Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels

UNIVERSITAT POLITÈCNICA DE CATALUNYA

# TRABAJO FINAL DE CARRERA

**TÍTULO DEL TFC: Simulación e implementación de una red de sensores inalámbrica multisalto para la medición de consumo energético en un edificio.**

**TITULACIÓN: Ingeniería Técnica de Telecomunicaciones, especialidad Telemática**

**AUTOR: Jose Manuel Linares Arenas**

**DIRECTORES: Miguel Ángel Vázquez y Mischa Dohler**
**SUPERVISOR: Carles Gómez Montenegro**

**FECHA: 28 de marzo de 2014**

**TÍTULO DEL TFC:** Simulación e implementación de una red de sensores inalámbrica multisalto para la medición de consumo energético en un edificio.

**AUTOR:** Jose Manuel Linares Arenas

**DIRECTORES:** Miguel Ángel Vázquez y Mischa Dohler
**SUPERVISOR:** Carles Gómez Montenegro

**FECHA:** 28 de marzo de 2014

## Resumen

En los últimos años han tomado especial relevancia en el ámbito de las I+D+I, las redes inalámbricas de sensores WSN (Wireless Sensor Networks), donde pequeños dispositivos compactos y económicos llamados nodos-sensores, tienen la capacidad de sensar, detectar y suministrar datos a otros sistemas. Las WSN se están aplicando en muchos ambientes y con distintos propósitos. Estas redes están compuestas por nodos-sensores de bajo costo y bajo consumo que cooperan entre sí, los cuales se comunican inalámbricamente y están especialmente diseñadas para redes con pérdidas y recursos limitados de memoria.

En este proyecto se propone implementar y simular una red inalámbrica de sensores multisalto en el edificio B4 del PMT (Parque Mediterráneo de la Tecnología) de Castelldefels, en el cual reside el centro de investigación CTTC (Centro Tecnológico de Telecomunicaciones de Catalunya).

Para poder llevar a cabo el escenario, se ha desarrollado una aplicación que permite a los nodos comunicarse entre ellos, la cual medirá las condiciones climáticas de cada estancia, el estado de la batería del propio sensor y el consumo eléctrico en tiempo real de los splits del aire acondicionado del edificio. Dado el corto alcance de estas redes y procurando cubrir todo el área de la construcción, se han estudiado e implementado protocolos de comunicaciones multisalto. Esta red estará constituida de varios nodos “senders” los cuales se encargarán de realizar las medidas y transmitir la información recolectada a un nodo central llamado “sink” para su procesamiento. Los senders también actúan como relays de tal modo que se forma una red mesh. El sink estará conectado a un PC al cual le enviará por serial todos los datos recolectados.

Con este TFC, se propone evaluar los diferentes tipos de sensores y sistemas operativos que existen en la actualidad dando especial importancia a las motas Z1 de Zolertia, con el sistema operativo Contiki que aún no se ha desarrollado) para dichas motas, con su simulador Cooja, y a su vez los algoritmos y protocolos de comunicación eficientes que este sistema operativo implementa, que permitan a los nodos recolectar, procesar y trasmitir los datos sobre la red. Los parámetros principales que se desean caracterizar del escenario son las pérdidas ocasionadas en la transmisión y posteriormente la latencia experimentada por los paquetes dentro de la red.

**Title:** Simulation and implementation of a multihop wireless sensor network for measuring energy consumption in a building.

**Author:** Jose Manuel Linares Arenas

**Directors:** Miguel Ángel Vázquez y Mischa Dohler

**Tutor:** Carles Gómez Montenegro

**Date:** March, 28th 2014

## Overview

In recent years it has taken special relevance in the field of R + D + I, wireless sensor networks WSN (Wireless Sensor Networks), where small compact and inexpensive devices called sensor nodes, have the ability to sense, detect and provide data to other systems. The WSN are being implemented in many environments and for different purposes. These networks are composed of inexpensive sensor nodes and low consumption that cooperate with each other, which communicate wirelessly and are specially designed for networks with memory loss and limited resources.

This project aims to implement and simulate a multi-hop wireless network sensors in the PMT building B4 (Mediterranean Technology Park) in Castelldefels, in which the research center CTTC (Centre Tecnològic de Telecomunicacions de Catalunya) resides.
To carry out the scenario, there has been developed an application that allows nodes to communicate with each other, which will measure the climatic conditions of each room, the battery status of the sensor itself and the power consumption in real time splits for the air conditioning of the building. Given the short range of these networks and trying to cover all the area of construction, it has been studied and implemented protocols for multi-hop communications. This network will consist of several "senders" nodes which are responsible for carrying out the measurements and transmit the collected information to a central node called "sink" for processing. The senders also act as relays so that they form a mesh network. The sink shall be connected to a PC which will send via serial all data collected.

With this TFC, it is proposed to evaluate the different types of sensors and operating systems that exist today giving special emphasis to the Z1 motes of Zolertia, the Contiki operating system that has not yet been developed for said motes, with its Cooja simulator, and in turn the algorithms and efficient communication protocols that this operating system implements, that allow the nodes collect, process and transmit the data over the network. The main parameters that characterize the desired scenario are the transmission losses and the latency caused by the transmission of packets within the network.

*Me gustaría dedicar este trabajo a las personas que siempre han estado a mi lado, apoyándome, como mi madre que siempre ha estado pendiente de cada paso a lo largo de mi carrera universitaria, y a mi padre que aunque ya no está entre nosotros fue el primero que me apoyo cuando decidí volver a estudiar y sé que ahora mismo estaría muy orgulloso de mi.*

*A mi pareja por soportar ciertos momentos de estrés y por darme ánimo en todo momento.*

*A mis directores y tutor por darme toda la ayuda que he necesitado a lo largo del proyecto, ofreciéndome todo el soporte necesario para que éste cogiera forma y que sin su apoyo no hubiera sido posible.*

*Al CTTC por brindarme todas las herramientas que he necesitado para llevar a cabo el proyecto y por toda la formación que me han ofrecido, donde he tenido la suerte de trabajar con todo un equipo muy humano y profesional, del que me siento orgulloso de haber formado parte.*

*Gracias a todos,*

*Jose Manuel Linares Arenas*

# Índice

# INTRODUCCIÓN

El avance tecnológico permite la fabricación de dispositivos electrónicos con un tamaño cada vez más compacto, de baja potencia y económicos, con capacidad de detectar, procesar datos y comunicarse con otro, de forma inalámbrica. Una red inalámbrica de sensores (WSN), es un conjunto de estos dispositivos que reciben el nombre de nodos-sensores de bajo coste y bajo consumo y se comunican entre sí, formando una red inalámbrica.

Esta clase de redes se caracterizan por su fácil despliegue y por ser autoconfigurables, pudiendo convertirse en todo momento en emisor, receptor, ofrecer servicios de encaminamiento entre nodos sin visión directa, así como registrar datos referentes a los sensores locales de cada nodo. Otra de sus características es su gestión eficiente de la energía, que les permite obtener una alta tasa de autonomía que las hacen plenamente operativas.

En este proyecto se propone implementar y simular una red inalámbrica de sensores multisalto en el edificio B4 del PMT (Parque Mediterráneo de la Tecnología) de Castelldefels, en el cual reside el centro de investigación CTTC (Centro Tecnológico de Telecomunicaciones de Catalunya). Para poder llevar a cabo el escenario, se ha desarrollado una aplicación que permite a los nodos comunicarse entre ellos, la cual medirá las condiciones climáticas de cada estancia, el estado de la batería del propio sensor y el consumo eléctrico en tiempo real de los splits del aire acondicionado del edificio. Dado el corto alcance de estas redes y procurando cubrir todo el área de la construcción, se han estudiado e implementado protocolos de comunicaciones multisalto. Esta red estará constituida de varios nodos “senders” los cuales se encargarán de realizar las medidas y transmitir la información recolectada a un nodo central llamado “sink” para su procesamiento. Los senders también actúan como relays de tal modo que se forma una red mesh. El sink estará conectado a un PC al cual le enviará por serial todos los datos recolectados.

Con este TFC, se propone evaluar los diferentes tipos de sensores y sistemas operativos que existen en la actualidad dando especial importancia a las motas Z1 de Zolertia, con el sistema operativo Contiki que aún no se ha desarrollado (como si lo está TinyOS) para dichas motas, con su simulador Cooja, y a su vez los algoritmos y protocolos de comunicación eficientes que este sistema operativo implementa, que permitan a los nodos recolectar, procesar y trasmitir los datos sobre la red. Los parámetros principales que se desean caracterizar del escenario son las pérdidas ocasionadas en la transmisión y posteriormente la latencia experimentada por los paquetes dentro de la red.

En el primer capítulo se explican los conceptos teóricos, características, y aplicaciones de las redes inalámbricas de sensores.

En el segundo capítulo se explican las plataformas existentes de nodos de sensores, describiendo las motas comerciales y los sistemas operativos más comunes que existen en una WSN.

En el tercer capítulo se explica la simulación e implementación de una red de sensores inalámbrica multisalto, describiendo brevemente la distribución de las motas que se han implementado en el edificio, el material utilizado, tanto de hardware como de software, durante los ensayos experimentales y el porqué de su elección y los protocolos utilizados en ContikiOS que es el sistema operativo que se ha utilizado para las motas Z1 de la empresa Zolertia.

En el cuarto capítulo en primer lugar se explica el diseño de la aplicación en Contiki que se ha realizado, para poder medir la temperatura ambiental de los establecimientos del edificio, el estado de la batería de la mota y el consumo energético que está realizando en ése momento por el uso de los splits del aire acondicionado. Posteriormente se explica cómo se compila una aplicación en Contiki.

En el quinto capítulo se describe cómo funciona el simulador Cooja y el entorno de simulación de la red inalámbrica de sensores multisalto, simulando la distribución de los sensores de la red implementada en el edificio del CTTC.

En el sexto capítulo se describirá brevemente las pruebas y resultados que se han obtenido con el simulador Cooja, para poder evaluar la red de sensores inalámbrica multisalto. Primero se describe las pruebas realizadas para poder evaluar las pérdidas ocasionadas en la transmisión y posteriormente evaluar la latencia experimentada por los paquetes dentro de la red.

Por último en el séptimo capítulo se exponen las conclusiones a las que se ha llegado y las líneas futuras derivadas de este proyecto.

# CAPÍTULO 1. REDES DE SENSORES INALÁMBRICAS

En este primer capítulo se describe brevemente los conceptos teóricos de las redes de sensores inalámbricas, sus características principales, y sus ámbitos de aplicación.

## 1.1. Conceptos teóricos

Una red inalámbrica de sensores consiste en una gran cantidad de dispositivos, capaces de recoger información de su entorno, tales como humedad, luz, temperatura, etc., mediante el uso de los sensores que llevan incorporados estos dispositivos y además de éstos, también existen los nodos repetidores que se encargaran de encaminar los datos hacia la estación base que se encuentra conectado a un ordenador, que puede comunicarse hacia el exterior a través de Internet o una red de área local (LAN).

Los nodos sensores, se encuentran esparcidos por el escenario de interés en diferentes tipos de topologías, dependiendo de la aplicación. En la **figura 1.1**. se puede ver un ejemplo de una red de sensores.

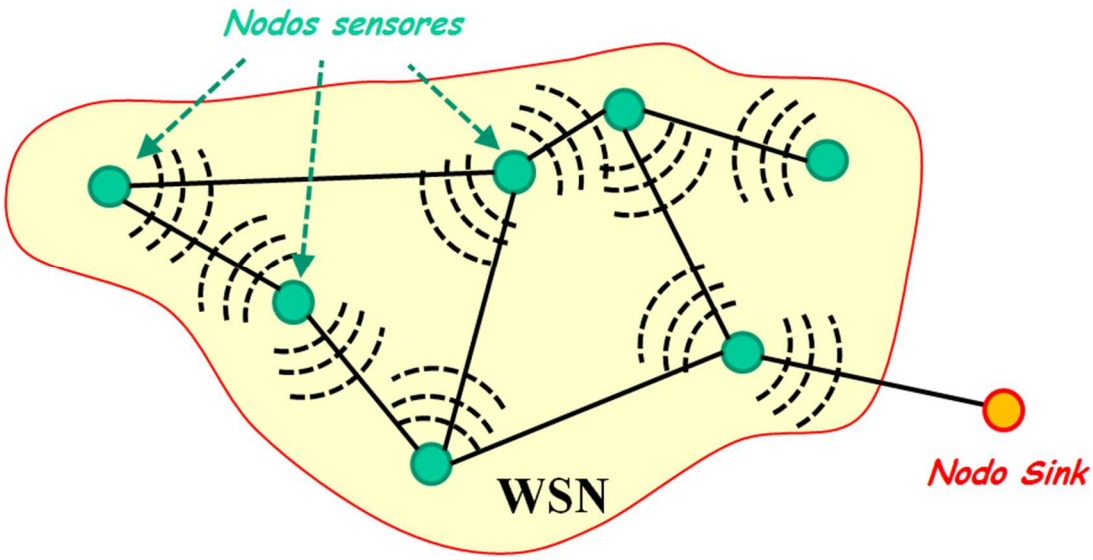



**Figura 1.1.:** Ejemplo de una red de sensores

## 1.2. Características

A la hora de diseñar una red de sensores se ha de tener en cuenta las siguientes características:

- **Topología:** Es la forma en que los nodos de una red está distribuida. Existen varios tipos de topología tales como, estrella, malla, árbol entre otras. En nuestro caso la topología utilizada es de jerarquía en árbol en el

que distinguimos sensores, repetidores y nodo base. La topología siempre es cambiante, debido a que los nodos pueden dejar de funcionar en cualquier momento, y éstos tienen que adaptarse para poder comunicar nuevos datos adquiridos.

- **No se utiliza infraestructura de red:** Una red de sensores no tiene necesidad alguna de infraestructura para poder operar, ya que sus nodos pueden actuar de emisores, receptores o router.

- **Medio de transmisión:** En una red de sensores, los nodos están conectados de manera inalámbrica mediante radio. En este proyecto, utilizamos el transceptor CC2420 que opera en la banda de 2.4Ghz bajo el estándar IEEE 802.15.4. en el que existen una serie de fenómenos como pueden ser la atenuación, desvanecimientos rápidos, desvanecimientos lentos e interferencias que pueden producir errores en la transmisión.

- **Consumo energético:** Los nodos sensores al tratarse de un dispositivo de escaso tamaño, poseen una fuente energética bastante limitada, dependiendo de las baterías utilizadas. Por lo que el tiempo de vida de un nodo viene condicionado por la batería y el ciclo útil de trabajo de cada nodo, ya que en algunos casos la recarga de ellas depende de los módulos de alimentación adicionales, utilizando paneles solares para pilas recargables, aunque esta solución se encuentra fuera del alcance de este trabajo. Como anteriormente hemos comentado, los nodos sensores tienen la función de recolectar la información y posteriormente enrutarlas hacia el nodo base si es necesario, y en el caso de que un nodo se quedase sin batería, podría suponer un cambio significativo de la topología de la red y se requerirá un nuevo enrutamiento y una reorganización de la red.

- **Tolerancia a errores**: Una red se sensores tiene que ser capaz de seguir funcionando en el caso de que un nodo sensor o varios no funcionen adecuadamente o dejen de funcionar.

- **Comunicaciones multisalto o broadcast**: En las aplicaciones de redes de sensores siempre es característico el uso de algún protocolo que permita comunicaciones multisalto, aunque también es posible utilizar mensajería basada en broadcast.

- **Limitaciones hardware**: Para poder conseguir un consumo ajustado, se hace indispensable que el hardware sea lo más sencillo posible, así como su transceptor de radio, esto nos deja una capacidad de proceso limitada y una reducida cantidad de memoria.

- **Reducido tamaño de mensajes:** Usualmente los mensajes en estas redes son reducidos en comparación con las redes tradicionales.

- **Características de Tráfico**: En las WSN el tráfico principal generalmente fluye en sentido ascendente (upstream) desde los nodos sensores hasta

el nodo recolector o sumidero (sink), aunque ocasionalmente el sink puede generar tráfico en sentido contrario (downstream) para realizar tareas de gestión y control. En el caso de tráfico ascendente, se realiza una comunicación de muchos a uno. Además, dependiendo de la aplicación el tráfico puede ser entregado al sink de forma periódica, cada vez que ocurra un evento, de forma continua en tiempo real, o cualquier combinación de éstas.

- **Costes de producción**: Dado que la naturaleza de una red de sensores tiene que ser en número muy elevado, para poder obtener datos con fiabilidad. Una vez definida su aplicación, la fabricación de nodos sensores es económica, si éstos se producen en grandes cantidades.

## 1.3. Aplicaciones

El ámbito de aplicación de las redes de sensores es muy amplio y diverso. Cada una de estas aplicaciones tiene sus requerimientos particulares En particular cabe destacar las siguientes áreas de aplicación:

- **Medicina**: Con la reducción de tamaño que están sufriendo los nodos sensores, la calidad de vida de pacientes que tengan que tener controlada sus constantes vitales (pulsaciones, presión arterial, tensión, nivel de azúcar en sangre, etc.), podrá mejorar substancialmente.
- **Sensores ambientales**: El control ambiental de áreas de bosque o de océano. El control de múltiples variables, como temperatura, humedad, fuego, actividad sísmica así como detectar y prevenir condiciones climáticas adversas.
- **Tráfico:** Pueden informar a conductores de la situación del tráfico, en caso de atasco o accidente, con lo que estos tienen capacidad de reacción para tomar rutas alternativas.
- **Agricultura:** Control de condiciones climáticas, recolección de datos sobre el terreno, control de riegos, suministro de datos para los agricultores, cálculo de agua.
- **Entornos de alta seguridad:** Entornos de alta seguridad, tales como aeropuertos (medición de ruido acústico), centrales nucleares (sensores para la medición de radiaciones ionizantes), edificios de gubernamentales (sensores de movimiento).
- **Domótica:** Su tamaño, economía y velocidad de despliegue, las convierten en una tecnología ideal para automatizar tareas cotidianas en el hogar y tener controlado un hogar mediante sensores
- **Militares:** Una red de sensores puede ser utilizada para la detección y demolición o desactivación de minas.
- **Estructuras:** Identifica y monitoriza fallos en estructuras como puentes, edificios y otras construcciones edificadas.

## 1.4. Elementos de una red de sensores inalámbrica

En relación a los elementos que componen una WSN (**ver Fig. 1.2.**), caben destacar los nodos sensores, el nodo sink o gateway, y la red inalámbrica. El nodo sink interconecta la red de sensores inalámbrica y la red de datos TCP/IP. Los nodos sensores toman del medio la información y la convierten en señales eléctricas y envían la información a la estación base. La estación base recolecta los datos en un ordenador o sistema embebido.

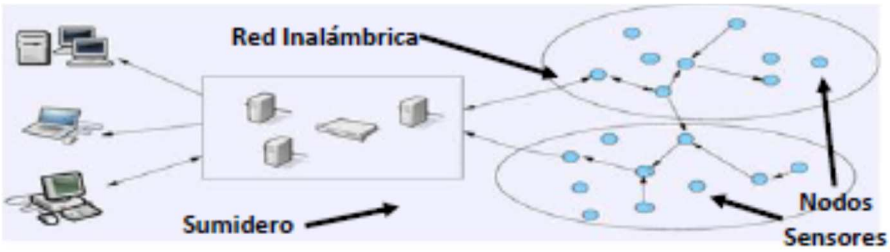



**Figura 1.2.:** Elementos de una red de sensores.

Los dispositivos utilizados para formar una red de sensores inalámbrica, se denominan motas. Estas motas son pequeños dispositivos hardware típicamente equipados con los siguientes componentes:

- **Sensores:** Miden una magnitud física como por ejemplo temperatura, humedad, o luz, entre muchas otras.
- **Actuadores:** Dispositivo mecánico cuya función es proporcionar fuerza para mover o "actuar" sobre otro dispositivo mecánico.
- **Conversor analógico-digital**: Su función es convertir la magnitud analógica que proviene de los sensores en una señal digital que pueda ser procesada por un microcontrolador. Habitualmente, el conversor analógico-digital está integrado en el microcontrolador.
- **Microcontrolador**: Procesa la información que proviene de los sensores.
- **Transceptor de comunicaciones radio**: Su función es enviar y recibir información desde/hacia otras motas utilizando el canal radio.
- **Conector USB:** Su función es conectar la mota con un PC para compilar la aplicación y extraer resultados.

En la **figura 1.3.** se puede ver una mota con los elementos que la componen.

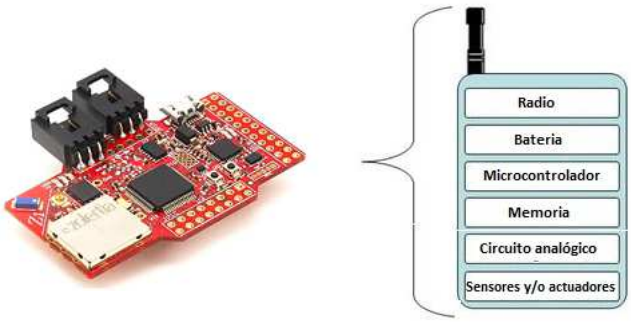



**Figura 1.3.:** Elementos de un nodo sensor.

# CAPÍTULO 2. PLATAFORMAS PARA NODOS SENSORES

Existen una gran variedad de sensores en el mercado que nos permiten desarrollar aplicaciones sobre redes de sensores. Además, estas motas sirven para poder realizar trabajos de investigación basados en el estudio y diseño de protocolos de comunicaciones para redes de corto alcance.

Las necesidades que tiene un nodo de una WSN son totalmente distintas a las que pueda tener cualquier otro dispositivo como puede ser un PC, por lo tanto estos nodos tienen sus propios sistemas operativos.

En este capítulo se describen las motas comerciales y los sistemas operativos más comunes que existen en una WSN.

## 2.1. Motas comerciales

### Mica2 y Mica2dot

La funcionalidad de Mica2 y Mica2dot es similar pero son muy diferentes en factor de forma, como se puede apreciar en la **figura 2.1.** Funcionan ambos a 4 MHz con un microprocesador de 8 bits de la marca Atmel. Tiene 128 KB de memoria para el programa y 4KB de memoria RAM. Su velocidad de transmisión radio es de 19,2 Kb/s sobre un canal CSMA/CA y utiliza el estándar IEEE 802.15.4. También están equipados con una memoria externa de tipo Flash no volátil con 512 KB que puede ser usada para guardar otros datos.

**Figura 2.1.:** Mica2. Mica2dot.

A la placa principal del procesador se le pueden adherir placas con sensores/actuadores por medio de unos pines, en el caso del Mica2, o por medio de unos conectores, para el modelo Mica2dot. Como por ejemplo, una placa con sensores de temperatura, luminosidad, un micrófono, acelerómetro y un sensor magnético, entre otros.

### Intel Mote 2

Las motas de Intel son pequeñas, autónomas, alimentadas por medio de baterías y con comunicación vía radio, de forma que son capaces de compartir información con otros y organizarse automáticamente dentro de una red AdHoc y tiene total compatibilidad con el sistema operativo TinyOS (**ver fig. 2.2.**).

Intel Mote 2 es una plataforma avanzada para la creación de una WSN. La plataforma está construida alrededor del procesador de bajo consumo XScale PXA27x con comunicación radio 802.15.4 sobre la placa principal y una antena de 2.4 GHz. Contiene dos interfaces de sensores básicos y dos interfaces avanzados para sensores.


**Figura 2.2.:** Intel Mote.

El Intel Mote 2 es una plataforma modular y escalable. La placa principal contiene el procesador y el módulo radio y se le pueden agregar diferentes módulos sensores dependiendo de la aplicación final de la red de sensores.
El procesador que contiene la placa principal puede funcionar a bajo voltaje (0.85 V) y una frecuencia también baja (13 MHz), cuando se habilita el modo de bajo consumo, mientras que el valor máximo de operación es de 416 MHz. También integra una memoria SRAM de 256 KB y diferentes opciones de E/S, lo que lo hace extremadamente flexible para soportar diferentes sensores A/D, opciones de radio. Estas E/S incluye I2C, 3 puertos serie síncronos, 3 puertos UART, E/S digitales, puerto USB. Además, el propio procesador incluye diversos temporizados y un reloj en tiempo real.
Una de las características más importantes y que lo diferencian del resto de motes comerciales, es que existe una placa de expansión (**ver fig. 2.3.**) que incorpora una cámara CMOS, sensor PIR, micrófono y altavoz capaz de reproducir sonidos con un alto nivel de detalle. Lo cual unido a las 64 MB de memoria disponibles y a su compatibilidad con TinyOS (sólo el altavoz, sensor PIR y micrófono), le permite desarrollar aplicaciones que son impensables en otros dispositivos comerciales.
El principal problema de este dispositivo, es su elevado precio, alrededor de 1000 € el Imote2 con la placa de expansión, se antoja muy elevado en la actualidad.

