Universidad de la República
Facultad de Ingeniería

RSItrust

Red de Sensores Inalámbricos para monitoreo de condiciones microclimáticas en cultivos de cítricos

Memoria de proyecto presentada a la Facultad de Ingeniería de la Universidad de la República por

Cecilia Cardozo, Ignacio Camps, Martín Driedger

en cumplimiento parcial de los requerimientos para la obtención del título de Ingeniero Electricista.

Tutores

Dr. Ing. Leonardo Steinfeld ..... Universidad de la República
Ing. Javier Schandy ..... Universidad de la República

Tribunal

Dr. Ing. Fernando Silveira ..... Universidad de la República
Dr. Ing. Pablo Belzarena ..... Universidad de la República
Dr. Ing. Leonardo Steinfeld ..... Universidad de la República

Montevideo
domingo 16 agosto, 2015

RSItrust

Red de Sensores Inalámbricos para monitoreo de condiciones microclimáticas en cultivos de cítricos, Cecilia Cardozo, Ignacio Camps, Martín Driedger.

Esta tesis fue preparada en LaTeX usando la clase iietesis (v1.1).
Contiene un total de 183 páginas.
Compilada el domingo 16 agosto, 2015.
http://iie.fing.edu.uy/

Agradecimientos

Agradecemos a nuestros padres, familiares y amigos por brindarnos su apoyo para seguir estudiando y lograr el objetivo de finalizar la carrera. Por acompañarnos durante el proyecto y el transcurso de la etapa universitaria. A todos los que alguna vez le dijimos: "No puedo ir, tengo que estudiar".

Agradecemos a nuestros tutores Leonardo Steinfeld y Javier Schandy por guiarnos durante el proyecto y compartirnos su conocimiento.

A Mauricio Gonzalez, Agustina Pieruccioni y Florencia Arbío por la ayuda brindada.

Al grupo de Microelectrónica del IIE y el FPTA-INIA por permitirnos formar parte del proyecto GERVASIO.

A Miguel Barreto, Gabriel Eirea y Gregory Randall por darnos las herramientas de gestión necesarias para llevar a cabo el proyecto.

A todos aquellos docentes que alimentaron nuestras mentes vacías de conocimiento durante la carrera.

Agradecemos a Google por todas las herramientas desarrolladas que nos fueron de gran ayuda.

Resumen

En el proyecto de fin de carrera RSItrust se diseñó y se implementó una red de sensores inalámbricos, para monitoreo de condiciones microclimáticas en cultivos de cítricos. Las magnitudes que se adquieren incluyen humedad y temperatura del aire de manera de reducir el impacto de las heladas emitiendo alertas o posibilitando acciones post-cosecha, y humedad del suelo posibilitando la racionalización del riego, entre otras aplicaciones.

Se utilizó el sistema operativo Contiki OS. Se incluyó el stack de protocolos recomendado por los desarrolladores: IEEE 802.15.4 PHY en capa física, ContikiMAC para manejar el ciclo de trabajo de la radio, IEEE 802.15.4 MAC en capa de enlace, 6LoWPAN en capa de adaptación, IPv6 y RPL en capa de red, UDP en capa de transporte y CoAP en capa de aplicación. Durante el proyecto los parámetros de configuración de estos protocolos fueron optimizados para reducir el consumo energético en la comunicación y así aumentar la vida útil de las baterías de los nodos que componen la red. Para ello se empleó la herramienta Energest de Contiki OS para lograr este último objetivo.

La aplicación de software implementada permite solicitar a los nodos las medidas de los sensores mencionados, pudiendo modificar el periodo de sensado y la hora de los nodos. Además se puede realizar una solicitud en la cual el nodo envía periódicamente la medidas, siendo este periodo también configurable. Adicionalmente, se le puede solicitar su tabla de rutas y los periodos de tiempo en los cuales el microcontrolador permanece en sus distintos estados de funcionamiento (modo bajo consumo por ejemplo).

Se diseñó un nodo basado en el SoC CC2538 de Texas Instrument utilizando un módulo fabricado por EMBIT. Se integraron sensores de temperatura y humedad del aire, y humedad del suelo.

La RSI implementada fue sometida a diversas pruebas, tanto de software, hardware y del funcionamiento de la red. Dichas pruebas fueron pasadas con éxito, obteniendo así una red robusta. RSItrust se enmarca en el proyecto FPTA-INIA GERVASIO, que contiene una línea de trabajo para la aplicación de las redes. Por lo que se redactó un manual de fabricación como guía para su reproducción.

Tabla de contenidos

Tabla de contenidos

Capítulo 1

Introducción

1.1. Descripción del problema

Las Redes de Sensores Inalámbricas (RSI) se han propuesto para ser utilizadas en la producción agropecuaria para realizar de forma más eficiente diferentes procesos de producción. Debido a que la comercialización de esta tecnología es incipiente en el mundo, en nuestro país no existen productos comerciales disponibles. El grupo de microelectrónica del Instituto de Ingeniería Eléctrica (IIE) de la Universidad de la República (UdelaR), ha llevado adelante varios proyectos de utilización de RSI para la agricultura. Ejemplos de ellos son, FPTA-INIA SIMPA (2009-2011) [52], proyectos de fin de carrera como ContikiWSN (2011) [27] y Plagavisión (2014) [48]. Estos proyectos avanzaron en diversos aspectos y sirvieron para mostrar la potencialidad de esta tecnología y los beneficios que presenta. Sin embargo, estos pilotos no se desarrollaron considerando lograr un producto que fuera fácilmente producido a mayor escala.

RSItrust se enmarca en el proyecto FPTA-INIA GERVASIO [25], que contiene una línea de trabajo para la aplicación de las RSI para monitoreo de condiciones microclimáticas en cultivos de cítricos. Las magnitudes que se adquieren incluyen humedad y temperatura del aire de manera que permita reducir el impacto de las heladas, y humedad del suelo posibilitando la racionalización del riego, entre otras aplicaciones.

1.2. Objetivo General

El objetivo del proyecto es diseñar e implementar una RSI, con el fin de obtener un producto confiable (diseñado para ser tolerante a fallos, robusto y debidamente testeado) y pensado para ser producido en mayor escala. Dicho sistema debe contar con sensores de humedad y temperatura del aire, y humedad del suelo, permitiendo monitorear las condiciones microclimáticas. La red está formada por varios nodos a los cuales se conectan los sensores, y un nodo base a través del cual el usuario se comunica con los nodos sensores.

1.3. Alcance

El alcance del proyecto abarca los siguientes puntos:

1. Diseño de circuitos auxiliares para conexión e integración de sensores a cada nodo que conforma la red [1].

2. Diseño y construcción del montaje físico de protección de los dispositivos.

3. Diseño, análisis e implementación de la red. Este punto incluye la implementación de la aplicación y modificación de los parámetros necesarios de los protocolos de comunicación de red.

Queda fuera del alcance del proyecto:

1. Selección del microcontrolador (MCU), la radio y sensores a utilizar.

2. Diseño del módulo principal de los nodos de sensores inalámbricos (placa conteniendo MCU y radio).

3. Diseño hardware del nodo base.[2]

4. Integración del Gateway a la red.[3]

1.4. Criterios de éxito

El proyecto se considera exitoso si cumple con los siguientes criterios:

1. El sistema tanto hardware como software sea fácilmente reproducibles a partir de un manual de producción.

2. Cada uno de los nodos de la red tiene una autonomía estimada mayor a un año, contando con una capacidad de $2800mAh$ (equivalente a dos pilas AA).

3. Se logran enrutar el 100 % de los datos siempre que exista un camino hacia la base.

4. Los nodos cuentan con un grado de protección $IP55$ (Entrada de polvo en una cantidad tal que no interfiera con el correcto funcionamiento del equipamiento y protección completa contra chorros de agua desde todas las direcciones).

[1] Nodo: Es un punto de intersección o conexión en una red

[2] Dicho nodo iba a ser implementado en un curso de facultad, para utilizar si quedaba integrable.

[3] Se realizó una replanificación en la cual este punto quedó fuera del alcance del proyecto.

1.5. Actores

Los actores involucrados en el proyecto son los siguientes:

1. Agrisur-Urud'or.[4]
2. INIA.[5]
3. Grupo de Microelectrónica del IIE.
4. Tutores del Proyecto.
5. Equipo de Proyecto.

1.6. Supuestos

El proyecto se desarrolla bajo los siguientes supuestos:

1. *ContikiOS*[6] cuenta con documentación suficiente (tutoriales, wiki, ejemplos y código comentado) que permita desarrollar las aplicaciones y luego poder realizar modificaciones en la implementación de los protocolos.
2. Se cuenta con los sensores de humedad y temperatura del aire, de humedad del suelo y el MCU.
3. Se cuenta con la financiación de INIA para la realización del Proyecto.
4. Se cuenta con instrumentos para realizar las medidas y apoyo para pruebas en campo.

1.7. Restricciones

A continuación se enumeran las restricciones impuestas al grupo de Proyecto:

1. Se utilizan los sensores y módulos seleccionados.
2. Se utiliza *ContikiOS* como plataforma de software.

[4]Agrisur-Urud'or: `http://www.urudor.com.uy/espanol/f_organizacion.html`
[5]INIA: `http://www.inia.uy/`
[6]ContikiOS se explica en el capítulo 3

1.8. Especificación funcional del Proyecto

El sistema está compuesto por varios nodos que forman una RSI, los cuales cuentan con sensores de temperatura y humedad del aire y humedad del suelo. Las medidas se envían a través de la red hacia una aplicación en un PC a través de un Border Router, el cual procesa y muestra los datos. Además, se le puede solicitar a todos los nodos información del estado del mismo y su tabla de rutas.

