPI_SEL_PIN);//Se lee temperatura del MAX6675
n += sprintf(&(buf[n]),",\"max6675\":\"%u\"",temperature/4);
n += sprintf(&(buf[n]),"}");
buf[n] = 0;
```

Figura 6.40: Código que genera un mensaje codificado en JSON

6.1.5.2 Cliente CoAP

Como cliente CoAP, un nodo ya tiene preconfigurados los siguientes parámetros en el arranque:

- Intervalo entre mensajes: 10 segundos
- Recurso remoto donde enviar los mensajes: /sink
- Dirección IPv6 del servidor CoAP remoto: bbbb::150
- Temperatura máxima que activa una alarma: 50 grados centígrados.

```
/*Cuando el timer finaliza porque se ha cumplido el perído definido, es reniciado y
se incia el proceso de recopilar y enviar los datos al servidor CoAP de la Rpi*/

if(etimer_expired(&et_read_sensors)) {

        etimer_set(&et_read_sensors, sensor_cfg.post_interval * CLOCK_SECOND);
        process_start(&read_sensors, NULL);

}
```

Figura 6.41: Código que inicia la lectura periódica de datos

El proceso read_sensors recopila los datos y construye un mensaje JSON por cada dato recopilado de la misma forma que se muestra en la "F*igura 6.38: Envío de JSON* corto" del punto anterior. Cuando tiene un mensaje JSON completo lo envía (POST) al nodo sumidero en su recurso **sink**.

```
/*DoPOST*/
if (dag != NULL) {
        static coap_packet_t request[1];
        coap_init_message(request, COAP_TYPE_CON, COAP_POST, 0 );
        coap_set_header_uri_path(request, sensor_cfg.sink_path);
        coap_set_payload(request, buf, strlen(buf));

        post_count++;
        COAP_BLOCKING_REQUEST(&sensor_cfg.sink_addr, REMOTE_PORT, request, client_chunk_handler);
        if(erbium_status_code)
                PRINTF("status %u: %s\r\n", erbium_status_code, coap_error_message);
        if (con_ok == 0)
                PRINTF("CON failed\r\n");
        else
                PRINTF("coap done\r\n");

}
```

Figura 6.42: Código que envía los datos recopilados al nodo sumidero (POST)

6.1.5.3 Servidor CoAP

Para que el servidor CoAP sea configurable, se ha definido el recurso **config** que implementa dos manejadores uno para atender las peticiones de lectura de algún parámetro (GET) y otro para establecer un nuevo valor a un parámetro (POST).

Para poder realizar consultas sobre recursos del nodo, se han implementado cuatro recursos ubicados en el directorio **resources** del código fuente del programa.

En el siguiente punto se explica como implementar el recurso de leer la dirección MAC del nodo de forma genérica. De la misma forma se han implementado todos los recursos del servidor:

- res-mac.c
- res-cpu-temp.c
- res-max.c
- res-rssi.c

6.1.5.4 Implementación de un recurso genérico en el servidor CoAP del nodo

Para implementar los recursos del servidor CoAP se ha utilizado como base el ejemplo ubicado en los fuentes de Contiki dentro del directorio **/examples/er-rest-example**.

Modificaciones realizadas sobre ese código:

- Añadido cliente UDP para la correcta visualización de los nodos y como están conectados en el servidor web del Border Router.
- Modificado Makefile para activar el protocolo RPL
- Añadida la configuración necesaria para arrancar el cliente UDP si el flag **WITH_UDPCLIENT** está activo.
- Por cada nuevo recurso que se desee utilizar, se debe realizar lo siguiente:

```
extern resource_t nombre_recurso;

PROCESS_THREAD(er_example_server, ev, data)
{
…..
#Activar ese recurso
rest_activate_resource(&nombre_recurso, "uri/recurso");
….
}
```

- En la raíz del proyecto existen dos directorios:
  - apps: contiene el cliente udp utilizado para debug
  - resources: contiene los recursos definidos en er-example-server.c

- Ejemplo del contenido del recurso que lee la dirección MAC de un nodo en /resources/res-mac.c:

```
#include <stdlib.h>
#include <string.h>
#include "rest-engine.h"

#include "contiki-net.h"
#include "net/ip/uip.h"

/* Función Leer Mac */
static void mac_address_read(uint8_t * mac_addr, int len){…}

#Definición del recurso
RESOURCE(res_mac,
        "title=\"Mac Address: \";rt=\"Text\"",
        res_get_handler,
        NULL,
        NULL,
        NULL);

#Definición del manejador de ese recurso (cuando se recibe un GET, se envía la dirección MAC)
static void
res_get_handler(void *request, void *response, uint8_t *buffer, uint16_t preferred_size, int32_t *offset)
{
          static uint8_t buf_node_id[18];
          int length = 17;
          /* read current MAC address */
          mac_address_read(buf_node_id, 17); //17 es la talla de la dirección MAC representada en char *

          buf_node_id[17]='\0'; //buffer impreso como string
          memcpy(buffer, buf_node_id, length); #Se copia la MAC leída al buffer

          REST.set_header_content_type(response, REST.type.TEXT_PLAIN);
          REST.set_header_etag(response, (uint8_t *)&length, 1);
          REST.set_response_payload(response, buffer, length);
}
```

6.1.5.5 Ajustes de red estación de trabajo

Para poder acceder al servidor CoAP de los nodos cliente, es necesario añadir una ruta a la interfaz de red de la estación de trabajo funcionando con Ubuntu 16.04:

sudo route -A inet6 add aaaa::/64 gw bbbb::100

Con esta ruta estamos configurando como puerta de enlace hacia la red de los sensores la IPv6 del Border Router. Más adelante se configuraron las rutas en el fichero de configuración de las interfaces de red para mayor comodidad ya que el comando anterior es solo temporal hasta que reinicia la interfaz de red.

Contenido del fichero **/etc/networking/interfaces** de la estación de trabajo:

```
vpolo@zenderone:~$ cat /etc/network/interfaces
…
#Ruta a la red de nodos
post-up route -A inet6 add aaaa::/64 gw bbbb::100
….
```

6.1.5.6 Ajustes de red nodo sumidero

Para disponer de conectividad a la red de sensores (aaaa::) y recibir los mensajes de los nodos cliente, también se debe especificar la ruta para alcanzarla como en el punto anterior. Además se establece una IPv6 fija que corresponde con la IPv6 que tiene configurados los nodos cliente para notificar periódicamente.

```
pi@raspberrypi:~ $ cat /etc/network/interfaces
…
# IPv6 estática
iface eth0 inet6 static
address bbbb::150
netmask 64

#Ruta a la red de nodos
post-up route -A inet6 add aaaa::/64 gw bbbb::100
…
```

6.1.6 Nodo Sumidero: Implementación de Servidor CoAP

Para implementar el servidor CoAP del nodo sumidero, se han considerado diversas implementaciones en varios lenguajes de programación y finalmente se ha decidido utilizar libcoap, librería CoAP programada en C que permite tener listo un servidor CoAP a medida en poco tiempo.

Para decodificar los mensajes JSON enviados por los nodos cliente, se ha utilizado la librería jsmn desarrollada también en C. Se ha utilizado como base del servidor CoAP el ejemplo contenido en el directorio **/examples** de la librería libcoap.

Para las primeras pruebas simplemente se clonó el repositorio de libcoap y compilaron los ejemplos. Se ejecutó el binario correspondiente con el servidor CoAP de ejemplo y se estableció la IPv6 de la estación de desarrollo a bbbb::150, los nodos cliente mandaban la información al PC y el servidor CoAP la mostraba por la terminal.

El siguiente paso fue establecer el recurso **/sink** donde los nodos cliente envían sus lecturas periódicas y utilizar las funciones que proporciona la librería jsmn para decodificar los mensajes consiguiendo el siguiente resultado:

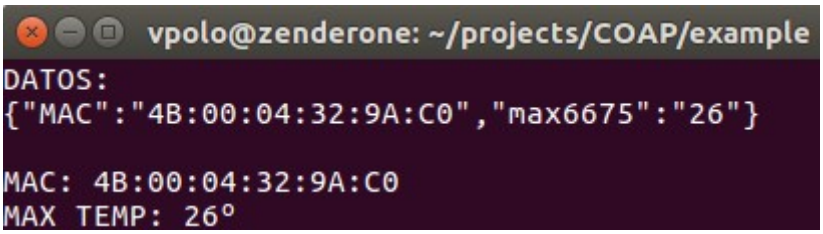


Figura 6.43: Mensaje recibido en servidor CoAP

Como se puede observar en la figura anterior se recibe una cadena de texto en formato JSON que al procesarse por el parser jsmn es separada en sus distintos atributos.

Una vez comprobado que el parser JSON funciona correctamente, se procedió a escribir la parte del servidor CoAP que gestiona una alerta a notificar por SMS. Antes de la implementación del Microservicio SMS se realizó un simulacro con simples printf.

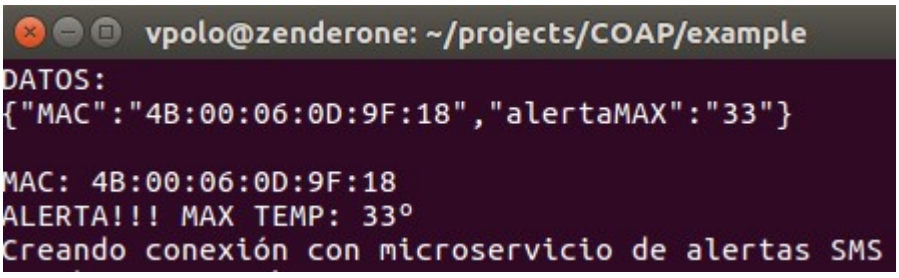


Figura 6.44: Alarma recibida en servidor CoAP

Problemas encontrados:

Como se realizó la implementación del servidor CoAP en la estación de trabajo y más adelante se compiló y ejecuto en la Raspberry Pi, apareció un problema que no había ocurrido hasta la fecha. El servidor CoAP se cerraba aleatoriamente tanto en el PC como en la Raspberry Pi, en el PC con menos frecuencia.

El problema era un error en tiempo de ejecución que fue encontrado utilizando la herramienta Valgrind. El problema en tiempo de ejecución era por utilizar sin casting previo una estructura size_t como int. En el compilador del PC no daba ningún warning sobre esto pero en la Raspberry Pi sí.

Una vez realizada la conversión de tipo en todos los usos a esa estructura, se volvieron a compilar los binarios. Se dejó en funcionamiento el servidor CoAP tanto en la Raspberry Pi como en el PC junto a varios nodos cliente enviándoles información cada segundo para forzar cualquier aparición de otro fallo.

El sistema estuvo funcionando sin incidencias unas 40 horas en ambos sistemas por lo que se dio el problema como solucionado.

6.1.7 Nodo Sumidero: Implementación de microservicio SMS

Para la implementación del Microservicio SMS primero se realizaron pruebas a mano realizando conexiones al módem utilizando picocom, se configuró utilizando comandos AT y se enviaron SMS. Una vez realizadas con éxito las primeras pruebas a mano se empezó a implementar el microservicio.

Primero se pasó a realizar una implementación de la librería encargada de conectarse con el módem por el puerto serie, configurarlo y enviar el SMS al móvil guardado en **/etc/telf.conf**. Esta implementación se realizó en los ficheros sms.c y sms.h, en el fichero de cabeceras se definieron los comandos AT fijos a utilizar para el envío del SMS y los ajustes para establecer la conexión serie con el módem:

```
// SMS conf
#define SMS_MODE "AT+CMGF=1\r\n"
#define SMS_END "\032"
#define SERIAL_PORT "/dev/ttyAMA0"
#define CHAR_SIZE 8
#define BAUD_RATE 9600
#define PARITY_BIT 'n'
#define STOP_BITS 1
```

En el fichero sms.c se definieron las siguientes funciones:

- open_serial_port(): función encargada de establecer una conexión serie con el módem /dev/ttyAMA0
- init_modem(): función encargada de establecer los ajustes del módem una vez establecida la conexión serie utilizando comandos AT.
- read_phone(): función encargada de leer el fichero **/etc/telf.conf** y extraer el teléfono móvil destino del SMS.
- set_phone(): función encargada de enviar el comando AT que permite generar SMS con el teléfono móvil destino.
- write_sms(): función encargada de enviar el texto recibido como parámetro al módem.
- init_SMS(): función encargada de utilizar las funciones anteriormente descritas para enviar el SMS.

La implementación descrita funcionaba como un simple binario que al ser ejecutado iniciaba el proceso de envío de un SMS con un texto preestablecido.

Una vez implementada la primera versión que funcionaba de forma local, se implementó un microservicio mediante socket stream para utilizar la versión local de envío de SMS.

```
int main(void)
{

   int len;
   synch_serversock_open(8000);
   len = 1;

   while (1) {

      if (len > 0) {
         printf("Waiting connection accept\n");
         synch_serversock_accept();
      }

      len = synch_serversock_recv_buf_alerta(1000);

      if (len > 0) { /*{printf("SMS Enviado");*/
          synch_serversock_accept_close();
      }
   }
}
```

El microservicio está escuchando el puerto 8000 hasta recibir un mensaje en formato JSON, el mensaje es decodificado utilizando la librería jsmn, una vez decodificado se construye el contenido del SMS y se envía a la librería SMS generada con una modificación en la función init_SMS() que le permite recibir el contenido del mensaje SMS generado como parámetro. Una vez recibido el contenido del SMS, el funcionamiento e implementación de sms.c es el mismo que en las pruebas manuales.

Para poder realizar pruebas sobre el microservicio sin depender de los nodos y el servidor CoAP, se establecían conexiones TCP con el microservicio utilizando el programa por comandos **netcat**, una vez establecida la conexión con el socket se escribían los mensajes JSON con el formato adecuado para simular envíos de alertas del servidor CoAP.

6.1.8 Nodo Sumidero: Comunicación entre Servidor CoAP y Microservicio SMS

Una vez implementado el microservicio de SMS, solo queda modificar el código fuente del servidor CoAP del nodo sumidero para establecer una conexión TCP con él. Como se especificaba en el apartado *"6.1.6 – Nodo Sumidero: Implementación de servidor CoAP"* ya se dejó el código fuente preparado para añadir la conectividad con el microservicio cuando este estuviera listo. Para añadir conectividad con el microservicio se realizaron los siguientes cambios:

```
else if (jsoneq(JSON_STRING, &t[i], "alertaMAX") == 0) {// JSON decodificado como alerta
        ret = synch_clientsock_connect("192.168.0.200", 8000); // Estableciendo conexión con microservicio
       SetSocketBlockingEnabled(1);