**Figura 2.3.:** Placa de expansión para Imote2 (cámara, altavoz, micrófono y sensor PIR).

**TelosB**

El dispositivo TelosB (**ver fig. 2.4.**), también llamado Tmote Sky consta del microcontrolador MSP430F1611. Este procesador RISC de 16 bits de bajo consumo está diseñado expresamente para el uso en redes de sensores inalámbricas.
Al igual que el Micaz, consta de un transceptor de radio Chipcon CC2420. Otra de las características del TelosB es que dispone de sensores de humedad, temperatura y luminosidad en la propia placa del dispositivo, lo cual permite realizar medir estas dos variables sin necesidad de incorporar una placa de expansión.

**Figura 2.4.:** TelosB.

Una de las principales cualidades de la mota TelosB es que es totalmente compatible con los dos sistemas operativos más utilizados en redes de sensores inalámbricas: Contiki y TinyOS. Lo cual le ofrece una gran versatilidad a la hora de elegir cuál de ellos se adapta mejor a las características de la aplicación que se desee desarrollar.

**Waspmote**

La empresa zaragozana Libelium lanzó al mercado en el año 2009 este dispositivo destinado a ser usado en redes de sensores inalámbricas denominado Waspmote (**ver figura 2.5.**).

**Figura 2.5.:** Mota Waspmote con módulo de radio, GPS y GPRS incorporados.

El módulo principal del Waspmote (**ver fig. 2.6.**) dispone de un microcontrolador ATmega1281 que funciona a 8 MHz y dispone de 8 KB de memoria RAM y 128 KB de memoria FLASH. En conjunto con el microcontrolador, la placa principal cuenta con un reloj en tiempo real con su batería de respaldo y de un slot microSD.

La idea de Libelium es vender el resto de componentes del dispositivo en función de las necesidades del cliente. Es por ello que el dispositivo cuenta con diversos conectores de expansión que le permiten añadir diferentes elementos en función de las necesidades finales.

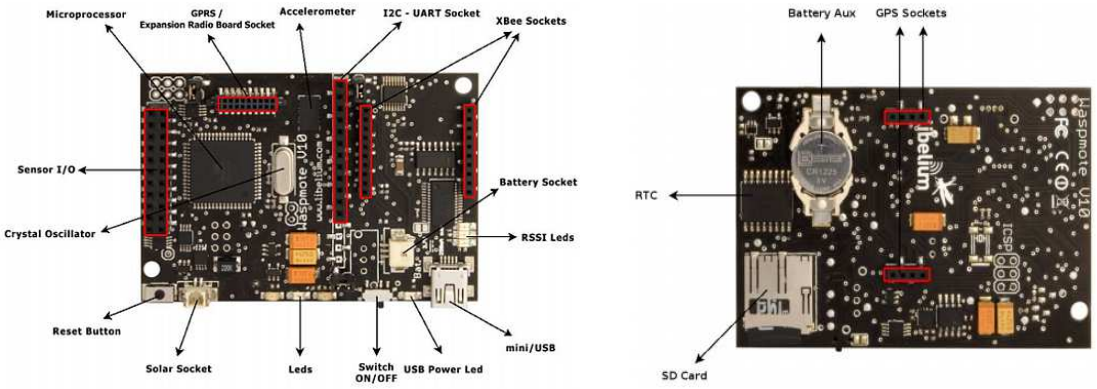



**Figura 2.6.:** Vista delantera y trasera de la placa principal del Waspmote de Libelium.

En lo que respecta a la conectividad vía radio, se le pueden añadir diferentes módulos XBee o bien un módulo Bluetooth. Además también permite añadir un módulo GSM/GPRS para el envío y recepción de datos a través de la red GSM. También es posible añadir un módulo GPS con su correspondiente antena.

El principal inconveniente del Waspmote es que no es compatible con Contiki ni con TinyOS. Lo cual limita en gran medida el uso del dispositivo. Ya que se depende totalmente del soporte del fabricante.

### Z1

Las motas Z1 han sido fabricadas a inicios del año 2009 en Cerdanyola del Vallés, por la empresa Zolertia (**ver fig. 2.7.).**

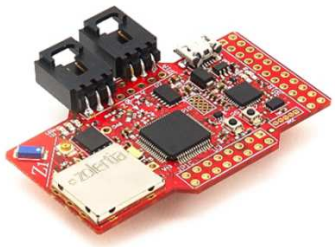


**Figura 2.7.:** Mota Z1 de Zolertia.

Las Z1 son dispositivos de red inalámbricos de bajo consumo y sirven como una plataforma de propósito general de desarrollo para diseñadores de WSN, investigadores, entusiastas y aficionados.

La mota Z1 dispone de soporte para sistemas operativos de código abierto implementados por la comunidad WSN, como TinyOS (disponibles en la

actualidad) y Contiki (en desarrollo). La mota Z1 es un módulo de bajo consumo inalámbrico compatible con protocolos del estándar IEEE 802.15.4 y Zigbee con la intención de ayudar a los diseñadores a practicar y desplegar sus propias aplicaciones y prototipos, con la mejor relación entre el tiempo de desarrollo y la flexibilidad hardware.
El núcleo de su arquitectura se basa en la familia de microcontroladores y transceptores radio MSP430+CC2420 de Texas Instruments, el cual lo hace compatible con motas basadas en esta misma arquitectura. Sin embargo la presente MCU (la función principal de un MCU es gestionar la comunicación entre diferentes terminales en un esquema de transmisión multipunto), en Z1 es el MSP430F2xxx de segunda generación en lugar de la MSP430F1xxx, como es habitual en otras motas como la TelosB de Crossbow, Tmote de Moteiv y similares. Este hecho implica algunas incompatibilidades debido a los cambios internos en el microcontrolador, sin embargo, estas pequeñas diferencias no se aprecian en el ámbito de aplicación si se utiliza un sistema operativo compatible como ContikiOS.

Las características principales de las motas Z1 son las siguientes:

- Rango de temperatura de operación (-40ºC a 85ºC).
- Conector de expansión de 52 pins para añadir otros sensores.
- Microcontrolador de la familia MSP430 de segunda generación de bajo consumo, de 16 bit y 16 Mhz.
- Trabaja en la frecuencia de 2.4 GHz.
- Transceptor RF: CC2420.
- Tasa máxima efectiva de transmisión de 250 Kbps.
- Compatible con 6LoWPAN y Zigbee.
- Sensor de temperatura digital de bajo consumo TMP102.
- Acelerómetro digital de 3 ejes ADXL345.
- Memoria flash de 16 MB.
- 3 leds y 2 pulsadores para interactuar con el usuario.
- Puerto micro-USB para poder obtener los datos o programar la mota.
- Antena externa opcional.
- Compatible con sensores Phidgets y una gran variedad de sensores analógicos y digitales.

La característica más importante de esta mota es que a pesar de ser relativamente reciente, es totalmente compatible tanto con TinyOS como con Contiki. Este hecho es esencial ya que es la única mota de los vistas hasta ahora que era compatible con ambos sistemas operativos era la Telosb. Sin embargo, las características técnicas (tanto transceptor de radio como microcontrolador) del Telosb son muy inferiores a las del Z1. Por tanto la mota Z1 es una opción muy interesante tanto por sus características técnicas como por su compatibilidad software.

En lo que respecta a la alimentación ofrece un amplio abanico de posibilidades de conexión, ya que se puede alimentar por 2 pilas AAA, una pila de botón, alimentación directa por USB o a una fuente de alimentación a través de dos cables.

En la **tabla 1.2.** se puede apreciar una comparativa entre los diferentes motas comerciales, mostrando sus características más importantes.

**Tabla 1.2.:** Comparativa entre motas comerciales ordenados por año de fabricación.

| Mota | | MICA2 | MICA2 Dot | TelosB | Imote2 | Zolertia Z1 | Waspmote |
|---|---|---|---|---|---|---|---|
| MCU | Chip | ATmega128L | | Msp30f1611 | FXA271 | MSP430F2167 | ATmega128L |
| | Tipo | 8 bits | | 16 bits | 32 bits | 16 bits | 8 bits |
| | Memoria Programa | 128 KB | | 48 KB | N/A | 96 KB | 128 KB |
| | RAM | 4 KB | | 10 KB | 32768 KB | 96 KB | 128 KB |
| Almacenamiento Externo No Volátil | Chip | AT45DB014B | | M25P80 | N/A | N/A | N/A |
| | Conexión | SPI | | SPI | N/A | N/A | SPI |
| | Tamaño | 512 KB | | 1024 KB | 32768 KB | N/A | SD hasta 2000 KB |
| Sistema de Alimentación | Tipo | 2xAA | Coin Cell | 2xAA | 3xAAA | 2xAA/A ó Coin Cell | Variable |
| | Capacidad Típica | 2850 mAh | 1000 mAh | 2850 mAh | 1100 mAh | 1100 mAh | 2000 mAh |
| RF | Chip | CC1000 | | CC2420 | CC2420 | CC2420 | Variable (Xbee) |
| | Frecuencia | 868/916,433 ó 315 MHz | | 2,4GHz | 2,4GHz | 2,4GHz | 2,4GHz |
| | Datarate | 38,4 Kbps | | 250 Kbps | 250 Kbps | 250 Kbps | 250 Kbps |
| | Potencia Transmisión | 5 dBm | | 0 dBm | 0 dBm | 0 dBm | 18 dBm |
| | Alcance Exterior | 152,4 m | 152,4 – 304,8 m | 75 – 100 m | ~30 m | 150 m | 300 m |
| Año comercialización | | 2002 | 2002 | 2005 | 2007 | 2009 | 2009 |

## 2.2. Sistemas operativos para motas

Los sistemas operativos para WSN son típicamente menos complejos que los de propósito general, tanto debido a los requisitos especiales de las aplicaciones en las que se usan, como a las restricciones de recursos encontradas en las plataformas hardware utilizadas. Por ejemplo, las aplicaciones de WSN no son tan interactivas como son las aplicaciones para PC y debido a esto, estos sistemas no necesitan incluir el soporte de interfaces de usuario.
Los nodos sensores incluyen un microcontrolador capaz de ejecutar tareas que requieren el acceso a elementos de hardware (sensores, memoria, radio, etc.).

Las funciones principales de un sistema operativo son:

- Gestionar eficientemente los recursos de hardware.
- Facilitar la programación de aplicaciones de alto nivel.

Estos sistemas operativos están diseñados específicamente teniendo en cuenta las restricciones de hardware de los nodos sensores. Ya que los sistemas operativos diseñados para otro tipo de sistemas empotrados no se adaptan a las fuertes restricciones de los nodos sensores, sobretodo relacionadas con el tamaño de memoria, consumo y requisitos de las aplicaciones.

En el marco de los sistemas operativos existentes para el trabajo con dispositivos destinados a ser usados en redes inalámbricas de sensores, TinyOS fue el primer sistema operativo diseñado específicamente para ellas. A diferencia de la mayoría sistemas operativos, TinyOS se basa en un modelo de programación controlado por eventos, en lugar de multiprocesos.

Existen varios sistemas operativos, entre ellos se encuentran los siguientes:

**TinyOS**

TinyOS es un sistema operativo orientado a trabajar con redes de sensores, desarrollado en la Universidad de Berkeley. TinyOS puede ser visto como un conjunto de programas avanzados, el cual cuenta con un amplio uso por parte de comunidades de desarrollo, dadas sus características de ser un proyecto de código abierto. Este conjunto de programas contiene numerosos algoritmos, que nos permiten generar enrutamientos, así como también aplicaciones pre construidas para sensores. Además soporta diferentes plataformas de nodos de sensores, arquitecturas bases para el desarrollo de aplicaciones.
El lenguaje en el que se encuentra programado TinyOS es un meta-lenguaje que deriva de C, cuyo nombre es NesC. Además existen varias herramientas que ayudan al estudio y desarrollo de aplicaciones para las redes de sensores, que van desde aplicaciones para la obtención y manejo de datos, hasta sistemas complejos de simulación.

**MANTIS**

MANTIS (MultimodAl NeTworks In-situ Sensors). Ante el incremento de complejidad de las tareas realizadas por las redes de sensores como compresión, agregación y procesado de señales, los multiprocesos del sistema operativo MANTIS (MOS) permiten interpaginar complejas tareas con tareas susceptibles al tiempo para así mitigar los problemas en los saltos de buffers. Para conseguir una eficiencia en el uso de la memoria, MOS es implementado para que utilice una pequeña cantidad de RAM. Usa menos de 500 bytes de memoria, incluyendo el kernel, los controles de tiempo y la pila de comunicación. Para conseguir la eficiencia energética, el controlador de eficiencia energética de MOS hace que el microcontrolador duerma después de ejecutar todas las tareas activas, reduciendo el consumo de energía a un rango de μA.

Una de las características principales de MOS es la flexibilidad en el soporte de múltiples plataformas como PCs, PDAs y diferentes plataformas de microsensores. Otra de las características destacada del diseño de MOS es el soporte de control remoto, permitiendo una reprogramación dinámica y un acceso remoto.

**eCos**

eCos (embedded Configurable operating system) es un sistema operativo de código abierto, gratuito y de operación en tiempo real, desarrollado para sistemas embebidos y para aplicaciones que necesiten un procesador con múltiples sesiones. Puede ser personalizado para cumplir los requisitos que la aplicación precise, con cientos de opciones, pudiendo llegar a la mejor combinación entre el rendimiento en tiempo real y el hardware necesario. Este sistema es programable bajo lenguaje C y tiene capas y APIs compatibles para POSIX y µITRON. Ecos fue diseñado para aparatos con un tamaño de memoria sobre cientos de kilobytes o con requerimientos en tiempo real. Puede ser usado en hardware con muy poca RAM soportando Linux empotrado a partir de un mínimo de 2 MB de RAM, sin incluir las necesidades de la aplicación y del servicio.
eCos funciona correctamente en una amplia variedad de plataformas hardware como pueden ser ARM, CalmRISC, FR-V, Hitachi H8, IA-32, Motorola 68000, Matsushita AM3x, MIPS, NEC V8xx, Nios II, PowerPC, SPARC y SuperH.
Incluido con la distribución de eCos se dispone de RedBoot, una aplicación de código abierto que usa la capa de abstracción de hardware de eCos que provee soporte de arranque para sistemas embebidos.
Está definido como estable, completamente testeado, certificado y con soporte, sin embargo, algunas de sus características no han sido liberadas como software libre.

**ContikiOS**

Contiki es un pequeño sistema operativo de código abierto, altamente portable y multitarea, desarrollado para uso para uso en pequeños sistemas, desde ordenadores de 8-bits a sistemas embebidos sobre microcontroladores, incluyendo nodos de redes de sensores.
A pesar de incluir multitarea y una pila TCP/IP, Contiki sólo requiere varios kilobytes de código y unos cientos de bytes de RAM. Un sistema totalmente completo con una Interfaz Gráfica de Usuario (GUI) que requiere aproximadamente 30 kilobytes de RAM.
El núcleo básico y la mayor parte de las funciones principales fueron desarrollados por Adam Dunkels en el grupo de sistemas de redes embebidas en el instituto sueco de ciencias computacionales.
Contiki fue diseñado para sistemas embebidos con poca cantidad de memoria. Una configuración típica de Contiki es 2 kilobytes de RAM y 40 kilobytes de ROM. Contiki consiste en un núcleo orientado a eventos, el cual hace uso de protothreads, sobre el cual los programas son cargados y descargados dinámicamente. También soporta multihilado apropiativo opcional por proceso, comunicación entre procesos mediante paso de mensajes a través de eventos,

al igual que un subsistema GUI opcional, el cual puede usar un soporte directo de gráficos para terminales locales, terminales virtuales en red mediante VNC o sobre Telnet.
Contiki funciona en una variedad de plataformas, desde microcontroladores embebidos, como el MSP430 y el AVR, a viejos computadores domésticos. El tamaño del código está en el orden de los kilobytes y el uso de la memoria puede configurarse para que sea de sólo unas decenas de bytes.

### SOS

SOS es un sistema operativo para redes de sensores que procura remediar algunas de las limitaciones propias de la naturaleza estática de muchos de los sistemas precursores a éste (por ejemplo TinyOS).
SOS implementa un sistema de mensajería que permite múltiples hebras entre la base del sistema operativo y las aplicaciones, las cuales pasan a ser módulos que pueden ser cargados o descargados en tiempo de ejecución sin interrumpir la base del sistema operativo.
El principal objetivo de SOS es la reconfigurabilidad. Ésta se define como la habilidad para modificar el software de nodos individuales en una red de sensores, una vez que éstos han sido desplegados físicamente e iniciado su funcionamiento. En el caso de encontrar un problema, en caso de no contar con esta solución, habría sido necesario recolectar todos los nodos para poder modificar su software.

### Nano-RK

Nano-RK es un sistema operativo completamente preventivo basado en reserva bajo tiempo real (RTOS) con soporte para redes multisalto adecuado para el uso en redes de sensores inalámbricas.
Nano-RK soporta multitareas preventivas con prioridad para asegurar que los plazos de las tareas son conocidos, además de soporte de CPU, red y sensores y actuadores.
Las tareas pueden especificar las demandas de recursos y el sistema operativo provee el acceso controlado y garantizado para los ciclos de CPU y los paquetes de red. Todos estos recursos forman la reserva de energía virtual que permite al sistema operativo controlar el nivel de energía del sistema y de las tareas.

# CAPÍTULO 3. DISEÑO E IMPLEMENTACIÓN DE UNA RED DE SENSORES INÁLAMBRICA MULTISALTO.

En este tercer capítulo se explica el diseño e implementación de una red de sensores inalámbrica multisalto, en primer lugar se describe brevemente la distribución de las motas que se han implementado en el edificio, seguidamente se describe el material utilizado, tanto de hardware como de software, durante los ensayos experimentales y el porqué de su elección y por último, los protocolos utilizados en ContikiOS, que es el sistema operativo que se ha utilizado para las motas Z1 de la empresa Zolertia.

## 3.1. Distribución de las motas en el edificio

La red inalámbrica de sensores, se implementa en el edificio B4 del PMT (Parque Mediterráneo de la Tecnología) de Castelldefels (Barcelona), que está formado por tres plantas, la planta baja, primera planta y segunda planta.

En la **figura 3.1.**, se muestra el plano correspondiente a cada planta, dónde se indican los sensores utilizados con su ID que los identifica. Cada sensor tiene su función como indica en la **tabla 3.1.:**

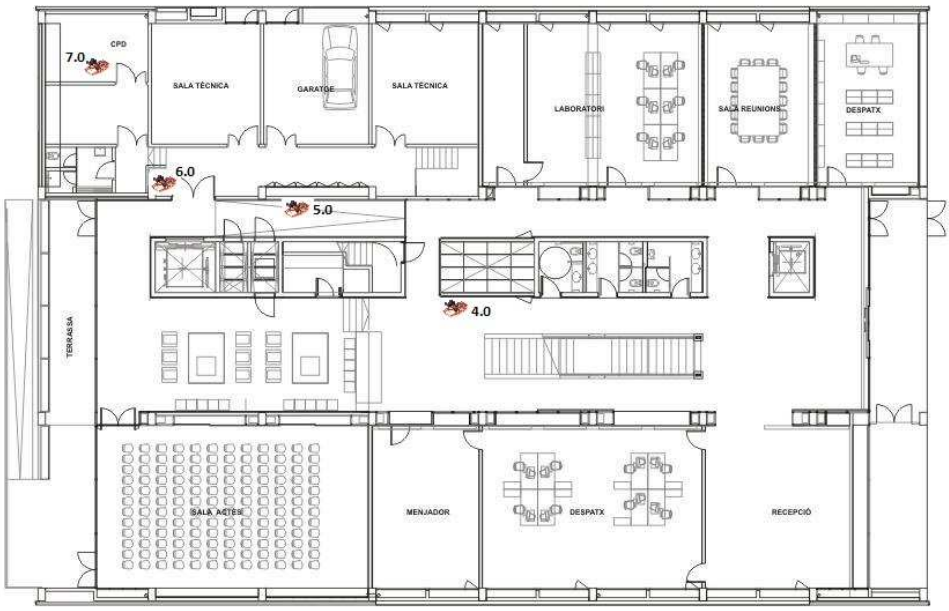
**Tabla 3.1.:** Función de cada sensor con su ID que los identifica.

| ID sensor | Operación |
|---|---|
| **1.0** | Sensor de temperatura y estado de su batería (Sink Node) |
| **2.0** | Sensor de temperatura y estado de su batería (Router Node) |
| **3.0** | Sensor de temperatura y estado de su batería (Router Node) |
| **4.0** | Sensor de temperatura y estado de su batería (Router Node) |
| **5.0** | Sensor de temperatura y estado de su batería (Router Node) |
| **6.0** | Sensor de temperatura y estado de su batería (Router Node) |
| **7.0** | Sensor de temperatura, estado de su batería consumo energético (Router Node) |

*El sensor con ID 7.0 mide la temperatura ambiente y el consumo del aire acondicionado.

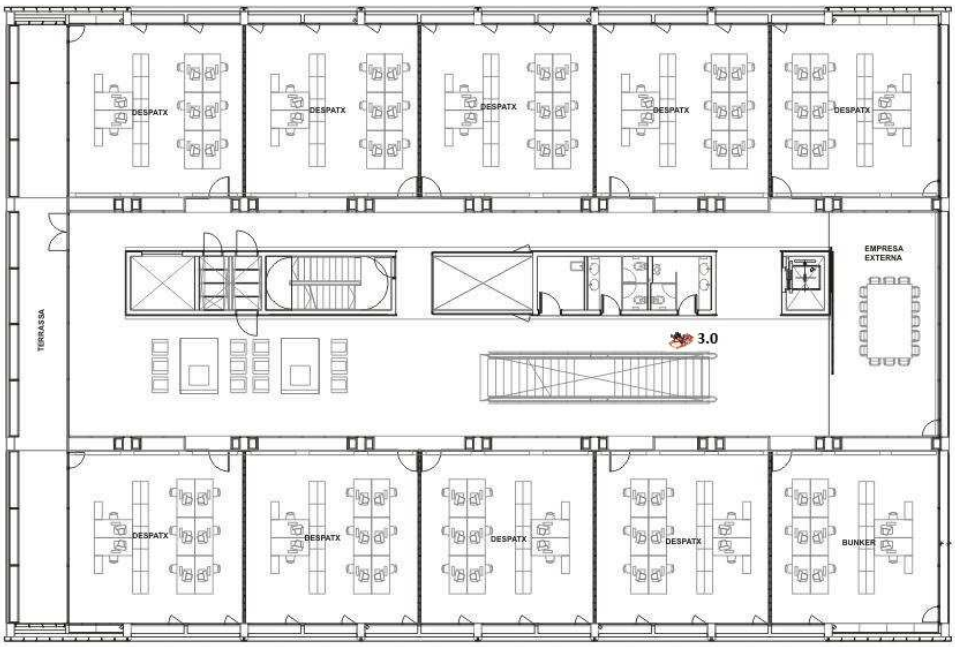
CENTRE TECNOLÒGIC TELECOMUNICACIONS DE CATALUNYA

## PLANTA BAJA




CENTRE TECNOLÒGIC TELECOMUNICACIONS DE CATALUNYA

## PRIMERA PLANTA



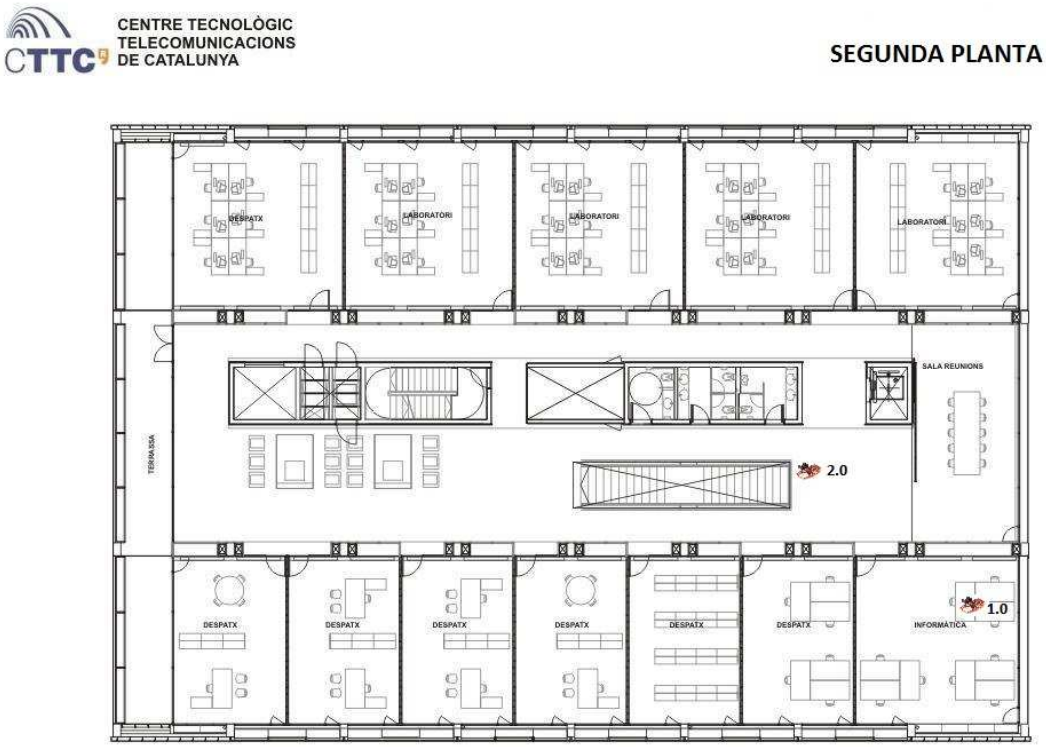



**Figura 3.1.:** Planos del edificio B4 del PMT (CTTC).

Como se puede observar se han utilizado 7 sensores para poder transportar la información de extremo a extremo del edificio.