El período de muestreo de los sensores es configurable, siendo esta configuración enviada remotamente a través del router hacia el nodo sensor de la red. De esta manera, también se realiza la configuración de la hora de cada nodo y del período de observación, el cual indica cada cuanto tiempo el nodo sensor envía la última medida registrada.

El sistema tiene los siguientes requerimientos no funcionales: robusto, de bajo consumo y tolerante a fallos.

1.9. Conclusiones principales

En RSItrust se logró diseñar e implementar una RSI, la cual cuenta con el diseño de nodos que poseen sensores de humedad y temperatura del aire y humedad del suelo. La aplicación de software implementada permite solicitar a los nodos las medidas de los sensores mencionados, pudiendo modificar el período de sensado y la hora de los nodos. Además se puede realizar una solicitud en la cual el nodo envía periódicamente la medidas, siendo este período también configurable. Adicionalmente, se le puede solicitar su tabla de rutas y los períodos de tiempo en los cuales el MCU permanece en sus distintos estados de funcionamiento (modo bajo consumo por ejemplo). También permite configurar intervalos de temperatura y humedades en los cuales el usuario considera que el cultivo no está en peligro. Esto último se puede utilizar en trabajos futuros para disparar alarmas, las cuales no fueron implementadas.

Con la optimización de protocolos se logró aumentar la vida útil de 7 a 9 meses, no llegando así a superar el año de vida que se había planteado como objetivo. Sin embargo, se analizó, que si se utilizara un módulo sin amplificación exterior, el tiempo de vida superaría ampliamente el año, a costo de disminuir la distancia entre nodos.

Los nodos cuentan con una caja con protección $IP65$ la cual tiene mayor protección que la deseada, cumpliendo con el criterio de éxito.

Se cuenta con manuales de usuario para la programación y configuración de software, así como también un manual de test y reproducción del diseño de hardware, haciendo que el sistema sea fácilmente reproducible.

1.10. Descripción de los capítulos

1.10.1. Capítulo 1 - Introducción

El primer capítulo describe formalmente el proyecto, marcando los objetivos, alcance, criterios de éxito, actores, supuestos, restricciones, especificación funcional y conclusiones principales. También se indican antecedentes, junto con referencias a dichos documentos.

1.10.2. Capítulo 2 - Diseño general de la solución

En el segundo capítulo se da una visión general del sistema que se creó y su arquitectura.

1.10.3. Capítulo 3 - Estado del arte

En el tercer capítulo se describen las RSI en general y sus aplicaciones al agro. Se introducen los temas de redes de bajo consumo, Contiki OS y el *stack* de protocolos utilizados, incluyendo la evolución histórica y terminología de cada uno de los puntos anteriores.

1.10.4. Capítulo 4 - Hardware

En el cuarto capítulo se describe el diseño de hardware, incluyendo el criterio de elección de los componentes y precauciones para el diseño del PCB y sus protecciones mecánicas. Se explica el proceso de elección de los sensores y del MCU, y la integración de los mismos. Además se detallan los test de funcionamiento, explicando los errores iniciales de diseño y cómo se resolvieron.

1.10.5. Capítulo 5 - Software

El quinto capítulo cuenta con una descripción funcional de la aplicación, explicando el diseño y desarrollo de la misma. Detalla los archivos utilizados y modificados de ContikiOS, así como también describe los drivers de los sensores utilizados. Además se especifican los test de la aplicación que se realizaron y sus resultados.

1.10.6. Capítulo 6 - Optimización de Protocolos

El sexto capítulo describe la estrategia de pruebas y modificaciones de los parámetros de los protocolos del *stack* utilizado, para lograr optimizar el consumo de los nodos. Explica tanto las simulaciones realizadas como las pruebas en campo, mostrando los datos obtenidos y su posterior análisis.

1.10.7. Capítulo 7 - Test de la Red

En el séptimo capítulo se describen las pruebas y resultados obtenidos al probar el sistema integrado de software y hardware. Se detalla tanto las pruebas del nodo como las pruebas de la red, verificando así su correcto funcionamiento.

1.10.8. Capítulo 8 - Conclusiones

Finalmente en el octavo capítulo se evalúan de manera global los resultados obtenidos, resaltando los aportes novedosos y comparando lo obtenido con los objetivos iniciales planteados anteriormente. También se detallan las dificultades y los trabajos a futuro que se pueden realizar a partir de este proyecto.

Capítulo 2

Diseño general de la solución

2.1. Introducción

En el presente capítulo se explica el funcionamiento e implementación del sistema creado, mostrando su arquitectura de hardware y software. Se explica brevemente de qué manera se desarrolló el sistema, logrando así el funcionamiento deseado.

2.2. Desarrollo del Sistema

El sistema creado para observar condiciones climáticas en una plantación de cítricos se compone de una red de nodos, los cuales son capaces de medir humedad del suelo, temperatura y humedad del aire. Estos se ubican distribuidos a $100m$ de distancia entre ellos aproximadamente, la cual es una distancia típica en estas instalaciones, instalando un nodo por hectárea.

Los nodos que componen la red cuentan con sensores para medir las magnitudes mencionadas anteriormente, los cuales se conectan con conectores $3{,}5mm$ en un PCB. El sensor de humedad del suelo es un sensor de interfaz analógica, mientras que los de temperatura y humedad del aire son de interfaz digital comunicándose a través del bus serial $I2C$. El PCB fue diseñado en base a un módulo compuesto por un SoC (chip que integra un MCU y una radio, del inglés: *System on a Chip*), amplificador de radio y antena. Además cuenta con switches para habilitación y deshabilitación de los sensores, porta pilas para alimentar el sistema con dos pilas AA, LED y botones de usuario y conectores UART y JTAG.

El sistema utiliza el *stack* de protocolos de comunicación que se introduce en el capítulo 3. Es un *stack* de protocolos orientado a las redes de sensores inalámbricos desarrollado para ser más liviano y adaptarse a las dificultades presentes en una red con recursos limitados y muchas pérdidas. Utilizando estos protocolos, cada nodo genera las rutas hacia el resto, buscando el mejor camino para enrutar

los paquetes. En la Tabla 3.1 se puede observar el *stack* de protocolos seleccionado.

En este SoC se ejecuta una aplicación programada utilizando el sistema operativo Contiki OS (ver en capítulo 3). Cada sensor conectado al nodo, periódicamente realiza una lectura, y esta se guarda en la memoria del MCU, para luego enviarla hacia el nodo base cuando éste se lo solicite. El período de muestreo de los sensores es configurable, siendo esta configuración enviada remotamente a través del router por la aplicación en un PC, utilizando el protocolo CoAP de capa de aplicación, el cual se presenta en el capítulo 3. Previo a cada lectura el sistema habilita el switch que energiza cada sensor y realiza una comprobación de si hay sensor conectado o no mediante el sensor de presencia con que cuentan los conectores a los cuales se conectan los mismos. Si no hay sensor conectado el sistema envía un mensaje de error. A su vez, para el caso de los sensores de humedad y temperatura del aire, si hay una falla de comunicación $I2C$ se obtiene el mensaje de error correspondiente. En estos casos el usuario en lugar de la medida recibe un mensaje de error del tipo *SERVICE UNAVAILABLE* (Servicio no disponible).

Se implementó un recurso CoAP para cada sensor, uno para configuración, otro para los estados del MCU y otro para obtener la tabla de rutas de cada nodo. Los recursos encargados de tomar las medidas de los sensores pueden ser observados por el nodo base si éste le envía una solicitud mediante un mensaje CoAP del tipo observe. De esta manera, periódicamente cada nodo sensor le envía la última medida de cada uno de sus sensores, siendo este período configurable en cada nodo de forma similar al período de muestreo de los sensores.

Cada medida enviada por un nodo contiene una marca de tiempo, indicando en qué momento se realizó la medición. Para esto, de la misma manera que al nodo se le configura el período de muestreo y el período en que envía las medidas, se le debe configurar la hora actual para que la marca de tiempo indique la hora real de la medición. Adicionalmente, se le puede solicitar su tabla de rutas y los períodos de tiempo en los cuales el MCU permanece en sus distintos estados de funcionamiento (modo bajo consumo por ejemplo). También permite configurar intervalos de temperatura y humedades en los cuales el usuario considera que el cultivo no está en peligro. Esto último se puede utilizar en trabajos futuros para disparar alarmas, las cuales no fueron implementadas. El capítulo 5 describe en detalle la implementación de la aplicación en el nodo.

La Figura 2.1 muestra un diagrama de bloques del módulo compuesto por el SoC y un amplificador de radio conectado a una antena. El SoC cuanta con una potencia de transmisión máxima de $7dBm$. Dicho amplificador permite alcanzar mayores distancias entre nodos, aumentando la potencia de transmisión del sistema de 7dBm ($5mW$) a 22dBm ($158mW$). La antena a utilizar es una antena omnidireccional, la cual recibe las señales procedentes de cualquier dirección con la misma ganancia. Debido a esto, no es necesario orientarla para que los nodos se puedan comunicar [1].

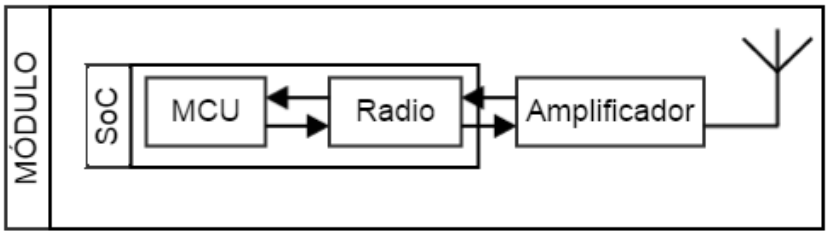


Figura 2.1: Estructura del módulo

Este módulo se integra con los sensores mencionados anteriormente mediante un PCB que se diseñó. En la Figura 2.2 se puede ver la estructura de un nodo, el cual está conformado por la integración del PCB con los sensores. Los circuitos auxiliares de los sensores cuentan con resistencias de pull up y capacitores especificados en la hoja de datos de cada sensor. Dentro del esquema del PCB se encuentran los switches para la alimentación de los sensores, el módulo, LEDs y botones de usuario, porta pilas, conectores de los sensores, así como conectores UART y JTAG para programación y pruebas. El diseño del nodo puede verse en mayor detalle en el capítulo 4.