         if (ret != RET_OK) { // No se ha podido establecer la conexión con el microservicio
            printf("error connecting to server\n");
         }
         else if (ret == RET_OK) { // Conexión establecida correctamente con el microservicio
            ret = synch_clientsock_send_buf(JSON_STRING, size, 0); // Reenvío de mensaje JSON
            synch_clientsock_close(); // Cerramos conexión con el microservicio
         }
         i++;
      }
```

En el momento que se decodifica un mensaje JSON con la etiqueta "alertaMAX", se inicia una conexión TCP con el microservicio de SMS, si la conexión se ha establecido correctamente, se reenvía el mensaje JSON que contiene la alerta.

6.2 Verificación y validación

En este apartado se detallan las pruebas realizadas para comprobar el correcto funcionamiento de las implementaciones por separado y del sistema en conjunto.

6.2.1 Pruebas de estabilidad de la red

Para realizar estas pruebas se utilizó la interfaz web que proporciona el Border Router, gracias a ella se puede consultar en tiempo real el estado de la red de sensores y se pueden observar los cambios producidos en ella.

El punto de partida para estas pruebas es la red de sensores conectada y funcionando, para forzar desconexiones de la red, se quitaron las antenas de varios nodos cliente y se ubicaron en distintas zonas de la oficina de Thermesys. Para poder realizar las pruebas con mayor comodidad, los nodos utilizados para moverse se alimentaron a pilas o con un powerbank.

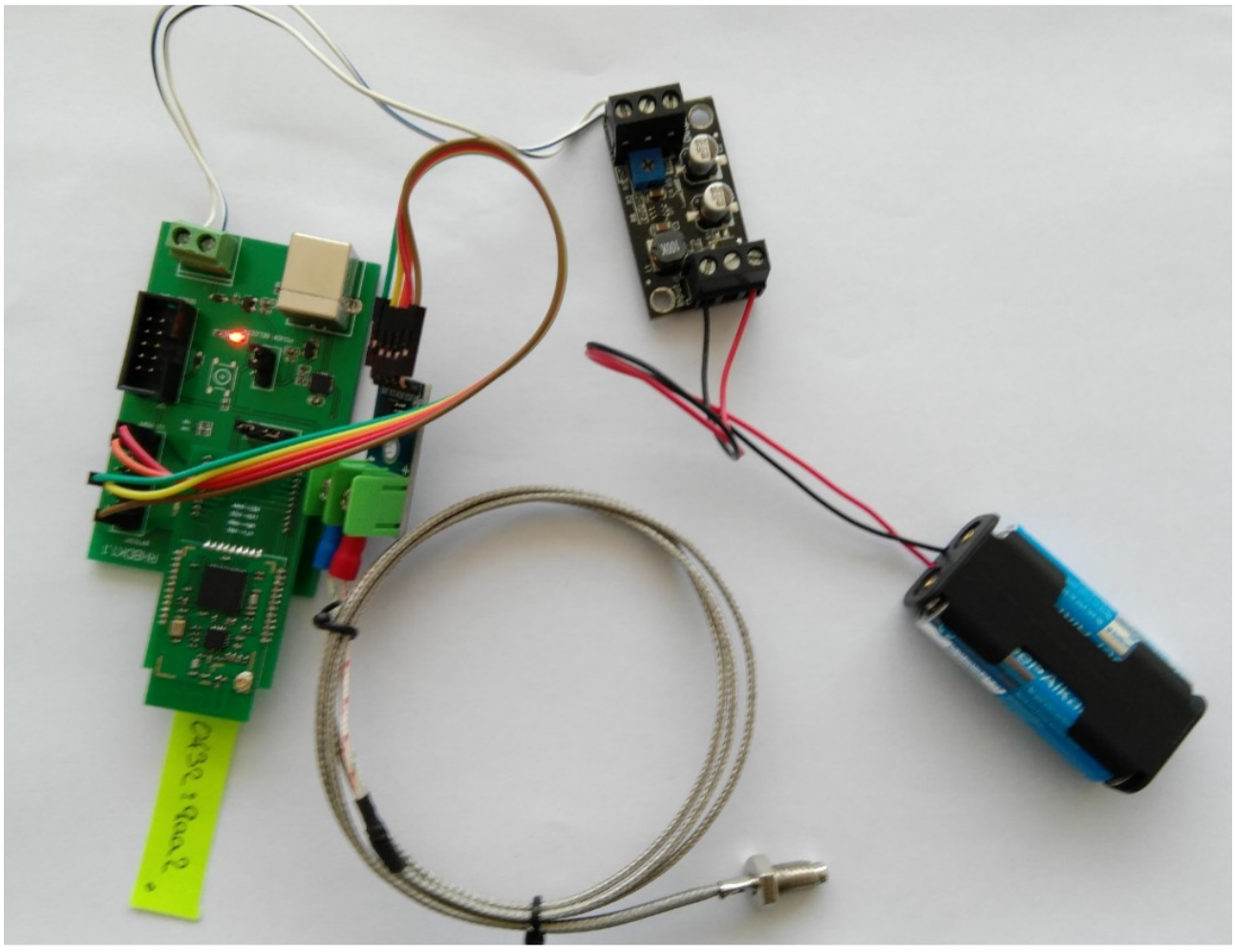


Figura 6.45: OpenChina programado como nodo cliente alimentado a pilas

El estado inicial de la red observando el Border Router sin realizar ninguna distribución de los nodos era el que se muestra en la siguiente figura:

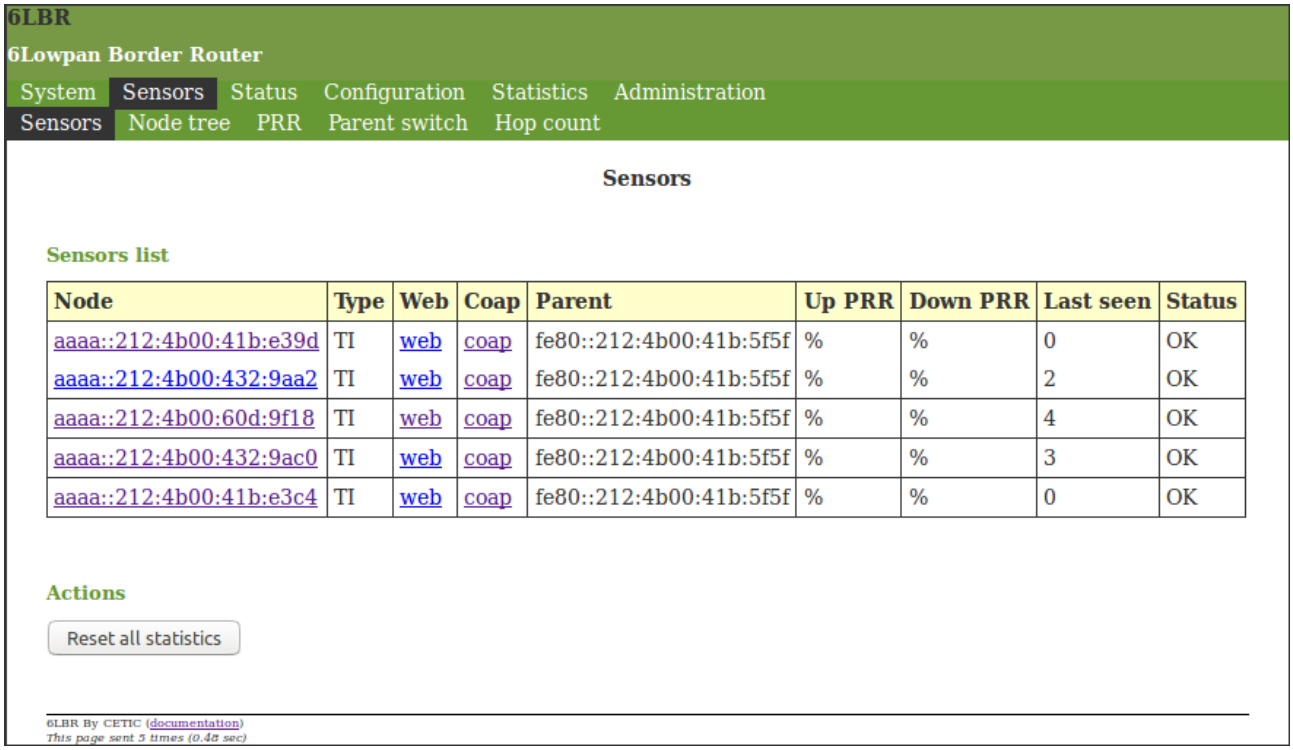


Figura 6.46: Listado de nodos conectados al Border Router

Como se puede ver en la columna de “Parent”, todos los nodos tienen un salto de distancia hasta alcanzar el Border Router. En la siguiente figura que se construye dinámicamente en el Border Router se aprecia mejor que en la tabla:

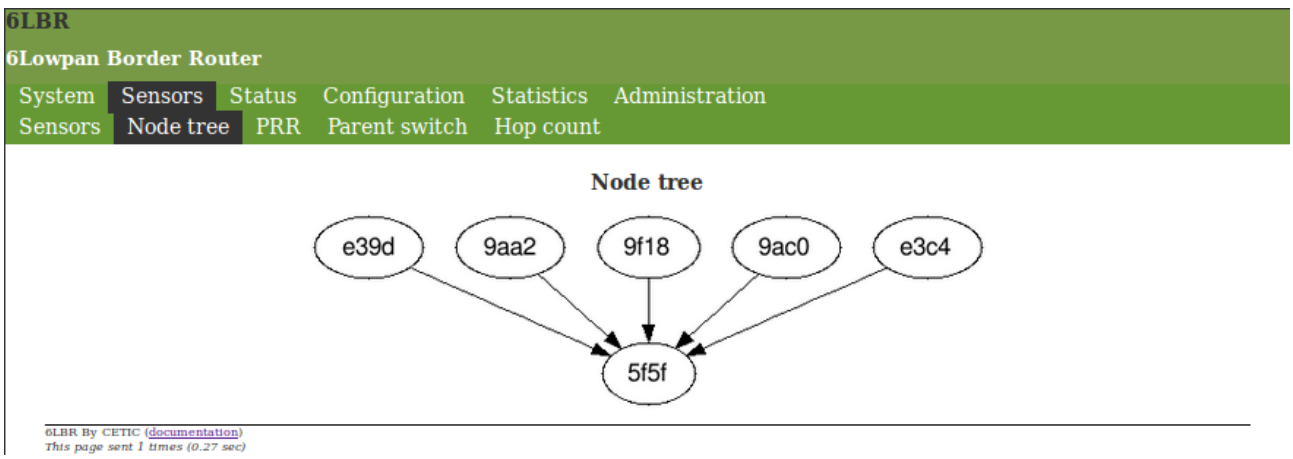


Figura 6.47: Nodos conectados directamente al Border Router

El nodo utilizado para forzar su desconexión directa del Border Router es el identificado como **9f18** en la anterior figura. Este nodo corresponde a una placa OpenMote a la que se le va a quitar la antena para que pierda mucho alcance de radio y se vea obligado a utilizar a uno de sus vecinos como salto. El nodo **9f18** es ubicado en un extremo opuesto de la oficina a donde esta el Border Router, se comprueba que el nodo ya no se conecta con él y se ubica en puntos intermedios de la oficina varios nodos alimentados a pilas para conseguir que alguno de ellos sea utilizado como salto.

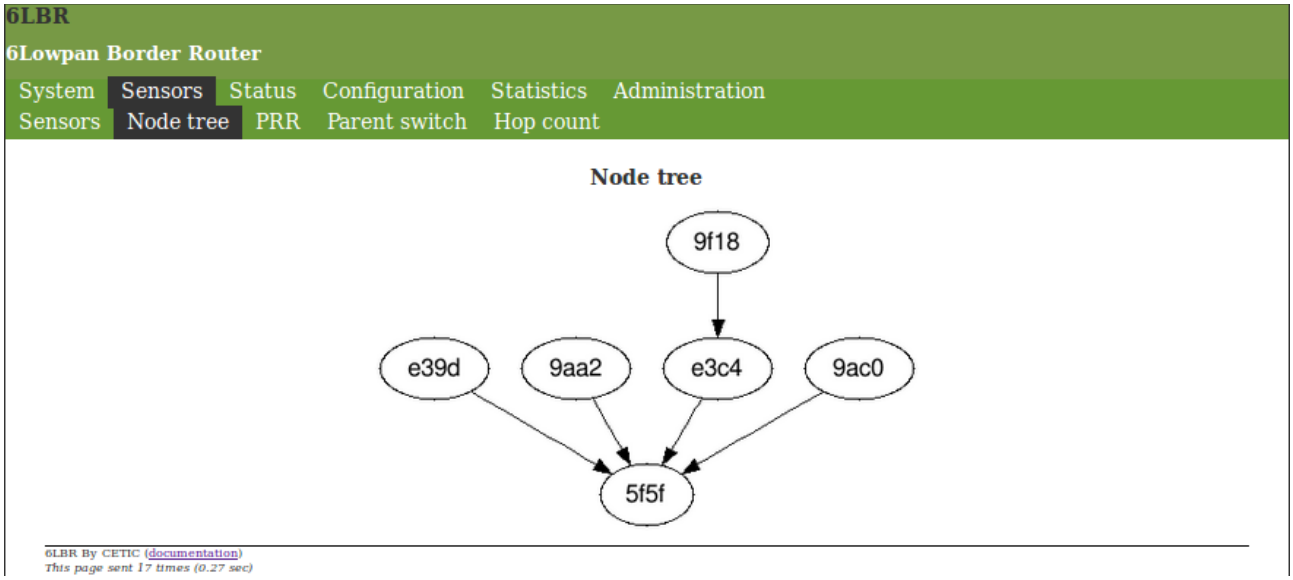


Figura 6.48: Nodo 9f18 conectado al Border Router indirectamente mediante el nodo e3c4

Como se puede ver en la figura anterior, el nodo **9f18** en este instante utiliza a su vecino **e3c4** como salto para alcanzar su destino.

Durante la misma prueba, el mismo nodo pasado un tiempo pasó a utilizar otro vecino como salto:

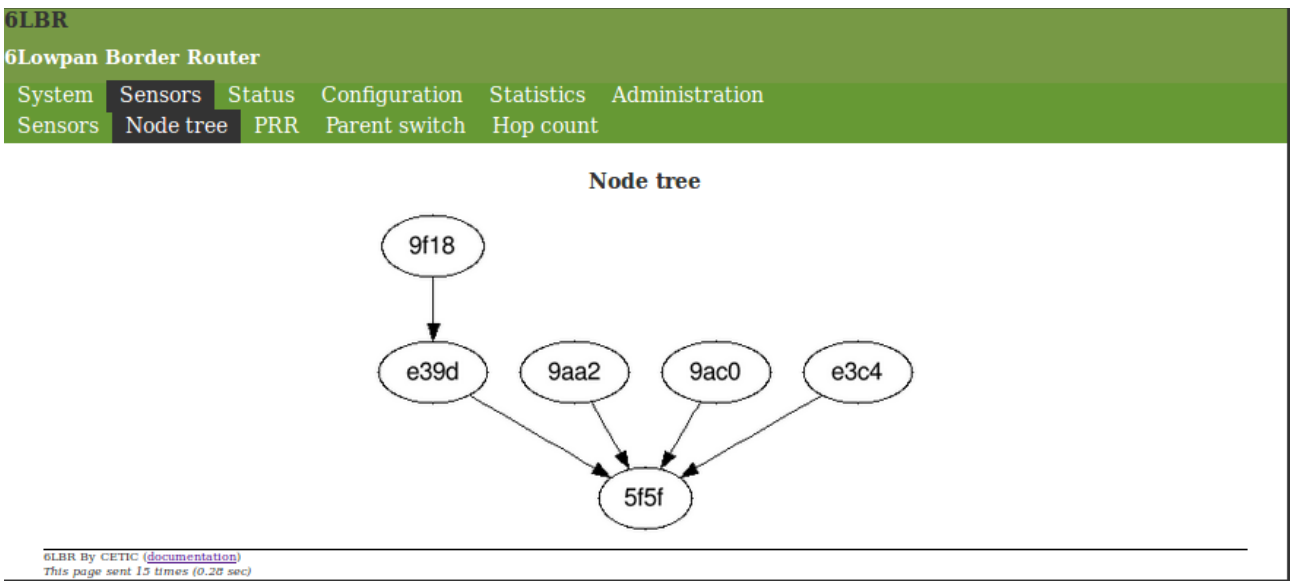


Figura 6.49: Nodo 9f18 conectado al Border Router indirectamente mediante el nodo e39d

Mientras un nodo tenga algún vecino a su alcance éste siempre termina llegando a su destino y sigue siendo accesible su servidor CoAP y sus notificaciones siguen llegando al nodo sumidero.