## 3.2. Hardware y software utilizado

En esta sección se describe brevemente el material utilizado para implementar la red de sensores, tanto de hardware como de software, durante los ensayos experimentales y el porqué de su elección.
El hardware que se ha utilizado son las motas Z1 de Zolertia para formar la red, el portátil Dell Vostro, para poder programar la aplicación que se ejecuta en las motas y visualizar los datos recibidos por el sink, adaptadores de corriente y pilas para alimentar las motas y cables USB – miniUSB.
En cuanto al software que se ha empleado, se describe brevemente el sistema operativo Contiki y su simulador Cooja.

### 3.2.1. Motas Z1 de Zolertia

Se han elegido las motas Z1, ya que las fabrica la empresa Zolertia, la cual reside en Cerdanyola del Vallés y así impulsar la industria catalana. Otra razón es que la mota Z1 es un módulo de bajo consumo inalámbrico compatible con protocolos del estándar IEEE 802.15.4 y Zigbee con la intención de ayudar a los diseñadores a practicar y desplegar sus propias aplicaciones y prototipos, con la

mejor relación entre el tiempo de desarrollo y la flexibilidad hardware. El núcleo de su arquitectura se basa en la familia de microcontroladores y transceptores radio MSP430+CC2420 de Texas Instruments, el cual lo hace compatible con motas basadas en esta misma arquitectura y es totalmente compatible tanto con TinyOS como con Contiki. En la sección 2.1 (motas comerciales) se ha descrito más detalladamente las características principales de la mota Z1.

### 3.2.2. Portátil Dell Vostro 130

Se ha empleado un portátil Dell de gama Vostro, modelo 130 y dispone de las siguientes características:

- Microprocesador Intel Core i3-380UM Arrendale, 1.33 GHz. de 2 núcleos, modelo ULV (Ultra Low Voltage).
- GPU integrada en la CPU.
- Memoria RAM: 4 GB DDR3.
- Pantalla de 13.3 pulgadas, 1.366×768 píxeles de resolución y salidas de vídeo HDMI y VGA.
- Microsoft Windows 7 Professional de 64 bits y Ubuntu 12.04 LTS Desktop de 64 bits (Dual boot).
- Se ha instalado Ubuntu 12.04 LTS, ya que es una versión con soporte técnico extendido.

Para poder configurar y alimentar las motas se han utilizado los siguientes materiales:

- Adaptadores de corriente – USB para poder alimentar directamente las motas que están ubicadas cerca a las tomas de corriente, sin utilizar pilas.
- Cables USB – mini USB para poder conectar la mota al adaptador de corriente y para poder configurarlos con el PC.
- Pilas AAA de 1.2 V y 2700 mAh de capacidad, para alimentar las motas que no tienen la opción de ser alimentadas directamente por USB.

### 3.2.3. Sistema Operativo Contiki 2.5.

Contiki es un sistema operativo open source, multi-tarea desarrollado para plataformas con recursos limitados de procesamiento, energía y memoria. Implementado por un grupo de académicos del “Swedish Institute of Computer Science” liderado por Adam Dunkels. Sus creadores lo desarrollaron pensando en “Internet of things”, que tiene como idea que en un futuro los objetos que nos rodean tengan la capacidad de comunicarse de forma inalámbrica y de esta manera se podrán ubicar y controlar remotamente.

Se ha elegido el sistema operativo Contiki, ya que se adapta mejor a las motas Z1 que el más popular sistema operativo TinyOS. La adaptación de TinyOS para las motas z1 no se ha completado y cuando el módulo está dormido presenta un consumo energético de 1 mA, y debería ser del orden de µA.

Contiki aún no se ha desarrollado por la comunidad WSN, que es una de las facetas que se pretende realizar con el estudio de éste TFC. Otras de las razones de su elección es que soporta una gran variedad de protocolos MAC y de enrutamiento, está escrito en el lenguaje de programación C y dispone de una herramienta llamada Cooja, que consiste en un simulador, que permite compilar programas realizados y simular la red.

### 3.2.4. El simulador COOJA

El simulador Cooja viene incorporado en el sistema operativo de Contiki. Cooja nos ofrece una visión general de alto nivel de simulación de redes de sensores y nos ayuda a entender más fácilmente sus problemas, limitaciones y características.
La simulación de una mota con Contiki en Cooja, es un sistema Contiki real compilado y en ejecución. El sistema se controla y se analiza por Cooja mediante la compilación de Contiki para la plataforma nativa como una librería compartida, y carga la librería en Java utilizando Java Native Interfaces (JNI). En la misma simulación se pueden compilar y cargar diferentes librerías Contiki, representando diferentes tipos de nodos de sensores (redes heterogéneas). Cooja controla y analiza Contiki a través de algunas funciones como por ejemplo, informar al sistema Contiki para tratar un evento, o recoger la información para poder analizarla.

## 3.3. Protocolos utilizados en ContikiOS

Debido a la limitación temporal del proyecto, únicamente se han utilizado los protocolos de comunicaciones que están disponibles en Contiki, sin embargo, Contiki es un sistema operativo ampliamente utilizado, y sus protocolos también. A continuación se describen brevemente.

El sistema operativo Contiki ofrece tres stacks de comunicaciones. El stack TCP/IP, que proporciona una red IPv4, el stack RPL-IPv6 que proporciona una red IPv6 y el stack Rime, que es un conjunto de protocolos de red ligeros personalizados diseñados específicamente para redes inalámbricas de baja potencia, por ello se ha elegido estudiar éste último, ya que es una pila de red alternativa que está destinada a ser utilizada cuando la sobrecarga de las pilas IPv4 o IPv6 es prohibitivo que es nuestro caso.

El stack Rime, es un stack modular estructurado en capas muy simples, de manera que los encabezados sean lo más pequeños posibles. Las distintas capas de Rime implementan funcionalidades asociadas a los servicios de comunicación que se quieren brindar.
Rime ofrece una amplia gama de primitivas y protocolos, como la recogida de datos multi-salto (multi-hop data collection), enrutamiento unicast multi-salto y comunicaciones broadcast multi-salto. Las primitivas se pueden utilizar solas o combinadas para formar protocolos y mecanismos más complejos. En el anexo se explica detalladamente la pila de protocolos Rime.

En la **figura 3.2.** se puede ver la pila de protocolos Rime que se ha utilizado en el proyecto, consta de la capa radio, la capa RDC, la capa MAC y la capa de red. En el siguiente esquema se muestra los diferentes protocolos estudiados agrupados en dichas capas.

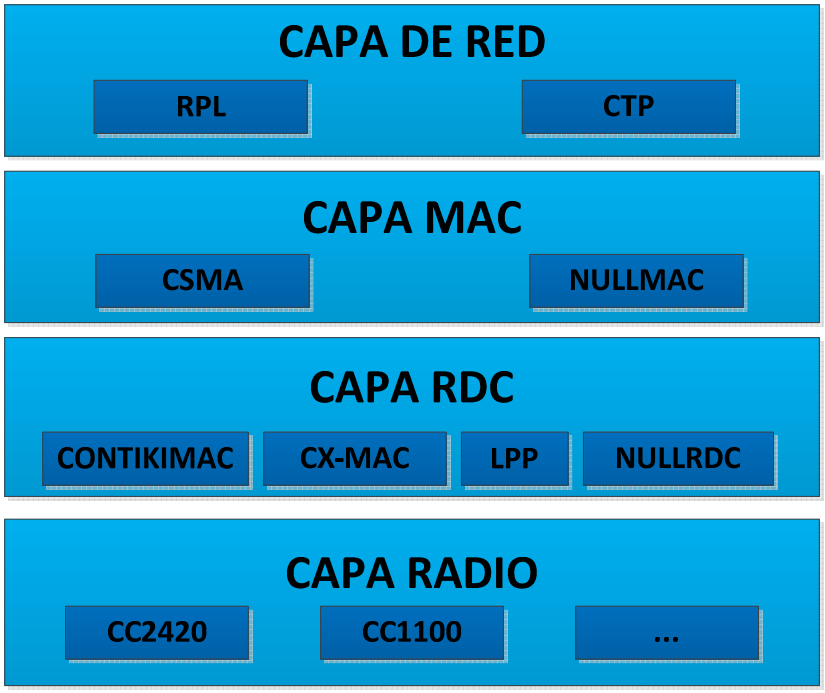



**Figura 3.2.:** Pila de protocolos de comunicaciones.

### 3.3.1. Capa Radio en Contiki

La pila de protocolos Rime empieza con el driver de la radio, el cual se encarga de enviar y recibir datos por el aire. En el proyecto se utiliza la radio CC2420, que opera en la banda de 2.4Ghz bajo el estándar IEEE 802.15.4.

### 3.3.2. Capa RDC en Contiki

En las redes de baja potencia, el transceptor radio debe estar apagado tanto como sea posible para ahorrar energía. En Contiki, esto se hace con la capa de Radio Duty Cycling (RDC). Contiki tiene varios drivers o controladores de RDC. Los más utilizados son ContikiMAC, X-MAC, CX-MAC, LPP y NullRDC. Seguidamente se explican más detalladamente los protocolos estudiados en el proyecto.

ContikiMAC es un protocolo de radio duty cycling el cual usa wake-ups periódicos para escuchar la transmisión de paquetes de sus vecinos. Si un paquete de transmisión es detectado durante un wake-up, el receptor es capaz de recibirlo. Cuando el paquete es recibido satisfactoriamente, el receptor envía un acuse de recibo. Para transmitir un paquete, el emisor va enviando el paquete de datos repetido hasta recibir un paquete de reconocimiento del receptor. Los

paquetes que se envían en broadcast no esperan ningún ACK. En su lugar el emisor envía repetidamente el paquete completo durante el intervalo para garantizar que todos los vecinos lo han recibido.
En la **figura 3.3.** se muestra como los nodos duermen la mayor parte del tiempo y periódicamente se despiertan para ver detectar la transmisión de paquetes, el receptor permanece despierto para recibir el siguiente paquete y envía su correspondiente reconocimiento (ACK).

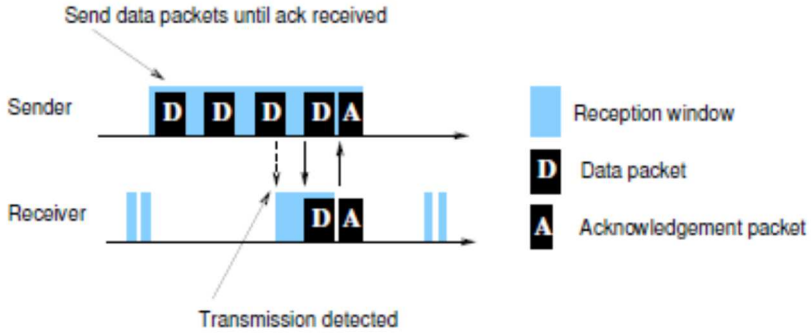



**Figura 3.3.:** Funcionamiento del protocolo de radio duty cycling ContikiMAC.

En la **figura 3.4.** se puede ver una transmisión broadcast en el que se envían los paquetes repetidos durante todo el intervalo en el que está en estado wake-up.

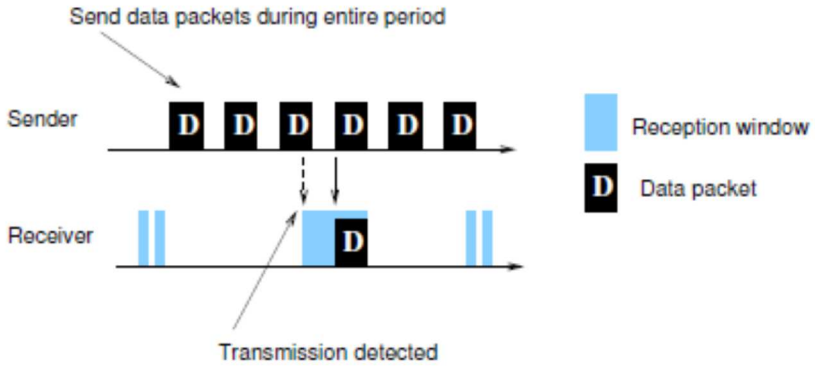



**Figura 3.4.:** Transmisión Broadcast en ContikiMAC.

Contiki X-MAC (CX-MAC) se basa en el diseño del X-MAC original (explicado en el anexo III detalladamente), pero añade soporte para el tráfico broadcast, sensibilización de fase, flujo de transporte, y retransmisiones opcionales de capa MAC.
En Contiki X-MAC los nodos mantienen un periodo estricto en el que encienden su radio, escuchan los paquetes beacon entrantes de los vecinos, apagan su radio si no se ha recibido un paquete beacon, y mantienen la radio apagada durante el resto del período del ciclo de trabajo. Si se ha recibido un paquete beacon, el nodo transmite un paquete de acuse de recibo al emisor. Cuando el emisor recibe un beacon de confirmación desde el nodo al que está enviando beacons, el emisor envía inmediatamente el paquete de datos.
Para implementar un protocolo de difusión en X-MAC original, hay al menos dos alternativas. La primera cuando un nodo escucha un paquete beacon con la dirección de broadcast, no transmite un paquete beacon de confirmación, sino

que enciende su radio. El emisor envía un período completo de paquetes beacons, seguido por el paquete de datos. Este mecanismo garantiza que todos los nodos están despiertos en el momento en que llegue el paquete de difusión, pero se desperdicia energía, ya que todos los nodos tienen que estar despiertos hasta que llegue el paquete de datos.
La segunda forma de implementar broadcast en X-MAC es más simple. Cuando se envía un paquete broadcast, en lugar de enviar beacons, el emisor se limita a enviar a sí mismo el paquete de broadcast. Esto tiene la ventaja de que los vecinos no necesitan mantener su radio durante un tiempo más largo que un intervalo de comprobación.
En Contiki X-MAC, la emisión en broadcast se realiza con transmisiones repetidas del paquete de datos. Cada vez que un nodo envía un paquete a su vecino, el emisor registra el período en que el receptor estaba despierto. El emisor sabe el periodo en que el vecino estaba despierto, ya que a continuación el receptor envía un paquete beacon de confirmación. Cuando el emisor es consciente del instante en el que su vecino estaba despierto, sabrá su estado.
Cuando un emisor está a punto de enviar un paquete a un vecino y es consciente de su fase, el emisor no empieza a enviar sus beacons hasta que el vecino está despierto. Si el emisor no encuentra al vecino despierto simplemente seguirá enviando beacons durante todo el periodo. Así, el mecanismo será correcto sin tener en cuenta si la fase la memoriza correctamente o no. Sin embargo con una información correcta de fase, el mecanismo es más eficiente. Si se implementa bien, el emisor sólo tendrá que enviar un solo beacon cuando se conoce la fase del vecino.
Contiki X-MAC proporciona compatibilidad con la capa enlace y el estándar 802.15.4, mediante la formación de sus mensajes de acuerdo con la especificación 802.15.4. Todos los mensajes, tanto de paquetes beacons, como mensajes de datos, utilizan la trama 802.15.4 y su formato de direccionamiento. Esto hace que sea posible poder inter-operar en una red 802.15.4 y en una red Contiki X-MAC.

En el proyecto se ha decidido utilizar el controlador CX-MAC como mecanismo de RDC, así se evita posibles errores de transmisión radio, ya que da la posibilidad de que funcione con un conjunto más amplio de radios y es compatible con X-MAC, pero sus requerimientos de sincronismo son menores. Se probó con ContikiMAC, con X-MAC y con CX-MAC y la red se comportaba mejor con CX-MAC, ya que se observaba menores pérdidas de paquetes.

Los controladores RDC mantienen la radio apagada tanto como sea posible y comprueban periódicamente el medio, para ver si existe actividad radio. Cuando se detecta actividad, la radio se mantiene encendida para recibir el paquete. La frecuencia de comprobación del medio se da en Hz, especificando el número de hercios por segundo, y por defecto es de 8 Hz. Los ajustes típicos son 2, 4, 8 y 16 Hz.

En el proyecto se ha ajustado a 16Hz, así el receptor observará el medio con mayor frecuencia para ver si es un paquete que es para él y así tendrá menos perdidas de paquetes. Esto implica un aumento en el consumo energético ya que tendrá que encender la radio más a menudo.

### 3.3.3. Capa Mac en Contiki

La capa MAC (Control de Acceso al Medio) se encuentra en la parte superior de la capa de RDC. La capa MAC es responsable de evitar colisiones en el medio radio y retransmitir los paquetes si hubo una colisión. Contiki proporciona dos capas MAC: un mecanismo CSMA (Carrier Sense Multiple Access) y un mecanismo NullMAC, que no hace ningún tipo de procesamiento de nivel MAC.
La principal diferencia entre los mecanismos CSMA y NullMAC es que NullMAC se basa en el protocolo Aloha que se limita a enviar los paquetes si tiene para enviar y si colisiona con otra transmisión los intenta enviar más tarde y con CSMA cada nodo anuncia opcionalmente su intención de transmitir antes de hacerlo para evitar colisiones entre los paquetes de datos. De esta forma, el resto de nodos de la red sabrán cuando hay colisiones y en lugar de transmitir la trama en cuanto el medio está libre, se espera un tiempo aleatorio adicional corto y solamente si, tras ese corto intervalo el medio sigue libre, se procede a la transmisión reduciendo la probabilidad de colisiones en el canal.
CSMA es el mecanismo por defecto en Contiki. La capa MAC recibe los paquetes entrantes desde la capa RDC y utiliza la capa de RDC para transmitir los paquetes. Si la capa de RDC o la capa radio detecta una colisión de radio, la capa MAC puede retransmitir el paquete en un punto posterior en el tiempo. El mecanismo CSMA es actualmente la única capa MAC que retransmite los paquetes si se detecta una colisión.

En el proyecto se ha escogido el mecanismo CSMA ya que es el protocolo MAC por defecto en Contiki y permite detectar posibles colisiones para luego poder hacer sus respectivas retransmisiones, si hubo colisión y así tener menores perdidas de paquetes.

### 3.3.4. Capa de Red en Contiki

Por último esta la capa de red. La capa de red se encarga del enrutamiento de los paquetes y se ha utilizado el algoritmo CTP (Collection-Tree-Protocol), ya que para la aplicación estudiada, solo se necesita tráfico upward y, a diferencia del protocolo RPL, que se explica en el anexo III, CTP solo proporciona éste tipo de enrutamiento. Por lo tanto CTP es la opción idónea para la aplicación, ya que tiene menos overhead de mensajes de control.
Otra causa de la elección de este protocolo es que es el protocolo que trae incorporado Contiki para utilizar con el stack RIME. CTP fue desarrollado especialmente para redes con pérdidas y recursos limitados de memoria. Se encarga de alimentar las tablas de ruteo de los nodos y designar los nodos padres (preferred parents).

Seguidamente se explica el algoritmo CTP más detalladamente.

CTP es un protocolo de recolección de datos con topología tipo árbol. Los nodos forman una red de tipo árbol, dónde uno de ellos es la raíz, llamado sink. Con CTP un nodo no envía un paquete a una raíz en particular, en su lugar se elige

de forma implícita el nodo que corresponde al siguiente salto generando la ruta hacia la raíz.

El protocolo CTP asume que la capa de enlace de datos proporciona una dirección de difusión local eficiente, reconocimientos síncronos para paquetes unicast, soporte de protocolos de nivel superior y que tenga una fuente de un solo salto y los campos de destino.

CTP asume que tiene estimaciones de calidad del enlace de varios vecinos cercanos y proporciona una estimación del número de transmisiones que se necesita para que un nodo al enviar un paquete de unicast, reciba un acuse de recibo con éxito.

CTP dispone de varios mecanismos para mejorar la fiabilidad de entrega, pero no garantiza una entrega fiable al 100% y está diseñado para tasas de tráfico relativamente bajas.

CTP utiliza la métrica ETX (Expected Transmission Count). ETX tiene en cuenta la calidad del enlace. Una raíz tiene un ETX de 0. El ETX de un nodo es el ETX de su padre más el ETX del enlace con su padre. Dada la opción válida de rutas, CTP debe elegir la ruta que tenga el valor de ETX más bajo.

Los bucles de enrutamiento son un problema que puede surgir en una red CTP, y generalmente ocurren cuando un nodo elige una ruta que tiene un ETX significativamente superior a su antigua ruta, tal vez en respuesta a la perdida de conectividad con su padre. Si la nueva ruta incluye un nodo que era un descendiente, entonces se produce un bucle.

La duplicación de paquetes es otro problema adicional que puede ocurrir en CTP, y se produce cuando un nodo recibe un paquete de datos con éxito y transmite un ACK, pero el ACK no se recibe. El remitente retransmite el paquete y receptor recibe el mismo paquete por duplicado. Esto puede tener efectos desastrosos sobre múltiples saltos, ya que la duplicación es exponencial. Por ejemplo, si cada tramo en promedio produce un duplicado, a continuación, en el primer salto habrá dos paquetes, en el segundo habrá cuatro, en el tercero ocho…

Los bucles de enrutamiento complican la supresión de duplicados y puede causar que un nodo reciba un mismo paquete más de una vez. Por lo tanto si un nodo suprime duplicados basándose únicamente en la dirección de origen y numero de secuencia, se eliminaran los bucles de enrutamiento.

# CAPÍTULO 4. DISEÑO DE LA APLICACIÓN EN CONTIKI

En este cuarto capítulo, en primer lugar se explica cómo se estructura el archivo de la aplicación en Contiki, y las funciones que se han diseñado, para poder hacer la medición de la temperatura ambiental de los establecimientos del edificio, el estado de la batería de la mota y el consumo energético que está realizando en ése momento por el uso de los splits del aire acondicionado. Posteriormente se explica cómo se compila una aplicación en Contiki.