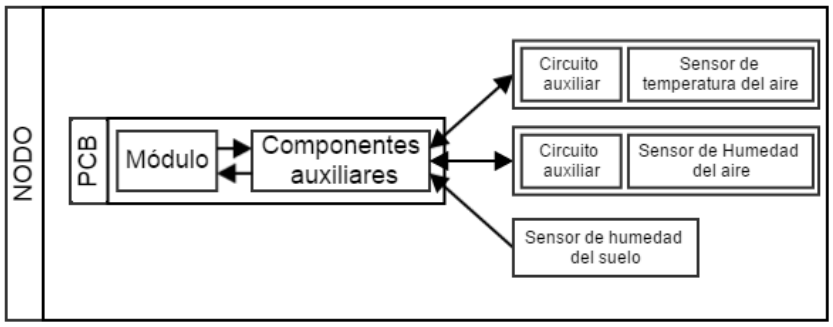


Figura 2.2: Estructura del nodo

Por último en la Figura 2.3 se muestra un esquema de la red, compuesta por los nodos y la PC comunicándose a través de un router. Los nodos intercambian mensajes RPL entre ellos, manteniendo así las tablas de ruta de cada nodo actualizada incluso cuando hay cambios en la topología de la red. Además, cada nodo se comunica con el router realizando comunicación multisalto, y de esta manera cualquier nodo que esté en la red podrá comunicarse con el router sin importar la distancia a la que se encuentre del mismo.

En lo que resta de este documento se describe en detalle cómo fue desarrollado cada punto del sistema. En el capítulo 4 se detalla la elección de los componentes

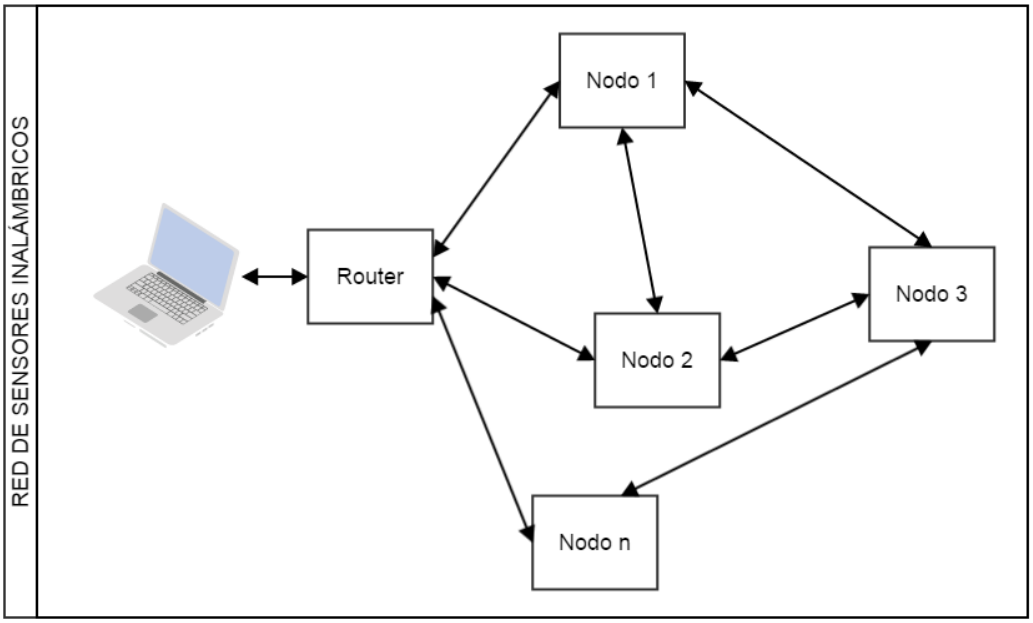


Figura 2.3: Red de Sensores Inalámbricos

a utilizar en el hardware, así como el diseño del PCB y las pruebas para testear su correcto funcionamiento. En el capítulo 5 se describe el desarrollo de la aplicación, explicando el diseño y las pruebas de la misma. El capítulo 6 explica las pruebas y modificación de las variables de los protocolos para lograr bajar el consumo de los nodos, mientras que en capítulo 7 se describen las pruebas de la red donde se utilizan los nodos y la aplicación diseñados durante el proyecto.

Capítulo 3

Estado del Arte

3.1. Introducción

En el presente capítulo se describen las RSI en general y sus aplicaciones al agro. Se introducen los temas de redes de bajo consumo, el sistema operativo Contiki OS y el *stack* de protocolos utilizados, incluyendo la evolución histórica y terminología de cada uno de los puntos anteriores.

3.2. Redes de Sensores Inalámbricas

Desde los años 90, las redes han revolucionado la forma en la que las personas y las organizaciones intercambian información y coordinan sus actividades. La disponibilidad de microsensores y comunicaciones inalámbricas han permitido desarrollar redes de sensores para un amplio rango de aplicaciones.

Cada nodo de la red consta de un dispositivo con MCU, sensores, transmisor y receptor, formando una red junto con otros nodos. Estos sistemas deben ser de bajo consumo y larga duración; tanto cuando operan como cuando permanecen a la espera.

Otra característica para las redes de sensores es la capacidad de procesamiento distribuido, el cual implica, que varios procesos se ejecuten en paralelo. Siendo la comunicación el principal consumidor de energía, un sistema distribuido implica que algunos sensores necesitan comunicarse a través de largas distancias, lo que se traducirá en mayor consumo. Por ello, es una buena idea el procesar localmente la mayor cantidad de información, para minimizar el número de bits transmitidos, utilizando la estructura de red de *multihop*[1] y no punto a punto. El consumo es

[1] *Multihop*: Consiste en un conjunto de nodos que colaboran para crear una red. Un nodo puede actuar como router y/o como usuario, donde los datos se mueven de la fuente al destino saltando entre nodos.

importante, ya que al ser nodos inalámbricos alimentados a baterías, necesitan administrar bien su energía para poder aumentar su vida útil.

Las redes de sensores tienen una amplia variedad de aplicaciones:

- Monitoreo de un hábitat (para determinar la población y comportamiento de animales y plantas).
- Monitoreo del medio ambiente, observación del suelo o agua.
- Mantenimiento de ciertas condiciones físicas (temperatura, luz).
- Control de parámetros en la agricultura.
- Detección de incendios, terremotos o inundaciones.
- Sensorización de edificios inteligentes.
- Control de tráfico.
- Asistencia militar o civil.
- Detección acústica, etc.

De hecho las RSI tienen el potencial de revolucionar los complejos sistemas de control u observación, tal y como hoy los entendemos [54].

3.2.1. Plataformas hardware

Tmote Sky [43] es una plataforma para aplicaciones en redes de sensores de muy bajo consumo y alta tasa recolección de datos. Lleva integrados tanto los sensores como la radio, antena y MCU, además puede ser fácilmente programado (Figura 3.1). Tmote Sky cuenta con sensores para medir luminosidad, y un sensor STH11 [56], el cual mide tanto humedad como temperatura del aire. Las operaciones de baja energía en el Tmote Sky son realizadas gracias al MCU MSP430F1611 [32]. Este procesador RISC de 16 bits consume muy poca energía tanto en el estado activo, como en modo de bajo consumo. Utiliza un conector USB para comunicarse con el PC si es que así se desea, y maneja una radio Chipcon CC2420 [31], la cual utiliza el estándar IEEE 802.15.4 que provee una comunicación inalámbrica fiable para las RSI.

Los avances tecnológicos han permitido desarrollar otras plataformas más eficientes y con mayor capacidad.

Una de ellas es OPENMOTE-CC2538 (Figura 3.2) el cual posee un MCU CC2538 [33], el cual es un SoC ya que integra un MCU y una radio. Este SoC contiene un procesador ARM Cortex-M3 de 32 bit, diseñado para aplicaciones de alto

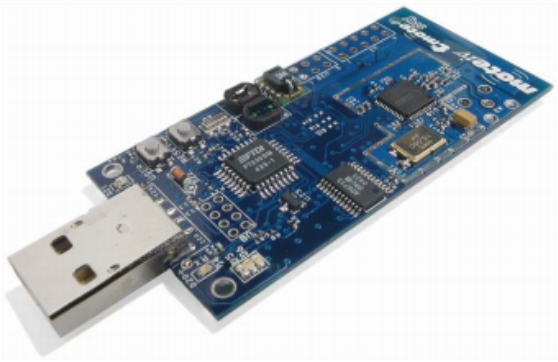


Figura 3.1: Tmote Sky

rendimiento y bajo consumo. La radio integrada en el CC2538 es un *transceiver RF* (transmisor/ receptor de radiofrecuencia) 2.4-GHz IEEE 802.15.4 compatible, con una sensibilidad de recepción de $-97dBm$, y potencia de transmisión configurable de hasta $7dBm$.