En la siguiente figura se pueden observar el número de saltos que realiza cada nodo para alcanzar el Border Router:

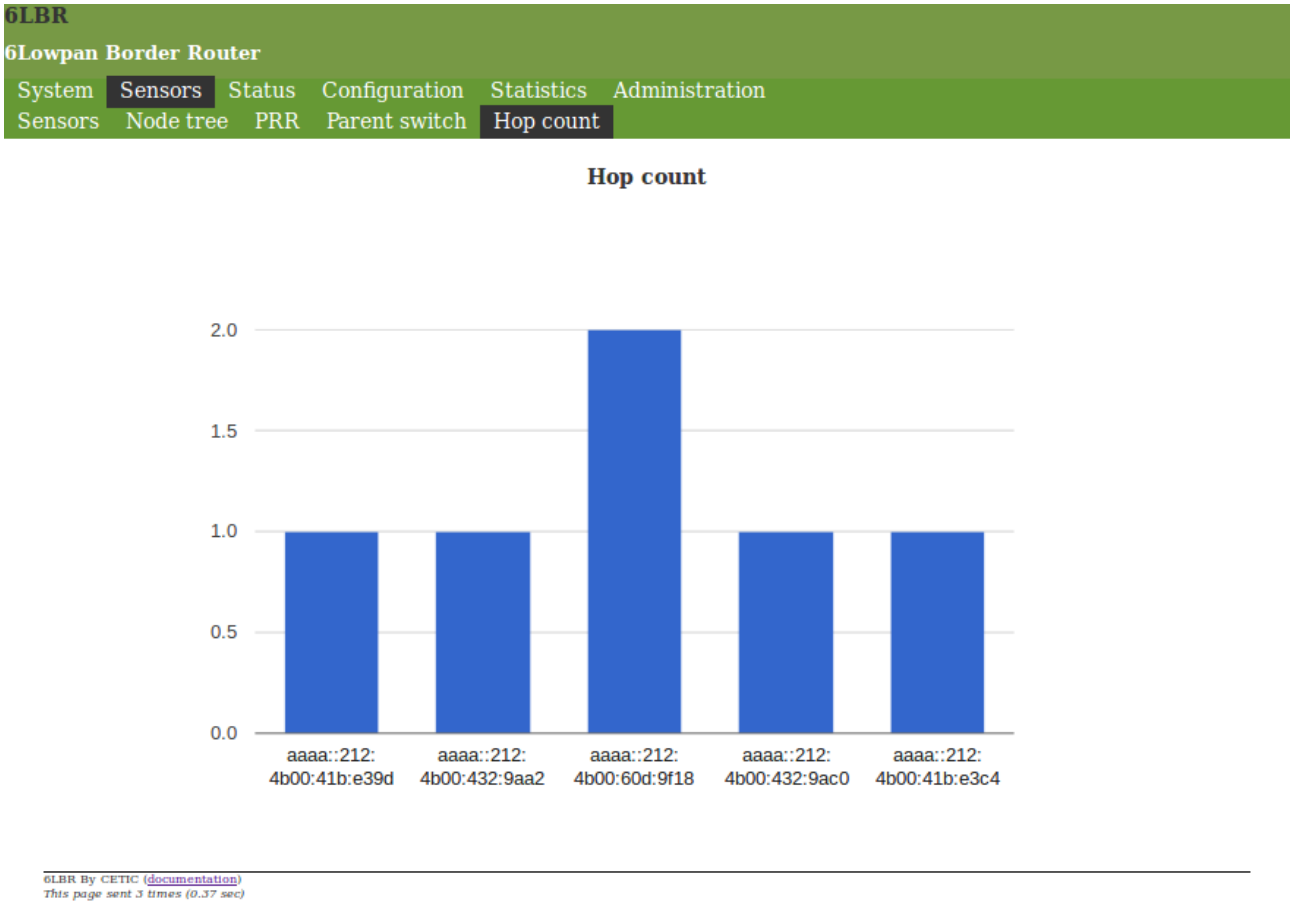


Figura 6.50: Número de saltos por nodo hasta alcanzar el Border Router

Los nodos que no han sido forzados a perder alcance de radio, tienen visión directa (1 salto) y el nodo que ha sido forzado, tiene visión indirecta (2 saltos).

6.2.2 Nodo Cliente: Pruebas servidor CoAP

El acceso a un servidor CoAP de un nodo puede realizarse accediendo a la página web del Border Router y acceder directamente a la dirección CoAP de los nodos conectados a la red. También es posible acceder a los nodos utilizando algún cliente CoAP por comandos. Como se puede ver en la siguiente figura podemos consultar todos los nodos cliente que tenemos enlazados con el Border Router y acceder a su dirección CoAP.

Sensors list

Node	Type	Web	Coap	Parent	Up PRR	Down PRR	Last seen	Status
aaaa::212:4b00:41b:e39d	TI	web	coap	fe80::212:4b00:41b:5f5f	%	%	2	OK
aaaa::212:4b00:432:9aa2	TI	web	coap	fe80::212:4b00:41b:5f5f	%	%	4	OK
aaaa::212:4b00:60d:9f18	TI	web	coap	fe80::212:4b00:41b:e3c4	%	%	1	OK
aaaa::212:4b00:432:9ac0	TI	web	coap	fe80::212:4b00:41b:5f5f	%	%	2	OK
aaaa::212:4b00:41b:e3c4	TI	web	coap	fe80::212:4b00:41b:5f5f	%	%	0	OK

Figura 6.51: Listado de nodos cliente conectados al Border Router con acceso a sus servidores CoAP

En la siguiente figura podemos ver los recursos de un nodo cliente en concreto utilizando Copper y pulsando en el botón “Discover”, si no se observa ningún recurso en este punto, es que el nodo cliente no tiene implementado un servidor CoAP o que no funciona correctamente.

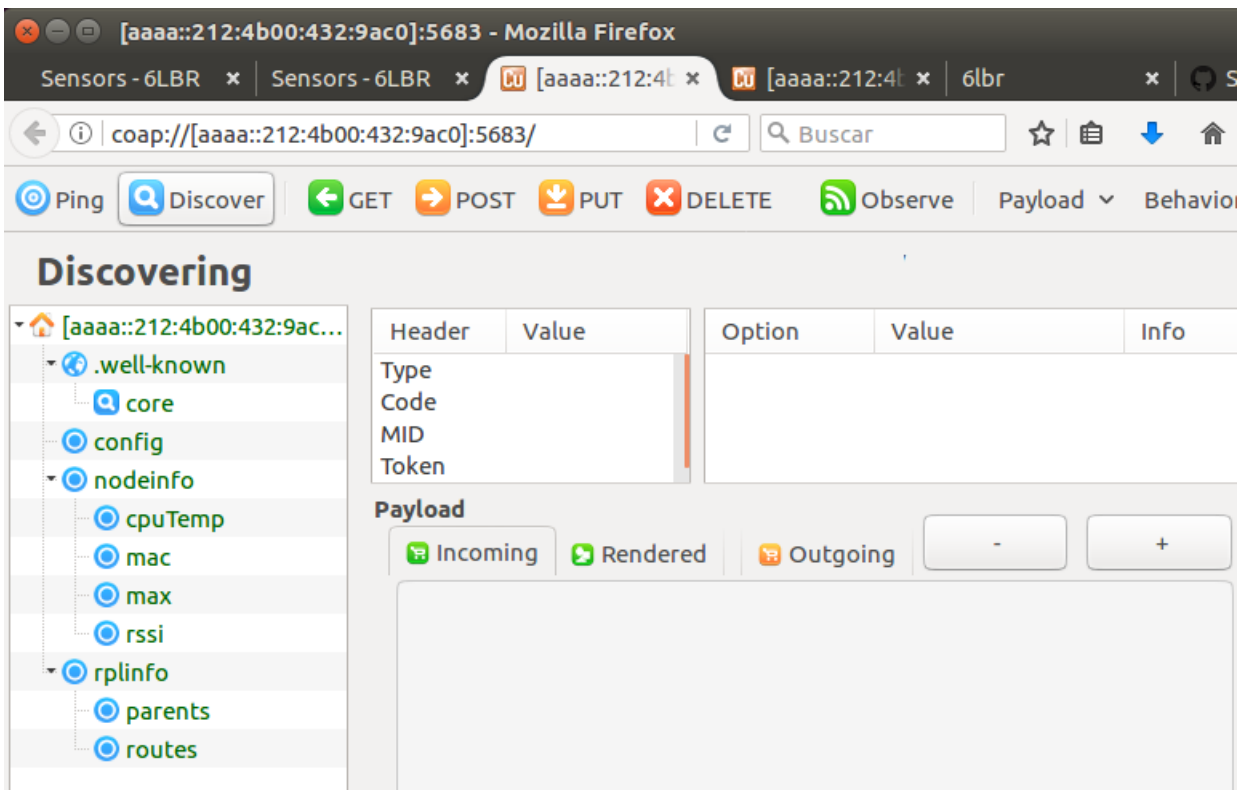


Figura 6.52: Cliente Copper conectado a servidor CoAP nodo cliente

Para consultar recursos de un nodo (GET), se elije el recurso que quedemos consultar y se pulsa el botón GET. En la siguiente figura podemos observar que una consulta GET sobre el recurso “max” devuelve la temperatura actual del sensor MAX6675 que tiene conectado el nodo.

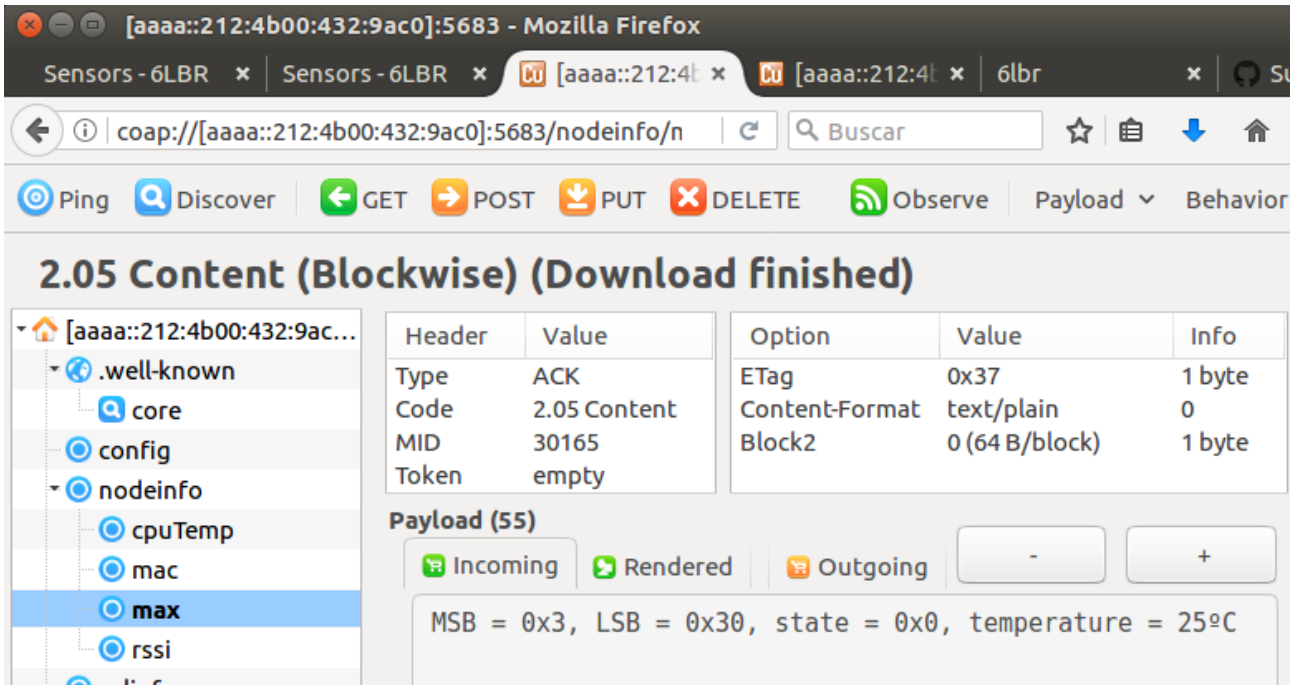


Figura 6.53: Consulta (GET) sobre recurso max que devuelve la temperatura del sensor MAX6675

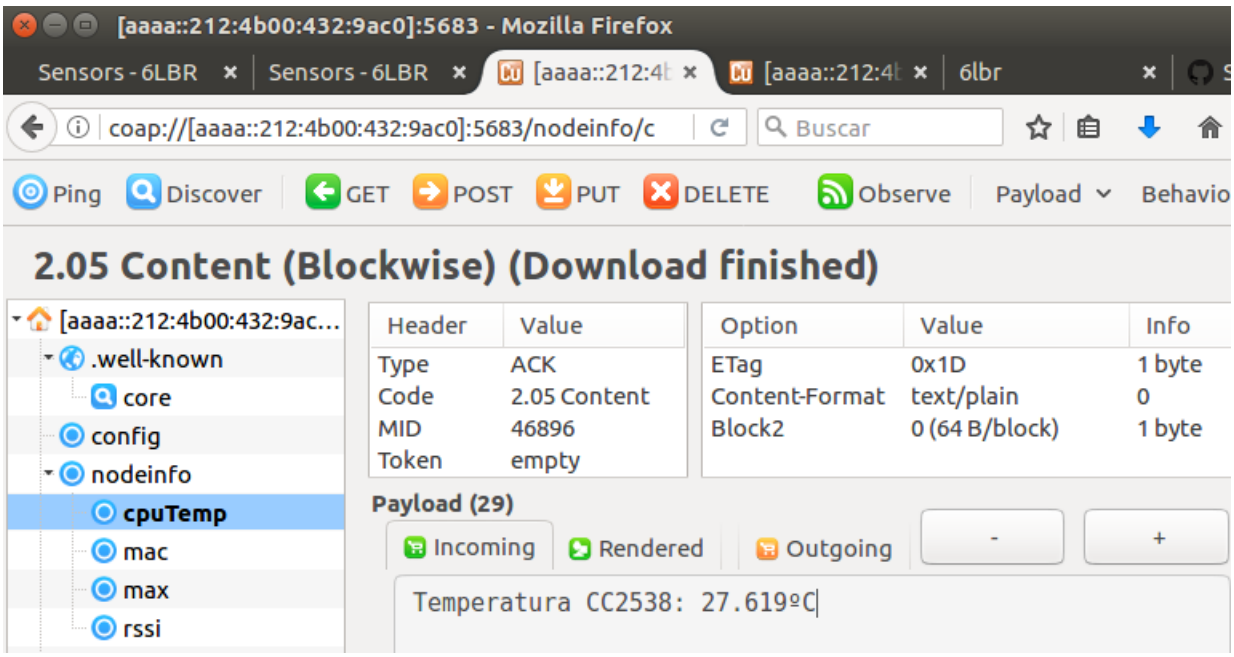


Figura 6.54: Consulta (GET) sobre recurso cpuTemp que devuelve la temperatura de la CPU del nodo

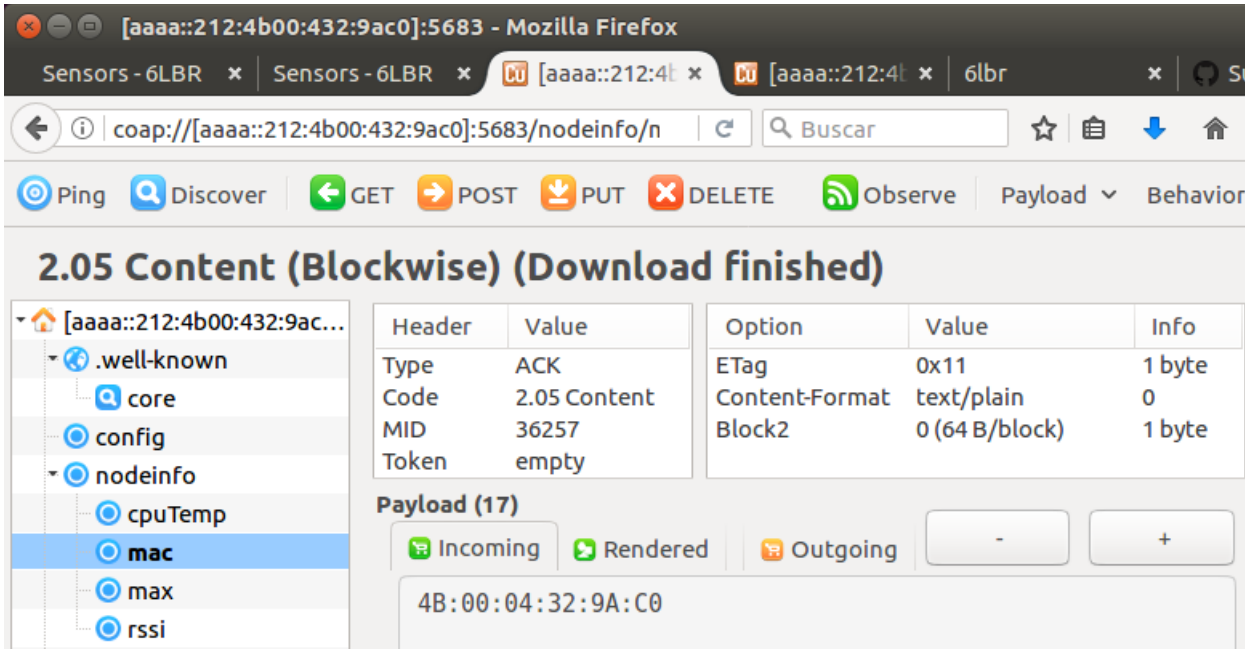


Figura 6.55: Consulta (GET) sobre recurso mac que devuelve la dirección MAC del nodo

Para realizar ajustes en el nodo debemos realizar POST sobre las variables disponibles; **param**, **ip**, **sink**, e **interval**. En la siguiente figura se puede observar como establece un nuevo periodo de notificaciones al nodo sumidero.

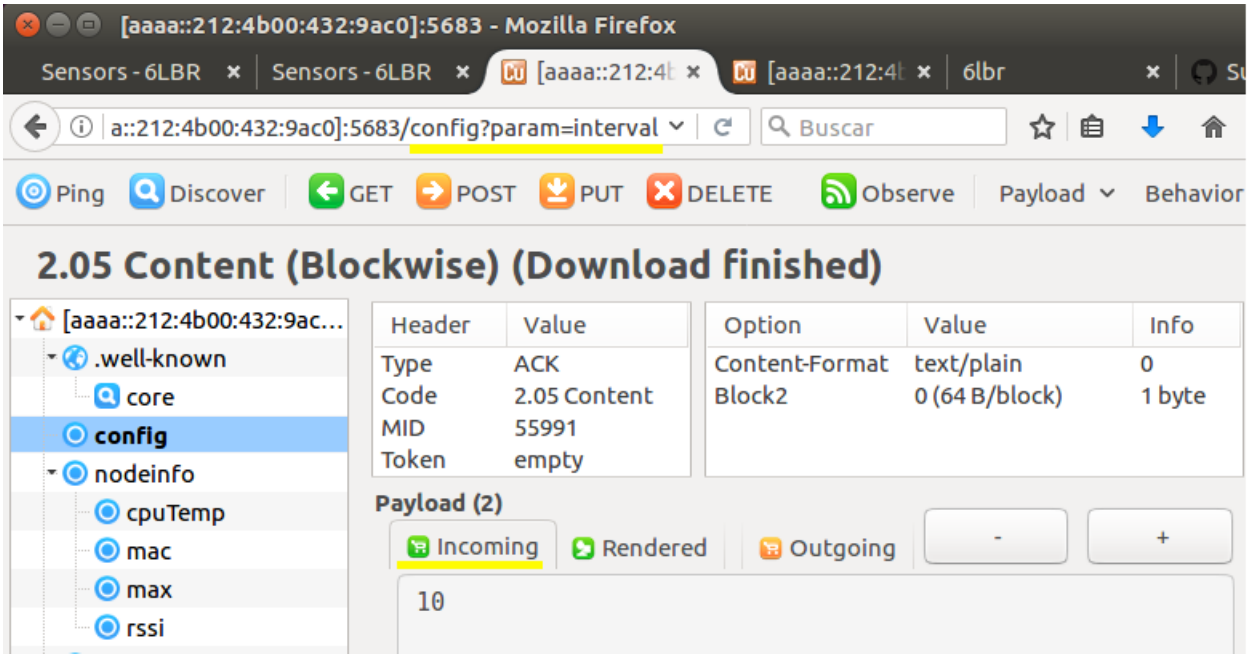


Figura 6.56: Ajuste sobre parámetro (POST) interval a 10 segundos

En la siguiente figura se realiza un GET sobre la variable interval para comprobar que se ha establecido un nuevo periodo correctamente.

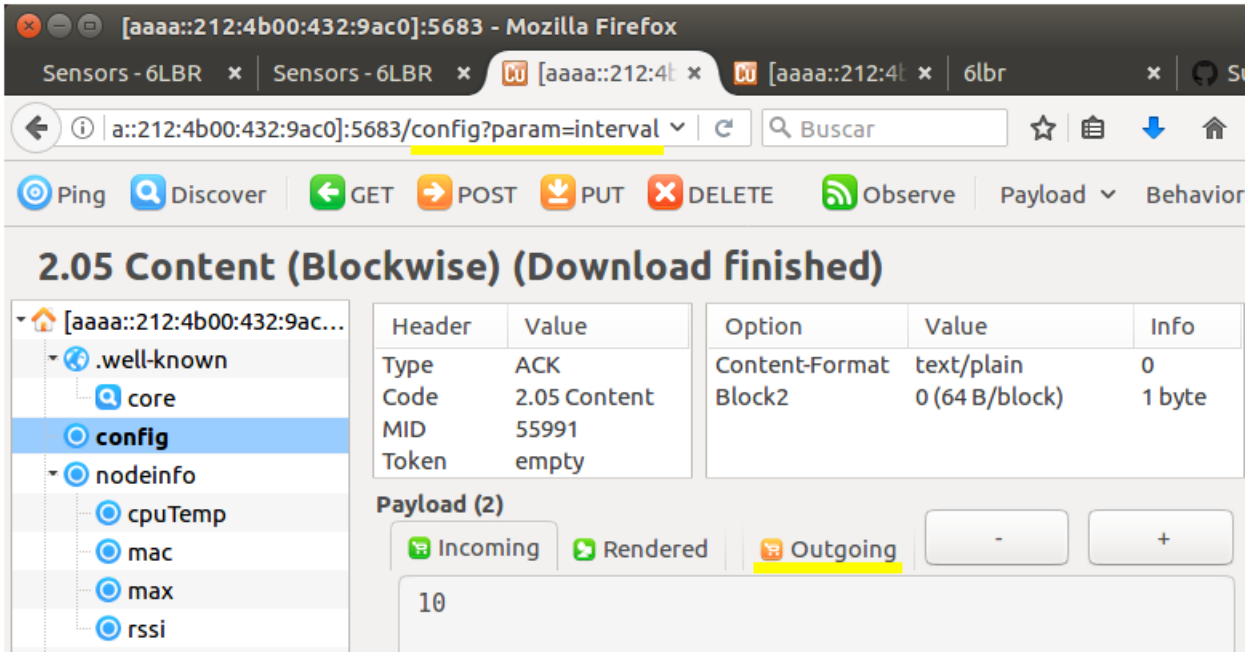


Figura 6.57: Consulta (GET) sobre parámetro interval a 10 segundos

6.2.3 Nodo Sumidero: Pruebas servidor CoAP

Para realizar pruebas con el servidor CoAP del nodo sumidero se conectaron varios nodos cliente que envían periódicamente datos al nodo sumidero, si el servidor CoAP funciona correctamente éste muestra los datos recibidos en la terminal que ejecuta el servidor CoAP codificados en JSON y luego decodificados. Como se puede observar en la siguiente figura, el servidor CoAP recibía y decodificaba correctamente los mensajes de los nodos.