## 4.1. Archivo de la aplicación en Contiki

La estructura básica de una aplicación en Contiki se basa como se muestra a continuación en la **figura 4.1.** Esta estructura es genérica para Contiki y se puede utilizar como punto de partida para programar las aplicaciones. Para comprender la estructura del proceso, se comenta después de cada línea. Las palabras resaltadas en rojo, pueden ser reemplazadas con el nombre y la descripción de la aplicación que se realice.

```
#include "contiki.h" // Filesystem file, ALWAYS include
#include "/dev/leds.h // leds driver
#include "/dev/sensor-button.h"// user button driver

PROCESS(blink_process, "blink example"); // Definition of the process, and a reference name when debugging
AUTOSTART_PROCESSES(&blink_process);  // Load the proccess  at boot
PROCESS_THREAD(blink_process, ev, data) // Content of the process
{
        /* Specify an action when a process exits, always comes immediately before PROCESS_BEGIN() */
  PROCESS_EXITHANDLER(goto exit);
  PROCESS_BEGIN();  //Defines the beginning of a process, always comes first
  /* Initialize variables and stuff here */
    while(1) {  // Infinite loop
          /* User space for programming the application */
  exit:
     PROCESS_END();  //Defines the end of a process
              }
}
```

**Figura 4.1.:** Estructura básica de una aplicación en Contiki.

El archivo de la aplicación Contiki, es el archivo que contiene nuestro código fuente en el que programamos la aplicación para poder hacer la medición de los siguientes parámetros:

- **Temperatura:** Indica la temperatura ambiental que hay en ese momento en el establecimiento.
- **Batería:** Indica el estado de la batería de la mota. En el caso de que esté alimentado directamente con una fuente eléctrica, indica un voltaje de 3,2V.
- **Consumo:** Indica el consumo energético que se está realizando en ése momento por el uso de los splits del aire acondicionado.

Para ello se ha dividido la aplicación en tres funciones:

```
static const char * getTemperature()
static const char * getBattery()
static const char * getConsumption ()
```

En el anexo se encuentra el archivo de la aplicación diseñada, dónde se pueden ver las tres funciones diseñadas.
En la misma aplicación, se ha programado la frecuencia a la que un nodo envía un paquete con destino al sink, sea de 30 segundos, de la siguiente forma:

```
/* Send a packet every 30 seconds. */
if(etimer_expired(&periodic)) {
     etimer_set(&periodic, CLOCK_SECOND*30);
     etimer_set(&et, random_rand()%(CLOCK_SECOND*30);
     }
```

## 4.2. Compilar una aplicación en Contiki

Para compilar una aplicación en Contiki existen 3 archivos: el archivo de la aplicación Contiki, explicado en el apartado anterior, que contiene la aplicación que se va a emplear, el archivo opcional de configuración de nuestra aplicación y el Makefile. La estructura Makefile utilizada por Contiki es bastante simple:

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

La primea línea indica la ubicación de la raíz de las fuentes Contiki, en la segunda línea se especifica las aplicaciones que va a compilar, y en la tercera línea incluye todo el sistema Contiki, que contiene las definiciones del core de Contiki y señala el Makefile específico de nuestra plataforma de destino. Makefile.include siempre debe estar ubicado en la parte superior del código fuente Contiki.

Nuestro Makefile quedaría así:

```
CONTIKI = ../..
all: example-abc example-mesh example-collect example-trickle example-polite\
     example-rudolph0 example-rudolph1 example-rudolph3 example-rucb \
     example-runicast example-unicast example-neighbors example-collectTBC

include $(CONTIKI)/Makefile.include
CFLAGS += -DPROJECT_CONF_H\”project-conf.h\”
```

En la última línea se ha añadido el archivo project-conf.h al Makefile. Un proyecto en Contiki puede tener un archivo opcional de configuración llamado project-conf.h. Este archivo no está activo por defecto, para activarlo, hay que añadir la siguiente línea al fichero Makefile del proyecto:

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

Para crear el archivo de project-conf.h debe estar ubicado en el directorio del proyecto y llamarse project-conf. El archivo project-conf.h puede activar una serie de opciones de configuración de Contiki.

Con el archivo project-conf.h podemos especificar la frecuencia del canal RDC. Esto se hace mediante la adición de un #define en el archivo project-conf.h que especifica la frecuencia de del canal, en Hz:

```
#define NETSTACK_CONF_RDC_CHANNEL_CHECK_RATE 16
```

El valor por defecto de la frecuencia de canal, si no se especifica en el archivo project-conf.h es de 8 Hz, como ya se ha comentado anteriormente.

También se puede especificar el controlador RDC que se utiliza en el proyecto, añadiendo otro #define en el archivo project-conf.h:

```
#define NETSTACK_CONF_RDC nullrdc_driver
```

La nomenclatura de los controladores RDC en Contiki son las siguientes:

```
contikimac_driver

xmac_driver

cxmac_driver

lpp_driver

nullrdc_driver
```

También se puede especificar el controlador MAC, añadiendo otro #define al archivo project-conf.h:

```
#define NETSTACK_CONF_MAC nullmac_driver
```

En el ejemplo anterior se especifica que no se quiere utilizar el controlador MAC, pero se puede especificar que se utilice CSMA así:

```
#define NETSTACK_CONF_MAC csma_driver
```

En conclusión con el archivo project-conf.h se puede especificar el controlador RDC y los protocolos de control de acceso al medio (MAC) ya que son una parte importante de la pila de protocolos en Contiki, ya que determinan el consumo de energía de los nodos y su comportamiento cuando la red está congestionada.

Nuestro archivo de configuración project-conf.h quedaría así:

```
#define NETSTACK_CONF_RDC_CHANNEL_CHECK_RATE 16

#define NETSTACK_CONF_RDC cxmac_driver
```

Donde se especifica una frecuencia de canal a 16 Hz y el controlador RDC CX-MAC, y no se especifica el protocolo de acceso al medio, ya que por defecto es CSMA.

Después de haber especificado el archivo de proyecto-conf.h en el Makefile, es necesario limpiar las dependencias existentes con el comando make clean:

```
make TARGET= example-collectTBC clean
```

Esto solo es necesario la primera vez, después se compila como de costumbre.

Para compilar una aplicación hay que especificar la plataforma de destino utilizando el TARGET = plataforma de sintaxis, y si no se especifica de forma predeterminada se compilará la plataforma nativa.

```
make TARGET=z1 example-collectTBC
```

Para compilar y cargar la aplicación en la mota z1:

```
make TARGET=z1 example-collectTBC.upload
```

Una vez compilado para lanzar la aplicación:

```
make login
```

El Makefile.z1 proporciona comandos útiles en platform/z1/Makefile.z1, que se incluye al target = z1. A continuación se describe algunas funcionalidades:

- **savetarget:** Guarda la plataforma z1 como destino predeterminado en el archive Makefile.target, a partir de ahora podemos compilar sin tener que teclear TARGET=z1**.**
- **z1-motes:** Lista las motas z1 conectadas al ordenador.
- **z1-motelist:** Anota la referencia de las motas z1 conectadas al ordenador, junto con el dispositivo.
- **z1-reset:** Restablece los nodos conectados.
- **linslip:** Realiza una conexión sobre IP hacia el nodo.
- **serialdump:** Imprime la marca de tiempo y la salida de la mota z1.
- **serialview:** Igual que el serialdump, pero permite la interacción con la mota z1.
- **login:** Imprime la salida del nodo conectado, sin la marca de tiempo.

Para modificar manualmente el ID de nodo y la dirección MAC (y por tanto la dirección IPv6, ya que se forma a partir de la dirección MAC), viene preparado un programa que te permite grabar la dirección MAC y el ID del nodo en la memoria flash, así:

```
make clean && make burn-nodeid.upload nodeid=158 nodemac=158 && make z1-reset
&& make login
```

Dónde ahora el nodo sensor tendrá la dirección MAC y el ID número 158.

# CAPÍTULO 5. SIMULACION DE LA RED INALÁMBRICA EN COOJA

En este capítulo se describe cómo funciona el simulador Cooja y el entorno de simulación de la red inalámbrica de sensores multisalto, simulando la distribución de los sensores de la red implementada en el edificio del CTTC.

Cooja se encuentra en la carpeta de herramientas de Contiki y se ejecuta en el terminal con “ant run” como se muestra en la **figura 5.1.:**


**Figura 5.1.:** Comando para lanzar Cooja desde Contiki.

En la **figura 5.2.** se muestra la pantalla principal de Cooja:


**Figura 5.2.:** Pantalla principal de Cooja.

Para crear la simulación, se ha de ir al menú File / New Simulation y se abrirá una ventana, como se muestra en la **figura 5.3.** en la que primero da la opción de poner el nombre a la simulación y posteriormente en ajustes avanzados, se puede elegir el tipo de radio de propagación que se utilizará en la simulación, entre los cuales esta UDGM (Unit Disk Graph Medium) siendo probablemente el tipo más simple, ya que sólo depende de la distancia y de la transmisión de energía.

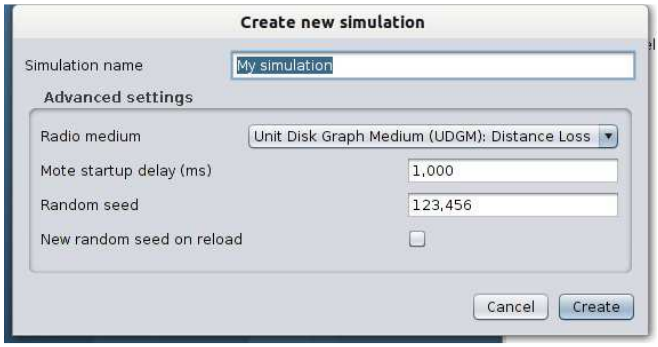



**Figura 5.3.:** Ventana de creación de una nueva simulación.

Al clicar en “Crear” se abrirá la simulación, como se muestra en la **figura 5.4.:**

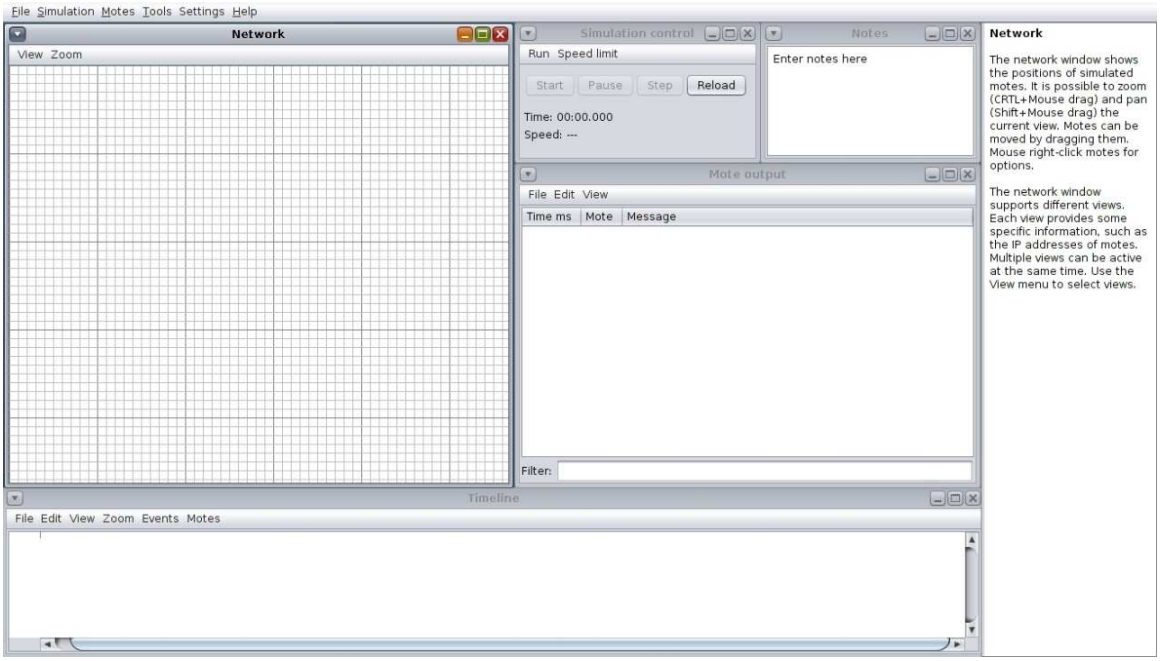



**Figura 5.4.:** Primera visión de la simulación.

El siguiente paso es establecer el tipo de mota que se simulará. En Cooja, se puede crear una lista de tipos de motas, definir sus parámetros y asignar el código del programa, para ello se ha de ir a “Types of Motes / Create Mote Type/z1 Mote Type”, z1, para poder simular la mota de Zolertia, se le da una pequeña descripción y se especifica el archivo fuente (el archivo .c) y así el campo de comandos de compilación se activa y se puede especificar instrucciones específicas para la compilación. Posteriormente se pulsa el botón de compilar “compile”, que compilará el archivo .c mostrando la salida de resultados de la compilación y si se tiene éxito, se dispondrá del botón de crear la simulación como se muestra en la **figura 5.5.:**

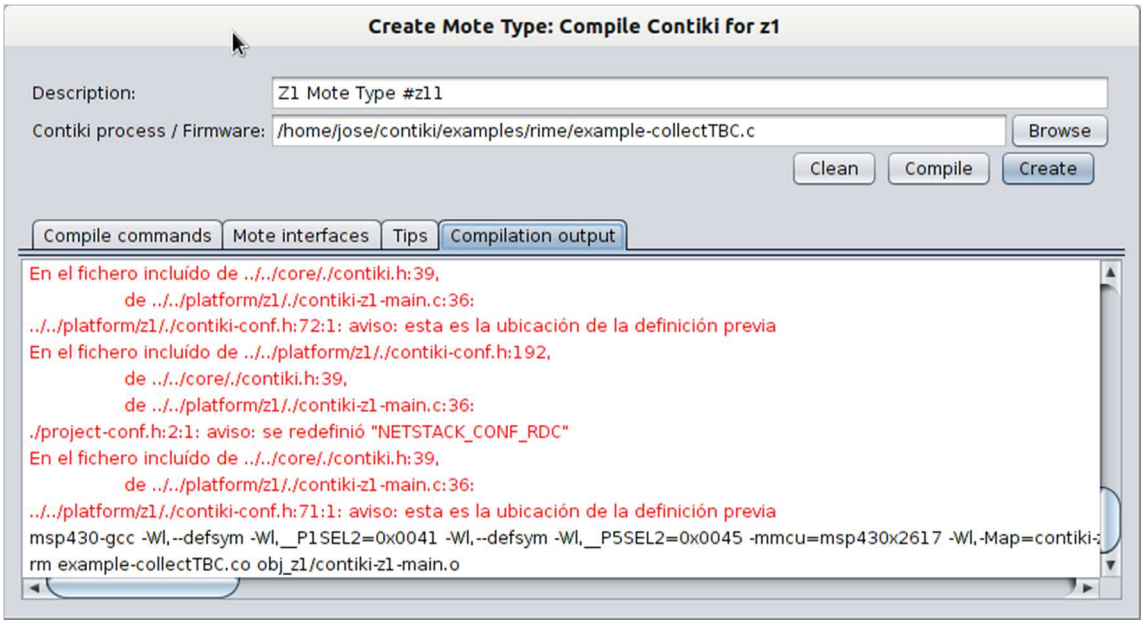



**Figura 5.5.:** Creación del tipo de mota Z1, elección del código y compilación del código.

En la figura anterior se puede ver como se ha compilado la aplicación que se ha creado, llamada example-collectTBC.c.

Una vez compilada la aplicación y se ha creado el tipo de mota, se ofrece la oportunidad de añadir motas en la pestaña de “Add motes” como indica en la **figura 5.6.**, La cual permite establecer el número de motas y su posicionamiento que posteriormente se podrá modificar, por ello una opción es posicionarlas aleatoriamente y luego modificar su posición.

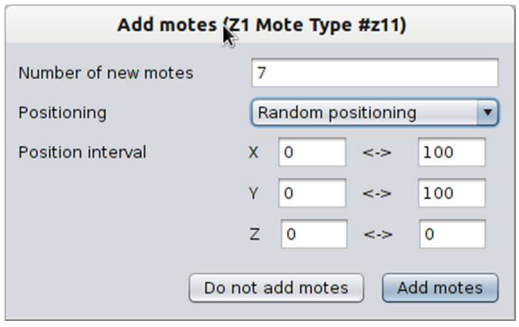



**Figura 5.6.:** Adición de motas Z1.

En la **figura 5.7.** se muestra la ventana Network, un ejemplo de cómo quedarían las motas de forma aleatoria.

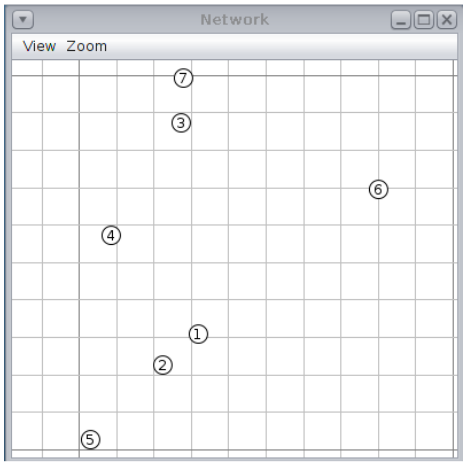



**Figura 5.7.:** Ventana Network.

En el entorno de simulación existe una ventana llamada “Control Panel” como se muestra en la **figura 5.8.,** permite ajustar varios parámetros de tiempo. La principal, es el retraso de la simulación. Se puede ejecutar la simulación a cualquier velocidad, con velocidades que son clave “Sin demora” y “Tiempo real”.


**Figura 5.8.:** Ventana Control Panel.

Desde la ventana del visualizador de simulación “Simulator Visualizer”, se puede seleccionar los perfiles que se pueden ver por cada mota mostrando una lista de opciones que se pueden seleccionar. Así se podrá ver lo que está pasando en la simulación. En **la figura 5.9.** se muestra a la izquierda las opciones visuales y a la derecha dos nodos comunicándose.

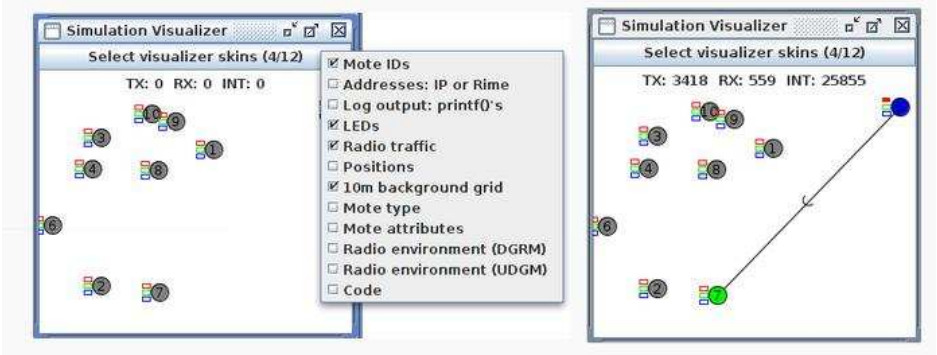



**Figura 5.9.:** Lista de opciones de visionado de las motas y dos nodos comunicándose.

Los LEDs muestran el estado de la mota, en azul cuando la mota transmite, en verde cuando está recibiendo y en rojo cuando la señal está siendo interferida.

La ventana Mote output muestra la salida de texto de lo que sería la interfaz USB a través del comando printf(), se utiliza para depurar el funcionamiento de un programa. La información se presenta en tres columnas: Time ms, Mote, Message. La columna Time ms muestra el tiempo en ms de lo que está ocurriendo, la columna Mote indica la ID de la mota y la columna Message contiene el resultado de las líneas que los nodos hayan impreso con su printf(). En la **figura 5.10.** se muestra la ventana Mote output.

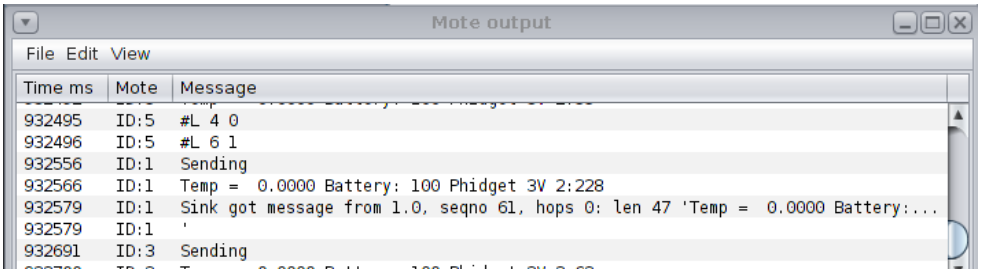



**Figura 5.10.:** Ventana Mote output.

Dada la gran cantidad de información que se muestra es de gran utilidad poder seleccionar que se quiere visualizar. Con el campo Filter podemos escribir una palabra para utilizarla como criterio de filtrado de los mensajes printados. Si se filtra por ejemplo escribiendo ^ID:[4]$, podremos ver exclusivamente los mensajes que printa el nodo 4, como se muestra en la **figura 5.11.**

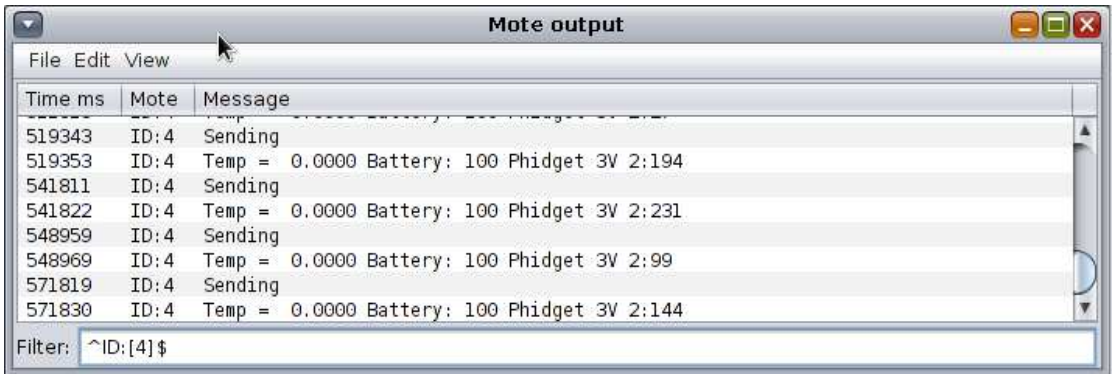



**Figura 5.11.:** Filtrado de mensajes.

Con el botón derecho del ratón sobre una fila dentro de la ventana, permite visualizar las opciones disponibles:


**Figura 5.12.:** Opciones de la ventana Mote output.

Se dispone de la opción Focus (Space) que aparece en oscuro, la cual permite marcar la ocurrencia de ese evento en la ventana Timeline que se describe más adelante. Si se elige la opción Mote-specific coloring COOJA diferenciará los mensajes de los distintos nodos asignándoles un color, como se muestra en la **figura 5.13.**

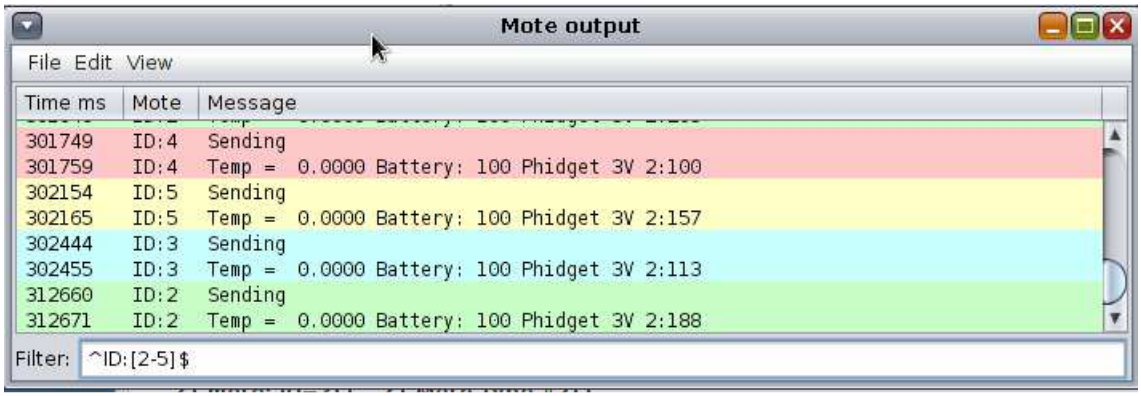



**Figura 5.13.:** Ventana Mote output con la opción de Mote-Specific coloring.

En la figura anterior se ha realizado un filtro para que solo muestre los mensajes de las motas 2, 3, 4, y 5. Existen varios filtros entre los que se encuentran los siguientes:

- **Hello:** Muestra en el log solo los mensajes que contienen el string “Hello”.
- **^Contiki:** Muestra en el log los mensajes que empiezan con el string “Contiki”.
- **^[CR]:** Muestra en el log los mensajes que empiezan con una C o con una R.
- **Hello$:** Muestra en el log los mensajes que acaban con el string “Hello”. *logs ending with 'Hello'*
- **ID:[2-5]$:** Muestra en el log los mensajes de las motas comprendidas entre la 2 y la 5.
- **ID:[2-5] Contiki:** Igual que el anterior y que empiecen con “Contiki”.