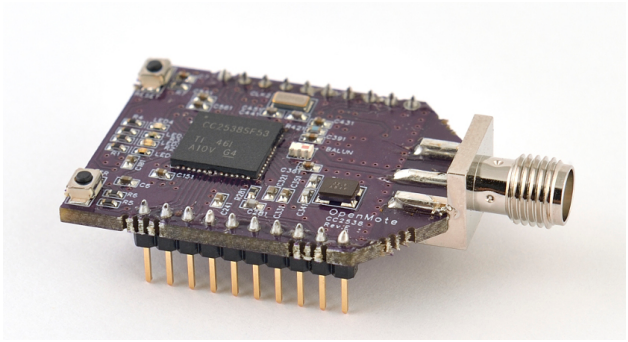
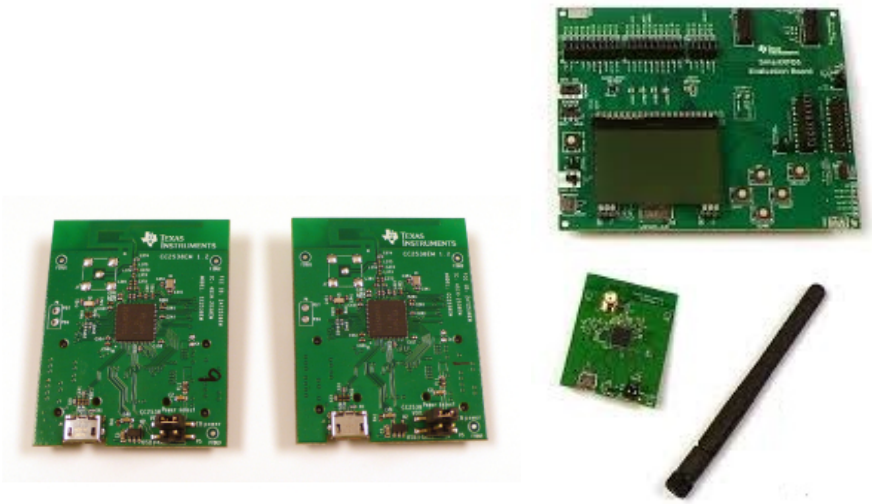


Figura 3.2: Open Mote

En el presente proyecto se utiliza el módulo de evaluación CC2538 Evaluation Modules (CC2538EM, Figura 3.3a) como herramienta de desarrollo. Este módulo se puede programar utilizando la plataforma CC2538DK [34], el cual es un kit de desarrollo para *Low Power RF SoC* de Texas Instrument. En la Figura 3.3b se pueden apreciar el kit de desarrollo nombrado anteriormente. En el capítulo 4 se profundiza sobre la utilización del SoC CC2538, así como el uso de la plataforma de desarrollo mencionada.



(a) CC2538 Evaluation Modules (b) CC2538DK

Figura 3.3: Módulos comerciales recientes

3.3. Sistema operativo Contiki

Contiki es un sistema operativo multihilos de código abierto para la Internet de las cosas, el cual permite conectar MCU con radio de bajo costo y consumo a Internet [3].

3.3.1. Principales características

Este sistema operativo fue diseñado para sistemas de bajos recursos y cuenta con un reducido *footprint* de memoria[2]. Es muy eficiente en cuanto al uso de memoria y cuenta con diferentes mecanismos para su manejo. Está diseñado para operar en sistemas de bajo consumo, por lo cual es ideal para las RSI, cuyos nodos tienen que operar durante años con baterías. Además, cuenta con mecanismos para estimar el consumo, entender dónde se genera el mayor gasto de energía y poder minimizarlo.

Contiki actualmente soporta tanto el estándar IPv4 como el IPv6 así como los estándares 6lowpan, RPL y CoAP especialmente desarrollados para las redes de bajo consumo y con pérdidas (LLN). El código fuente de Contiki cuenta con numerosos ejemplos que permiten probar el funcionamiento de los diferentes protocolos [26].

Las aplicaciones en Contiki se escriben en el lenguaje de programación C. Contiki cuenta con un simulador llamado Cooja (ver anexo E) que permite simular

[2]Footprint de memoria: cantidad de memoria que se utiliza durante la ejecución de un programa.

una red con múltiples nodos (sin necesidad de contar con el hardware). Este simulador es una herramienta muy potente que permite incluso crear redes formadas por nodos simulados y nodos reales. Incluye la opción de captura y análisis de los paquetes intercambiados entre los nodos y cuenta con modelos para la red que permiten simular pérdida de paquetes y la existencia de obstáculos entre los nodos.

Contiki está portado a una gran cantidad de dispositivos y sus desarrolladores, liderados por Adam Dunkels [21], cuentan con el apoyo de colaboradores de las principales compañías desarrolladoras de MCUs. Esto permite que el sistema operativo esté en constante desarrollo y sea fácil portarlo a nuevas plataformas.

En Contiki se ejecutan múltiples hilos en paralelo. Básicamente, estos hilos esperan determinadas condiciones para activarse, luego ejecutan sus tareas y posteriormente vuelven a quedar a la espera de otro evento que los active. Los eventos son guardados en una cola circular y se procesan en orden cronológico, del más antiguo al más reciente. Cada evento generado tiene un hilo asociado para despertar , al cual le puede enviar datos cuando lo despierta.

3.3.2. Procesos

El código en Contiki puede ser ejecutado en dos contextos: cooperativo o preemptivo. El código cooperativo se ejecuta secuencialmente con respecto a otro código cooperativo y tiene que ejecutarse en su totalidad antes de que otro código cooperativo pueda ejecutarse. El código preemptivo detiene temporalmente al código cooperativo. Cuando esto sucede el código cooperativo no puede retomar su ejecución hasta que no se haya ejecutado completamente el código preemptivo. Los procesos de Contiki se ejecutan en el contexto cooperativo, mientras que las interrupciones y los *timers* se ejecutan en el contexto preemptivo.

Todo código ejecutable en Contiki cuenta con procesos que comienzan típicamente cuando el sistema se inicia o cuando un módulo que contiene un proceso es cargado al sistema. Los procesos se ejecutan cuando, por ejemplo, un *timer* termina su cuenta o sucede un evento externo.

Un proceso de Contiki está compuesto por dos partes: un bloque de control y un *thread*. El bloque de control se almacena en la RAM y contiene información de runtime del proceso como su nombre, estado (activado, desactivado, llamado) y un puntero a su thread. El *thread* se almacena en la memoria de programa (típicamente no volátil), es utilizado por el usuario y contiene el código de ejecución del proceso. Está formado por un *protothread* que es invocado por el *process scheduler*[3]. En la siguiente subsección se explica el concepto de *protothread*.

[3]Scheduler: es el encargado de repartir el tiempo disponible de un microprocesador entre todos los procesos que están disponibles para su ejecución.

3.3.3. Protothreads

Los *protothread* son funciones capaces de esperar por eventos. Es una forma de estructurar el código que le permite al sistema ejecutar otras actividades mientras el código está esperando que ocurra un evento. La noción de evento se describe en la siguiente subsección.

Cuando un evento inicializa un proceso, se ejecuta su código desde el principio y el proceso queda en estado *llamado*. Cuando el *protothread* deba esperar por un evento, le devuelve el control al *process scheduler* y su proceso pasa al estado *activo*. Cuando un evento llame al proceso se ejecutará si el evento es el esperado y en caso afirmativo se ejecutará el código del *protothread* desde donde se había quedado anteriormente.

3.3.4. Eventos

En Contiki los procesos son ejecutados cuando reciben un evento. Los eventos tienen asociado el proceso que van a despertar y puede enviarle datos al mismo. Hay dos tipos de eventos: asíncronos y síncronos.

Cuando ocurre un evento asíncrono, éste es encolado en una cola de eventos del Kernel y será entregado al proceso al llegar tu turno. Son eventos que pueden ser enviados o posteados a un proceso o a todos.

Cuando un evento síncrono es posteado el evento se entrega inmediatamente al proceso. Estos eventos solo pueden tener como destinatario un único proceso.

3.3.5. Timers

Contiki provee librerías de *timers* que pueden ser usados tanto por las aplicaciones creadas por usuarios como por el propio sistema operativo. Estas bibliotecas contienen funciones para chequear si el tiempo del timer expiró, despertar el sistema de un modo de bajo consumo en un momento determinado o agendar tareas en tiempo real.

Los diferentes tipos de *timers* implementados en Contiki son:

- *Timer*: Se utilizan para chequear si transcurrió un cierto período de tiempo. Cuando expira no toma ninguna iniciativa, por lo cual hay que consultarlo. Utiliza los *ticks* del reloj del sistema como unidad de medida, por lo que sirven para medir períodos cortos de tiempo (tiene gran resolución).

- *sTimer*: Es similar al *Timer*, a diferencia que utiliza segundos como medida, por lo que pueden medir grandes cantidades de tiempo a cambio de tener menor resolución.

- *eTimer*: Permite generar eventos temporizados, utilizando *clock_time*() para obtener la hora del sistema. El *eTimer* se implementa como un proceso, el cual postea al proceso que lo seteó el evento $PROCESS\_EVENT\_TIMER$ cuando vence su tiempo.

- *cTimer*: Este timer llaman una cierta función, función de *callback*, cuando expiran, utilizando *clock_time*() para obtener la hora del sistema.

- *rTimer*: Permite agendar y ejecutar tareas de tiempo real, utilizando su propio módulo de reloj para permitir mayor resolución. Las tareas se ejecutan en modo preemptivo. Generalmente se usa para llamar un *process_poll*(...)

3.3.6. Energest

Energest es una herramienta que brinda Contiki, la cual permite saber el tiempo durante el cual el MCU permanece en distintos estados. El mismo permite saber los intervalos siguientes:

- CPU: El tiempo acumulado en el que el MCU estuvo encendido en modo activo.
- LPM: Intervalo de tiempo durante el cual el MCU estuvo encendido en modo bajo consumo sumado al tiempo durante el cual atendió rutinas de atención a interrupciones.
- LISTEN: Intervalo de tiempo durante el cual la radio del MCU estuvo encendida escuchando el canal.
- TRANSMIT: Intervalo de tiempo durante el cual la radio del MCU estuvo encendida enviando paquetes por el canal, desde que se encendió el MCU hasta el momento en el cual se lee la variable.
- IRQ: Intervalo de tiempo durante el cual el MCU ejecutó rutinas de atención a las interrupciones.