```
pi@raspberrypi:~/COAP $ ./example/coap-srv -A bbbb::200 -v 3
ENTRO EN DATA POST HANDLER
RES = 1
SIZE: 42

DATOS:
{"MAC":"4B:00:06:0D:9F:18","max6675":"27"}

MAC: 4B:00:06:0D:9F:18
MAX TEMP: 27º
ENTRO EN DATA POST HANDLER
RES = 1
SIZE: 52
```

Figura 6.58: Servidor CoAP recibe mensaje de nodo y es decodificado

Para realizar las pruebas de detección de una **alertaMax**, se calentó uno de los termopares tipo K de uno de los sensores MAX6675 con un mechero para alcanzar una temperatura superior a los 50º centígrados a los que están configurados por defecto los nodos cliente para alertar. En otras pruebas en lugar de calentar el termopar tipo K, se reducía el parámetro del nodo cliente para que se activara la alarma con la temperatura ambiente (unos 23º centígrados).

Como se puede ver en la siguiente figura, el servidor CoAP reconocía correctamente una alarma e iniciaba el proceso de envío del SMS reenviando el JSON al microservicio de SMS.

```
DATOS:
{"MAC":"4B:00:06:0D:9F:18","alertaMAX":"61"}

MAC: 4B:00:06:0D:9F:18
ALERTA!!! MAX TEMP: 61º
Creando conexión con microservicio de alertas SMS
(SEND sd = 4 CONNECT 192.168.0.200, 8000)
connected
Conectado con microservicio SMS, enviando JSON...
(SEND sd = 4) >>>>>>>>>>>>>>>>>>>> (0x7b, 0x22, 0x4d, 0x41, 0x43, 0x22, 0x3a, 0x
22, 0x34, 0x42, 0x3a, 0x30, 0x30, 0x3a, 0x30, 0x36, 0x3a, 0x30, 0x44, 0x3a, 0x39
, 0x46, 0x3a, 0x31, 0x38, 0x22, 0x2c, 0x22, 0x61, 0x6c, 0x65, 0x72, 0x74, 0x61,
0x4d, 0x41, 0x58, 0x22, 0x3a, 0x22, 0x36, 0x31, 0x22, 0x7d, )
        ----->Buffer ENVIADO:
{"MAC":"4B:00:06:0D:9F:18","alertaMAX":"61"}

JSON enviado...
(RECV CLOSED CONNECTION sd = 4)
```

Figura 6.59: Servidor CoAP recibe mensaje de nodo, es decodificado como alertaMAX y conecta con microservicio

En la siguiente figura a la izquierda se puede observar como el microservicio de SMS recibe el JSON reenviado por el servidor CoAP, lo decodifica, genera el contenido del SMS y posteriormente lo envía. En la figura de la derecha se puede observar el SMS recibido al número de móvil configurado en **/etc/telf.conf**.

```
pi@raspberrypi:~ $ ./SMS_Service/simpc
Waiting connection accept
                        Current TIME: 1500571920
synch_serversock_recv_buf: data received
Read 44 bytes
synch_serversock_recv_buf: data received
(RECV) read 0 bytes (remote peer disconnected)
(RECV CLOSED ACCEPTED CONNECTION)
RECIBO BUFF
----->Buffer recibido:
{"MAC":"4B:00:06:0D:9F:18","alertaMAX":"84"}
        -->CONTENIDO SMS:
ALERTA
NODO: 4B:00:06:0D:9F:18
Temperatura alcanzada: 84 grados centigrados
```

Figura 6.60: Microservicio recibe JSON reenviado por servidor CoAP, genera y envía SMS

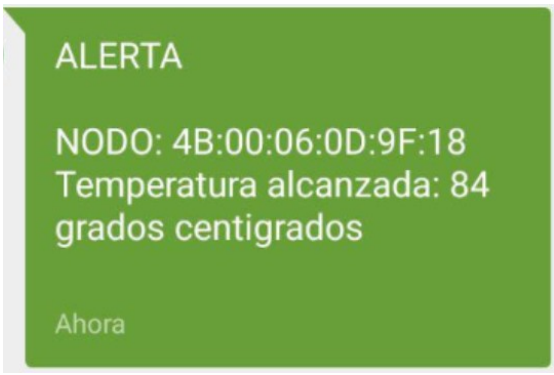


Figura 6.61: Mensaje SMS generado por microservicio recibido en teléfono móvil configurado

6.2.4 Pruebas a campo abierto

El punto de partida de estas pruebas de campo eran tres nodos cliente, un OpenMote alimentado con una powerbank y dos Openchina alimentados a pilas. Uno de los Openchina tenía conectada una antena de alta ganancia mientras que el otro funcionaba con su antena integrada. Como Border Router se utilizó un OpenChina conectado a una OpenBase para aprovechar que dispone del extensor de alance CC2592. Ver *Anexo Nodos X: Adaptar Border Router para Openchina utilizando ENC28J60* para más detalles de su configuración.

Figura 6.62: Border Router (Openchina) conectado a antena de alta ganancia

En las siguientes figuras se pueden observar los nodos Openchina utilizados en la prueba, la figura izquierda es el Openchina con antena de alta ganancia mientras que la figura derecha es el Openchina que utiliza su antena integrada.

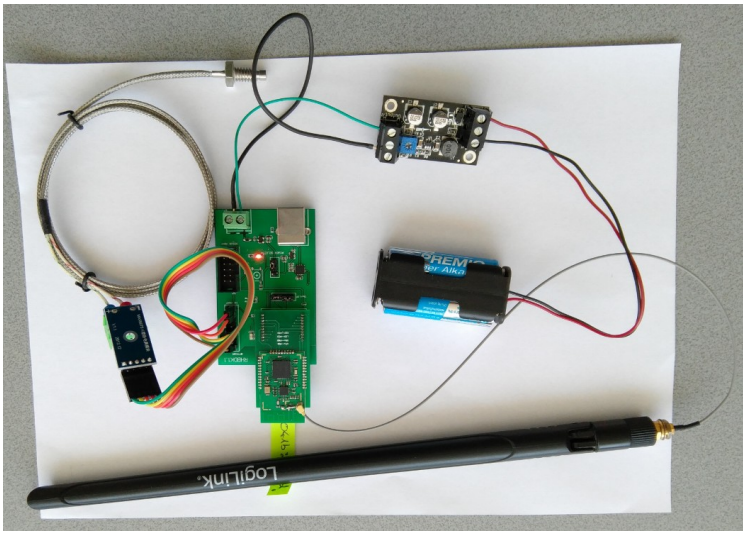


Figura 6.63: Nodo Openchina alimentado a pilas con antena de alta ganancia

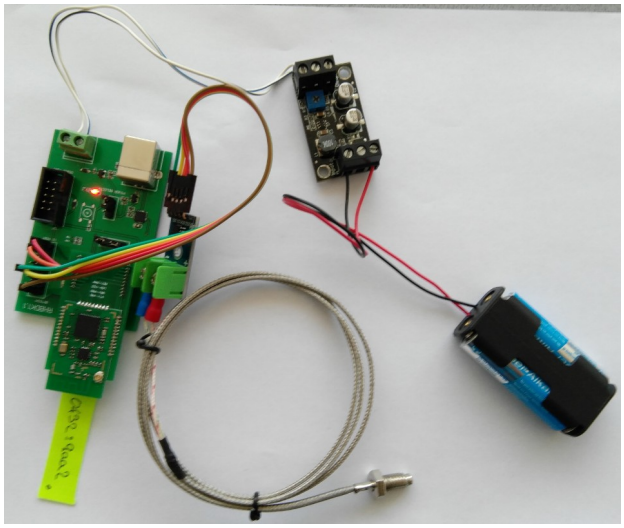


Figura 6.64: Nodo Openchina alimentado a pilas utilizando antena integrada

Para las pruebas de alcance en un entorno real tuve la ayuda del supervisor del proyecto, yo estaba en la oficina monitorizando los nodos que él se llevo para realizar la prueba, estábamos en contacto mediante una llamada telefónica.

Para verificar la conectividad de los nodos con el Border Router, se realizaban ping a cada uno de los nodos, se consultaba la página web del Border Router para ver cuanto tiempo hacía de su última conexión y se realizaban peticiones de temperatura del sensor MAX6675 utilizando el cliente Copper.

Mientras los nodos estaban en movimiento había variaciones de tiempo en sus respuestas al ping y a las peticiones CoAP. Una vez los nodos se quedaban quietos, la conexión se estabilizaba. Cabe recordar que el sistema no está pensado para que los nodos cliente se muevan sino que se ubican en su posición óptima y de allí no se mueven. Pero es destacable que incluso en movimiento, los nodos seguían respondiendo.

Cuando los nodos sin antena de alta ganancia no alcanzaban al Border Router inmediatamente pasaban a utilizar el nodo con antena de alta ganancia como salto.

Primero se realizó una prueba de visibilidad directa, llegando alcanzar los nodos una distancia de unos 500 metros antes de perder definitivamente la conectividad con el Border Router. Luego se realizó una prueba de visibilidad indirecta para ver que alcance tenían los nodos utilizando saltos.

Llegando a la conclusión de que mientras un nodo tenga visibilidad directa con otro, se alcanzan perfectamente distancias de 500 metros entre nodos, pero en el momento que existe un obstáculo la distancia alcanzable se reduce. La solución para los puntos ciegos, es ubicar un nodo cliente en ellos y de esta forma evitar el obstáculo.

Figura 6.65: Mapa de distancias alcanzadas en la prueba

En la figura anterior se pueden observar las distancias obtenidas durante la prueba de campo, el nodo verde es el Border Router. Los enlaces rojos detallados en verde indican visibilidad directa y los enlaces rojos detallados en rojo visibilidad indirecta con el Border Router. Los nodos rojos indican visibilidad indirecta y el nodo amarillo indica visibilidad directa con el Border Router.

El nodo amarillo (en la parte izquierda de la figura) está ubicado en un punto ciego dónde se perdía la visibilidad con el Border Router y permitiendo a los nodos rojos utilizarlo como salto hasta alcanzarlo. El nodo rojo que tiene un enlace de 589,60 metros, se dejó de seguir monitorizando en esa dirección (calle recta) debido a la conclusión comentada anteriormente de que mientras dos nodos se vean, la conectividad está asegurada por lo que se siguió por la Calle Dinamarca hasta perder la conectividad con el nodo amarillo por otro obstáculo. Llegados a este punto se decidió terminar con la prueba y comentar los resultados obtenidos.

6.2.5 Monitorización de la red mallada utilizando un Sniffer

Se programó uno de los OpenMote como sniffer utilizando uno de los ejemplos disponibles en los fuentes de Contiki. Un nodo programa en este modo permite se utilizado en un pc como interfaz de red WSN por USB, de esta forma utilizando el programa de monitorización de red WireShark y eligiendo como interfaz a monitorizar el OpenMote se puede capturar y analizar el tráfico de la red de nodos. Se realizó una captura de datos de la red de nodos durante unos 20 minutos iniciando la red desde cero para poder capturar los momentos iniciales de la red, mover los nodos, desconectar antenas y apagar uno de ellos para posteriormente poder visualizar las consecuencias.

Con los datos capturados, se utilizó la utilidad de Cetic Foren6[20] para visualizar gráfica como interaccionan los nodos de la red y las rutas alternativas que se generan.

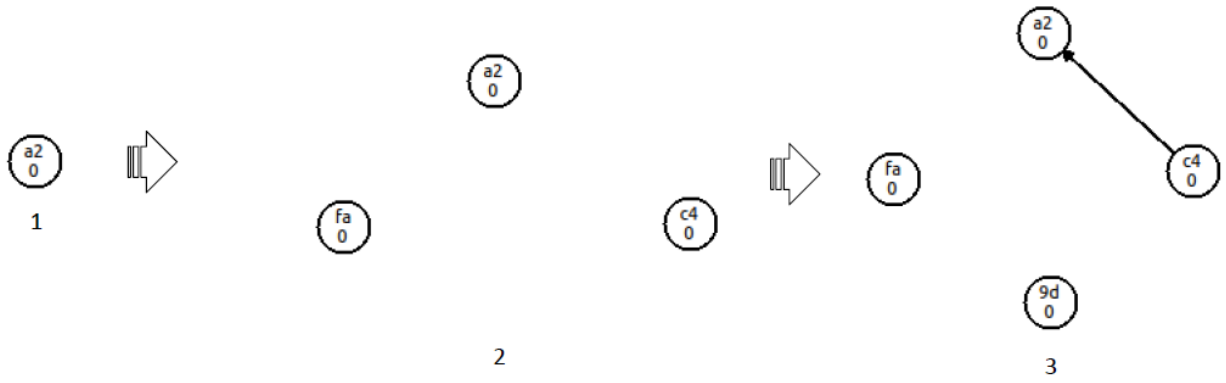


Figura 6.66: Foren6 Conectividad entre nodos (I)

Como se puede observar en la figura anterior, en un primer momento solamente aparece el nodo **a2** unos instantes más tarde (2) aparecen los nodos **fa** (el Border Router) y **a2**. Luego (3) aparece el nodo **c4** que se conecta al nodo **a2**.

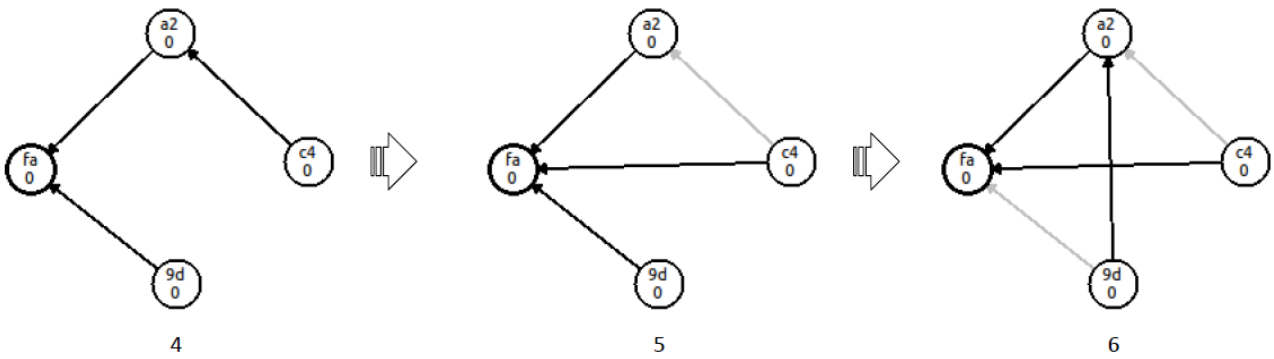


Figura 6.67: Foren6 Conectividad entre nodos (II)

En la figura anterior ya se puede observar como todos los nodos cliente alcanzan directa o indirectamente al nodo **fa** (4), en el punto 5 todos los nodos alcanzan directamente al nodo **fa** y el nodo **c4** tiene como ruta alternativa (enlace gris) al nodo **a2**. En el punto 6 el nodo **9d** utiliza el nodo **a2** para alcanzar el nodo **fa** el motivo de este cambio es que el nodo **a2** se cambió de ubicación para quedar en un punto central entre el Border Router y el resto de nodos.

20 Enlace Cetic Foren6: http://cetic.github.io/foren6/

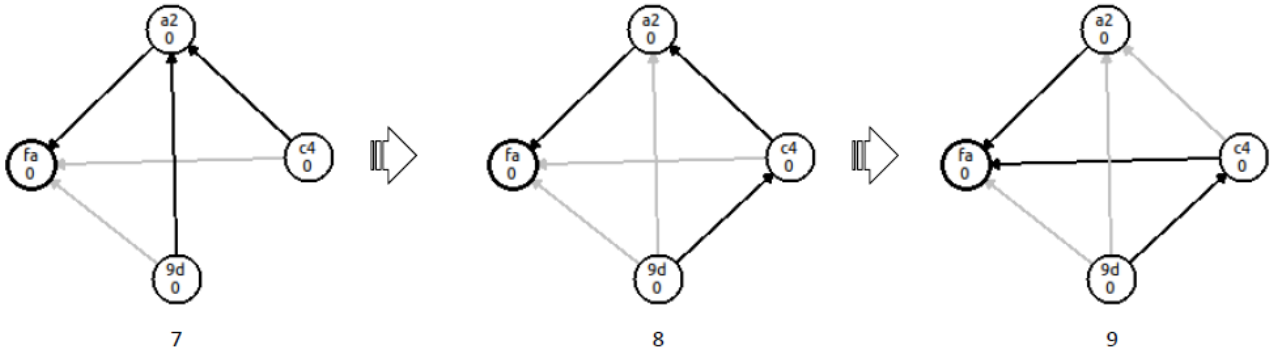


Figura 6.68: Foren6 Conectividad entre nodos (III)

En la figura anterior se puede observar como en el punto 7 el nodo **c4** también decide utilizar el nodo **a2** como salto, en el punto 8 el nodo **9a** conecta con el **c4** el motivo de este cambio es que al nodo **9d** se le desconectó la antena externa lo que redujo su alcance y su vecino más próximo era el nodo. En el punto 9, el nodo **c4** se ubicó al lado del nodo **a2** por lo que dejó de depender de él y se conectó directamente al Border Router.

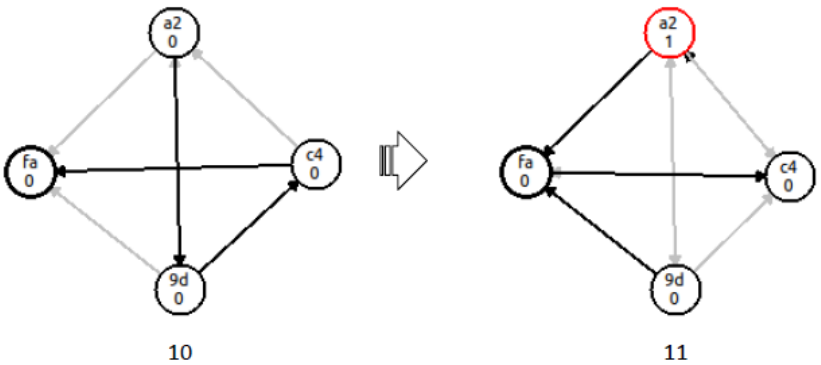


Figura 6.69: Foren6 Conectividad entre nodos (IV)

En la figura anterior, en el punto 10, el nodo **a2** utilizó al nodo **9d** como salto y finalmente en el punto 11 el nodo **a2** (resaltado en rojo) fue apagado.

Todas las figuras mostradas anteriormente pertenecen a un intervalo de captura de unos 8 minutos en los que era más destacable mostrar los cambios en la red.

Capítulo 7

Conclusiones

La versión actual de la red de monitorización posee la capacidad de mostrar los datos por pantalla o realizar un volcado de datos a un fichero de texto una vez recibidos en el nodo sumidero, al mismo tiempo permite enviar un SMS de alerta cuando uno de los nodos detecta una temperatura superior al máximo que tiene configurado. Además, los nodos cliente son capaces de recuperarse de forma autónoma ante desconexiones o pérdida de visión directa con el Border Router. Los nodos cliente son configurables de forma remota, se permite la consulta remota gracias a que implementan un servidor CoAP y envían mensajes periódicos al nodo sumidero. Por lo tanto se cumplen los objetivos del proyecto planteados por la empresa.

Sin embargo, en una futura revisión, se deberían corregir algunas carencias del sistema:

- Aprovechar la memoria flash que implementa el microprocesador CC2538, para almacenar los parámetros de configuración. Ahora mismo, si un nodo se desconecta y vuelve a iniciarse, carga los parámetros establecidos por defecto perdiendo los ajustes realizado por el usuario. Se debería guardar en la flash los ajustes cada vez que uno de los parámetros configurables es modificado y cargarlos al iniciar el nodo.

- Añadir seguridad a la red de nodos, actualmente no existe seguridad alguna en ella por lo que cualquiera puede monitorizar la red y capturar datos sin mucha complicación.

- Los datos enviados por los nodos al nodo sumidero actualmente no se almacenan en ningún sitio, se debería utilizar la base de datos postgresql que tiene Thermesys para su versión comercial.

- Integrar el sistema con la página web que tiene Thermesys implementada para su versión comercial que permite visualizar los datos recibidos por los sensores de forma gráfica seleccionando los datos a mostrar. Thermesys tiene prevista una nueva versión de la página web que además permita configurar los parámetros de los nodos individualmente o conjuntamente.

En lo que respecta al apartado personal, el desarrollo de este proyecto ha supuesto un reto, donde muchas de las tecnologías utilizadas han sido nuevas para mí pero a medida que me documentaba y comprendía su funcionamiento pude realizar pequeñas pruebas que me permitieron luego hacer las tareas del proyecto. He podido aplicar conocimientos vistos en mi itinerario de una forma práctica.

También he mejorado mi nivel de programación en C, donde tienes que ser mucho más cuidadoso de los recursos utilizados, ya que me he encontrado con problemas en tiempo de ejecución por no tratar correctamente el tipo de estructura de datos utilizado, ha sido muy entretenido poder utilizar tanta variedad de dispositivos hardware y hacer que se comuniquen entre ellos.

Estoy muy contento de haber podido terminar el proyecto y ver como todo el trabajo realizado durante la estancia de las practicas ha dado como resultado un sistema funcional.

Bibliografía

[1] Las redes de sensores inalámbricos y el Internet de las cosas: Revista INGE CUC,Volumen 8, Número 1, Octubre de 2012, pp. 163-172.
https://dialnet.unirioja.es/descarga/articulo/4869014.pdf7 [Consulta Julio 2017]

[2] IPv6 @ UJI - Luís Peralta - 1 de octubre de 2002
http://docplayer.es/3634156-Ipv6-uji-luis-peralta-http-spisa-act-uji-es-peralta-ipv6.html [Consulta Julio 2017]

[3] IoT in 5 days: an easy guide to Wireless Sensor Networks (WSN), IPv6 and the Internet of Things (IoT)
https://github.com/marcozennaro/IPv6-WSN-book [Consulta Julio 2017]

[4] SIM800 Series_AT Command Manual_V1.05
https://www.itead.cc/wiki/images/6/6f/SIM800_Series_AT_Command_Manual_V1.05.pdf [Consulta Julio 2017]

[5] CC2538 System-on-Chip Solution for 2.4-GHz IEEE 802.15.4 and ZigBe/ZigBee IP Applications
http://www.ti.com/lit/ug/swru319c/swru319c.pdf [Consulta Julio 2017]

[6] AN130 - Using CC2592 Front End With CC2538
http://www.ti.com/lit/an/swra447/swra447.pdf [Consulta Julio 2017]

[7] MAXIM Cold-Junction-Compensated K-Thermocouple-to-Digital Converter
https://cdn-shop.adafruit.com/datasheets/MAX6675.pdf [Consulta Julio 2017]

[8] Contiki Wiki github
https://github.com/contiki-os/contiki/wiki [Consulta Julio 2017]

[8] Cetic 6LBR Wiki github
https://github.com/cetic/6lbr/wiki [Consulta Julio 2017]

Anexos

Anexo Nodos I: Preparación del entorno de desarrollo

Para preparar el entorno de desarrollo es necesario un PC con un sistema Linux instalado, en este caso se ha comprobado su correcto funcionamiento en los siguientes sistemas operativos:

- Ubuntu 14.04.5 LTS
- Ubuntu 16.04.2 LTS

En este caso se prepara el entorno de desarrollo para el microcontrolador CC2538.

Instalación de paquetes necesarios

Solamente es necesario instalar los siguientes paquetes:

```
sudo apt-get install gcc-arm-none-eabi
sudo apt-get install gdb-arm-none-eabi
sudo apt-get install libnewlib-arm-none-eabi
sudo apt-get install libstdc++-arm-none-eabi-newlib
```

Al instalar estos paquetes ya podemos compilar y debug de programas para el CC2538.

Descarga de Contiki

Para empezar a desarrollar con el sistema operativo Contiki, debemos clonar el repositorio a un directorio de nuestra estación de trabajo:

```
git clone https://github.com/contiki-os/contiki.git
```

Compilación de un ejemplo

Accedemos al directorio **Contiki/examples/hello-world**, estos ejemplos están listos para ser compilados. En este caso se va a realizar la compilación para la placa openmote.

Ejecutamos el siguiente comando:

```
Make TARGET=openmote-cc2538
```

```
victor@victor-VPCEB1S8E:~/Thermesys/contiki/examples/hello-world$ make TARGET=openmote-cc2538
mkdir obj_openmote-cc2538
  CC      ../../cpu/cc2538/./ieee-addr.c
  CC      ../../cpu/cc2538/cc2538.lds
  CC      ../../cpu/cc2538/./startup-gcc.c
```

```
  CC       ../../platform/openmote-cc2538/./contiki-main.c
….....................................................................................
CC       ../../core/net/mac/nordc.c
  CC       ../../core/net/mac/nullmac.c
  CC       ../../core/net/mac/nullrdc.c
  CC       ../../core/net/mac/nullrdc-noframer.c
  CC       ../../core/net/mac/phase.c
  CC       ../../core/net/mac/contikimac/contikimac.c
  CC       ../../core/net/mac/contikimac/contikimac-framer.c
  CC       ../../core/net/llsec/anti-replay.c
  CC       ../../core/net/llsec/ccm-star-packetbuf.c
  CC       ../../core/net/llsec/nullsec.c
  CC       ../../core/net/llsec/noncoresec/noncoresec.c
  CC       hello-world.c
  LD       hello-world.elf
arm-none-eabi-objcopy -O ihex hello-world.elf hello-world.hex
arm-none-eabi-objcopy -O binary --gap-fill 0xff hello-world.elf hello-world.bin
cp hello-world.elf hello-world.openmote-cc2538
rm obj_openmote-cc2538/startup-gcc.o hello-world.co
```

Tras unos segundos o minutos dependiendo del código que se esta compilando, obtendremos los binarios compilados con varias extensiones.

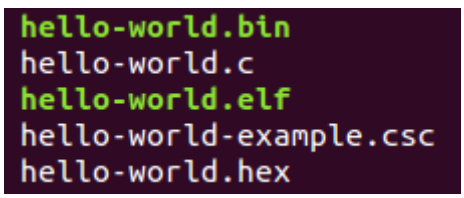


Figura AN I.70: Binarios generados

El fichero en formato .bin se puede programar utilizando la utilizad “cc2538-bsl”. (Ver Anexo V: Programación de nodos por USB) y el fichero en formato .elf se puede programar utilizando el puerto JTAG (Ver *Anexo IV: Programación de nodos por JTAG*).

Para limpiar nuestra compilación debemos ejecutar el comando:

```
Make TARGET=openmote-cc2538 clean
```

De esta forma se eliminan objetos generados para la compilación del ejemplo y binarios generados:

Antes:

```
victor@victor-VPCEB1S8E:~/Thermesys/contiki/examples/hello-world$ ls -l
total 1348
-rwxrwxr-x 1 victor victor 516096 jul  4 17:50 hello-world.bin
-rw-rw-r-- 1 victor victor   2307 jun  8 19:19 hello-world.c
-rwxrwxr-x 1 victor victor 158303 jul  4 17:50 hello-world.elf
-rw-rw-r-- 1 victor victor   5708 jun  8 19:19 hello-world-example.csc
-rw-rw-r-- 1 victor victor 125560 jul  4 17:50 hello-world.hex
-rwxrwxr-x 1 victor victor 158303 jul  4 17:50 hello-world.openmote-cc2538
-rw-rw-r-- 1 victor victor 449407 jul  4 17:50 hello-world-openmote-cc2538.map
-rw-rw-r-- 1 victor victor    107 jun  8 19:19 Makefile
drwxrwxr-x 2 victor victor  12288 jul  4 17:50 obj_openmote-cc2538
-rw-rw-r-- 1 victor victor   1231 jun  8 19:19 README.md
```

Figura AN I.71: Directorio programa compilado antes de MAKE clean

Después:

```
victor@victor-VPCEB1S8E:~/Thermesys/contiki/examples/hello-world$ ls -l
total 20
-rw-rw-r-- 1 victor victor 2307 jun  8 19:19 hello-world.c
-rw-rw-r-- 1 victor victor 5708 jun  8 19:19 hello-world-example.csc
-rw-rw-r-- 1 victor victor  107 jun  8 19:19 Makefile
-rw-rw-r-- 1 victor victor 1231 jun  8 19:19 README.md
```

Figura AN I.72: Directorio programa compilado después de MAKE clean

Anexo Nodos II: Estructura de directorios Contiki

En este anexo se describe la estructura de directorios del los fuentes de Contiki:

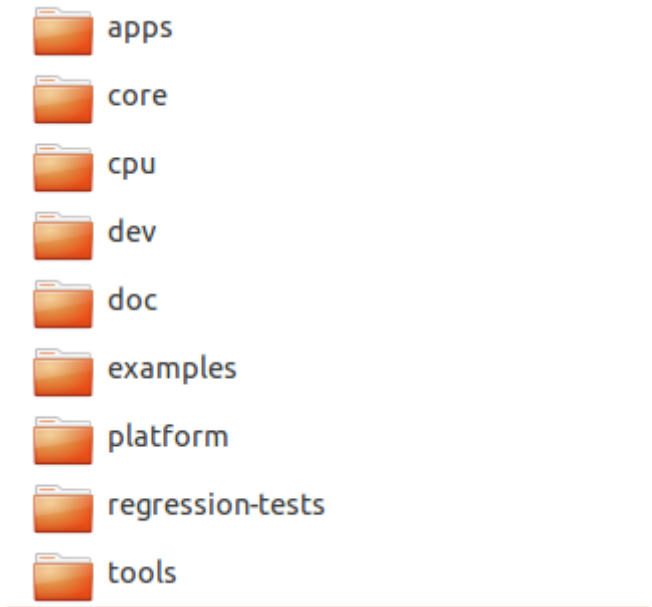


Figura AN II.73: Estructura de directorios código fuente Contiki

- **apps**: Contiene aplicaciones implementadas que se pueden utilizar en el programa principal.
- **core**: Contiene el core del sistema operativo Contiki. En este directorio se pueden encontrar la implementación de los protothreads, los timers, uIP, Rime, RPL y todo lo que se refiere al sistema operativo.
- **cpu**: En este directorio se encuentra todo el código que depende del microcontrolador. Como se implementa la flash, los timers, los modos de energía, etc.
- **dev**: Contiene implementaciones de driver para varios sensores.
- **doc**: contiene todo lo referente a la documentación del código.
- **examples**: contiene códigos de ejemplo que utilizan el sistema operativo Contiki.
- **platform**: contiene las definiciones específicas de cada plataforma soportada por Contiki como la configuración de los botones, leds, sensores integrados, etc.
- **tools**: En este directorio se encuentran varias herramientas que no forma parte del sistema operativo Contiki como el simulador de nodos COOJA y el script de programación cc2538-bsl utilizado en el proyecto.

Anexo Nodos III: Añadir nuevo platform a Contiki

En este anexo se va a explicar como se ha añadido el platform OpenChina definido en el proyecto a los fuentes de contiki.

Como la placa Openchina comparte microcontrolador con el OpenMote, se ha utilizado el platform/openmote como base para generar el platform “openchina”.

Primero de todo se ha duplicado el directorio contiki/platform/openmote como contiki/platform/openchina.

Contenido del directorio Openchina actual:

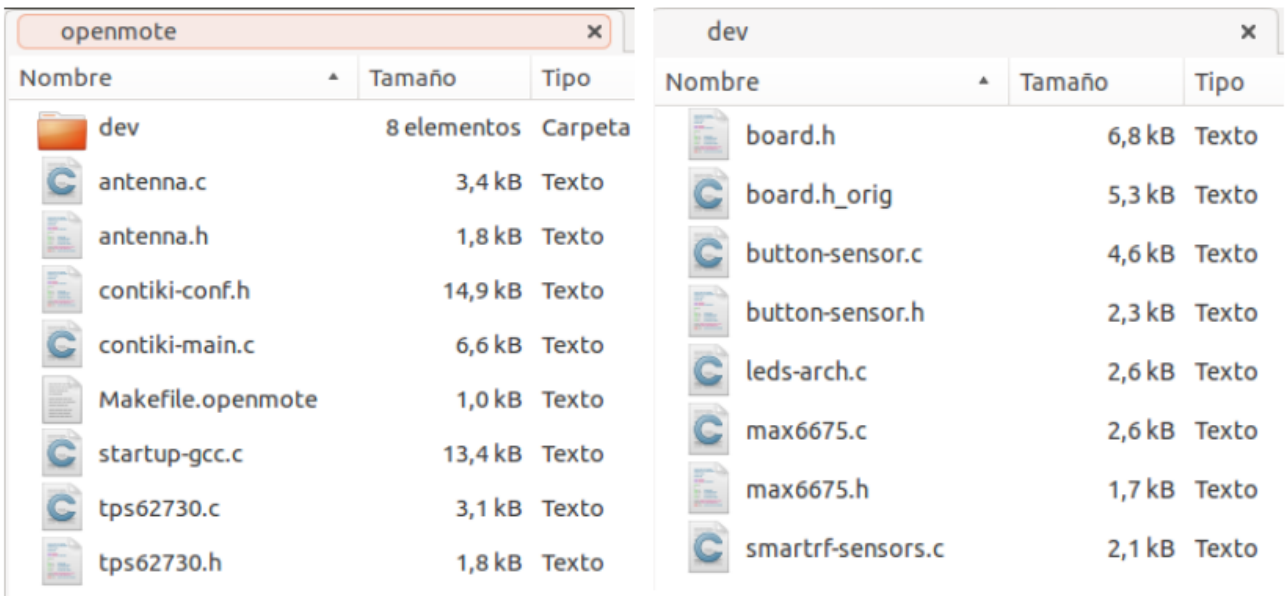


Figura AN III.74: Estructura platform Openchina

El fichero que se debe adaptar es: /dev/board.h, en este fichero se definen los pines y puertos del microcontrolador asignados a: protocolos, leds, botones, etc.

En este caso se han editado los pines correspondientes a los puertos SPI, se ha definido un flag “SPIX_IN_USE, se han eliminado las definiciones de los led que no tiene y se ha renombrado la placa a “OpenChina-CC2538-CC2592”.

Para poder establecer los pines y puertos, se ha consultado el datasheet proporcionado por el fabricante de la placa Openchina. (Ver *Anexo Nodos IX: Pines placa OpenChina*)

Se han definido los pines y puertos del puerto SPI/SSI y su flag de activación, de esta forma solamente se van a activar los pines si es necesario utilizarlos. El flag de activación se define en el fichero “project-conf.h” de cada programa o en el momento de realizar la compilación se puede añadir el flag a activar.

SPI/SSI0