En la ventana Radio Messages se dispone de la información vinculada a la interfaz de aire IEEE 802.15.4. como puede verse en la **figura 5.14.** Las columnas muestran el tiempo de simulación en milisegundos, el ID del nodo transmisor y el del receptor y el dato transmitido.

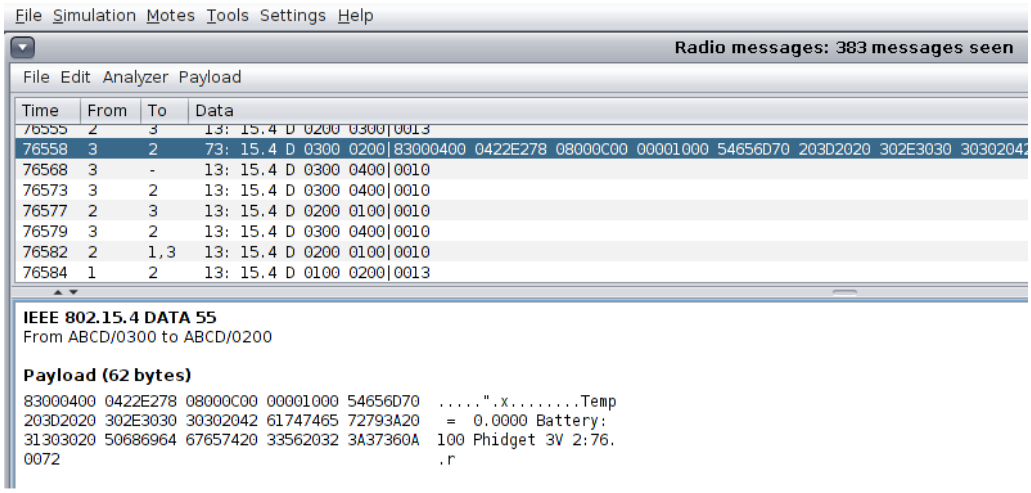



**Figura 5.14.:** Ventana de Radio Messages.

En esta ventana también están disponibles opciones con click-derecho como se puede ver en la **figura 5.15.**

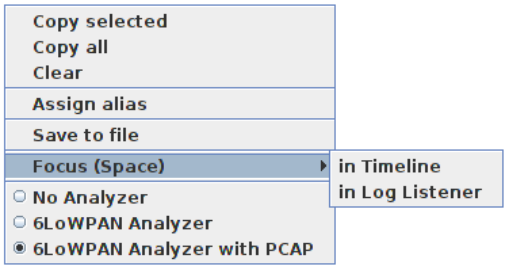



**Figura 5.15.:** Opciones del Radio Messages.

Las tres opciones inferiores permiten elegir si se quiere visualizar algo en el cuadro inferior. La opción No Analyzer deja el cuadro inferior vacío, la opción 6LoWPAN Analyzer muestra detalles de los paquetes correspondiente al protocolo 6LoWPAN. La tercera opción permite que además de mostrar esta información, que se guarde en un archivo con extensión .pcap que pueda abrirse con el analizador de protocolos Wireshark.

La línea de tiempo (Timeline) muestra de manera simultánea la actividad de todos los nodos en relación a la actividad de la radio (on/off, recepción, transmisión,…). La línea de tiempo se puede utilizar para inspeccionar las actividades de los nodos individualmente, así como las interacciones entre los nodos. Para cada mota, se muestra sus eventos de simulación en una línea de color. Los diferentes colores corresponden a diferentes eventos. Para obtener la información sobre el evento en particular, se pasa el ratón por encima de línea. Todas las motas se muestran por defecto en la línea de tiempo, pudiendo eliminar la que no interese. En la **figura 5.16.**, se muestra la ventana de Timeline.

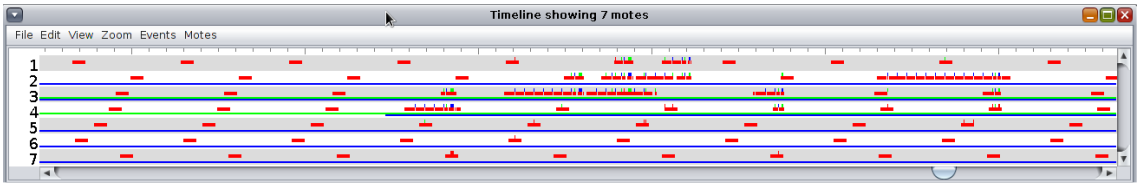



**Figura 5.16.:** Ventana Timeline.

La información relativa a cada nodo aparece en franjas horizontales diferenciadas por colores. En gris oscuro se indica que la radio está encendida. Si marcamos con el ratón en la franja blanca que está por encima de la franja del nodo 1 se obtiene la información del tiempo. Al deslizarlo horizontalmente veremos que el valor varía y el valor que muestra entre paréntesis cambiará indicando el tiempo transcurrido desde el instante correspondiente al instante que se marcó inicialmente y el actual como indica en la **figura 5.17.:**

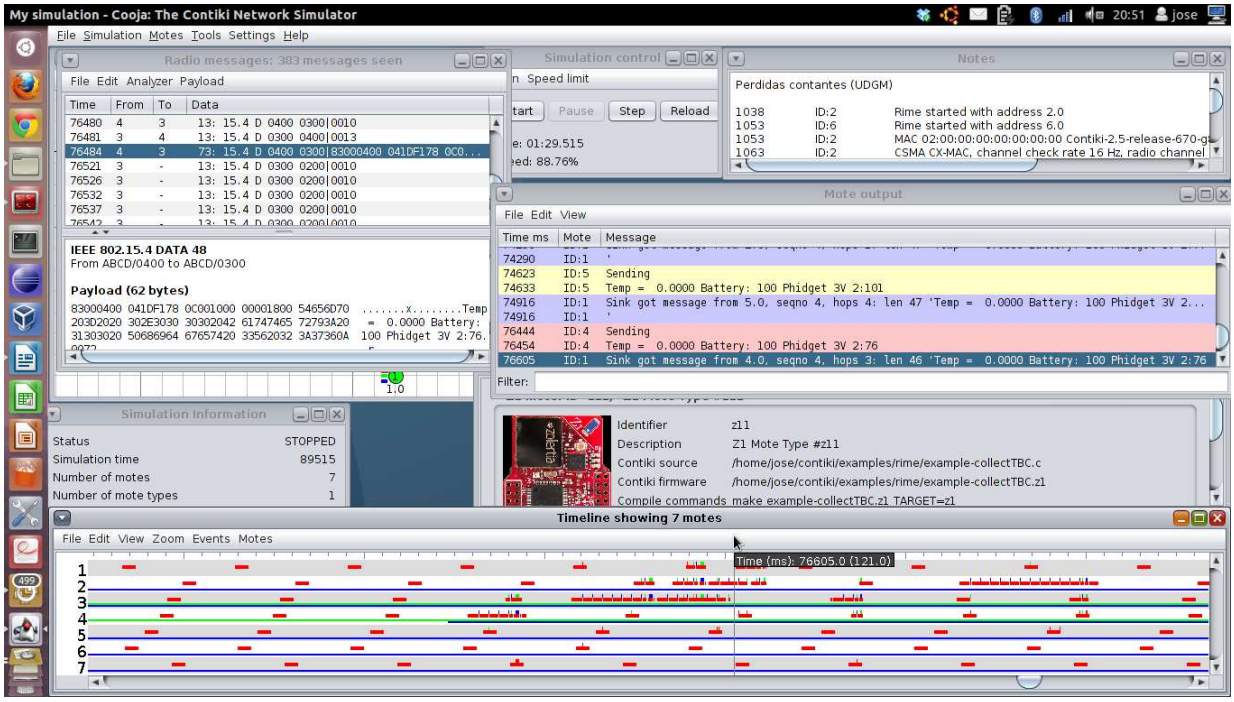



**Figura 5.17.:** Medición de tiempo de transmisión de un paquete desde el nodo 4 hasta el sink (nodo 1).

En la figura anterior, a modo de ejemplo se puede observar el tiempo transcurrido desde el inicio de la transmisión realizada por el nodo 4 en 120 ms. En la ventana Radio Messages se puede ver como el nodo 3 hace transmisiones broadcast y como la mota 4 envía un paquete a su vecino con ID 3. En la ventana Mote output se ve como el paquete es recibido por el nodo con ID 1 que es el sink de la red, indicando que es un paquete recibido de la mota 4, que ha realizado 3 saltos, que la longitud del paquete es de 46 bits y el mensaje indicando el valor de temperatura, el valor del estado de su batería y el valor de consumo que está realizando en ese momento el aire acondicionado.

Toda la información representada en la ventana Timeline se puede guardar en un archivo de texto utilizando la opción Save raw data to file para luego ser procesada independientemente si se quiere.

En la **figura 5.18.**, se muestra una captura de la simulación de la red inalámbrica multisalto, simulando la distribución en la que se han ubicado los sensores en el edificio del CTTC.

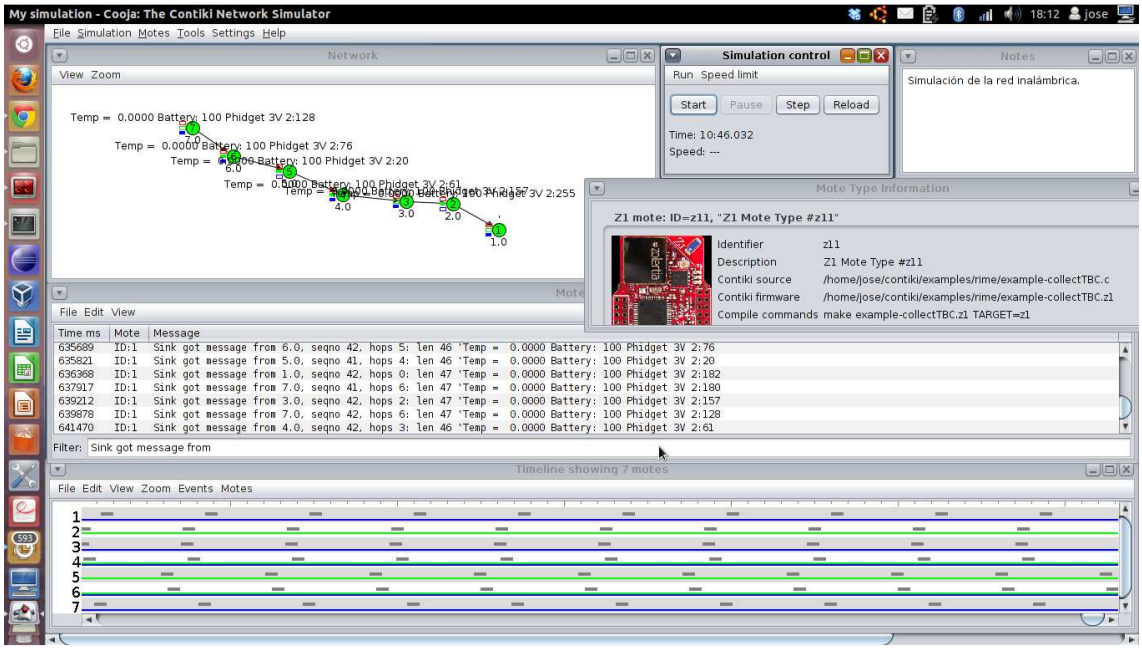



**Figura 5.18.:** Captura de la simulación de la red inalámbrica multisalto del CTTC.

# CAPÍTULO 6. ESCENARIOS Y RESULTADOS

En este capítulo, se describirá brevemente las pruebas y resultados experimentales que se han realizado con en el simulador Cooja, para poder evaluar la red de sensores inalámbrica multisalto. Primero se describirá las pruebas realizadas para poder evaluar las pérdidas ocasionadas en la transmisión y posteriormente la latencia experimentada por los paquetes dentro de la red.

Seguidamente se describirán diferentes escenarios que se han simulado, para poder evaluar la eficiencia de la red de sensores inalámbricos multisalto.

En la **figura 6.1.** se puede observar cómo se ha distribuido las motas simulando la distancia real que existe entre ellas y así poder evaluar la red.

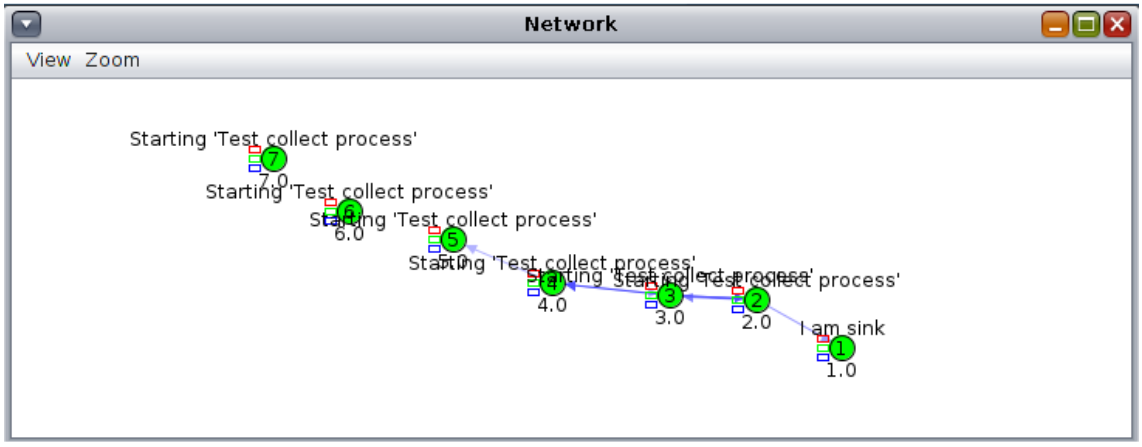



**Figura 6.1.:** Distribución de las motas (escenario 1).

Para poder evaluar las pérdidas ocasionadas en la transmisión, se ha tratado el archivo .txt que previamente se ha guardado, dónde se encuentra la salida de mensajes de las motas, para poder ver los mensajes que nos interesan más fácilmente y así obtener resultados rápidamente.
Para poder hacer la medición, se ha evaluado los 200 primeros paquetes enviados desde cada mota, los cuales tienen su número de secuencia y posteriormente se han filtrado por cada mota los mensajes que han llegado al sink (mota 1) y así obtener el número de paquetes perdidos.
Para poder hacer el filtrado de mensajes por cada mota y obtener el número de paquetes que se han perdido, se ha introducido por línea de comandos lo siguiente:

```
grep "Sink got message from 2.0" prueba16hz.txt > prueba16hzperdidas.txt; sleep 3; wc -l prueba16hzperdidas.txt
200
```

El comando grep busca por cada línea del archivo prueba16hz.txt los mensajes que contienen “Sink got message from 2.0” y los devuelve a otro archivo llamado prueba16hzperdidas.txt, filtrando solo los mensajes que en éste caso serán los mensajes que ha recibido el sink desde la mota 2 y por último se hace un word

count –linea (wc –l) del archivo, que cuenta el número de líneas que tiene y al resultado se le resta los 200 paquetes enviados, obteniendo como resultado el número de paquetes perdidos.

El archivo prueba16hzperdidas.txt al ser tratado con el comando grep quedaría así:

```
8847  ID:1  Sink got message from 2.0, seqno 0, hops 1: len 47 'Temp =
0.0000 Battery: 100 Phidget 3V 2:152
31700 ID:1  Sink got message from 2.0, seqno 1, hops 1: len 46 'Temp =
0.0000 Battery: 100 Phidget 3V 2:56
44573 ID:1  Sink got message from 2.0, seqno 2, hops 1: len 47 'Temp =
0.0000 Battery: 100 Phidget 3V 2:119
66009 ID:1  Sink got message from 2.0, seqno 3, hops 1: len 47 'Temp =
0.0000 Battery: 100 Phidget 3V 2:225
```

Donde cada línea representa que es un paquete recibido por el sink, indicando que es un mensaje enviado desde la mota 2.0, con su nº de secuencia, los saltos realizados hasta alcanzar el sink, la longitud de bits del mensaje y la información que nos interesa, temperatura de la estancia, estado de su batería y el consumo de los splits del aire acondicionado de ese momento.

Los resultados que se han obtenido se muestran en la siguiente tabla:

**Tabla 6.1.:** Resultados escenario 1.

| Nº Saltos | ID Mota | Paquetes enviados | Paquetes recibidos | Promedio de paquetes perdidos | Paquetes perdidos (%) |
|---|---|---|---|---|---|
| 6 | **Mota 2** | 200 | 200-200 | 0 | 0 |
| | **Mota 3** | 200 | 200-200 | 0 | 0 |
| | **Mota 4** | 200 | 176-194 | 20 | 10 |
| | **Mota 5** | 200 | 159-187 | 27 | 13,5 |
| | **Mota 6** | 200 | 141-137 | 61 | 30,5 |
| | **Mota 7** | 200 | 159-63 | 89 | 44,5 |

Como se puede ver en la tabla anterior, en la columna de paquetes recibidos aparecen dos valores, que reflejan los resultados obtenidos de hacer dos simulaciones con el mismo escenario obteniendo el promedio de paquetes perdidos.

En la **figura 6.2.** se puede apreciar que a medida que las motas están más lejos del sink, la información tiene que realizar más saltos, produciendo más pérdidas de paquetes.

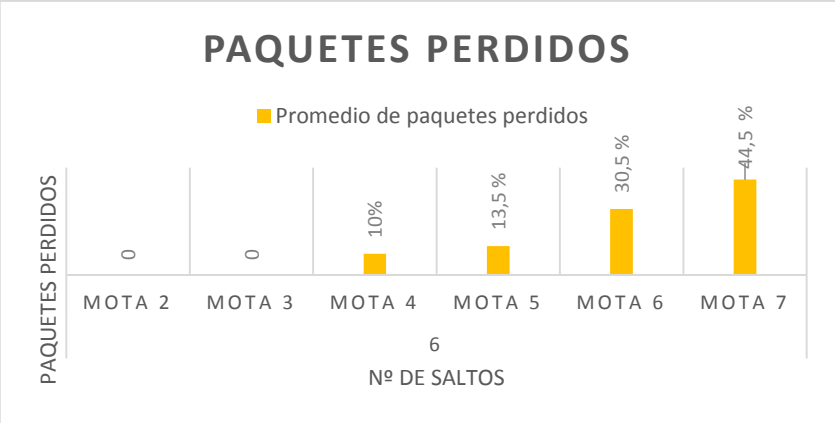



**Figura 6.2.:** Promedio de paquetes perdidos (escenario 1).

Para poder ver cómo se comporta la red inalámbrica según el número de saltos, se han realizado varias simulaciones de la red dando lugar a diferentes escenarios. Inicialmente se ha simulado con el número mínimo de motas, es decir 2, para ver las pérdidas ocasionadas por la mota con ID2, posteriormente se ha sumado otra mota y así, hasta completar el número de saltos que se ha necesitado para cubrir la zona.

La siguiente tabla representa los resultados obtenidos, inicialmente con una red de un salto con dos motas, hasta un total de seis saltos con siete motas.

**Tabla 6.2.:** Resultados escenario 2.

| Nº Saltos | ID Mota | Paquetes enviados | Paquetes recibidos | Paquetes perdidos | Paquetes perdidos (%) |
|---|---|---|---|---|---|
| **1** | **Mota 2** | 200 | 200 | 0 | 0 |
| **2** | **Mota 2** | 200 | 200 | 0 | 0 |
| | **Mota 3** | 200 | 200 | 0 | 0 |
| **3** | **Mota 2** | 200 | 200 | 0 | 0 |
| | **Mota 3** | 200 | 200 | 0 | 0 |
| | **Mota 4** | 200 | 199 | 1 | 0,5 |
| **4** | **Mota 2** | 200 | 200 | 0 | 0 |
| | **Mota 3** | 200 | 200 | 0 | 0 |
| | **Mota 4** | 200 | 200 | 0 | 0 |
| | **Mota 5** | 200 | 197 | 3 | 1,5 |
| **5** | **Mota 2** | 200 | 200 | 0 | 0 |
| | **Mota 3** | 200 | 199 | 1 | 0,5 |
| | **Mota 4** | 200 | 186 | 14 | 7 |
| | **Mota 5** | 200 | 173 | 27 | 13,5 |
| | **Mota 6** | 200 | 181 | 19 | 9,5 |
| **6** | **Mota 2** | 200 | 200 | 0 | 0 |
| | **Mota 3** | 200 | 200 | 0 | 0 |
| | **Mota 4** | 200 | 193 | 7 | 3,5 |
| | **Mota 5** | 200 | 163 | 37 | 18,5 |
| | **Mota 6** | 200 | 168 | 32 | 16 |
| | **Mota 7** | 200 | 159 | 41 | 20.5 |

En la **figura 6.3**. se puede apreciar que en una red inalámbrica multisalto en la que hay pocos saltos, se ocasionan pocas perdidas y a medida que necesitamos abarcar más distancia añadiendo más motas, aumentan las perdidas.

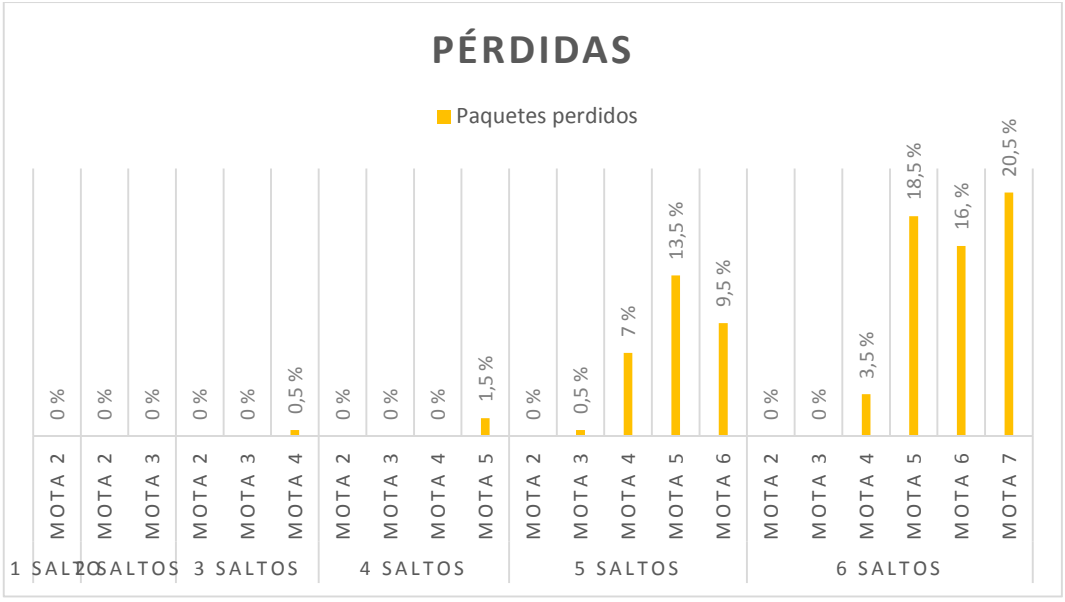



**Figura 6.3.:** Pérdidas escenario 2.

Para poder evaluar la latencia experimentada por los paquetes dentro de la red, se ha decidido escoger la mota en el mejor caso, la mota en el peor caso y una mota intermedia, que serían las motas identificadas con IDs 2, 7 y 5 respectivamente, ya que la mota 2 es la mota más cercana al sink y la mota 7 es la más lejana al sink. Para ello se ha vuelto a tratar los mensajes por cada mota filtrando los mensajes que interesan, que serían los mensajes en los que cada mota emite su información y los mensajes que contienen “Sink got message from 2.0”, para ello se ha introducido por línea de comandos lo siguiente:

```
grep -e "ID:2" -e "Sink got message from 2.0" prueba16hz.txt > prueba16hzlatencia.txt; sleep 4;  grep -e "Temp" -e "Sink got message from 2.0"  prueba16hzlatencia.txt > prueba16hzlatenciaok.txt
```

El comando grep busca por cada línea del archivo prueba16hz.txt, los mensajes que contienen  ID:2 y “Sink got message from 2.0” y los devuelve a otro archivo llamado prueba16hzlatencia.txt, filtrando todos los mensajes relacionados con la mota con ID:2.0 y posteriormente se vuelve a tratar el archivo para poder filtrar solo los mensajes que nos interesan en el archivo prueba16hzlatenciaok.txt, que tendrá los mensajes enviados por la mota 2 y los mensajes recibidos por el sink desde la mota 2 y así poder calcular el instante en el que se recibe el paquete y el instante en el que se ha enviado, restarlos y obtener como resultado el tiempo que transcurrido desde que se ha enviado el paquete hasta que lo ha recibido el sink.