Los parámetros anteriores se miden en *ticks* del *rtimer*, los cuales se convierten a segundos dividiendo por la variable de Contiki $RTIMER\_ARCH\_SECOND$.

Esta herramienta fue fundamental en la fase de optimización de protocolos, ya que como uno de los objetivos es crear un nodo autónomo, se estimó el consumo del nodo usando Energest, tal como se describe en el capítulo 6. Como se observó que

el componente del sistema de mayor consumo es la radio, se procedió a analizar cómo variaba el consumo al variar los parámetros de los protocolos de red. Se incluyó la herramienta Energest en la aplicación desarrollada en el presente proyecto para poder saber cuánto tiempo estaba transmitiendo y recibiendo el nodo, y de esta manera modificar las variables de los protocolos de comunicación para mejorar dichos tiempos y aumentar el tiempo de vida útil.

3.3.7. Directorios de Contiki

Los archivos fuente de Contiki están organizados en directorios. Algunos de ellos son:

- contiki/core

  Código fuente de sistema

  net: MAC, RDC, Rime, IP

  sys: Processes

  cfs: File System

- contiki/examples

  Contiene varios ejemplos para las diferentes plataformas.

- contiki/apps

  System apps (telnet, shell, deluge, servreghack, tunslip)

- contiki/platform

  Código específico para de cada plataforma.

- contiki/cpu

  Código específico para cada CPU (un subdirectorio por CPU).

- contiki/tools

  Contiene herramientas como por ejemplo el simulador Cooja mencionado anteriormente.

3.4. Capas de la red

Típicamente las implementaciones de las redes TCP/IP requieren mucha memoria y paquetes de gran tamaño debido a sus encabezados. Esto hace que no sean útiles para redes de bajos recursos [8]. En las redes tradicionales, como las redes de computadoras, generalmente los nodos están conectados a la red eléctrica y los enlaces son estables, sin embargo en las redes de bajos recursos los nodos suelen

tener una alimentación limitada. Este hecho, sumado a un entorno con mucho ruido, suele generar enlaces muy inestables. Es por esto que los requerimientos para una red de sensores inalámbricos son muy distintos de las redes tradicionales [20].

El *stack* de protocolos orientado a las redes de sensores inalámbricos fue desarrollado para ser más liviano y adaptarse a las dificultades presentes en una red con recursos limitados y muchas pérdidas. Un modelo de capas aplicable a este tipo de redes es el que se presenta en la Tabla 3.1 [9].

Capa de Red	**Protocolo**
APLICACIÓN	CoAP
TRANSPORTE	UDP
RED	RPL IPv6
ADAPTACIÓN	6LoWPAN
ENLACE	IEEE 802.15.4 MAC
RDC	ContikiMAC
FÍSICA	IEEE 802.15.4 PHY

Tabla 3.1: *Stack* de protocolos

Para saber cuáles son los drivers utilizados por defecto en cada capa en Contiki según las diferentes plataformas hay que referirse al archivo:

contiki/platform/<plataforma>/contiki-conf.h

3.4.1. Capa Física

La capa física suministra el servicio a la capa de enlace. Codifica los datos de la trama de enlace en un patrón de unos y ceros para transmitir a través del medio. En esta capa se especifican las características de tensión, frecuencia, tiempos, etc. Existen diferentes estándares para definir la capa física en las redes de bajos recursos con muchas pérdidas. Ejemplos de estos son IEEE 802.15.4 y *Power Line Communication* (PLC). En el presente proyecto se trabajó con el estándar IEEE 802.15.4 debido a que se caracteriza por facilitar la comunicación inalámbrica con baja potencia.

IEEE 802.15.4 es un estándar especificado por el Institute of Electrical and Electronics Engineers (IEEE) para las redes de área personal (Personal Area Networks - PAN) cuya tasa de transferencia varía entre $20kbps$ y $250kbps$ dependiendo de la frecuencia. En IEEE 802.15.4 se usa el rango de frecuencia 2400-2483,5 MHz. Esta banda de $2400MHz$ está dividida en 16 canales y pueden solaparse con señales de Wi-Fi, haciendo que aumente la posibilidad de tener interferencia. El estándar

utiliza modulación por desplazamiento de fase binaria (BPSK) o en cuadratura (QPSK).

3.4.2. Radio Duty Cycle

La capa Radio Duty Cycle (RDC) es la responsable de manejar el ciclo de trabajo de la radio. La radio tiene un consumo elevado tanto en transmisión como en recepción, por lo que mantenerla apagada el mayor tiempo posible es indispensable para obtener un bajo consumo. Los mecanismos con los que cuenta Contiki para el manejo del ciclo de la radio son:

- NullRDC
- ContikiMAC
- XMAC
- LPP

En este proyecto se utilizó ContikiMAC, el cual fue desarrollado y presentado por los creadores de Contiki como un mecanismo eficiente para el manejo del ciclo de la radio en redes de sensores que utilizan el sistema operativo Contiki. Al hacer, en la mayoría de los casos, que el mayor consumo de la comunicación esté en el transmisor, permite que los nodos solo despierten su radio periódicamente para escuchar el canal. El emisor transmite varias veces la misma trama a lo largo de un ciclo completo o hasta recibir un reconocimiento del receptor (ACK). En este mecanismo la sincronización no es necesaria pero es de suma utilidad para hacer que el emisor comience a transmitir justo antes del momento en el que el receptor se enciende. En las Figuras 3.4 se muestra este funcionamiento, donde transmisiones Broadcast son enviadas con paquetes de datos repetidos durante un ciclo de escucha o hasta que se recibe un reconocimiento. Los mensajes broadcast son paquetes de datos los cuales tienen como dirección destino la dirección MAC FF.FF.FF.FF.FF.FF. De esta manera el mensaje es enviado a todos los dispositivos de la red.

Resumidamente, ContikiMAC hace que cada nodo se despierte periódicamente cada T_{cycle}. Al despertarse el nodo hace dos sensados de canal (CCA, clear channel assessment) separados un tiempo T_c (se requieren dos CCA en caso de que el primero ocurra entre dos tramas). Si no detecta señal se vuelve a estado inactivo. Si detecta señal mantiene la radio encendida hasta que la trama completa haya sido recibida y envía un reconocimiento al emisor. La frecuencia con la cual se despierta el nodo se conoce como tasa de sensado del canal (*Channel Check Rate* - CCR) que es $1/T_{cycle}$. En el presente proyecto se modificó el CCR para disminuir el consumo sistema. De esta manera al disminuir el CCR el periodo de tiempo en el que el MCU permanece en estado inactivo aumenta y por ende el consumo

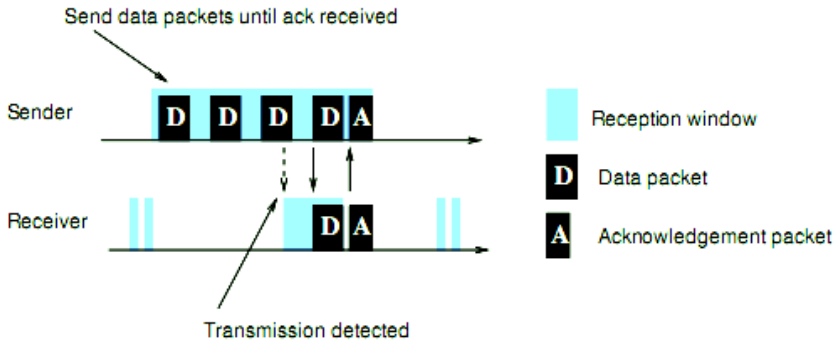


Figura 3.4: Ciclo de ContikiMAC

disminuye.

En la Figura 3.5 se pueden observar los tiempos requeridos, siendo los mismos:

- ti: intervalo entre paquetes transmitidos.
- tr: tiempo requerido para obtener una RSSI (indicador de intensidad de señal recibida) estable.
- CCA: sensados del canal.
- tc: intervalo entre CCA.
- ta: tiempo entre que se recibe un paquete y se manda un ACK.
- td: tiempo requerido para hacer una detección exitosa de un ACK.
- ts: tiempo de envío de un paquete de datos

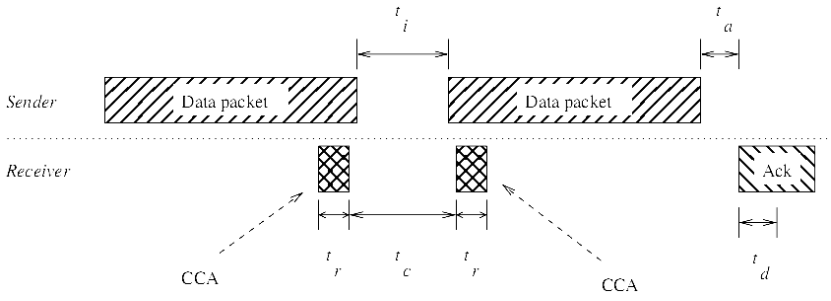


Figura 3.5: Tiempos de ContikiMAC

Como consecuencia de este funcionamiento se debe cumplir la siguiente restricción para los tiempos mencionados: [4]

$$T_a < T_i < T_c < T_c + 2T_r < T_s$$

Esto significa que el tiempo necesario para detectar un reconocimiento tiene que ser menor que el intervalo entre las tramas. Este intervalo además tiene que ser menor que el intervalo entre los CCA. De lo contrario, dos CCA no garantizarían que se detecte la secuencia de tramas. Los últimos dos términos implican que una trama debe ser mayor que el tiempo que toma hacer dos CCA.

En cuanto a los valores numéricos, según el estándar IEEE 802.15.4, T_a es $0{,}352ms$, T_r depende del hardware, mientras que T_c y T_i pueden ser determinados por el usuario siempre que se cumpla con la desigualdad anterior.