```
#ifndef SPI0_IN_USE
#define SPI0_IN_USE             0
#endif
#if SPI0_IN_USE


#define SPI_CLK_PORT            GPIO_C_NUM
#define SPI_CLK_PIN             4
#define SPI_MOSI_PORT           GPIO_A_NUM
#define SPI_MOSI_PIN            7
#define SPI_MISO_PORT           GPIO_A_NUM
#define SPI_MISO_PIN            5

#endif
```

Figura AN III.75: Definición pines SPI/SSI0 y flag de activación en board.h

SPI/SSI1

```
#ifndef SPI1_IN_USE
#define SPI1_IN_USE             0
#endif

#if SPI1_IN_USE


#ifndef SPI1_CLK_PORT
#define SPI1_CLK_PORT           GPIO_C_NUM
#endif
#ifndef SPI1_CLK_PIN
#define SPI1_CLK_PIN            4
#endif
#ifndef SPI1_TX_PORT
#define SPI1_TX_PORT            GPIO_C_NUM
#endif
#ifndef SPI1_TX_PIN
#define SPI1_TX_PIN             3
#endif
#ifndef SPI1_RX_PORT
#define SPI1_RX_PORT            GPIO_C_NUM
#endif
#ifndef SPI1_RX_PIN
#define SPI1_RX_PIN             7
#endif

#endif
```

Figura AN III.76: Definición pines SPI/SSI1 y flag de activación en board.h

Con estos cambios, ya se pueden compilar programas utilizando el comando “Make TARGET=openchina”. Y utilizar los mismos programas que usan el sensor MAX6675 en los OpenMote.

Aparte de esta modificación, para poder aprovechar el amplificador de señal CC2592 se ha modificado el fichero “contiki/cpu/cc2538/dev/cc2538-rf.c” para añadir los ajustes especificados por Texas Instruments. (Ver *Anexo Nodos VII: Añadir compatibilidad en Contiki para el extensor de alcance CC2592*).

Anexo Nodos IV: Programación de nodos por JTAG

Instalar software proporcionado por el fabricante del J-Link, en este caso se ha utilizado Ubuntu 16.04 y Ubuntu 14.04.

```
JLink_Linux_V614b_x86_64.deb
```

Antes de empezar con la programación, debemos conectar el nodo a programar por su puerto JTAG al J-link como se muestra en la siguiente figura.

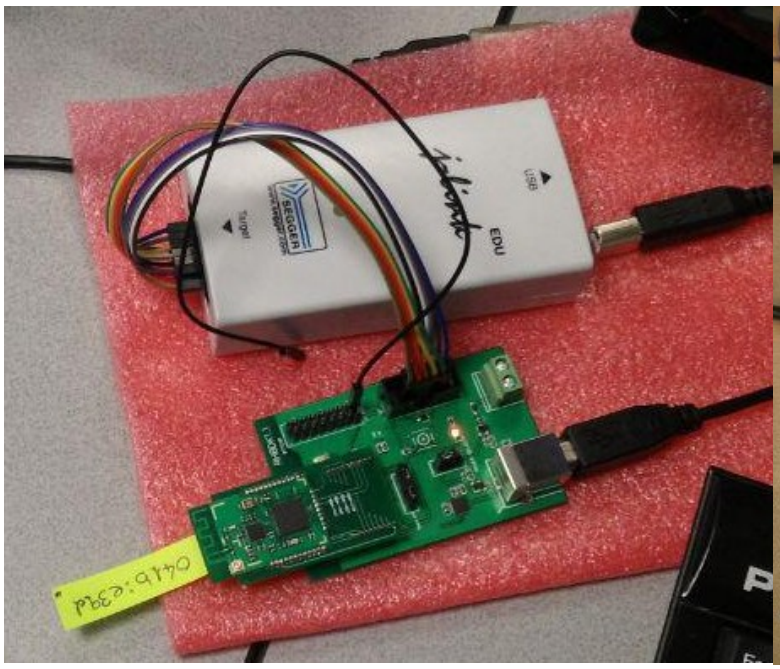


Figura AN IV.77: Openchina conectado a JLink

Figura AN IV.78: OpenMote conectado a JLink

Abrimos una terminal para iniciar el servidor JlinkGDBServer:

```
JLinkGDBServer -device CC2538SF53
```

Una vez iniciado el servidor, abrimos otra terminal y ejecutamos:

```
arm-none-eabi-gdb
```

Con esta terminal abierta podemos programar el nodo CC2538 conectado al programador J-link utilizando esta secuencia de comandos:

```
target remote localhost:2331
monitor interface jtag
monitor speed 1000
monitor endian little
monitor flash download = 1
monitor flash breakpoints = 1
monitor reset
```

```
load ~/projects/chn_6lbr_dev-1.4.x/examples/er-rest-client/er-example-server.elf
```

Una vez ejecutado load, podremos reiniciar el nodo programado y comprobar su correcto funcionamiento.

Anexo Nodos V: Programación de nodos por USB

Contiki incluye en su directorio contiki/tools/cc2538-bsl un script escrito en python que permite programar los nodos con el microcontrolador CC2538 de forma sencilla utilizando los ficheros compilados con extensión .bin.

```
victor@victor-VPCEB1S8E:~/Thermesys/6lbr/examples/6lbr/bin_openmote$ sudo ~/Ther
mesys/cc2538-bsl/cc2538-bsl.py -p /dev/ttyUSB0 -ewv cetic_6lbr_router.bin
Opening port /dev/ttyUSB0, baud 500000
Reading data from cetic_6lbr_router.bin
Connecting to target...
    Target id 0xb964, CC2538
Erasing 524288 bytes starting at address 0x200000
    Erase done
Writing 524288 bytes starting at address 0x200000
    Write done
Verifying by comparing CRC32 calculations.
    Verified (match: 0xa1f96c8b)
```

Figura AN V.79: Programación nodo satisfactoria

-p: especifica el puerto serie asignado al dispositivo que queremos programar.
-e: borra el dispositivo.
-w: escribe en el dispositivo el binario especificado como último argumento.

Si la programación por usb falla, es posible que no se haya realizado correctamente el puente a gnd o vcc del pin que activa el bootloader backdoor o que accidentalmente se haya cerrado el bootloader backdoor al programar alguno de los ejemplos de contiki. (Ver *Anexo Nodos VI: Activar Bootloader Backdoor*)

```
victor@victor-VPCEB1S8E:~/Thermesys/6lbr/examples/6lbr/bin_openmote$ sudo ~/Ther
mesys/cc2538-bsl/cc2538-bsl.py -p /dev/ttyUSB0 -ewv cetic_6lbr_router.bin
Opening port /dev/ttyUSB0, baud 500000
Reading data from cetic_6lbr_router.bin
Connecting to target...
ERROR: Can't connect to target. Ensure boot loader is started. (no answer on syn
ch sequence)
```

Figura AN V.80: Programación nodo fallida

Anexo Nodos VI: Activar Bootloader Backdoor

Al activar el flag del bootloader backdoor, podemos programar las Openchina por el puerto usb.

Simplemente debemos compilar alguno de los ejemplos de contiki para el OPENMOTE, de esta forma:

make TARGET=openmote FLASH_CCA_CONF_BOOTLDR_BACKDOOR=1

Al activar este flag, se ejecuta el código ubicado en /platform/openmote/startup-gcc.c que activa el backdoor al pin de la placa PA6 a nivel bajo (GND), en el caso de las Openchina nos sirve tal como esta ese código porque tenemos acceso a ese pin.

Si en la placa a activar, no tenemos acceso a ese pin, simplemente debemos editar el fichero "contiki-conf.h" ubicado dentro de platform/openmote/ con el pin que deseemos. O definir los pines en el momento de ejecutar el Make.

Contenido del fichero:

```
/** @} */
/*---------------------------------------------------------------------------*/
/**
 * \name Serial Boot Loader Backdoor configuration
 *
 * @{
 */
#ifndef FLASH_CCA_CONF_BOOTLDR_BACKDOOR
#define FLASH_CCA_CONF_BOOTLDR_BACKDOOR         1 /**<Enable the boot loader backdoor */
#endif

#ifndef FLASH_CCA_CONF_BOOTLDR_BACKDOOR_PORT_A_PIN
#define FLASH_CCA_CONF_BOOTLDR_BACKDOOR_PORT_A_PIN 6 /**< Pin PA_6 (On/Sleep) activates the boot loader */
#endif

#ifndef FLASH_CCA_CONF_BOOTLDR_BACKDOOR_ACTIVE_HIGH
#define FLASH_CCA_CONF_BOOTLDR_BACKDOOR_ACTIVE_HIGH 0 /**< A logic low level activates the boot loader */
#endif
/** @} */
```

Una vez compilado, lo cargamos mediante el jlink a la placa y ya podremos utilizar el cable usb para programar los motes. (Ver *Anexo Nodos IV: Programación de nodos por JTAG*)

Anexo Nodos VII: Añadir compatibilidad en Contiki para el extensor de alcance CC2592

Por defecto los fuentes de Contiki no tienen implementado el extensor de alcance CC2592, para añadir esta compatibilidad a los fuente de Contiki que se han utilizado para la realización del proyecto se ha consultado el fichero “Using_CC2592_Front_End With_CC2538” de Texas Instruments y el datasheet proporcionado por el fabricante de los OpenChina.

Estos ficheros especifican por que pines del microcontrolador CC2538 esta conectado el CC2592 y como activar sus distintos modos de funcionamiento.

El fichero que se ha editado para añadir el uso del CC2592 es: **contiki/cpu/cc2538/dev/cc2538-rf.c**

Se ha añadido el siguiente código para que se pueda controlar el CC2592:

```
#if CC2592_EN

/*CC2538 Registers for CC2592 Control */
/*https://github.com/retfie/contiki/commit/c6ca74d8740b9b41b0dbf7a44e7abd74fed0704b*/
/
*https://github.com/retfie/contiki/blob/c6ca74d8740b9b41b0dbf7a44e7abd74fed0704b/cpu/cc2538/dev/cc2538-rf.c*/
/*---------------------------------------------------------------------------*/

/* RF observable control register value to output PA signal */
#define RFC_OBS_CTRL_PA_PD_INV       0x6A
/* RF observable control register value to output LNA signal */
#define RFC_OBS_CTRL_LNAMIX_PD_INV    0x68
// RF observable control register value to output LNA signalfor CC2591 compression workaround.
#define RFC_OBS_CTRL_DEMOD_CCA       0x0D
/* OBSSELn register value to select RF observable 0 */
#define OBSSEL_OBS_CTRL0           0x81
/* OBSSELn register value to select RF observable 1 */
#define OBSSEL_OBS_CTRL1           0x80
#define RFC_OBS_CTRL0 REG(RFCORE_XREG_RFC_OBS_CTRL0)
#define RFC_OBS_CTRL1 REG(RFCORE_XREG_RFC_OBS_CTRL1)
#define OBSSEL3   REG(CCTEST_OBSSEL3)
#define OBSSEL2   REG(CCTEST_OBSSEL2)
#define EN_LNA_PIN  (1 << 2)
#define PAEN_PIN    (1 << 3)
#define HGM_PIN     (1 << 2)

  GPIO_SET_OUTPUT(GPIO_C_BASE, PAEN_PIN);
  GPIO_WRITE_PIN(GPIO_C_BASE, PAEN_PIN, 1);
  GPIO_SET_OUTPUT(GPIO_C_BASE, EN_LNA_PIN);
  GPIO_WRITE_PIN(GPIO_C_BASE, EN_LNA_PIN, 1);
  // PD2 -> 0/1 -> High/Low Gain
  GPIO_SET_OUTPUT(GPIO_D_BASE, HGM_PIN);

  #if RxLow
  GPIO_WRITE_PIN(GPIO_D_BASE, HGM_PIN, 0);
  #endif
```

```
  #if RxHigh
  GPIO_WRITE_PIN(GPIO_D_BASE, HGM_PIN, 1);
  #endif

  // PC3 -> PAEN
  RFC_OBS_CTRL0 = RFC_OBS_CTRL_PA_PD_INV;
  OBSSEL3 = OBSSEL_OBS_CTRL0;
  // PC2 -> EN (LNA control)
  RFC_OBS_CTRL1 = RFC_OBS_CTRL_LNAMIX_PD_INV;
  OBSSEL2 = OBSSEL_OBS_CTRL1;
  //
  printf("CONFIGURO PINES cc2592\n");