El archivo prueba16hzperdidas.txt al ser tratado quedaría así:

```
8732  ID:2  Temp =  0.0000 Battery: 100 Phidget 3V 2:152
8847  ID:1  Sink got message from 2.0, seqno 0, hops 1: len 47 'Temp =
0.0000 Battery: 100 Phidget 3V 2:152
31576 ID:2  Temp =  0.0000 Battery: 100 Phidget 3V 2:56
31700 ID:1  Sink got message from 2.0, seqno 1, hops 1: len 46 'Temp =
0.0000 Battery: 100 Phidget 3V 2:56
41115 ID:2  Temp =  0.0000 Battery: 100 Phidget 3V 2:119
44573 ID:1  Sink got message from 2.0, seqno 2, hops 1: len 47 'Temp =
0.0000 Battery: 100 Phidget 3V 2:119
```

Donde se ve más fácilmente los mensajes que nos interesan, que en este caso son los mensajes relacionados con la mota 2 con el instante en el que se ocasionaron. A modo de ejemplo podemos ver que en la primera línea la mota 2 emite su información en el instante 8732 ms y dicho paquete se ha recibido por el sink en el instante 8847 ms, obteniendo como resultado que el paquete ha tardado en llegar al destino 115 ms.

Similar a las pruebas realizadas anteriormente, para evaluar la latencia de los paquetes dentro de la red, inicialmente se ha simulado con el número mínimo de motas, es decir 2, para ver el tiempo que transcurre desde que el nodo envía el paquete hasta que se recibe por el sink, posteriormente se ha añadido otra mota al escenario, y así, hasta completar el número de saltos que se ha necesitado para cubrir la zona.

La siguiente tabla representa los resultados obtenidos, inicialmente con una red de un salto con dos motas, hasta finalizar con un total de seis saltos con siete motas.

**Tabla 6.3.:** Resultados escenario 3.

| **Red de 1 salto** | | | | |
|---|---|---|---|---|
| ID Mota | Nº Sec | Instante de paquete enviado (ms) | Instante de paquete recibido por el sink (ms) | Latencia (ms) |
| Mota 2 | 10 | 164334 | 164418 | 84 |
| | 100 | 1529787 | 1529860 | 73 |
| | 199 | 3002365 | 3002447 | 82 |
| **Red de 2 saltos** | | | | |
| ID Mota | Nº Sec | Instante de paquete enviado (ms) | Instante de paquete recibido por el sink (ms) | Latencia (ms) |
| Mota 2 | 10 | 164248 | 164329 | 81 |
| | 100 | 1529622 | 1529713 | 91 |
| | 199 | 3002412 | 3002502 | 90 |
| Mota 3 | 10 | 182114 | 182250 | 136 |
| | 100 | 1522615 | 1522759 | 144 |
| | 199 | 3003099 | 3003259 | 160 |

| Red de 3 saltos | | | | |
|---|---|---|---|---|
| ID Mota | Nº Sec | Instante de paquete enviado (ms) | Instante de paquete recibido por el sink (ms) | Latencia (ms) |
| Mota 2 | 10 | 155599 | 157143 | 1544 |
| | 100 | 1515381 | 1515497 | 116 |
| | 199 | 3002427 | 3002525 | 98 |
| Mota 3 | 10 | 155412 | 155573 | 161 |
| | 100 | 1529161 | 1529282 | 121 |
| | 199 | 3003099 | 3003543 | 444 |
| Mota 4 | 10 | 154829 | 155102 | 273 |
| | 100 | 1518134 | 1518384 | 250 |
| | 199 | 3002477 | 3004063 | 1586 |
| **Red de 4 saltos** | | | | |
| ID Mota | Nº Sec | Instante de paquete enviado (ms) | Instante de paquete recibido por el sink (ms) | Latencia (ms) |
| Mota 2 | 10 | 164467 | 164535 | 68 |
| | 100 | 1509053 | 1509138 | 85 |
| | 199 | 3002357 | 3002392 | 35 |
| Mota 3 | 10 | 177422 | 179508 | 2086 |
| | 100 | 1504977 | 1505933 | 956 |
| | 199 | 3003357 | 3003526 | 169 |
| Mota 4 | 10 | 165095 | 165297 | 202 |
| | 100 | 1515166 | 1515394 | 228 |
| | 199 | 3002564 | 3002779 | 215 |
| Mota 5 | 10 | 177220 | 177474 | 254 |
| | 100 | 1526617 | 1528395 | 1778 |
| | 199 | 3002906 | 3003212 | 306 |
| **Red de 5 saltos** | | | | |
| ID Mota | Nº Sec | Instante de paquete enviado (ms) | Instante de paquete recibido por el sink (ms) | Latencia (ms) |
| Mota 2 | 10 | 154733 | 156681 | 1948 |
| | 100 | 1526108 | 1526184 | 76 |
| | 199 | 3002350 | 3002458 | 108 |
| Mota 3 | 10 | 155826 | 156823 | 99 |
| | 100 | 1525778 | 1525956 | 178 |
| | 199 | 3003638 | 3003844 | 206 |
| Mota 4 | 10 | 160516 | 160744 | 228 |
| | 100 | 1506946 | 1507110 | 164 |
| | 199 | 3002994 | 3003346 | 352 |
| Mota 5 | 10 | 157032 | 159216 | 2184 |
| | 100 | 1516665 | 1516962 | 297 |
| | 199 | 3002954 | 3005157 | 2203 |
| Mota 6 | 10 | 174286 | 174620 | 334 |
| | 100 | 1502895 | 1506427 | 3532 |
| | 199 | 3002442 | 3004259 | 1817 |

| Red de 6 saltos | | | | |
|---|---|---|---|---|
| ID Mota | Nº Sec | Instante de paquete enviado (ms) | Instante de paquete recibido por el sink (ms) | Latencia (ms) |
| Mota 2 | 10 | 162053 | 162107 | 54 |
| | 100 | 1525693 | 1525915 | 222 |
| | 199 | 3002357 | 3002473 | 116 |
| Mota 3 | 10 | 165200 | 165369 | 169 |
| | 100 | 1514341 | 1514408 | 67 |
| | 199 | 3003443 | 3003658 | 215 |
| Mota 4 | 10 | 160321 | 160579 | 258 |
| | 100 | 1512563 | 1513285 | 722 |
| | 199 | 3002907 | 3003458 | 551 |
| Mota 5 | 10 | 177024 | 177341 | 317 |
| | 100 | 1516188 | 1517876 | 1688 |
| | 199 | 3003812 | 3005322 | 1510 |
| Mota 6 | 10 | 153919 | 159157 | 5238 |
| | 100 | 1525504 | 1527604 | 2100 |
| | 199 | 3002489 | 3002861 | 372 |
| Mota 7 | 10 | 165289 | 165715 | 426 |
| | 100 | 1514578 | 1515097 | 519 |
| | 199 | 3002835 | 3005549 | 2714 |

La siguiente tabla representa la latencia promedio obtenida de los escenarios anteriores.

**Tabla 6.4.:** Latencia promedio

| Nº Saltos | ID Mota | Latencia min. (ms) | Latencia máx. (ms) | Latencia promedio (ms) |
|---|---|---|---|---|
| **1** | **Mota 2** | 73 | 84 | 79,6 |
| **2** | **Mota 2** | 81 | 91 | 87,3 |
| | **Mota 3** | 136 | 160 | 146,6 |
| **3** | **Mota 2** | 98 | 1544 | 586 |
| | **Mota 3** | 121 | 444 | 726 |
| | **Mota 4** | 250 | 1586 | 703 |
| **4** | **Mota 2** | 35 | 85 | 62,6 |
| | **Mota 3** | 169 | 2086 | 1070,3 |
| | **Mota 4** | 202 | 228 | 215 |
| | **Mota 5** | 254 | 1778 | 779,3 |
| **5** | **Mota 2** | 76 | 1948 | 710,6 |
| | **Mota 3** | 99 | 206 | 161 |
| | **Mota 4** | 164 | 352 | 248 |
| | **Mota 5** | 297 | 2203 | 1561,3 |
| | **Mota 6** | 334 | 3532 | 1894,3 |
| **6** | **Mota 2** | 54 | 222 | 130,6 |
| | **Mota 3** | 67 | 215 | 150,3 |
| | **Mota 4** | 258 | 722 | 510,3 |
| | **Mota 5** | 317 | 1688 | 1171,6 |
| | **Mota 6** | 372 | 5238 | 2570 |
| | **Mota 7** | 426 | 2714 | 1219,6 |

En la **figura 6.4.** se representa la latencia máxima, mínima y promedio.

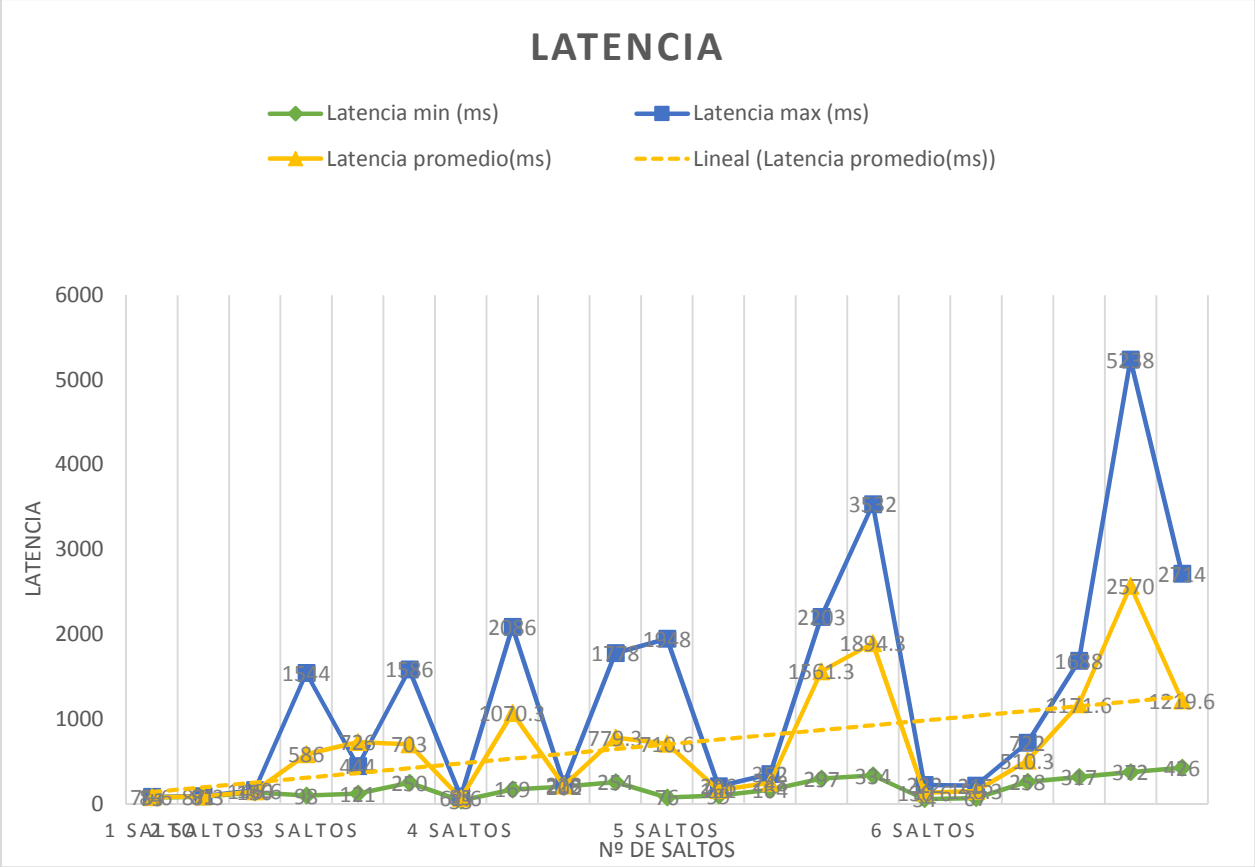



**Figura 6.4.:** Latencia máxima, mínima y promedio

En la gráfica se puede ver como la latencia aumenta en relación al número de saltos realizados.

El escenario siguiente es el escenario real que se ha implantado en el edificio del CTTC, dónde se puede apreciar que los valores obtenidos se asimilan mucho a las simulaciones anteriores, en la que la latencia aumenta en relación al número de saltos que se ha de realizar hasta llegar al sink.

**Tabla 6.5.:** Resultados red CTTC.

| Red de 6 saltos (red CTTC) | | | | |
|---|---|---|---|---|
| ID Mota | Nº Sec | Instante de paquete enviado (ms) | Instante de paquete recibido por el sink (ms) | Latencia (ms) |
| Mota 2 | 10 | 176186 | 176278 | 92 |
| | 100 | 1510857 | 1511093 | 236 |
| | 199 | 3002482 | 3002531 | **49** |
| Mota 3 | 10 | 164028 | 164091 | 63 |
| | 100 | 1531521 | 1531714 | 193 |
| | 199 | 3003497 | 3003607 | 110 |
| Mota 4 | 10 | 172407 | 172587 | 180 |
| | 100 | 1530555 | 1534289 | 3734 |
| | 199 | 3002938 | 3006972 | 4034 |
| Mota 5 | 10 | 181946 | 182164 | 218 |
| | 100 | 1519494 | 1523544 | 4050 |
| | 199 | 3143167 | 3148340 | 5173 |
| Mota 6 | 10 | 152966 | 162090 | 9124 |
| | 100 | 1530091 | 1538081 | 7990 |
| | 199 | 3002435 | 3002731 | 296 |
| Mota 7 | 10 | 189981 | 190534 | 553 |
| | 100 | 1502124 | 1511292 | **9168** |
| | 199 | 2998815 | 3007568 | 8753 |

| Nº Saltos | ID Mota | Latencia mín. (ms) | Latencia máx. (ms) | Latencia promedio (ms) |
|---|---|---|---|---|
| 6 | **Mota 2** | **49** | 236 | 125,6 |
| | **Mota 3** | 63 | 193 | 122 |
| | **Mota 4** | 180 | 4034 | 2649,3 |
| | **Mota 5** | 218 | 5173 | 3147 |
| | **Mota 6** | 296 | 9124 | 5803,3 |
| | **Mota 7** | 553 | **9168** | 6158 |

En la tabla anterior se puede ver como la latencia mínima (mejor caso) es de 49 ms y la latencia máxima (peor caso) es de 9168 ms, es decir, 9,2 segundos. También se puede observar que para poder enviar 200 paquetes se ha tardado 3007568 ms, que equivale a 0,066498 paq/s.

En la **figura 6.5.** se representa gráficamente la latencia promedio, en relación al número de saltos realizados.

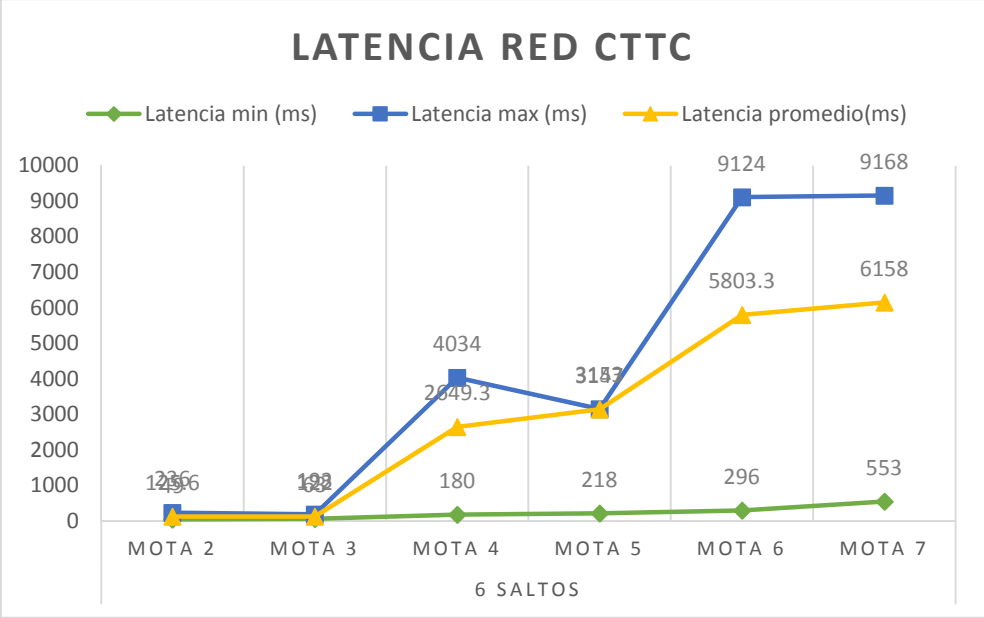



**Figura 6.4.:** Latencia máxima, mínima y promedio de la red CTTC

Con los resultados obtenidos anteriormente se observa que tanto en la simulación como en la implementación de una red de sensores inalámbrica multisalto, en la que hay pocos saltos, se ocasionan menos pérdidas y la latencia es baja. A medida que necesitamos abarcar más distancia añadiendo más motas aumentan las pérdidas y la latencia. Estas pérdidas y este aumento de latencia pueden ser ocasionadas por la calidad del enlace inalámbrico, fallos en el sensor y/o por congestión de la red. Se ha llegado a la conclusión de que podrían ser ocasionadas por la congestión de la red, ya que a medida que existen más nodos sensores en la red los paquetes que están en los buffers de las primeras motas permanecen más tiempo, llegando a ocupar todo el espacio disponible produciendo pérdidas de paquetes y aumentando la latencia de entrega de paquetes. Otra causa podría ser que la tasa de llegada de paquetes excede la tasa de servicio de un nodo, ya que generalmente esto suele ocurrir en los nodos que se encuentran más cerca del nodo “sink”, ya que estos nodos reciben un mayor tráfico.

# CAPÍTULO 7. CONCLUSIONES Y LÍNEAS FUTURAS

En este capítulo se presenta las conclusiones que se han obtenido del proyecto y las posibles líneas futuras.

El objetivo principal de este proyecto se ha centrado en implementar una red inalámbrica de sensores multisalto en el edificio B4 del PMT (Parque Mediterráneo de la Tecnología) de Castelldefels, dónde reside el centro de investigación CTTC (Centro Tecnológico de Telecomunicaciones de Catalunya). Esta red inalámbrica de sensores multisalto tiene como objetivo la medición en tiempo real del consumo eléctrico, producido por el aire acondicionado del edificio. Para poder alcanzar dicho objetivo, en primer lugar se ha estudiado las características principales de una red inalámbrica de sensores, su ámbito de aplicación y los elementos que la componen. Posteriormente se ha descrito brevemente las características principales de los sensores que existen en la actualidad y sus sistemas operativos, dando especial importancia a las motas Z1 de Zolertia, con su sistema operativo ContikiOS, que aún no ha sido desarrollado, como si lo ha sido TinyOS. Es por ello que en este proyecto se ha centrado en desarrollar posibles soluciones, para poder llevar la información de extremo a extremo del edificio, con una red inalámbrica de sensores multisalto con motas Z1 y así poder estudiar el sistema operativo Contiki.

Para poder realizar la comunicación extremo a extremo del edificio se ha tenido que estudiar y evaluar los diferentes protocolos de comunicaciones que existen para las redes inalámbricas de sensores, desde la capa física, hasta llegar a la capa de red.

Una vez asimilados los conceptos anteriores, se ha descrito el material utilizado para llevar a cabo la solución y los protocolos utilizados de ContikiOS necesarios para poder realizar la comunicación, ya que Contiki nos brinda la posibilidad de poder utilizar una gran variedad de protocolos. La comprensión del funcionamiento de este sistema operativo y la programación de la aplicación necesaria para poder realizar la medición de la temperatura ambiental de los establecimientos, el estado de la batería de la mota, y el consumo energético que está realizando en ése momento, por el uso de los splits del aire acondicionado, ha sido una faceta muy importante ya que se ha tenido que investigar cómo se estructura y como se compilan las aplicaciones en ContikiOS.

Por último, se ha presentado y analizado los resultados de las medidas que se ha obtenido en las diferentes pruebas y simulaciones de la solución desarrollada para el escenario. De esta manera se ha podido comprobar que a medida que la información tiene que realizar más saltos para poder llegar a su destino, se ocasionan más perdidas de paquetes, ocasionando retransmisiones y produciendo un retardo cada vez más elevado. Estos retardos y tasas de pérdidas de paquetes cumplen con los estándares de calidad esperados para el sistema. Por lo tanto, se recomienda el uso de estas redes para la medida de consumo eléctrico.

Como líneas futuras se podría investigar cómo reaccionaría una red inalámbrica de sensores similar a la solución descrita en el proyecto, en la que se tenga que utilizar una gran cantidad de sensores para poder abarcar una distancia mucho más grande, utilizando otro método de enrutamiento como es el modelo esquemático basado en clústeres, donde los protocolos usan técnicas de optimización para mejorar la eficacia del modelo multisalto. La red se divide en capas de clústeres y estos clústeres están compuestos de nodos que enrutan los datos hacia un recopilador o sink, que los envía a una capa superior, hasta llegar a la estación base. Los datos saltan de un clúster a otro por lo que cubre mayores distancias. Esto hace que demás los datos se transfieran más rápido a la estación base, ocasionando una latencia mucho menor que en las redes multisalto y por tanto será mejor para redes con gran cantidad de nodos en un espacio amplio (del orden de miles de sensores y cientos de metros de distancia).

Otra de las posibles vías de investigación sería optar por el protocolo de encaminamiento RPL descrito en el anexo III el cual soporta tráfico con mensajes bidireccionales, haciendo posible enviar mensajes desde los nodos hacia el sink y desde el sink hacia los nodos, y poder ver cómo se comporta la red inalámbrica multisalto y poder evaluar las pérdidas de paquetes y la latencia experimentada por los paquetes dentro de la red.