Los valores por defecto en Contiki son:

$$T_i = 0.4 \text{ ms}$$
$$T_c = 0.5 \text{ ms}$$
$$T_r = 0.884 \text{ ms}$$

Si bien la elección de T_{cycle} no tiene restricciones y puede ser elegido libremente, a partir de simulaciones se puede concluir que $0{,}250ms$ es un valor óptimo para numerosas aplicaciones de redes de sensores, incluida RSItrust.

Para hacer aún más eficiente el mecanismo, ContikiMAC cuenta con Phase-lock, donde cada nodo guarda información sobre la fase de sus vecinos para poder enviar las tramas justo antes de que ellos se despierten.

Si bien con ContikiMAC se logra mantener la radio apagada el 99 % del tiempo, el consumo energético depende fuertemente del hardware ya que depende de la capacidad del sensor de encender y apagar la radio y el MCU, además de la carga de la red.

La implementación de los diferentes protocolos mencionados en esta sección se encuentran en el directorio:

contiki/core/net/mac

3.4.3. Capa de Enlace

La capa de enlace es la responsable del intercambio de datos sin errores entre dos máquinas que están conectadas directamente mediante un cable. Esta capa recibe peticiones de la capa de red y utiliza servicios de la capa física. Además, es la encargada del control de flujo y el control de errores. Durante el proyecto se utiliza el protocolo MAC (control de acceso al medio), el cual se especifica en el estándar IEEE 802.15.4. El tamaño máximo de trama es de $128bytes$ y se aceptan direcciones MAC largas (de $64bits$) y cortas (de $16bits$). La capa MAC para realizar el control de acceso al medio utiliza CSMA/CA no persistente, es decir, si el medio está libre transmite, si no, espera un tiempo aleatorio para hacerlo.

3.4.4. Capa de Adaptación

La capa de adaptación es la encargada de adaptar la transmisión de paquetes IPv6 sobre IEEE 802.15.4. También realiza la compresión de encabezados, la fragmentación de los paquetes e informa a las capas superiores de las retransmisiones que necesita la capa MAC. A nivel de capa de adaptación en RSItrust se utilizó el estándar 6LoWPAN[4], por lo cual no se envía lo que el receptor ya sabe o puede calcular. Las direcciones IPv6 se pueden construir automáticamente a partir de las dirección de capa MAC, estando el *stack* IPv6 aislado de los detalles del nivel de capa física [9].

Una gran dificultad de usar IPv6 sobre IEEE 802.15.4 es que 802.15.4 tiene una unidad máxima de transferencia (MTU) de $127bytes$, mientras que la MTU de IPv6 es de $1280bytes$. Como consecuencia de esto, los paquetes IPv6 deben ser fragmentados antes de ser enviados. Otro problema es que, con un MTU tan reducido, los paquetes IPv6 encapsulados (con direcciones de $128bits$) requieren una gran parte de los $127bytes$ disponibles para su encabezado. Un solo encabezado IPv6 tiene $40bytes$ y si se usa TCP éste agrega otros $20bytes$, por lo que el *overhead* en una comunicación TCP/IP agrega $60bytes$. Además, el encabezado de 802.15.4 puede agregar otros 39 bytes, dejando solo $28bytes$ libres para los datos [20].

Para buscar una solución a estos problemas un grupo del IETF (Internet Engineering Task Force) llamado 6LowPAN, desarrolló una capa de adaptación entre IPv6 e IEEE 802.15.4. Su objetivo es hacer que los enlaces de 802.15.4 cumplan con todos los requerimientos de IPv6 [27].

Si el payload entra en un *frame* de 802.15.4, el mensaje no es fragmentado y el encapsulado de LoWPAN no tendrá el encabezado de fragmentación. Si el datagrama no entra en un *frame* IEEE 802.15.4, el mismo será fragmentado. Como el *offset* de los fragmentos sólo puede expresarse en múltiplos de 8 bytes, todos los fragmentos menos el último deben ser de 8 bytes. Mientras el primer fragmento debe contener el encabezado de fragmentación que se muestra en la Figura 3.6, los restantes deben contener el encabezado de la Figura 3.7. El valor de $datagram\_size$ debe ser el mismo para todos los fragmentos de un paquete IP. Para IPv6 este valor es 40 octetos, ya que es el tamaño sin comprimir de un encabezado IPv6. $Datagram\_tag$ debe ser IPv6 en este caso y $datagram\_offset$ el cual no está presente en el primer fragmento, indica el *offset* respecto al primer fragmento. Como no se espera que los fragmentos lleguen en orden cualquier paquete que no sea del paquete que se está armando será descartado [50].

En Contiki la implementación de 6LowPAN se llama *sicslowpan* y se encuentra en los archivos:

contiki/core/net/ipv6/sicslowpan.h
contiki/core/net/ipv6/sicslowpan.c

[4]IPv6 over Low power wireless Personal Area Networks [44]

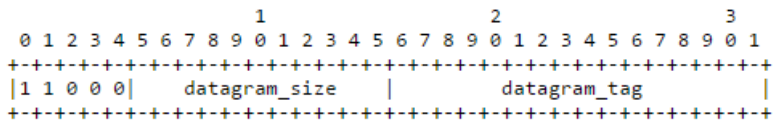


Figura 3.6: Encabezado del primer fragmento

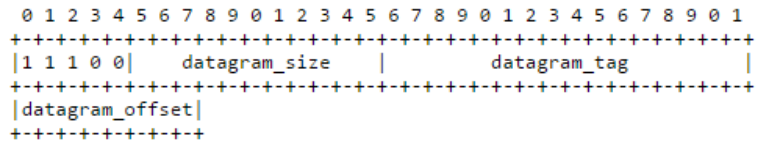


Figura 3.7: Encabezado de los subsiguientes fragmentos

3.4.5. Capa de Red

La capa de red hace posible la comunicación entre nodos que no son adyacentes. Además, envía los paquetes con métrica de ruteo ajustable en forma dinámica, realizando también, el encaminamiento y el control de congestión [9].

IPv6 es una actualización de IPv4 y su principal característica es que el tamaño de las direcciones IP aumentaron de 32 a 128 bits. Esta mejora posibilita crear $3{,}4 * 10^{38}$ direcciones únicas. Las direcciones IP permiten identificar de forma única a los dispositivos dentro de una red que utilice el protocolo IP.

El uso de IPv4 se extendió considerablemente gracias al uso del *Network Address Translation* (NAT), un mecanismo que permite que varios hosts a través de un router NAT compartan una única dirección de red global. La desventaja de esto, es que no se accede directamente a los hosts detrás de los routers NAT. En comunicaciones cliente-servidor esto no es un problema para los clientes ya que es el cliente el que inicializa la comunicación y es el servidor el que debe estar disponible en Internet. En el anexo D se describe en mayor detalle este protocolo.

En cuanto al ruteo, en Contiki se implementa el *Routing Protocol for Low-power and Lossy Networks* (RPL) desarrollado por el grupo *Routing over Low-power and Lossy networks* (ROLL) perteneciente a la IETF.

RPL conecta los nodos a través de un árbol denominado *Destination Oriented Directed Acyclic Graph* (DODAG). El nodo que se encuentra en la raíz del árbol de comunicación se llama Root, y se considera tráfico hacia arriba al que se dirige hacia el root, mientras que el que es desde el root se dice que es hacia abajo. La comunicación en un DODAG se puede dividir en 4 modos de operación (MOP) diferentes y es el root el que define qué MOP se usa:

1. Nodos no mantienen rutas hacia abajo.
2. Nodos no almacenan información de la ruta
3. Nodos almacenan información de la ruta pero no soporta multicast
4. Nodos almacenan información de la ruta y soporta *multicast*

En el modo 1 todos los nodos envían información al root, a nodos en el camino hacia el root o a otra red conectada al root. El modo 2 permite también el tráfico hacia abajo pero requiere que el ruteo lo haga el root. Esto quiere decir que el root tiene que especificar el camino completo hacia el nodo destino. Los nodos que no son root no almacenan información de la ruta y el tráfico en el DODAG siempre tiene que pasar por el root, ya que es el único capaz de rutear. Los modos 3 y 4 superan las restricciones anteriores haciendo que todos los nodos almacenen información de las rutas. Esto quiere decir que el tráfico puede ir hacia arriba o hacia abajo. La diferencia entre los modos 3 y 4 es que el segundo soporta multicast.

La red se forma intercambiando diferentes tipos de mensajes entre los nodos:

- DIO (DODAG Information Object): contienen información de la configuración del DODAG y sirven para crear, mantener y descubrir el DODAG.
- DAO (Destination Advertisement Object): se usan para propagar información hacia arriba y completar las tablas de ruteo de los nodos padres[5] .
- DIS (DODAG Information Solicitation Message): sirven para solicitar mensajes DIO a los nodos vecinos.

Además de los mensajes anteriores, para formar las tablas de ruteo se usa el Rank. Este parámetro sirve para evaluar qué tan buenos son los enlaces y evaluar sus potenciales padres, para poder determinar cuál es el mejor camino al root. El rank aumenta hacia abajo y cuanto más pequeño sea el rank mejor es el camino para llegar al root, siendo 256 el menor valor del rank (es el que tiene el root). Surge así el concepto de padre preferido, que es el nodo vecino con el cual se consigue el mejor camino para llegar al root.

El protocolo RPL utiliza un algoritmo llamado TRICKLE para la difusión de los mensajes DIO utilizado para mantener actualizadas las tablas rutas en todos los nodos de la red. El mismo chequea si la versión del mensaje que se recibe es más vieja, nueva o igual a la que se tiene registrada. Si la versión es más vieja o igual la recepción se considera consistente. Cabe destacar que el funcionamiento del algoritmo TRICKLE aquí descrito, es de carácter general, aplicable a otros usos que sea necesario, dependiendo de cada aplicación en particular. Este algoritmo evita el reenvío de paquetes con información repetida cuando la red está estable y

[5]Padre: Nodo seleccionado por un nodo, para enviar los mensajes hacia el nodo raíz con el menor costo posible.

todos sus nodos tienen la misma información.