#endif
```

En el inicio de este parche, están comentadas la fuentes del parche que ha sido adaptado para el correcto funcionamiento de las OpenChina.

Para poder activar o no el CC2592 y que los mismos fuentes de Contiki siguieran sirviendo para las plataformas OpenMote y OpenChina, se ha definido el flag “CC2592_EN ”, de esta forma se puede controlar que el parche solamente afecte a las placas OpenChina si esta especificado en el fichero “project-conf.h” del proyecto a compilar.

Otro dos flags han sido definido para utilizar los dos modos de funcionamiento del CC2592:

- RxLow: el CC2592 no actúa como extensor de señal, simplemente hace de puente sin amplificar la potencia de la señal.
- RXHigh: el CC2592 actúa como extensor de señal.

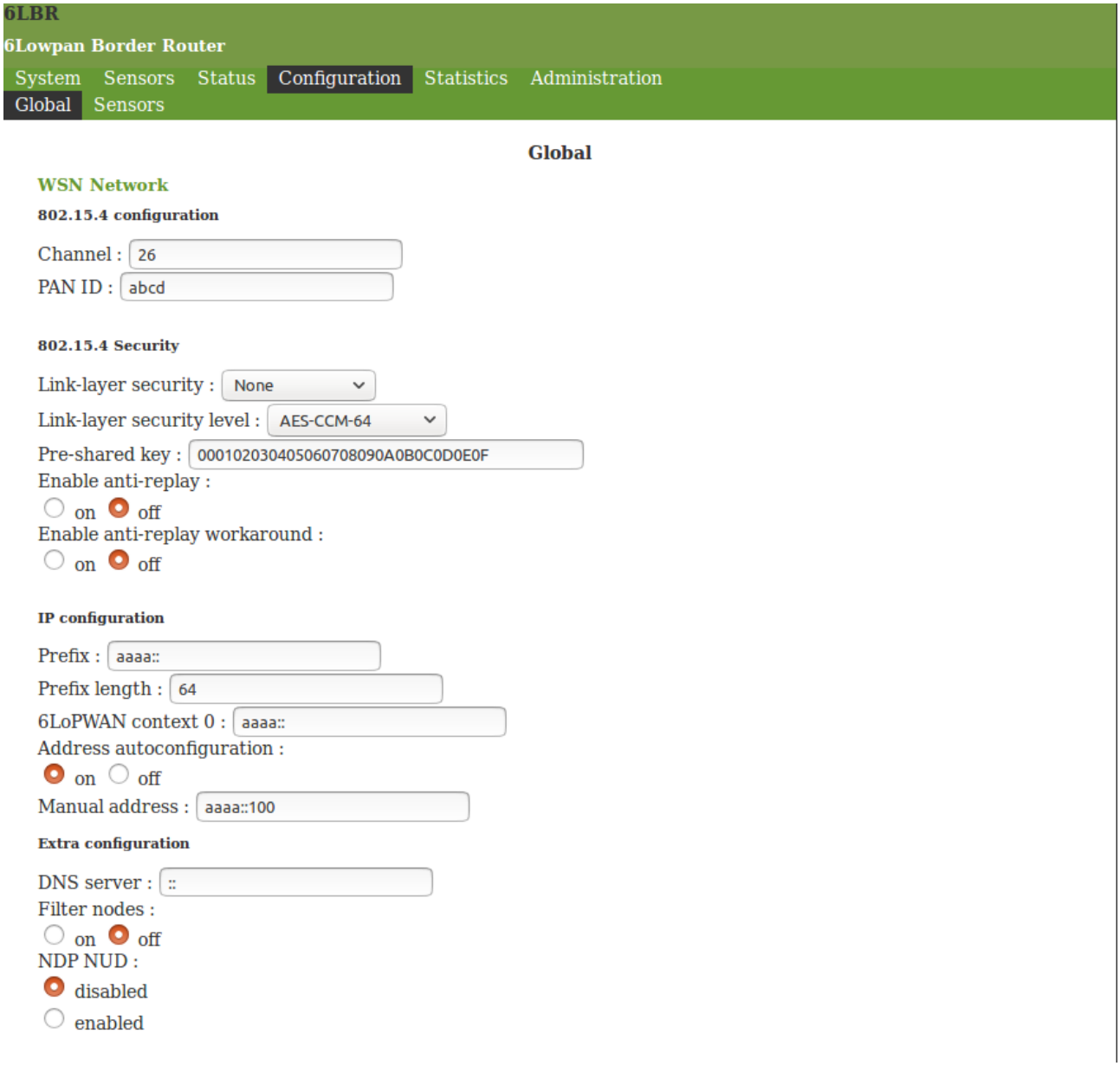
Ejemplo de “project-conf.h” para utilizar el CC2592:

```
/*---------------------------------------------------------------------------*/
/* OpenChina Board                                                           */
/*---------------------------------------------------------------------------*/
#if defined CONTIKI_TARGET_OPENCHINA

/*Enable CC2592*/

#define CC2592_EN 1

#define RxLow 1

#define RxHigh 0

#endif
```

En este caso, si la plataforma que se compila es OpenChina, se activa el CC2592 y se usa su modo de funcionamiento sin amplificar la señal.

Anexo Nodos VIII: Ajustes Border Router

En este anexo se adjuntan las capturas de pantalla que corresponden con los ajustes completos de la interfaz web del Border Router.



Eth Network

IP configuration

Prefix : bbbb::

Prefix length : 64

Address autoconfiguration :

○ on ◉ off

Manual address : bbbb::100

Peer router : ::

IP64

IP64 :

○ on ◉ off

DHCP :

○ on ◉ off

Address : 172.16.0.2

Netmask : 255.255.255.0

Gateway : 172.16.0.1

RFC 6052 prefix :

○ on ◉ off

Static port mapping :

○ on ◉ off

MDNS

MDNS publishing :

○ on ◉ off

Hostname : 6lbr

DNS-SD publishing :

◉ on ○ off

RA Daemon

RA Daemon :

◉ active

○ inactive

Router lifetime : 0

RA

Max interval : 600

Min interval : 200

Min delay : 3

RA Prefix

Send Prefix Information :
(•) on ( ) off
Prefix on-link :
(•) on ( ) off
Allow autoconfiguration :
(•) on ( ) off
Prefix valid time : 86400
Prefix preferred time : 14400

RA Route Information

Include RIO :
(•) on ( ) off
Route lifetime : 1800

RPL Configuration

Instance ID : 30
Manual DODAG ID :
( ) on (•) off
DODAG ID : ::
Global DODAG ID :
(•) on ( ) off
Preference : 0
DIO interval doubling : 8
DIO min interval : 12
DIO redundancy : 10
Min rank increment : 256
Max rank increase : 1792
Route lifetime : 30
Route lifetime unit : 256

Global configuration

Webserver :
( ) disabled
(•) enabled
UDP server :
( ) disabled
(•) enabled
DNS Proxy :
( ) disabled
(•) enabled

Submit

6LBR By CETIC (documentation)
This page sent 1 times (0.93 sec)

Anexo Nodos IX: Pines placa OpenChina

En la siguiente tabla se pueden ver los pines accesibles y en uso de la OpenChina junto a la placa de expansión RHBDK 1.1.

Pin IO Port	Pin OpenChina	Pin RHBDK1.1	Pin placa	Pin OpenChina	Pin RHBDK1.1
1	GND	GND	2	PB1	DIO0
3	PB2	DIO1	4	PB3	DIO2
5	PA0	UTx	6	PA1	URx
7	PD0	DIO14	8	PA7	DIO13
9	PA6	DIO12	10	PA5	DIO11
11	PC0	LED	12	PA4	KEY
13	PC4	DIO8	14	PC5	DIO7
15	VCC	VCC	16	GND	GND

Tabla AN IX.10: Pines placa OpenChina justo RHBDK 1.1

Anexo Nodos X: Adaptar Border Router para Openchina utilizando ENC28J60

Modificaciones realizadas en el código del Border Router para poder utilizar una OpenChina junto a una controladora Ethernet ENC28J60.

En **dev-1.4.x/examples/6lbr/platform/** se ha clonado el directorio openmote y ha sido renombrado a openchina, las modificaciones realizadas en los ficheros de ese directorio son las siguientes:

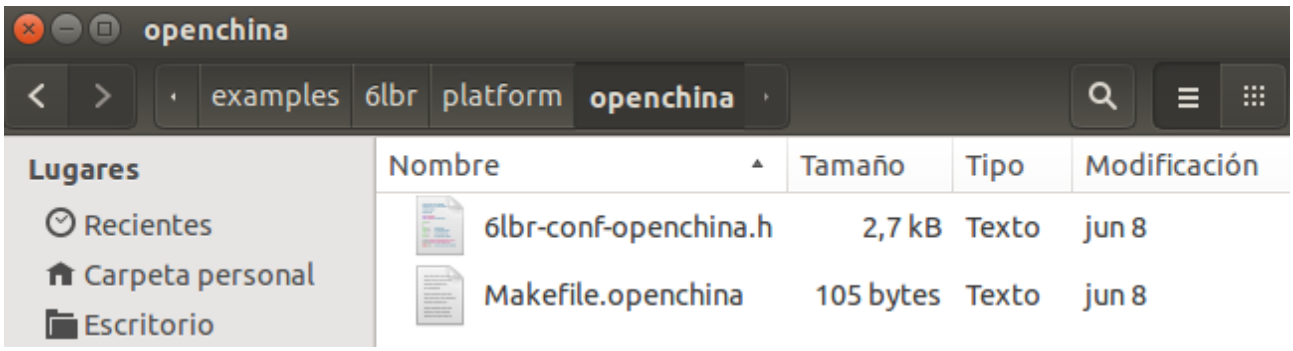


Figura AN X.81: Contenido directorio plataforma Border Router

El nombre openmote escrito en los ficheros ha sido cambiado a openchina, el fichero Makefile.openchina no ha sido modificado y en el fichero **6lbr-conf-openchina.h** se han realizado las siguientes modificaciones:

- Definido que puerto SPI que se va a utilizar (SSI0).
- Definidos los pines a utilizar para el ENC28J60.
- Activado el CC2592 en modo RxHigh .
- La disposición de los pines para el ENC28J60 es la siguiente:

ENC28J60	OpenChina	
Pin	Pin/Puerto CC2538	Pin/Puerto OpenChina
SCK	PC4	13
CS	PC5	14
MISO	PA7	8
MOSI	PA5	10

Tabla AN X.11: Definición pines Openchina utilizados para conectar ENC28J60

El contenido del fichero resultante es el siguiente:

```
/*---------------------------------------------------------------------*/
/* OpenChina 6LBR                                                      */
/*---------------------------------------------------------------------*/

/*Enable CC2592*/

#define CC2592_EN 1
#define RxLow 0
#define RxHigh 1


/*BUTTON USER VECTOR*/

#undef BUTTON_USER_VECTOR
#define BUTTON_USER_VECTOR      NVIC_INT_GPIO_PORT_A

/*SPI port 1 (SSI)*/

#undef SPI0_IN_USE
#define SPI0_IN_USE             1

/* Do not change lines below */

#define CC2538_ENC28J60_CONF_CLK_PORT GPIO_C_BASE
#define CC2538_ENC28J60_CONF_CLK_PIN 4

#define CC2538_ENC28J60_CONF_MOSI_PORT GPIO_A_BASE
#define CC2538_ENC28J60_CONF_MOSI_PIN 5

#define CC2538_ENC28J60_CONF_MISO_PORT GPIO_A_BASE
#define CC2538_ENC28J60_CONF_MISO_PIN 7

#define CC2538_ENC28J60_CONF_CS_PORT GPIO_C_BASE
#define CC2538_ENC28J60_CONF_CS_PIN 5
```

Figura AN X.82: Contenido fichero 6lbr-conf-openchina.h

Se ha utilizado el ENC28J60 integrado en una placa OpenBase para probar el correcto funcionamiento como Border Router.

La disposición de pines resultante ha sido la siguiente:

OpenBase		ENC28J60	RHBDK 1.1	
Pin/Puerto CC2538	Pin/Puerto OpenBase	Pin	Pin/Puerto CC2538	Pin/Puerto OpenChina
PA2	AD4/DIO4	SCK	PC4	13
PA3	CTS/DIO7	CS	PC5	14
PA4	AD5/DIO5	MISO	PA7	8
PA5	RTS/AD6/DIO6	MOSI	PA5	10

Tabla AN X.12: Pines OpenBase para conectar Openchina a ENC28J60

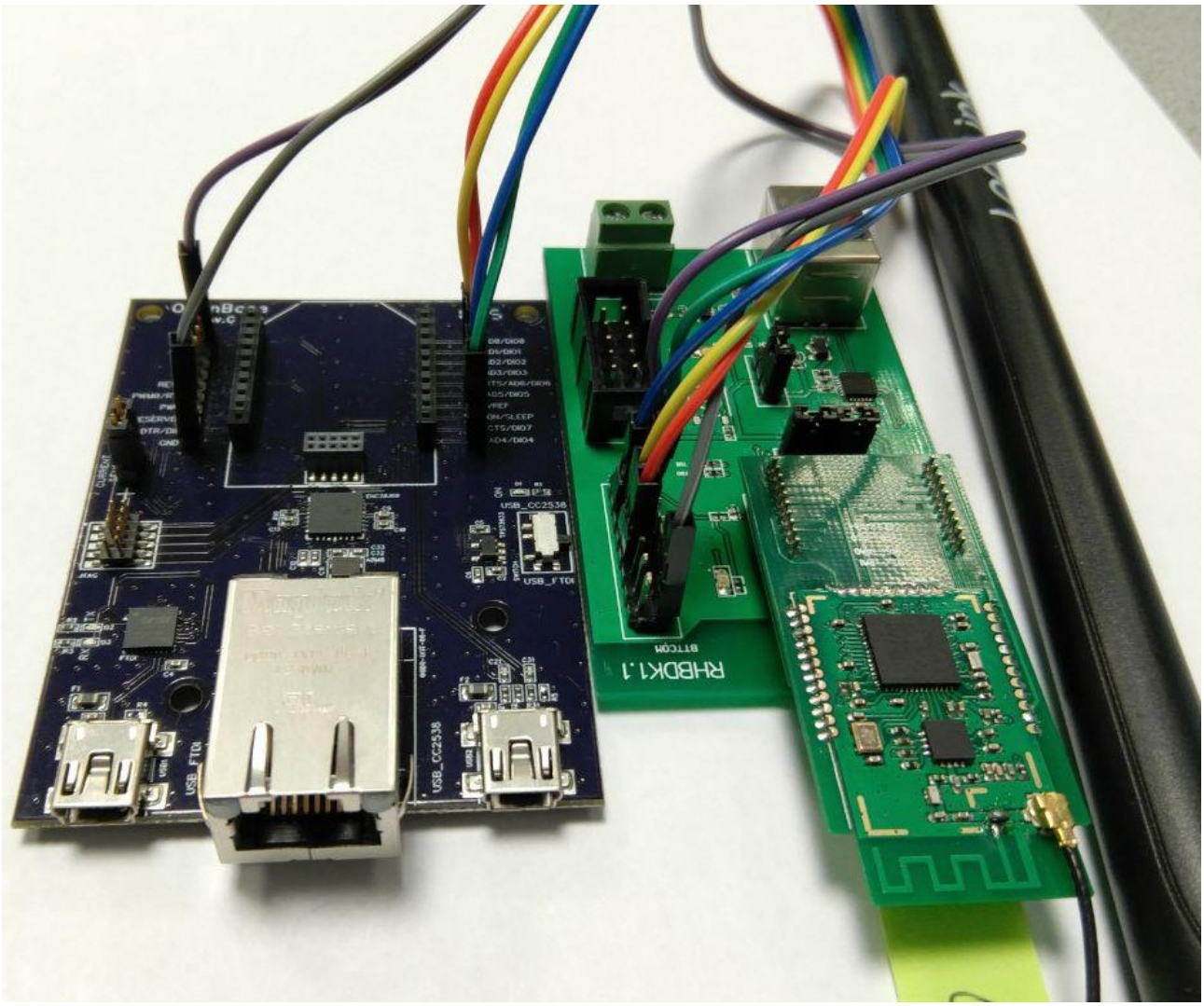


Figura AN X.83: Openchina conectado a OpenBase para usar su ENC28J60

Anexo Nodos XI: Fichero project-conf.h

En este fichero se pueden definir ajustes específicos de un programa en Contiki. De esta forma se permite tener un solo código para varios nodos con diferente hardware sin tener que adaptarlo para cada uno de ellos.

El fichero project-conf.h permite definir ajustes por plataforma sin afectar a otras, también permite sobreescribir las definiciones de cada plataforma en su fichero board.h sin tocar los fuentes de Contiki.

De esta forma se puede desacoplar por completo el código fuente de Contiki de un programa en concreto lo que permite seguir actualizando el código fuente sin tener que volver a realizar ajustes específicos para las plataformas soportadas.

Definición de pines utilizados para conectar el puerto SPI SSI1 de los OpenMote al sensor MAX6675:

```
/*---------------------------------------------------
/* OpenMote Board
/*---------------------------------------------------
#if defined CONTIKI_TARGET_OPENMOTE

/*BUTTON USER VECTOR*/

#define WITH_BUTTON_SENSOR 1

#undef BUTTON_USER_VECTOR
#define BUTTON_USER_VECTOR      NVIC_INT_GPIO_PORT_C

/*SPI port 1 (SSI)*/

#undef SPI1_CLK_PORT
#define SPI1_CLK_PORT            GPIO_A_NUM

#undef SPI1_CLK_PIN
#define SPI1_CLK_PIN             2

#undef SPI1_TX_PORT
#define SPI1_TX_PORT             GPIO_D_NUM

#undef SPI1_TX_PIN
#define SPI1_TX_PIN              3

#undef SPI1_RX_PORT
#define SPI1_RX_PORT             GPIO_D_NUM

#undef SPI1_RX_PIN
#define SPI1_RX_PIN              1
```

Figura AN XI.84: Pines utilizado para conectar MAX6675 a OpenMote

Definición de pines utilizados para conectar el puerto SPI SSI1 de los OpenChina al sensor MAX6675:

```
/*-----------------------------------------
/* OpenChina Board
/*-----------------------------------------

/*SPI port 1 (SSI)*/

#undef SPI1_IN_USE
#define SPI1_IN_USE             1

#undef SPI1_CLK_PORT
#define SPI1_CLK_PORT            GPIO_C_NUM

#undef SPI1_CLK_PIN
#define SPI1_CLK_PIN             4

#undef SPI1_TX_PORT
#define SPI1_TX_PORT             GPIO_D_NUM

#undef SPI1_TX_PIN
#define SPI1_TX_PIN              3

#undef SPI1_RX_PORT
#define SPI1_RX_PORT             GPIO_A_NUM

#undef SPI1_RX_PIN
#define SPI1_RX_PIN              7
```

Figura AN XI.85: Pines utilizado para conectar MAX6675 a Openchina

La activación del extensor de alcance CC2592 en los nodos OpenChina:

```
/*-------------------------------------
/* OpenChina Board
/*-------------------------------------
#if defined CONTIKI_TARGET_OPENCHINA