# Bibliografía

**Motas:**

**Mica2**, disponible on-line en: http://www.capsil.org/capsilwiki/index.php/MICA2
**Intel Mote**, disponible on-line en: http://docs.tinyos.net/tinywii/index.php/Imote2
**TelosB**, disponible on-line en: http://telosbsensors.wordpress.com/
**Libelium Company**, disponible on-line en: http://www.libelium.com/
**Zolertia**, disponible on-line en: http://www.zolertia.com/ti
**Crossbow**, disponible on-line en: www.crossbow.com

**Sistemas operativos:**

**TinyOS,** disponible online en: http://www.tinyos.net/
**MantisOS,** disponible on-line en: http://mantisos.org/index/tiki-index.php.html
**eCos,** disponible on-line en: http://ecos.sourceware.org/
**ContikiOS,** disponible on-line en: www.sics.se/contiki/
**NanoRK**, disponible on-line en: www.nanork.org
**Texas Instruments**, disponible on-line en: www.ti.com

**ContikiOS:**

**Compilar una aplicación en Contiki:**
http://zolertia.sourceforge.net/wiki/index.php/Mainpage:Contiki_Lesson_0
**Cambiar MAC y ID a la mota:**
http://zolertia.sourceforge.net/wiki/index.php/Mainpage:Contiki_apps#Change_the_default_MAC_address_and_Node_ID.2C_and_burn_it_to_flash
**Configuración del archive project-conf.h:**
https://github.com/contiki-os/contiki/wiki/Change-mac-or-radio-duty-cycling-protocols

**Cooja:**

http://www.george-smart.co.uk/wiki/Contiki_Cooja_Simulator
https://github.com/contiki-os/contiki/wiki/An-Introduction-to-Cooja

**Papers:**

**Collection Tree Protocol:**
http://soda.swedish-ict.se/5307/1/hassanzadeh12efficient.pdf
**Efficient Mobile Data Collection with Mobile Collect:**
http://soda.swedish-ict.se/5307/1/hassanzadeh12efficient.pdf
**The Evolution of MAC Protocols in Wireless Sensor Networks: A Survey**

http://ieeexplore.ieee.org/xpl/articleDetails.jsp?reload=true&tp=&arnumber=6188353&contentType=Early+Access+Articles&punumber%3D9739
**The ContikiMAC Radio Duty Cycling Protocol:**
http://dunkels.com/adam/dunkels11contikimac.pdf
**Performance Analysis of IEEE 802.15.4 Beacon-Enabled Mode:**
http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=5378538&url=http%3A%2F%2Fieeexplore.ieee.org%2Fxpls%2Fabs_all.jsp%3Farnumber%3D5378538
**MAC Essentials for Wireless Sensor Networks:**
http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=5451759&url=http%3A%2F%2Fieeexplore.ieee.org%2Fxpls%2Fabs_all.jsp%3Farnumber%3D5451759
**MAC Protocols for Wireless Sensor Networks: a Survey:**
http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=1632658&url=http%3A%2F%2Fieeexplore.ieee.org%2Fxpls%2Fabs_all.jsp%3Farnumber%3D1632658
**X-MAC: A Short Preamble MAC Protocol For Duty-Cycled Wireless Sensor Networks:**
http://www.cse.wustl.edu/~lu/cse521s/Papers/x-mac.pdf
**Routing Protocol for Low-Power and Lossy Wireless Sensor Networks:**
http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=6143534&url=http%3A%2F%2Fieeexplore.ieee.org%2Fxpls%2Fabs_all.jsp%3Farnumber%3D6143534
**Comparative performance study of RPL in Wireless Sensor Networks:**
http://ieeexplore.ieee.org/xpl/abstractAuthors.jsp?arnumber=6399404

# Anexo I. Instalación de Contiki en Ubuntu

Este método de instalación debe funcionar con cualquier distribución de Ubuntu. Primero se ha de instalar un conjunto de herramientas necesarias para comenzar a compilar el entorno de la mota Z1, básicamente se necesitan las siguientes librerías:

- GCC (la última versión es la 4.4.4).
- Binutils (la última versión es la 2.20.1)
- Libc support.
- Python-serial support (BSL).

Para comprobar la última versión de las librerías mencionadas anteriormente, ejecutando en un terminal:

```
dpkg  -l  '*' gcc  'binutils *'  |  grep ii
```

Zolertia ofrece en su wiki un tarball precompilado del msp430-gcc-4.4.5 (legacy), una vez se tenemos el paquete, se descomprime en la ubicación /opt y luego en un terminal:

```
echo  "PATH = / opt/msp430-gcc-4.4.5/bin: $ PATH"  >> ~ / bashrc.
```

Para comprobar la versión:

```
MSP430-gcc - version
```

Por último, para descargar la última fuente de Contiki, primero se instala el git, luego se clona el repositorio.

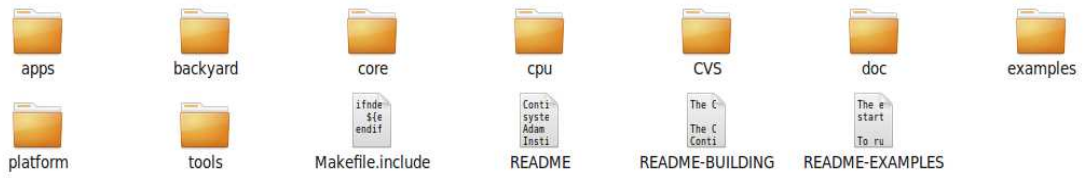
```
sudo apt-get install git-core
git clone git://contiki.git.sourceforge.net/gitroot/contiki/contiki
```

Después dentro del directorio  /contiki, se puede mantener actualizado:

```
git fetch origin master
git pull origin master
```

# Anexo II. Estructura de carpetas en Contiki

La siguiente figura muestra la estructura de carpetas de Contiki.




Estructura de carpetas de Contiki

- **apps**: En esta carpeta se encuentra las aplicaciones, son programas auxiliares que se pueden usar en el programa principal. Hay que tener cuidado de que no se puede usar cualquier aplicación con cualquier protocolo. Por ejemplo, hay muchas aplicaciones que están diseñadas para el stack Rime y no funcionan para uIP.
- **core**: En esta carpeta se encuentra el core del sistema operativo. Se recomienda no modificarlo. En esta carpeta se pueden encontrar la implementación de los protothreads, los etimers, uIP, Rime, RPL y todo lo que refiere al sistema operativo.
- **cpu**: En esta carpeta se encuentra todo el código que depende del microcontrolador. Algunos ejemplos pueden ser la configuración de los timer, el módulo serial, la memoria flash, las instrucciones para que el microcontrolador entre y salga del low-power mode. En nuestro caso se usa el msp430 F1611, por lo que se utiliza la carpeta msp430. Se puede cambiar fácilmente de microcontrolador eligiendo que se compile el código de una carpeta diferente.
- **doc**: Aquí se encuentran todo lo que refiere a documentación del código.
- **examples**: En esta carpeta se encuentran los programas de ejemplo que utilizan el sistema operativo.
- **platform**: En esta carpeta se encuentra todo el código que depende de la plataforma, como ser la configuración de los botones y los sensores. Además en esta carpeta se encuentra la rutina main.
- **tools**: En esta carpeta se encuentra varias herramientas extras que no forman parte del sistema operativo Contiki, para la depuración, la simulación y la mejora de las aplicaciones. Por ejemplo, esta carpeta tiene software complementario como el collect-view y el simulador COOJA, los cuales se compilan por separado al sistema operativo Contiki.

# Anexo III. Protocolos de redes inalámbricas de sensores

A lo largo de esta sección se hablará sobre la tecnología radio que utilizan las redes inalámbricas de sensores, el estándar IEEE802.15.4, la capa física, la capa de acceso al medio, algoritmos de duty cycling y protocolos de enrutamiento.

## Estándar de comunicaciones 802.15.4

El estándar IEEE 802.15.4-2006 define la capa física y la capa de control de acceso al medio (MAC) para redes inalámbricas de bajo consumo, como las redes de sensores. Esta especificación se desarrolló para dar respuesta a la demanda creciente de un protocolo más sencillo que no consumiera tantos recursos, ya que los estándares presentes hasta la fecha están orientados a aplicaciones que utilizan mucha energía y un elevado ancho de banda. Los protocolos más destacados en este sentido son el IEEE 802.11, comúnmente conocido como WiFi, y el IEEE 802.15.1 o Bluetooth.

En la tabla siguiente se muestra una comparativa entre los tres estándares, donde se reflejan cuantitativamente las diferencias existentes entre ellos.

Comparativa entre estándares inalámbricos.

| Estándar | Ancho de Banda | Consumo | Cobertura | Ventajas | Aplicaciones |
|---|---|---|---|---|---|
| **Wi-fi** | 54 Mbps | 400 mA en Tx 20 mA en reposo | 100 m | Gran ancho de banda | Internet, redes de ordenadores |
| **Bluetooth** | 1 Mbps | 40 mA en Tx 0,2 mA en reposo | 10 m | Interopera-bilidad, sustituto de cable | Wireless usb, móviles |
| **IEEE 802.15.4** | 250 Kbps | 1.8 mA en Tx 5.1µA en reposo | 100 m | Batería de larga duración, bajo coste | Control remoto, productos con batería, sensores |

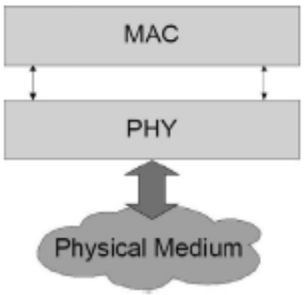
Al comparar sus parámetros, se ve como el estándar 802.15.4 se ha desarrollado para proveer las capas física y MAC de las redes LR-WPAN (Low-Rate Wireless Personal Area Network).

Como consecuencia de su bajo consumo tenemos un ancho de banda y una potencia de transmisión más reducidas. Por tanto esta tecnología inalámbrica sólo permite comunicaciones de corto alcance, típicamente con distancias de hasta 100 metros.

Este ancho de banda depende también de la frecuencia a la que trabaja el estándar. El 802.15.4 está diseñado para funcionar en la banda de 2,4 GHz a una tasa de 250 kbps, en la de 868 MHz a 40 kbps y en la de 915 MHz a 20 kbps. A pesar de tener estas tres alternativas, se suele utilizar la banda de 2,4 GHz, ya que es común en todo el mundo, mientras que las otras dos frecuencias sólo están disponibles en Europa y Estados Unidos respectivamente.
Gracias a su bajo consumo, un dispositivo que funcione con pilas debería poder mantener su autonomía durante periodos superiores a 6 meses, dependiendo de la aplicación. Por eso, para aplicaciones de monitorización y control, este estándar es más adecuado que otras tecnologías inalámbricas.

## Capa física

En la capa física del 802.15.4 se encuentra el transceptor de radiofrecuencia (RF). Por lo tanto es la capa que se encuentra más próxima al medio físico de transmisión y es la que se encarga de enviar y recibir los mensajes a través del medio radio. Como vemos en la figura siguiente, la capa física provee a la capa MAC de los servicios necesarios para una transmisión correcta.



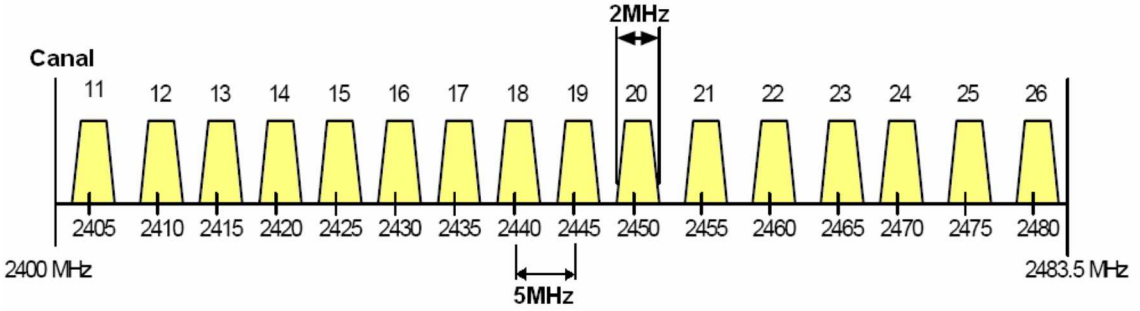

Arquitectura de protocolos de IEEE 802.15.4.

Las funciones principales que implementa la capa física de IEEE 802.15.4 son las de control del transceptor de radiofrecuencia, medición de la calidad del enlace (LQI), selección del canal y detección de portadora (CCA) para su uso en CSMA/CA a nivel MAC.
En la siguiente tabla, se puede ver cómo la capa física utiliza distintas configuraciones según la frecuencia en la que se trabaja. Hay que tener en cuenta que el alcance de la transmisión también varía según la frecuencia, puesto que la señal presenta más atenuación trabajando a mayores frecuencias.

Configuraciones en función de la banda utilizada.

| Física (MHz) | Frecuencia (MHz) | Tasa (kchip/s) | Modulación | Tasa de transmisión | |
|---|---|---|---|---|---|
| | | | | Kbps | Ksimbolos/s |
| **868/915** | 868-868 | 300 | BPSK | 20 | 20 |
| | 902-928 | 600 | BPSK | 40 | 40 |
| **2450** | 2400-2483,5 | 2000 | O-QPSK | 250 | 62,5 |

La banda de 2.4 GHz es la que dispone de más ancho de banda, está dividida en 16 canales de 2 MHz cada uno y una separación entre canales de 5 MHz, para evitar así la interferencia co-canal, como puede apreciarse en la **figura 1.22**.




División espectral de canales en la banda de 2.4Ghz.

## Capa de control de acceso al medio

El medio inalámbrico se emite por el aire y por lo tanto es propenso a las interferencias, por ello requiere protocolos de control de acceso al medio altamente optimizados. Particularmente en las redes de sensores en la que consiste en una gran cantidad de sensores alimentados con baterías para poder operar durante años sin intervención humana. Por ello ha crecido el interés en entender y optimizar los protocolos MAC en WSN, donde los escasos recursos y limitaciones han impulsado a reducir el consumo de energía de funcionalidades MAC.
La elección de un protocolo MAC se realiza según las estadísticas del tráfico generado y según los problemas que van surgiendo.
A día de hoy, la problemática más importante en el despliegue de una red de sensores inalámbrica es la dependencia de baterías limitadas. El principal objetivo es extender la vida útil de la red sin que peligren las comunicaciones fiables y eficientes desde el nodo sensor hasta los otros nodos como es el recolector de datos o sink.
La función principal de MAC es coordinar el acceso y la transmisión sobre un medio común a varias estaciones. En el contexto inalámbrico se complica por el hecho de que el medio común es el canal inalámbrico que se emite en la naturaleza, es decir, cualquier transmisión en curso interfiere con cualquier otra transmisión que haya en el mismo rango de frecuencias. La interferencia puede dar lugar a pérdidas de paquetes que necesitan ser atendidos con mecanismos adecuados de retransmisión.
MAC se puso en marcha para minimizar las interferencias y colisiones de paquetes. Tradicionalmente esto se logra mediante la optimización del acceso al canal, paquetes de transmisión, métodos de retransmisión, longitudes de paquetes, modulación y esquemas de codificación.

### Propiedades de la capa MAC relacionadas con una red de sensores

Un objetivo común de investigación es maximizar la vida útil de los sensores, para ello reducimos los derroches de la energía potencial. Los tipos de patrones de comunicaciones que se utilizan en aplicaciones de redes de sensores deben

ser estudiados, ya que nos pueden servir para extraer el comportamiento del tráfico de la red que tiene ser controlado por un determinado protocolo MAC.

Cuando un nodo receptor recibe más de un paquete al mismo tiempo, estos paquetes se dicen que colisionaron, incluso cuando coinciden parcialmente. Todos los paquetes que causan la colisión tienen que ser desechados ya que su retransmisión aumentaría el consumo de energía.
La segunda razón de la perdida de energía es cuando un nodo escucha el canal y los paquetes van destinados a otro nodo.
La tercera razón del gasto energético se produce como resultado de la sobrecarga de paquetes de control. Se debe utilizar un número mínimo de paquetes de control para hacer la transmisión de datos.
Una de las principales fuentes de derroche de energía es la escucha de inactividad, es decir, escuchar el canal libre para poder recibir el posible tráfico.
La última razón para la pérdida de energía se le llama overemitting, que es causado por la transmisión de un mensaje cuando el nodo de destino no está preparado.
Teniendo en cuenta los hechos anteriores, un buen diseño del protocolo MAC debe evitar esos derroches de energía.

Existen tres tipos de patrones de comunicación en las redes inalámbricas de sensores: Broadcast (Difusión), Convergecast y Local gossip (Charla local). El tipo broadcast lo utiliza generalmente una estación base llamado Sink (sumidero) para transmitir cierta información a todos los sensores de la red. El tipo de emisión de patrón de comunicaciones no debe confundirse con el tipo de paquete de broadcast. En el primer caso, todos los nodos de la red son el destino (receptores), mientras que para éste último, los receptores son los nodos del mismo rango del nodo que transmite. En algunos casos, los sensores detectan a un intruso comunicándose de forma local. Este tipo de modelo de comunicación se llama charla local, dónde un sensor envía un mensaje a sus nodos vecinos dentro de un rango. Los sensores que detectan al intruso tienen que enviar lo que perciben al centro de información. El patrón de comunicaciones convergecast es cuando un grupo de sensores se comunican con un sensor específico. Por último existe un cuarto tipo de patrón que se llama multicast, dónde un sensor envía un mensaje a un subconjunto específico de sensores.

## Protocolos de la capa MAC para redes de sensores.

Existe una amplia gama de protocolos MAC para redes de sensores, los cuales se describen brevemente, indicando su comportamiento principal, mencionando sus ventajas y desventajas que presentan y una comparativa. Al final se describe el protocolo X-MAC como mecanismo de Radio Duty Cycling, protocolo en el que se basa CX-MAC que se ha implementado en nuestro diseño.

### Sensor-MAC (S-MAC)

S-MAC, Sensor Media Access Control, es un protocolo basado en contención, añade a la capa MAC gestión del consumo, enlace a nivel de retransmisión, eliminación de paquetes duplicados, no presenta el problema del nodo oculto usando RTS/CTS y estimación de calidad del enlace. S-MAC duerme periódicamente, despierta y escucha el canal, y entonces vuelve a dormir. Está

diseñado para operar como “caja negra” optimizado para una carga de trabajo representativa.

Ventajas: El derroche de energía causado por la escucha de inactividad, se puede disminuir reduciendo los horarios de sueño. Además de sencillez de implementación, los gastos generales de tiempo de sincronización se pueden prevenir con los anuncios de horarios de sueño.
Desventajas: Los paquetes broadcast no utilizan RTS/CTS, aumentando la probabilidad de colisión. El sueño y los periodos de escucha están predefinidos y son constantes, lo que disminuye la eficiencia del algoritmo por la carga de tráfico variable.

**WiseMAC**
WiseMAC (Wireless Sensor MAC) se basa en una técnica de muestreo, en busca del preámbulo generado por el emisor, muestreando regularmente el medio con el fin de detectar actividad, manteniendo a la escucha el componente de radio durante un corto período de tiempo. Esta técnica denota un muy bajo consumo de energía cuando el canal está libre. Como desventaja, los largos periodos de encendido de la radio cuando detecta actividad en el canal. La idea que WiseMAC introduce para evitarlo consiste en que el emisor aprenda los periodos de muestreo del canal de los receptores y inicie su transmisión lo más cercano posible a estos periodos de muestreo, minimizando así la duración.

Ventajas: Los resultados de simulación muestran que WiseMAC se comporta mejor que S-MAC. Además su preámbulo dinámico ajusta los resultados para tener un mejor rendimiento en condiciones de tráfico variables. Además el protocolo se encarga de las fluctuaciones del reloj lo que disminuye el requisito de sincronización externa.
Desventajas: El principal inconveniente de WiseMAC radica en que la descentralización de los horarios resulta en diferentes tiempos para dormir y despertar de cada vecino en un nodo. Esto es específicamente un problema de la comunicación *broadcast* pues los paquetes enviados en este modo pueden ser almacenados en búfer en los nodos dormidos y entregado tantas veces como nodos dormidos se despierten. Esta redundancia de transmisión causará mayor latencia y consumo de energía.

**Tráffic-Adaptative MAC Protocol (TRAMA)**
Este algoritmo está basado en TDMA que busca incrementar la utilización del TDMA clásico de una manera energéticamente eficiente.
En TRAMA el tiempo es dividido en períodos de acceso aleatorio y acceso planificado. El primero es utilizado para establecer topologías de información a dos saltos donde el canal de acceso es basado en contención. Una suposición básica es que la capa MAC puede calcular la duración necesaria de la transmisión denominada SCHEDULE_INTERVAL. El nodo anuncia los intervalos que va a utilizar y los emisores con el paquete shedule.

Ventajas: Ofrece mayor porcentaje de intervalos de dormir y se consigue menos probabilidades de colisión comparadas con los protocolos basados en CSMA.
Desventajas: Los slots de transmisión están establecidos para ser siete veces mayores que los períodos de acceso aleatorio.

**SIFT**

Sift es un protocolo MAC propuesto para entornos de redes de sensores basados en eventos. Cuando se detecta un evento, se retransmite el primer mensaje posible de los N mensajes.

Sift se compara con el protocolo MAC 802.11 y se demostró que Sift reduce la latencia considerablemente cuando hay muchos nodos que tratan de enviar un mensaje. Sift es un método para el algoritmo de ranura de contención y puede coexistir con otros protocolos MAC como SMAC.

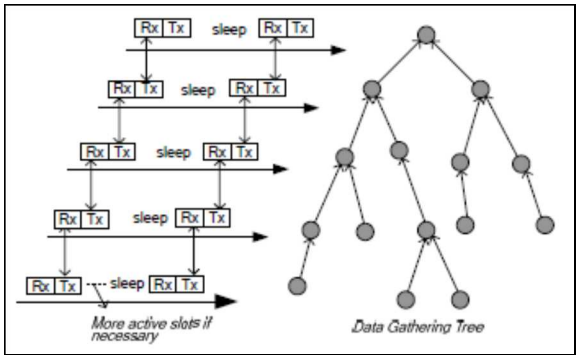
Ventajas: Se logra una latencia muy baja con muchas fuentes de tráfico. Sin embargo cuando la latencia es un parámetro importante del sistema, debe aceptarse un consumo mayor de energía.

Desventajas: Uno de los principales inconvenientes es causado por la escucha del canal cuando está desocupado, escuchando todas las ranuras antes de enviar. El segundo inconveniente es el aumento de escucha del canal cuando hay una transmisión en curso, los nodos deben escuchar hasta que acabe la transmisión. Además, es necesario el sistema de sincronización de tiempo para las ventanas de contención ranuradas. Por eso la complejidad de la implementación de Sift se incrementaría para los protocolos que no utilicen tiempos de sincronización.

**DMAC**

El tipo de patrón de comunicaciones convergecast, es el tipo de patrón más observado en de las redes de sensores. A partir de las fuentes posibles se crean rutas unidireccionales hacia el sumidero (sink), representado como una recopilación de datos tipo árbol. El objetivo principal de DMAC es lograr una baja latencia ahorrando energía.

DMAC se podría resumir como una mejora del algoritmo Aloha ranurado donde las ranuras se asignan a los conjuntos de nodos basados sobre una recopilación de datos tipo árbol como se muestra en la siguiente figura. Por lo tanto, durante el período de recepción de un nodo, todos sus nodos secundarios tienen que luchar por el medio para transmitir. La baja latencia se logra mediante la asignación de ranuras posteriores de los nodos que son sucesivas en el camino de transmisión de datos.




Recolector de datos tipo árbol y su implementación DMAC.

Ventajas: DMAC consigue una latencia muy buena en comparación con los periodos de dormir/escucha. La latencia de la red es crucial para ciertos escenarios, en el que DMAC podría ser un buen candidato.

Desventajas: Los métodos para evitar colisiones no se utilizan, por lo tanto, un número de nodos tiene el mismo periodo (mismo nivel en el árbol) intenten enviar al mismo nodo, se producirán colisiones. Además, los datos de enrutamiento no pueden ser conocidos de antemano, lo que impide la formación del árbol de recopilación de datos.

### Comparación de protocolos MAC

La siguiente tabla representa una comparación de los protocolos MAC. La columna tiempo de sincronización necesario, indica si el protocolo se obtiene la sincronización externamente. La columna de adaptabilidad a los cambios, significa que si tiene capacidad de adaptarse a los cambios de topología. Las dos variantes de S-MAC, T-MAC y DSMAC tienen las mismas características que S-MAC.

| Protocolo MAC | Tiempo de sincronización necesario | Patrón soportado | Tipo | Adaptabilidad a los cambios |
|---|---|---|---|---|
| **S-MAC / DSMAC/ WiseMAC** | No | All | CSMA | Buena |
| **TRAMA** | Si | All | TDMA/CSMA | Buena |
| **SIFT** | No | All | CSMA/CA | Buena |
| **DMAC** | Si | Convergecast | TDMA / Aloha ranurado | Débil |

**Tabla 2.3.:** Comparación de los protocolos MAC.

En la comunidad de investigación de redes de sensores, los protocolos de Radio Duty Cycling a menudo se llaman protocolos MAC ya que se trabajó en la capa MAC. En Contiki, se ha decidido dividir el ciclo de trabajo de la capa MAC y se trasladó a su propia capa, denominada capa de RDC.