TRICKLE cuenta con un timer definido dentro de un intervalo de tiempo I, un parámetro k de redundancia y un contador C. C incrementa cada vez que se tiene una recepción consistente, es decir le llega al nodo un paquete sin información nueva. Cada nodo envía mensajes TRICKLE con sus versiones mientras $C < k$. Dichos mensajes se envían en un tiempo aleatorio dentro del intervalo I, evitando que todos los nodos respondan a la vez.

Lo primero que realiza el algoritmo es definir los intervalos de tiempo en el cual funciona el TRICKLE timer. Se toma el intervalo de tiempo i, donde el intervalo de tiempo I pertenece a [i/2, i]. El I se incrementa al doble cada vez que expira el timer, es decir, el nodo no recibió información inconsistente durante el tiempo I del timer, y de esta manera los nodos mandan su información cada vez más distanciada en el tiempo. El intervalo I aumenta hasta llegar a I_{max}. Cuando todos tienen la misma versión queda estable la red sin envíos de paquetes Trickle ($C > k$), hasta que expira el Timer I. En este caso solo se resetea el contador C pero no se minimiza el intervalo, es decir vuelven a mandar mensajes TRICKLE hasta que $C > k$ pero con el mayor distanciamiento posible entre paquetes. Por el contrario cuando el nodo recibe un mensaje inconsistente (nueva versión), el nodo actualiza la información y se resetea el intervalo haciendo $I = I_{min}$, comenzando de nuevo el proceso [46].

El mecanismo que utiliza RPL para controlar el envío de mensajes DIOs es el TRICKLE, el cual le indica cuándo mandar mensajes *multicast* DIOs. Ciertos eventos son tratados como inconsistencias en la red, por ejemplo, cuando un nodo detecta un loop en la red. Los loops son detectados usando bits en el encabezado IPv6. El intervalo del TRICKLE timer se incrementa mientras la red se estabiliza lo que resulta en menos cantidad de envíos de mensajes DIO. Cuando una inconsistencia es detectada, se resetea el TRICKLE timer, enviando mensajes DIOs más seguido [45].

La creación de la topología de un DODAG se realiza a partir de la información contenida en la función objetivo OF. En la OF se define la métrica del enlace y cómo ésta es utilizada para determinar el *rank* de los diferentes nodos. En Contiki están definidas la OF0 y la OF MRHOF. La OF0 fue creada como una OF básica que no requiere realizar medidas, simplemente minimiza la cantidad de saltos. La MRHOF calcula el *rank* del nodo basándose en las métricas de cantidad de saltos, latencia o ETX. Por defecto en Contiki se utiliza la MRHOF con métrica de ETX.

En Contiki, los módulos principales en la implementación de RPL son:

contiki/core/net/rpl/rpl-conf.c
contiki/core/net/rpl/rpl-dag-root.c
contiki/core/net/rpl/rpl-dag.c
contiki/core/net/rpl/rpl-of0.c

contiki/core/net/rpl/rpl-mrhof.c

3.4.6. Capa de Transporte

La capa de transporte es la encargada de la calidad del servicio (fiabilidad, control de flujo y corrección de errores). En TCP/IP se cuenta con dos posibles protocolos, Transmission Control Protocol (TCP) y User Datagram Protocol (UDP). TCP proporciona una conexión confiable, ordenada, con confirmación y chequeo de errores. Por otro lado UDP es un protocolo basado en la transmisión de datagramas, permitiendo el intercambio de datagramas por la red sin la necesidad de establecer previamente una conexión. Tampoco cuenta con confirmación ni control de flujo como TCP. Los datagramas pueden llegar al receptor desordenados y no se tiene certeza si fueron recibidos ya que tampoco cuenta con confirmación. Al utilizar UDP cualquier tipo de garantías para la transmisión de datos deben ser implementadas en capas superiores. Por otro lado, TCP garantiza que los datos son entregados al destinatario y en el mismo orden que fueron enviados. Además, proporciona control de flujo y de congestión de la red, siendo por estos y otros motivos, que TCP da soporte a muchas de las aplicaciones y protocolos de aplicación más populares de Internet. [60]

A nivel de capa de transporte, en la red implementada se utilizó el protocolo UDP. La principal desventaja de TCP y las LLN, es que interpreta las pérdidas de segmentos como congestión, haciendo una reducción innecesaria de la velocidad de transferencia. Otro punto a tener en cuenta es que el encabezado de TCP es de 20 bytes, mientras que en UDP es de 8 bytes (solo envía puertos de origen y destino), generando UDP paquetes más pequeños y con menor riesgo de fragmentación. Finalmente, TCP provee confiabilidad a la transmisión mediante el envío de reconocimientos lo que, en la mayoría de las implementaciones, genera un tráfico extra en sentido opuesto al de los datos, provocando congestión y colisiones en las LLN.

En Contiki la implementación de TCP y UDP se encuentra en:

contiki/core/net/ip/tcpip.h
contiki/core/net/ip/tcpip.c
contiki/core/net/ip/udp-socket.h
contiki/core/net/ip/udp-socket.c
contiki/core/net/ip/tcpip-socket.h
contiki/core/net/ip/tcpip-socket.c

3.4.7. Capa de Aplicación

La capa de aplicación es la encargada de manejar protocolos de alto nivel, los cuales tienen en cuenta la representación, codificación y control de diálogo. El protocolo más utilizado es el protocolo de transferencia de hipertexto (HTTP). Sin embargo este protocolo en las redes con recursos limitados, presenta además

del problema de grandes encabezados, la falta de soporte para mensajes *multicast* y mecanismos de suscripción. Para superar estas carencias la IETF desarrolló el *Constrained Application Protocol* (CoAP) [2].

El protocolo CoAP está pensado para comunicar dispositivos de bajos recursos que necesitan ser controlados o monitoreados a través de Internet. CoAP fue diseñado para poder traducir fácilmente los paquetes a HTTP, para ser integrado con la web.

Como se mencionó anteriormente, se utiliza UDP en capa de transporte ya que es muy simple y solo agrega un encabezado con puertos de origen y destino a diferencia de TCP que agrega *handshakes*, control de flujo y manejo de errores. UDP no provee una comunicación libre de errores pero, de ser necesario, CoAP proporciona una mecanismo de retransmisión. Los mensajes de CoAP, contienen el campo confirmable que solicita que se retransmitan los paquetes corruptos, logrando obtener de esta forma, una comunicación confiable usando UDP.

Algunas de las principales características de CoAP son:

- Protocolo RESTful[6] con mapeo sencillo a HTTP
- Bajo *overhead* en el encabezado
- Baja complejidad en el parseo o análisis sintáctico
- Soporte para descubrir recursos[7] usando un servicio conocido
- Suscripción simple a recursos

Como el servidor no tiene memoria, y por ende no recuerda ninguna de las solicitudes previas, utilizando CoAP los recursos son accedidos y manipulados usando protocolos de capa de aplicación basados en diálogos cliente-servidor solicitud-respuesta. Este tipo de comunicación forma el modelo REST. Dicho modelo no especifica el protocolo a usar pero solicita que el mismo tenga comandos POST, GET, PUT, DELETE. Las aplicaciones que usan REST se les conoce como RESTful, y en el modelo REST aplicado a IoT o M2M los recursos son identificados por un identificador de recursos uniforme (URI), el cual es una cadena de caracteres que identifica los recursos de una red de forma unívoca.

Como se mencionó, CoAP es un protocolo RESTful, por lo cual utiliza los comandos REST para la comunicación [59].

A continuación se detallan los comandos del protocolo:

[6]RESTful: Se le llama a las aplicaciones que usan el modelo de comunicación REST, siendo el mismo un modelo en donde los recursos son accedidos y manipulados usando protocolos de capa de aplicación basados en diálogos cliente-servidor solicitud-respuesta.

[7]Recurso: Cualquier elemento con identidad [13].

- GET: Comando con el cual se recupera una representación de la información que corresponde al recurso identificado por la URI.
- POST: Envía una acción al recurso. Esta acción puede ser crear, modificar o eliminar un recurso.
- PUT: Solicita que el recurso identificado por la URI se actualice o se cree.
- DELETE: Elimina el recurso identificado por la URI.

Los comando anteriores son una pieza clave para la comunicación de los nodos sensores, realizando la configuración de recursos y solicitud de información con estos comandos.

En Contiki, la implementación de CoAP se llama *erbium* y fue implementada como una aplicación, por lo tanto, sus archivos fuente se encuentran en el directorio apps/er-coap [17]. Esta aplicación utiliza archivos de una aplicación denominada rest-engine que implementa un motor de REST. Por más detalles del funcionamiento de CoAP referirse al anexo C.

Capítulo 4

Hardware

4.1. Introducción

Los nodos que componen la red están basados en un MCU el cual permite la ejecución de una aplicación programada en Contiki OS, y los protocolos de comunicación definidos en este proyecto. Dado que se debe medir la humedad y temperatura del aire, y la humedad del suelo, es necesario integrar al MCU sensores que realicen estas mediciones. Para que el nodo se integre a una red, debe contar con una radio y antena. De esta manera se debe conformar un nodo que integre en un PCB un MCU, radio, sensores y antena.

Para cumplir dicho objetivo se utilizó como base un módulo que integra un MCU y radio embebidos en un SoC, un amplificador de radio y antena. Se diseñó un PCB para integrar dicho módulo con circuitos auxiliares para los sensores, proveer alimentación, elementos que permitan la programación del MCU y elementos para poder realizar pruebas de funcionamiento. Adicionalmente se agregaron elementos extra previendo una reutilización del diseño.