/*Enable CC2592*/

#define CC2592_EN 1

#define RxLow 0

#define RxHigh 1
```

Figura AN XI.86: Activación de extensor de alcance CC2592 en modo RxHigh

En la figura anterior se puede observar como se activa el extensor de rango CC2592 en un nodo OpenChina utilizando el flag CC2592_EN a 1 con el modo de funcionamiento RxHigh.

Para más detalles sobre los modos de funcionamiento del CC2592 consultar *Anexo Nodos VII: Añadir compatibilidad en Contiki para el extensor de alcance CC2592*.

Para utilizar aplicaciones implementadas en el directorio **apps** de los fuentes de Contiki, se debe editar el fichero Makefile del programa para añadir las aplicaciones al momento de compilar el firmware. El uso de aplicaciones implementadas en Contiki como el servidor/cliente UDP utilizado en los nodos cliente y Border Router:

```
#UDP client DEBUG
# Applications
WITH_UDPCLIENT?=1

# UDP client configuration

UDP_PERIOD?=5
WITH_UDP_CLIENT_AUTOSTART?=1

PROJECTDIRS += $(PROJECTDIR)/apps/udp-client

ifneq ($(WITH_UDPCLIENT),0)
CFLAGS += -DUDPCLIENT=1
PROJECT_SOURCEFILES += udp-client.c
else
CFLAGS += -DUDPCLIENT=0
endif

ifneq ($(UDP_PERIOD),)
CFLAGS += -DCETIC_6LBR_UDP_PERIOD=$(UDP_PERIOD)
endif

ifneq ($(WITH_UDP_CLIENT_AUTOSTART),0)
CFLAGS += -DUDP_CLIENT_AUTOSTART=1
endif
```

Figura AN XI.87: Activación de cliente UDP implementado en Contiki

El flag WITH_UDPCLIENT a 1 indica que se va a utilizar la aplicación UDP implementada en el directorio apps.

Los ajustes específicos de la red de nodos para conectar con el Border Router:

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

#define IEEE802154_CONF_PANID 0xABCD
#define RF_CHANNEL 26
#define CC2538_RF_CONF_CHANNEL RF_CHANNEL

#define NETSTACK_CONF_MAC               csma_driver
#define NETSTACK_CONF_RDC               nullrdc_driver

/*---------------------------------------------------------------
/* 6LoWPAN
/*---------------------------------------------------------------

#define CETIC_6LBR_6LOWPAN_CONTEXT_0
        { 0xAA, 0xAA, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 }

/*---------------------------------------------------------------
/* RPL & Network
/*---------------------------------------------------------------

#define RPL_CONF_INIT_LINK_METRIC                       2
#define RPL_MAX_DAG_PER_INSTANCE        2
#define RPL_MAX_INSTANCES               1
```

Figura AN XI.88: Ajustes de red nodo cliente

Anexo Raspberry Pi I: Configuración GSM Add-on v2.0 como acceso a Internet

En este anexo se adjunta un manual realizado durante la estancia en prácticas de como utilizar el módem GSM Add-on v2.0 como punto de acceso a Internet.

Configurar Raspbian para utilizar Raspberry PI GSM Add-on V2.0

Editar fichero de /boot/ para configurar el puerto serie correctamente

Editar fichero /boot/config.txt añadiendo al final del mismo estos ajustes:

```
#GSM Add-on V2.0
dtoverlay=pi3-miniuart-bt
enable_uart=1
force_turbo=1
```

Editar fichero /boot/cmdline.txt comentado (con #) o eliminando la única linea y añadiendo esta:

```
dwc_otg.lpm_enable=0 root=/dev/mmcblk0p2 rootfstype=ext4 elevator=deadline fsck.repair=yes rootwait
```

Antes de empezar, actualizar Raspbian

```
sudo apt-get update
sudo apt-get upgrade
sudo rpi-update
sudo reboot
```

Instalar el paquete ppp

```
sudo apt-get install ppp
```

Configurar ppp

Acceder como superusuario

```
sudo  -i
```

Cambiar al directorio de configuración de ppp dónde se especifican las diferentes conexiones:

```
cd /etc/ppp/peers/
```

Descargar fichero “plantilla” de fona para configurar el apn, pin, módem, baud rate, etc

```
wget https://raw.githubusercontent.com/adafruit/FONA_PPP/master/fona
```

El contenido del fichero es el siguiente:

```
# Example PPPD configuration for FONA GPRS connection on Debian/Ubuntu.

# MUST CHANGE: Change the -T parameter value **** to your network's APN value.
# For example if your APN is 'internet' (without quotes), the line would look like:
# connect "/usr/sbin/chat -v -f /etc/chatscripts/gprs -T internet"
connect "/usr/sbin/chat -v -f /etc/chatscripts/gprs -T ****"
```

```
# MUST CHANGE: Uncomment the appropriate serial device for your platform below.
# For Raspberry Pi use /dev/ttyAMA0 by uncommenting the line below:
#/dev/ttyAMA0
# For BeagleBone Black use /dev/ttyO4 by uncommenting the line below:
#/dev/ttyO4

# Speed of the serial line.
115200

# Assumes that your IP address is allocated dynamically by the ISP.
noipdefault

# Try to get the name server addresses from the ISP.
usepeerdns

# Use this connection as the default route to the internet.
defaultroute

# Makes PPPD "dial again" when the connection is lost.
persist

# Do not ask the remote to authenticate.
noauth

# No hardware flow control on the serial link with FONA
nocrtscts

# No modem control lines with FONA.
local
```

Las modificaciones que se deben realizar en el fichero son las marcadas en amarillo:
- En la primera de ellas, se debe especificar el apn de la operadora a la que nos vamos a conectar, en este caso freedompop por lo que su apn es: “freedompop.foggmobile.com” (sin las comillas).
- La siguiente, es el dispositivo que vamos a utilizar, en este caso, solamente es necesario descomentar esa línea puesto que el “GSM Add-on” se identifica en /dev como ttyAMA0.
- En la tercera, se debe especificar el baud rate actual del módem que por defecto es 9600

El contenido del fichero para los ajustes actuales debe quedar así:

```
# Example PPPD configuration for FONA GPRS connection on Debian/Ubuntu.

# MUST CHANGE: Change the -T parameter value **** to your network's APN value.
# For example if your APN is 'internet' (without quotes), the line would look like:
# connect "/usr/sbin/chat -v -f /etc/chatscripts/gprs -T internet"
connect "/usr/sbin/chat -v -f /etc/chatscripts/gprs -T freedompop.foggmobile.com"

# MUST CHANGE: Uncomment the appropriate serial device for your platform below.
# For Raspberry Pi use /dev/ttyAMA0 by uncommenting the line below:
/dev/ttyAMA0
# For BeagleBone Black use /dev/ttyO4 by uncommenting the line below:
#/dev/ttyO4

# Speed of the serial line.
9600
```

```
# Assumes that your IP address is allocated dynamically by the ISP.
noipdefault

# Try to get the name server addresses from the ISP.
usepeerdns

# Use this connection as the default route to the internet.
defaultroute

# Makes PPPD "dial again" when the connection is lost.
persist

# Do not ask the remote to authenticate.
noauth

# No hardware flow control on the serial link with FONA
nocrtscts

# No modem control lines with FONA.
local
```

Otro fichero de configuración que voy a comentar pero que para este caso se deja por defecto, es el siguiente: “etc/chatscripts/gprs”:

```
# You can use this script unmodified to connect to cellular networks.
# The APN is specified in the peers file as the argument of the -T command
# line option of chat(8).

# For details about the AT commands involved please consult the relevant
# standard: 3GPP TS 27.007 - AT command set for User Equipment (UE).
# (http://www.3gpp.org/ftp/Specs/html-info/27007.htm)

ABORT         BUSY
ABORT         VOICE
ABORT         "NO CARRIER"
ABORT         "NO DIALTONE"
ABORT         "NO DIAL TONE"
ABORT         "NO ANSWER"
ABORT         "DELAYED"
ABORT         "ERROR"

# cease if the modem is not attached to the network yet
ABORT         "+CGATT: 0"

""            AT
TIMEOUT       12
OK            ATH
OK            ATE1

# +CPIN provides the SIM card PIN
#OK           "AT+CPIN=1234"
```

```
# +CFUN may allow to configure the handset to limit operations to
# GPRS/EDGE/UMTS/etc to save power, but the arguments are not standard
# except for 1 which means "full functionality".
#OK             AT+CFUN=1

OK              AT+CGDCONT=1,"IP","\T","",0,0
OK              ATD*99#
TIMEOUT         22
CONNECT         ""
```

Marcado en amarillo está el comando AT necesario para desbloquear una sim con el código pin activado.

Ya se ha terminado toda la configuración.

Poner en marcha la red GSM

Ejecutamos el siguiente comando:

```
sudo pon fona
```

Para comprobar que todo ha ido correctamente, podemos consultar el log:

```
cat /var/log/syslog | grep pppd
```

Salida correcta:

```
May 11 09:46:03 raspberrypi pppd[6842]: pppd 2.4.6 started by root, uid 0
May 11 09:46:03 raspberrypi pppd[6842]: Device ttyAMA0 is locked by pid 5640
May 11 09:46:07 raspberrypi pppd[4806]: Device ttyAMA0 is locked by pid 5640
May 11 09:51:42 raspberrypi pppd[2725]: pppd 2.4.6 started by root, uid 0
May 11 09:51:42 raspberrypi pppd[2725]: Serial connection established.
May 11 09:51:42 raspberrypi pppd[2725]: Using interface ppp0
May 11 09:51:42 raspberrypi pppd[2725]: Connect: ppp0 <--> /dev/ttyAMA0
May 11 09:51:44 raspberrypi pppd[2725]: PAP authentication succeeded
May 11 09:51:45 raspberrypi pppd[2725]: Could not determine remote IP address: defaulting to
10.64.64.64
May 11 09:51:45 raspberrypi pppd[2725]: not replacing default route to wlan0 [192.168.0.1]
May 11 09:51:45 raspberrypi pppd[2725]: local  IP address 10.186.74.112
May 11 09:51:45 raspberrypi pppd[2725]: remote IP address 10.64.64.64
May 11 09:51:45 raspberrypi pppd[2725]: primary   DNS address 8.8.8.8
May 11 09:51:45 raspberrypi pppd[2725]: secondary DNS address 8.8.4.4
```

Si hay algún error, podemos consultar en que comando AT ha fallado:

```
cat /var/log/syslog | grep chat
```

Salida correcta:

```
May 11 09:51:42 raspberrypi chat[2730]: abort on (BUSY)
May 11 09:51:42 raspberrypi chat[2730]: abort on (VOICE)
May 11 09:51:42 raspberrypi chat[2730]: abort on (NO CARRIER)
May 11 09:51:42 raspberrypi chat[2730]: abort on (NO DIALTONE)
May 11 09:51:42 raspberrypi chat[2730]: abort on (NO DIAL TONE)
May 11 09:51:42 raspberrypi chat[2730]: abort on (NO ANSWER)
May 11 09:51:42 raspberrypi chat[2730]: abort on (DELAYED)
May 11 09:51:42 raspberrypi chat[2730]: abort on (ERROR)
```

```
May 11 09:51:42 raspberrypi chat[2730]: abort on (+CGATT: 0)
May 11 09:51:42 raspberrypi chat[2730]: send (AT^M)
May 11 09:51:42 raspberrypi chat[2730]: timeout set to 12 seconds
May 11 09:51:42 raspberrypi chat[2730]: expect (OK)
May 11 09:51:42 raspberrypi chat[2730]: AT^M^M
May 11 09:51:42 raspberrypi chat[2730]: OK
May 11 09:51:42 raspberrypi chat[2730]:  -- got it
May 11 09:51:42 raspberrypi chat[2730]: send (ATH^M)
May 11 09:51:42 raspberrypi chat[2730]: expect (OK)
May 11 09:51:42 raspberrypi chat[2730]: ^M
May 11 09:51:42 raspberrypi chat[2730]: ATH^M^M
May 11 09:51:42 raspberrypi chat[2730]: OK
May 11 09:51:42 raspberrypi chat[2730]:  -- got it
May 11 09:51:42 raspberrypi chat[2730]: send (ATE1^M)
May 11 09:51:42 raspberrypi chat[2730]: expect (OK)
May 11 09:51:42 raspberrypi chat[2730]: ^M
May 11 09:51:42 raspberrypi chat[2730]: ATE1^M^M
May 11 09:51:42 raspberrypi chat[2730]: OK
May 11 09:51:42 raspberrypi chat[2730]:  -- got it
May 11 09:51:42 raspberrypi chat[2730]: send
(AT+CGDCONT=1,"IP","freedompop.foggmobile.com","",0,0^M)
May 11 09:51:42 raspberrypi chat[2730]: expect (OK)
May 11 09:51:42 raspberrypi chat[2730]: ^M
May 11 09:51:42 raspberrypi chat[2730]:
AT+CGDCONT=1,"IP","freedompop.foggmobile.com","",0,0^M^M
May 11 09:51:42 raspberrypi chat[2730]: OK
May 11 09:51:42 raspberrypi chat[2730]:  -- got it
May 11 09:51:42 raspberrypi chat[2730]: send (ATD*99#^M)
May 11 09:51:42 raspberrypi chat[2730]: timeout set to 22 seconds
May 11 09:51:42 raspberrypi chat[2730]: expect (CONNECT)
May 11 09:51:42 raspberrypi chat[2730]: ^M
May 11 09:51:42 raspberrypi chat[2730]: ATD*99#^M^M
May 11 09:51:42 raspberrypi chat[2730]: CONNECT
May 11 09:51:42 raspberrypi chat[2730]:  -- got it
May 11 09:51:42 raspberrypi chat[2730]: send (^M)
```

También podemos consultar si otra interfaz de red nos ha aparecido (ppp0):

```
ifconfig
```

Si todo ha ido correctamente:

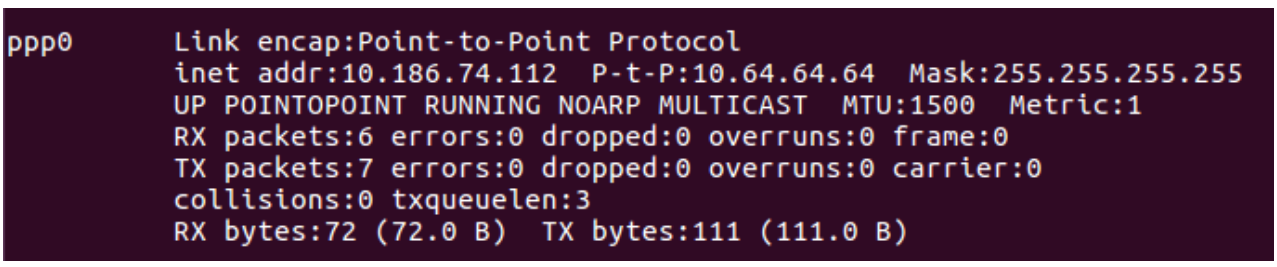


Figura AR I.89: Interfaz de red modem GSM

Para comprobar que realmente tenemos conectividad, podemos realizar un ping forzando su salida por la nueva interfaz de red:

```
ping -I ppp0 8.8.8.8
```

```
pi@raspberrypi:~ $ ping -I ppp0 www.google.es
PING www.google.es (216.58.201.227) from 10.186.74.112 ppp0: 56(84) bytes of data.
64 bytes from par10s33-in-f3.1e100.net (216.58.201.227): icmp_seq=1 ttl=53 time=824 ms
64 bytes from par10s33-in-f3.1e100.net (216.58.201.227): icmp_seq=2 ttl=53 time=604 ms
64 bytes from par10s33-in-f3.1e100.net (216.58.201.227): icmp_seq=3 ttl=53 time=605 ms
64 bytes from par10s33-in-f3.1e100.net (216.58.201.227): icmp_seq=4 ttl=53 time=601 ms
```

Figura AR I.90: Prueba de conectividad ping de la nueva interfaz de red

Para desconectar la conexión GSM utilizamos el siguiente comando:

```
sudo poff fona
```

Automatizar la conexión para que se realice en cada arranque de la raspberry

Editar el fichero /etc/network/interfaces:

```
# interfaces(5) file used by ifup(8) and ifdown(8)

# Please note that this file is written to be used with dhcpcd
# For static IP, consult /etc/dhcpcd.conf and 'man dhcpcd.conf'

# Include files from /etc/network/interfaces.d:
source-directory /etc/network/interfaces.d

auto lo
iface lo inet loopback

iface eth0 inet manual

allow-hotplug wlan0
iface wlan0 inet static
    address 192.168.0.200
    netmask 255.255.255.0
    gateway 192.168.0.1
    wpa-conf /etc/wpa_supplicant/wpa_supplicant.conf

allow-hotplug wlan1
iface wlan1 inet manual
    wpa-conf /etc/wpa_supplicant/wpa_supplicant.conf
auto fona
iface fona inet ppp
          provider fona
```

Añadidos los ajustes marcados en amarillo que permiten activar la interfaz ppp0 en cada inicio.

Documentación utilizada para la realización de este manual:

Ajustes GSM

https://learn.adafruit.com/fona-tethering-to-raspberry-pi-or-beaglebone-black?view=all

https://raspberrypi.stackexchange.com/questions/44597/how-to-use-internet-using-pppd-and-sim800-gsm-addon

https://www.modmypi.com/blog/how-to-connect-your-raspberry-pi-to-a-3g-network

http://surfero.blogspot.com.es/2017/04/freedompop-orange-pi-2g-iot.html

Documentación Módulo GSM

https://www.itead.cc/wiki/RPI_SIM800_GSM/GPRS_ADD-ON_V2.0

https://www.itead.cc/blog/raspberry-pi-3-add-on-supplementary-document