**X-MAC**

X-MAC es un protocolo MAC de bajo consumo de energía para redes de sensores inalámbricas. Los estándares desarrollados con métodos de duty cycling tales como BMAC, el cual es el protocolo MAC por defecto para TinyOS, emplean un preámbulo muy extenso y provocando un exceso de consumo de energía en los receptores. X-MAC propone una solución empleando un método abreviado de preámbulo, que conserva las ventajas las comunicaciones de bajo consumo, simplicidad, y la planificación de los momentos de sleep de los transmisores y receptores. X-MAC acortando el preámbulo reduce significativamente el consumo de energía tanto en el transmisor como en el receptor, reduciendo la latencia por salto y ofreciendo ventajas adicionales como la adaptación flexible a los datos de los sensores tanto si se transmite a ráfagas como periódicamente.

Cada nodo sensor tiene ciclos periódicos en los cuales están entre el modo escucha y el modo sleep. El periodo de un ciclo de trabajo es equivalente al tiempo en el que está en modo escucha más el tiempo que está en modo sleep. El desafío al que se enfrentan los diseñadores de protocolos es la forma de lograr un alto rendimiento, bajo retardo y la eficiencia energética cuando los nodos están despertando y durmiendo en la red. X-MAC es un protocolo que adapta la energía eficientemente a WSNs con mecanismos de duty cycling.

Los protocolos MAC desarrollados para duty cycled WSNs pueden clasificarse en síncronos, asíncronos y híbridos, motivados por el deseo de reducir la escucha en reposo, que es el tiempo en que el nodo está despierto escuchando el medio a pesar de que no haya paquetes destinados a él. La escucha de inactividad consume energía y se debe evitar. Protocolos sincronizados como S-MAC y T-MAC, negocian el momento en que los nodos deben estar despiertos para comunicarse, reduciendo el tiempo y el derroche de energía en la escucha inactiva. Los protocolos asíncronos, como B-MAC, y WISEMAC, se basan en la escucha de baja potencia, denominada muestreo de preámbulo, para poder enlazar con los datos de un emisor con los datos de un receptor el cual utiliza el mecanismo de duty cycling. La escucha inactiva se reduce en los protocolos asíncronos al transferir la sincronización con el emisor. Cuando un emisor tiene datos, el emisor transfiere un preámbulo que al menos tiene una longitud del tamaño del periodo sleep del receptor. El receptor se despierta, detecta el preámbulo y permanece despierto para recibir los datos. Esto permite una comunicación de baja potencia sin la necesidad de una sincronización explicita entre los nodos. El receptor sólo se despierta durante un corto tiempo para escuchar el medio, lo que limita la escucha de inactividad.

Una ventaja clave en los protocolos asíncronos de baja potencia en modo escucha es que el emisor y el receptor pueden estar completamente sin sincronizar sus ciclos de trabajo. La simplicidad de este diseño elimina la necesidad de planificar los tiempos de sincronización entre los tiempos de escucha y sueño.

Mientras que el bajo consumo en la escucha es simple, asíncrono y eficiente en consumo, el largo preámbulo presenta varias desventajas en términos de consumo de energía, tanto en el emisor como en el receptor, ya que están sujetos a oírlo, y provoca un exceso de consumo en los receptores y se introduce un exceso de latencia en cada salto. En primer lugar, el receptor tiene que esperar el período completo hasta que el preámbulo se termina antes de que empiecen a transmitirse los datos/ACK, incluso si el receptor se ha despertado en el empiezo del preámbulo derrochando energía en el receptor y el en emisor. En segundo lugar, la escucha de bajo consumo sufre el problema de sobre escucha (overhearing), el cual los receptores que no son objetivos del emisor también se despierten durante el largo preámbulo y tiene que permanecer despierto hasta el final del preámbulo para averiguar si el paquete está destinado a ellos. Esto desperdicia energía en todos los receptores. En tercer lugar, debido a que el receptor tiene que esperar a que se reciba todo el preámbulo antes de recibir el paquete de datos, la latencia por salto se delimita por la longitud del preámbulo. En una ruta multisalto, esta latencia se puede acumular y puede llegar a ser bastante sustancial.

Con X-MAC se propone un nuevo enfoque a la escucha de baja potencia empleando un preámbulo corto para reducir el consumo energético y reducir la latencia. La primera idea es incluir información de la dirección de destino en el preámbulo para que los receptores que no sean objetivos se puedan ir a dormir rápidamente. La segunda idea es utilizar un preámbulo estroboscópico* reduciendo el tiempo y el desperdicio de energía esperando a que se reciba el preámbulo completo. Se ha demostrado a través de la implementación en un banco de pruebas de sensores inalámbricos que se traduce en ahorros significativos utilizando X-MAC, en términos de latencia y energía. Por último X-MAC incluye un algoritmo automatizado para adaptar el ciclo de trabajo de los nodos para acomodar la carga de tráfico en la red.

X-MAC nos puede aportar:

- X-MAC introduce paquetes cortos de preámbulo incluyendo información de dirección de destino, evitando así el problema de escucha de baja potencia y ahorrando energía en los receptores que no son objetivos del emisor.
- X-MAC inserta una pausa en los paquetes cortos de preámbulo, creando un preámbulo estroboscópico o beacon, que permite al receptor especifico acortar el beacon a través de un rápido paquete de reconocimiento, lo que se consigue un ahorro adicional de energía tanto en el emisor como en el receptor, así como una reducción en la latencia por salto.
- X-MAC describe un algoritmo adaptativo que ajusta automáticamente el ciclo de trabajo de los receptores a la carga de tráfico, reduciendo más aun la latencia por salto.
- Evaluaciones experimentales han validado que X-MAC mejora el rendimiento y el ahorro de energía en comparación con técnicas de duty cycle asíncronas

Los objetivos de diseño del protocolo X-MAC son:

- Eficiencia energética
- Simple y de baja sobre escucha, implementación distribuida
- Baja latencia para datos
- Alto rendimiento para datos
- Adaptabilidad a la carga ofrecida de datos.
- Aplicable en todos los tipos de radio tanto con flujo de bits digitales como paquetes.

En X-MAC, el preámbulo extenso se corta cuando el receptor se despierta, proporcionando ahorro energético y reduciendo la latencia por salto, ya que el nodo receptor puede empezar a reenviar el paquete después de recibirlo. La reducción de la latencia con X-MAC es aproximadamente del 50%.

## Protocolos de enrutamiento

Los nodos no tienen un conocimiento de la topología de la red, deben descubrirla. La idea básica es que cuando un nuevo nodo, al aparecer en una red, anuncia su presencia y escucha los anuncios broadcast de sus vecinos. El nodo se informa acerca de los nuevos nodos que tiene a su alcance y de la manera de encaminarse a través de ellos, a su vez, puede anunciar el resto de nodos que pueden ser accedidos desde él. Transcurrido un tiempo, cada nodo sabrá que nodos tiene alrededor y una o más formas de alcanzarlos.

Los algoritmos de enrutamiento en redes de sensores inalámbricas tienen que cumplir las siguientes normas:

- Mantener una tabla de enrutamiento razonablemente pequeña.
- Elegir la mejor ruta para un destino dado (ya sea el más rápido, confiable, de mejor capacidad o la ruta de menos coste).
- Mantener la tabla regularmente actualizada para actualizar la caída de nodos, su cambio de posición o su aparición.

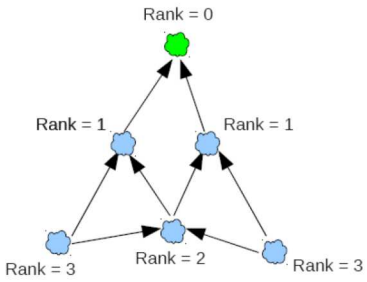
Existen varios modos de enrutamiento, como es el modelo de difusión directa (modelo de un salto) y el modelo multisalto (multi-hop).
El modelo de difusión directa o de un salto es más simple y representa la comunicación directa. Todos los nodos de la red transmiten a la estación base recolectora y tienen una distancia máxima de radio, por ello la comunicación directa no es una buena solución para las redes inalámbricas.
En modelo multisalto, un nodo transmite a la estación base recolectora o sink, reenviando sus datos a uno de sus vecinos, el cual está más próximo, hasta llegar al sink. Entonces la información viaja de la fuente al destino salto a salto desde un nodo a otro hasta que llega al destino.

En este modelo se basa el protocolo RPL que es un protocolo diseñado para redes inalámbricas que se implementa en un entorno con pérdidas donde es probable que se pierdan mensajes y en redes que consisten en nodos alimentados con baterías.

RPL es un protocolo que consiste en técnicas de encaminamiento que se organizan en mensajes DAG (Directed Acyclic Graphs - Gráficos acíclicos dirigidos). DAG es una estructura en la que todos los nodos están conectados, pero no hay un solo camino de ida y vuelta disponible de un nodo a otro. Los nodos están conectados entre sí y todos los mensajes se enrutan hacia un nodo formando nuevas estructuras llamadas DODAG (destination-oriented DAG). El nodo que recoge todos los mensajes DODAG se llama raíz o sink (sumidero). La siguiente figura representa una estructura DODAG con un nodo en verde que es el nodo DODAG raíz.
La formación de la estructura DODAG empieza desde la raíz. La raíz transmite un mensaje DIO (DODAG Information Object - Objeto de Información DODAG), hacia sus vecinos. Cada nodo en la red tiene su propio rango que representa su nivel en la red. El rango es una representación escalar de la posición de los nodos en la DODAG actual y no permite bucles en la red. Después de haber recibido el mensaje DIO desde la raíz, cada nodo calcula su propio rango y envía

sus mensajes de DIO con los nuevos valores de rango a sus vecinos para que puedan calcular sus propios rangos. El rango aumenta a través de la red con el aumento de la distancia hacia la raíz. El DODAG raíz tiene la opción de iniciar de nuevo el cálculo del rango de los nodos. Para iniciar el recalculo, la raíz envía mensajes DIO con el número de secuencia superior al mensaje anterior y avisa a los demás nodos que quiere rehacer la estructura DODAG.

Cuando un nodo se incorpora por primera vez a la estructura DODAG, tendrá que esperar a escuchar un mensaje DIO desde el nodo más cercano.




DODAG (Destination oriented DAG).

Un nodo también puede trasmitir su propio mensaje de solicitud de información DODAG (DODAG Information Solicitation) desde el nodo más cercano.
Uno de los campos del mensaje DIO tiene información sobre la función objetivo. La función objetivo se usa para calcular el costo de envío de mensajes por el enlace hacia la raíz y para asegurar que el mensaje se envía a través del enlace con el menor coste. Hay muchos tipos de funciones objetivo y todos los nodos de la red tienen que saber el tipo de función objetivo utilizado.

Según la función objetivo y los rangos de sus vecinos recibidos en el mensaje DIO, los nodos podrían unirse a la DODAG o mantenerse en la DODAG existente. La función objetivo depende sólo de la aplicación actual que se está utilizando en la red, sin embargo no disminuye las posibilidades de utilizar RPL.

El tráfico RPL soporta mensajes bidireccionales, haciendo posible enviar mensajes desde los nodos hacia el sink y desde el sink hacia los nodos. Los mensajes que envían los nodos hacia el sink con trayectoria hacia arriba a través de DODAG, elige su camino mediante la selección de “padre preferido”. La raíz DODAG en los mensajes DIO puede tener anunciado un conjunto de prefijos de destino a los que les proporciona conectividad. Estos prefijos se pueden utilizar para rellenar las tablas de enrutamiento de la red o como ruta por defecto a través de padres preferidos.

RPL de diseña como protocolo de comunicaciones multipunto-a-punto (MP2P) para entregar los datos de los nodos de la red a la raíz DODAG (o sink en WSNs). Con tráfico adicional, RPL puede utilizar también comunicaciones punto-a-multipunto (P2MP) y de punto a punto (P2P). El tráfico adicional significa que RPL tendría que tener algún otro tipo de mensajes, además de mensajes DIO.

Los mensajes DIO se utilizan para la configuración y el mantenimiento de la red y el mensaje DIS se utiliza para añadir nuevos nodos al DODAG. Otro tipo de mensajes que define RPL es el DAO (Destination Advertisement Object). Los mensajes DAO se envían desde un nodo hacia la raíz y permite que los nodos anuncien su presencia a los nodos que se encuentran entre el nodo y la raíz. Cada nodo transmite el mensaje de DAO a su padre, el cual escribe su propia dirección en el mensaje recibido y lo reenvía a su propio padre. El mismo procedimiento se repite en todo el camino hasta llegar a la raíz. El padre que recibe el primer mensaje DAO desde un nodo se denomina padre preferido y para cada nodo no puede haber más de un padre preferido. Si la raíz DODAG recibe el mensaje de DAO, lee las direcciones de los nodos en orden inverso, creando así el camino desde el nodo hacia la raíz. La raíz guarda esta ruta en su memoria y lo utiliza cada vez que se quiere enviar un mensaje a ese nodo.

Resumiendo, RPL es un protocolo que proporciona una configuración rápida de la red y la distribución del conocimiento a través de la red y la adaptación efectiva de la red incluso si la topología de la red cambia. Se recomienda su uso en escenarios de smart grids y WSNs.

# Anexo IV. Pila de protocolos Rime

Rime es un stack modular, estructurado en capas muy simples de manera de que los encabezados sean lo más pequeños posibles. Las distintas capas de Rime implementan funcionalidades asociadas a los servicios de comunicación que se quieren brindar.
Rime es una pila de comunicación ligero diseñado para radios de baja potencia que ofrece una amplia gama de primitivas y protocolos, como la recogida de datos multi-salto (multi-hop data collection), enrutamiento unicast multi-salto y comunicaciones broadcast multi-salto.

Seguidamente se nombran las primitivas de comunicaciones menos complejas:

**abc (Anonymous Best-effort Single-hop Broadcast)**
Es la primitiva de comunicación más básica. Permite a las capas superiores enviar paquetes a todos los vecinos que escuchan el canal en el que se envió el paquete. No se envía información de quien envió el paquete. El resto de las primitivas están basadas en esta.
**ibc (Identified Best-effort Single-hop Broadcast)**
Utiliza la primitiva abc y agrega la dirección del nodo remitente.
**uc (Best-effort Single-hop Unicast)**
Envía un paquete a un nodo vecino determinado. Usa la primitiva ibc y agrega la dirección del destinatario. Cuando un nodo recibe un paquete chequea la dirección del destinatario, si esta no coincide con la propia, descarta el paquete.
**stuc (Stubborn Single-hop Unicast)**
Envía repetidamente un paquete usando uc hasta que la capa superior cancela la transmisión.
**ruc (Reliable Single-hop Unicast)**
Utiliza acks y retransmisiones para asegurar que el nodo vecino recibió exitosamente un paquete.
Cuando el destinatario responde el ack, el modulo ruc notifica a la capa superior.
Utiliza la primitiva stuc para realizar retransmisiones y números de secuencia para confirmar los acks de los paquetes enviado.
**polite (Polite Single-hop Broadcast)**
Está diseñado para reducir la cantidad de paquetes enviados no repitiendo un mensaje que otro nodo ya envió. La capa superior que usa este protocolo setea un intervalo de tiempo en el cual se debe evitar el paquete. Si cuando este intervalo expira no se recibió el paquete que se está por enviar, entonces es enviado.
**Ipolite (Identified Polite Single-hop Broadcast)**
Utiliza la primitiva polite y agrega el identificador de remitente.
**mh (Best-effort Multi-hop Unicast)**
Envía un paquete a un nodo especifico de la red usando reenvió de multi-hop en cada nodo de la red.
La capa superior (o aplicación) que usa mh pone el algoritmo de ruteo para seleccionar el próximo el próximo paso. Si a la primitiva se le solicita enviar un paquete para el cual no se encuentra ningún nodo vecino adecuado, se le notifica a la capa superior y esta pude iniciar el proceso de descubrimiento de ruta.

Cuando el next-hop neighbor es encontrado la primitiva mh manda el paquete usando uc.

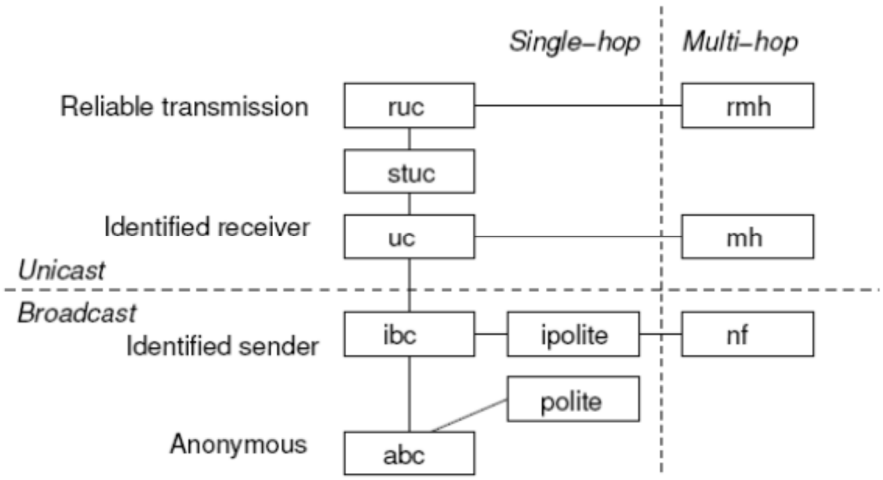
**rmh (Hop-by-hop Reliable Multi-hop Unicast)**

Es similar a mh salvo que usa la primitiva ruc en vez de uc para comunicarse confiablemente con los nodos vecinos.

**nf (Best-effort Network Flooding)**

Envía un paquete a todos los nodos de la red. Usa un polite broadcast para cada hop para reducir el número de transmisiones redundantes. Usa el atributo “time to live” para eliminar paquetes.

En la siguiente figura se muestra las primitivas de comunicación de la pila Rime y la forma en la que se superponen.




Primitivas de comunicación de la pila Rime

Las aplicaciones o protocolos que se ejecutan en la parte superior de la pila Rime se comunican en cualquier capa de la pila y usa cualquiera de las primitivas de comunicación.

La pila Rime soporta primitivas de un solo salto y primitivas de comunicación multi-salto. Las primitivas multi-salto no especifican cómo se enrutan los paquetes a través de la red. En su lugar, como el paquete se envía a través de la red, el protocolo de capa de aplicación o la capa superior se invoca en cada nodo para elegir el vecino del siguiente salto. Esto hace que sea posible la aplicación de los protocolos de enrutamiento arbitrarias en la parte superior de las primitivas multi-salto.

Los protocolos o aplicaciones que se ejecutan en la parte superior de la pila de Rime, pueden implementar protocolos adicionales que no están en la pila Rime. Si un protocolo o aplicación que se ejecuta en la parte superior de la pila de Rime necesitara una primitiva de comunicación que no está actualmente en la pila Rime, la aplicación o el protocolo puede implementarse directamente sobre la parte superior de otro tipo de primitiva de comunicación de la pila.

# Anexo V. Aplicación

```
* \file
 *        Aplicación para la medición de la temperatura ambiental
          de los establecimientos, el estado de la batería de la mota y
          el consumo energético que está realizando en ése momento por
          el uso de los splits del aire acondicionado del edificio B4
          del PMT de Castelldefels (CTTC).
 * \author
 *          Jose Manuel Linares Arenas [jlinaresarenas@gmail.com]
 */

#include <stdio.h>
#include "contiki.h"
#include "lib/random.h"
#include "net/rime.h"
#include "net/rime/collect.h"
#include "dev/leds.h"
#include "dev/button-sensor.h"
#include "dev/i2cmaster.h"
#include "dev/tmp102.h"
#include "net/netstack.h"
#include "dev/battery-sensor.h"
#include "dev/leds.h"
#include "dev/z1-phidgets.h"

#define TMP102_READ_INTERVAL (CLOCK_SECOND/2)

static struct collect_conn tc;
static struct etimer et;

float floor(float x) {
  if(x >= 0.0f) {
    return (float) ((int) x);
  } else {
    return (float) ((int) x - 1);
  }
}

static const char * getBattery() {
    char result[32]="";
    uint16_t bateria = battery_sensor.value(0);
    float mv = (bateria * 2.500 * 2) / 4096;
    sprintf(result,"Battery: %i ", bateria);
return &result;
}

static const char * getConsumption() {
    char result[32]="";
    leds_toggle(LEDS_GREEN);
```

```
    if (phidgets.value(PHIDGET3V_1) < 100) {
       leds_on(LEDS_RED);
     } else {
        leds_off(LEDS_RED);
     }
    sprintf(result,"Phidget 3V2:%d\n",phidgets.value(PHIDGET3V_2));
   return &result;
}

static const char * getTemperature() {
    int16_t tempint;
    uint16_t tempfrac;
    int16_t raw;
    uint16_t absraw;
    int16_t sign;
    char minus = ' ';
    char cadena[32]="";
    sign = 1;
         raw = tmp102_read_temp_raw();
         absraw = raw;
         if(raw < 0) {
              absraw = (raw ^ 0xFFFF) + 1;
              sign = -1;
         }
    tempint = (absraw >> 8) * sign;
    tempfrac = ((absraw >> 4) % 16) * 625;
    minus = ((tempint == 0) & (sign == -1))? '-' : ' ';
    sprintf(cadena,"Temp = %c%d.%04d ", minus, tempint, tempfrac);
    return &cadena;
}

/*---------------------------------------------------------------*/
PROCESS(example_collect_process, "Test collect process");
AUTOSTART_PROCESSES(&example_collect_process);
/*---------------------------------------------------------------*/
static void recv (const rimeaddr_t *originator, uint8_t seqno,
uint8_t hops)
{
  printf("Sink got message from %d.%d, seqno %d, hops %d: len %d
'%s'\n",  originator->u8[0], originator->u8[1],seqno, hops,
packetbuf_datalen(), (char *)packetbuf_dataptr());
}
/*---------------------------------------------------------------*/
static const struct collect callbacks callbacks = { recv };
/*---------------------------------------------------------------*/
PROCESS_THREAD(example_collect_process, ev, data)
{
  static struct etimer periodic;
```

```
  PROCESS_BEGIN();
  SENSORS_ACTIVATE(battery_sensor);
  SENSORS_ACTIVATE(phidgets);
  SENSORS_ACTIVATE(button_sensor);
  collect_open(&tc, 130, COLLECT_ROUTER, &callbacks);
if(rimeaddr_node_addr.u8[0] == 1 && rimeaddr_node_addr.u8[1] == 0)
{
    printf("I am sink\n");
    collect_set_sink(&tc, 1);
}

static struct etimer et;
tmp102_init();
etimer_set(&et, TMP102_READ_INTERVAL);
PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&et));
while(1) {
    /* Send a packet every 30 seconds. */
    if(etimer_expired(&periodic)) {
      etimer_set(&periodic, CLOCK_SECOND * 30);
      etimer_set(&et, random_rand() % (CLOCK_SECOND * 30));
    }
    PROCESS_WAIT_EVENT();
    if(etimer_expired(&et)) {
      static rimeaddr_t oldparent;
      const rimeaddr_t *parent;
      printf("Sending\n");
      packetbuf_clear();
      char result[128] ="";
      strcat(result,getTemperature ());
      strcat(result,getBattery());
      strcat(result,getConsumption());
      packetbuf_set_datalen(sprintf(packetbuf_dataptr(),"%s",
    result )+2);
    printf(result);
      collect_send(&tc, 15);
      parent = collect_parent(&tc);
      if(!rimeaddr_cmp(parent, &oldparent)) {
        if(!rimeaddr_cmp(&oldparent, &rimeaddr_null)) {
          printf("#L %d 0\n", oldparent.u8[0]);
        }
        if(!rimeaddr_cmp(parent, &rimeaddr_null)) {
          printf("#L %d 1\n", parent->u8[0]);
        }
        rimeaddr_copy(&oldparent, parent);
}
    }
  }
SENSORS_DEACTIVATE(battery_sensor);
PROCESS_END();
}
/*---------------------------------------------------------------*/
```