Para el diseño se puso especial consideración en el bajo consumo del sistema, se buscó versatilidad en el diseño, obteniendo así un producto reutilizable. Fue necesario prever la utilización de elementos que permitieran realizar la programación del MCU y su depuración. El PCB cuenta además con una etapa de alimentación al módulo y los sensores, los cuales se integran mediante conectores, permitiendo así que sean fácilmente reemplazables en caso de que se dañen.

Otro punto a tener en cuenta a la hora del diseño, fue el hecho de que el nodo debería estar a la intemperie, por lo tanto se consideró la utilización de una caja estanca para la protección del mismo.

Para la fabricación de los nodos se comenzó por una selección de los componentes a utilizar en el circuito y los sensores, siguiendo por el diseño del PCB para luego realizar un prototipo, sobre el cual se realizaron pruebas de funcionamiento

del hardware.

Se encontraron errores de diseño durante el test del prototipo, estos errores fueron identificados y enmendados, realizando luego un nuevo diseño del PCB corregido.

Este capítulo describe el proceso de construcción del nodo. Comenzando por el diseño, fabricación de prototipo y prueba del hardware. Muestra cómo reparar los errores de diseño detectados, y finalmente, se presenta el diseño final corregido.

4.2. Diseño de hardware

Para la construcción del nodo se buscó un diseño versátil, reutilizable y apuntando al bajo consumo. El primer paso del diseño fue la selección del MCU, radio y sensores a utilizar, para luego pasar a la etapa de diseño de un PCB que cumpla con las características deseadas, así como la integración de los sensores y elementos necesarios para la programación.

4.2.1. Selección de microcontrolador y radio

La primera decisión a tomar fue la elección del MCU y la radio a utilizar, los cuales fueron elegidos por los tutores del proyecto. El MCU seleccionado fue el CC2538 de Texas Instruments, el cual tiene una buena capacidad de memoria, tanto de RAM ($32kB$) como de memoria Flash ($512kB$). Este microcontrolador tiene incorporado un transceiver RF (transmisor/ receptor de radio frecuencia) 2.4-GHz IEEE 802.15.4 compatible. Entre sus cualidades, se destacan sus eficientes modos de bajo consumo, ya que se puede lograr un consumo de tan solo $1{,}7\mu A$ reteniendo $16kB$ de su memoria RAM cuando opera en estos modos. Otra característica importante de este dispositivo es su capacidad de cálculo, soportando una frecuencia de reloj de hasta $32MHz$.

Para el desarrollo de la aplicación y pruebas de hardware se utilizó el kit de desarrollo CC2538DK de Texas Instruments, el cual permite desarrollar aplicaciones cuando no se cuenta con el hardware definitivo a utilizar. Este kit está compuesto por dos plataformas de desarrollo SmartRF06, y dos módulos CC2538 Evaluation Modules (CC2538EM). Estos son módulos que integran el SoC con una antena, elementos auxiliares y puertos de entrada y salida. Estos módulos se conectan a la plataforma SmartRF06 conformando así un kit de desarrollo y evaluación completo, con herramientas suficientes para grabar, ejecutar y depurar programas en el CC2538.

El SoC necesita circuitos externos para su correcto funcionamiento, tales como un circuito de adaptación de señales de radio y conexión a una antena, y un osci-

lador externo. Soldar el chip y el diseño de su circuito externo está por fuera del alcance de este proyecto. Para esto se utilizó un módulo que resuelve esta conexión.

Para el PCB diseñado se utilizó el módulo Embit: EMB-Z2538PA [36] que integra el SoC seleccionado y un amplificador de radio PA/LNA (del inglés: Power Amplifier / Low Noise Amplifier) CC2592 de Texas Instruments [35]. Este último aporta ganancia de transmisión obteniendo una potencia de transmisión de hasta $22dBm$ (PA) y una ganancia de recepción (LNA) de $+11dB$ o $+6dB$, que se puede seleccionar desde el MCU. Teniendo en cuenta que el CC2538 tiene una sensibilidad de recepción de $-97dBm$, con el CC2592 se obtiene una sensibilidad de recepción configurable entre $-108dBm$ y $-103dBm$.

Otro módulo disponible, por ejemplo es el CC2538EM, pero este módulo pertenece al kit de desarrollo CC2538DK y no se pueden comprar en grandes cantidades debido a que están destinados a usos didácticos y desarrollo de aplicaciones. Por este motivo no fueron utilizados para la aplicación RSItrust, la cual está pensada para su reproducción a gran escala.

4.2.2. Sensores

Los nodos cuentan con sensores de temperatura y humedad del aire y humedad del suelo, los cuales fueron debidamente seleccionados. Aunque su selección no está dentro del alcance, el equipo de proyecto participó en la selección de estos sensores. Se le propuso al equipo un conjunto de sensores posibles, para luego evaluarlos, observando sus principales características. Se buscó minimizar la tensión de alimentación al nodo para lograr un menor consumo de energía del sistema. Debido a ello, se tuvo en consideración la tensión de alimentación necesaria para los sensores, buscando que fuera lo menor posible.

Elección de sensores

La tabla 4.1 contiene los sensores de humedad del suelo propuestos por los tutores y sus características más relevantes. Se puede observar que ambos tienen la misma precisión, aunque el EC-05 necesita una menor tensión de alimentación y tiene un menor consumo de corriente. También tiene la ventaja de ser más económico[1], por lo tanto este fue el sensor finalmente seleccionado para humedad del suelo. Algo importante a tener en cuenta es que para este sensor el fabricante da información de su calibración a $2{,}5V$, por lo tanto las curvas de porcentaje de humedad en función de la tensión a la salida son correctas si se alimenta a esta tensión. De otra manera, utilizar estas curvas para el cálculo del porcentaje de humedad daría resultados erróneos. Por lo tanto, es necesario que al medir humedad

[1]Basado en precios del distribuidor Aliexpress `http://www.aliexpress.com/`

del suelo la tensión del circuito sea 2,5V.

Sensor	EC-05 [19]	10HS [18]
Alimentación (V)	2,5 a 3	3 a 15
Consumo (mA)	10	12(@3V) a 15(@15V)
Precisión	$\pm 2\,\%$	$\pm 2\,\%$
Comunicación	Analógica	Analógica
Precio por cantidad ($U\$S$)	81	108

Tabla 4.1: Sensores de humedad de suelo

Los sensores de temperatura para medir la temperatura del aire propuestos por los tutores, se listan en la tabla 4.2.

Sensor	TMP275 [38]	TMP75C [39]
Alimentación (V)	2,7 a 5,5	1,4 a 3,6
Consumo (mA)	0,1	0,025
Precisión	$\pm 0{,}0625°C$	$\pm 0{,}5°C$
Comunicación	$I2C$	$I2C$
Precio por cantidad ($U\$S$)	2,04	0,967

Tabla 4.2: Sensores de temperatura

El sensor de temperatura seleccionado fue el TMP75C, dado que necesita una tensión de alimentación menor al TMP275, y un menor consumo de corriente. A esto se le agrega que es más económico[2]. Este sensor utiliza el protocolo serie Inter Integrated Circuit (I2C) de dos hilos para la transmisión de datos.

Finalmente, en la tabla 4.3 se listan los sensores de humedad que fueron evaluados, para utilizar en la medición de humedad del aire.

[2]Basado en precios del distribuidor DigiKey `http://www.digikey.com/`

Sensor	SHT75 [58]	SHT25 [57]
Alimentación (V)	2.4 a 5.5	2.1 a 3.6
Consumo (mA)	0.55	0.3
Tolerancia	±1.8 %RH	±1.8 %RH
Comunicación	Propietario	$I2C$
Precio por cantidad ($U\$S$)	28,64	8,34

Tabla 4.3: Sensores de humedad

Se eligió el SHT25 frente al SHT75 como sensor de humedad del aire, dado que es más económico[3], necesita una menor tensión de alimentación y tiene un menor consumo de corriente. Además, este sensor utiliza el protocolo serie $I2C$. De esta manera se puede utilizar el mismo bus de comunicación tanto para el sensor de temperatura como para el sensor de humedad, ya que $I2C$ admite múltiples esclavos en el bus.

A modo de resumen, la tabla 4.4 muestra cuáles fueron los sensores seleccionados para integrar al nodo.

Tipo de sensor	**Modelo**
Humedad del suelo	EC-05
Humedad del aire	SHT25
Temperatura del aire	TMP75C

Tabla 4.4: Sensores utilizados

Integración de sensores al nodo

La elección de los puertos a utilizar para cada sensor se basó en el tipo de comunicación de cada uno de ellos. El SHT25 y el TMP75C se conectan a GPIO configurados como $I2C$, utilizando el mismo bus para conectar ambos sensores, ya que este protocolo admite múltiples esclavos distinguiéndose mediante su dirección. El EC-05 se conectó a un GPIO, el cual se debe configurar como conversor analógico a digital (ADC, del inglés: Analog to Digital Converter).

Se utilizó un módulo que contiene el sensor de humedad SHT25 y sus elementos auxiliares para simplificar la etapa de construcción del nodo, teniendo en cuenta

[3]Basado en precios del distribuidor Mouser `http://uy.mouser.com/`

que su encapsulado no es sencillo de soldar. Este es el MOD-1018 de Embedded Adventures [11], cuyo esquemático se puede ver en la Figura 4.1. El módulo cuenta con resistencias de pull-up[4] para el bus de comunicación $I2C$, un condensador de desacople en la alimentación y tiene disponible las señales que se describen en la tabla 4.5. Si bien no se diseñó un PCB para este sensor, se puede realizar siguiendo las recomendaciones del fabricante.

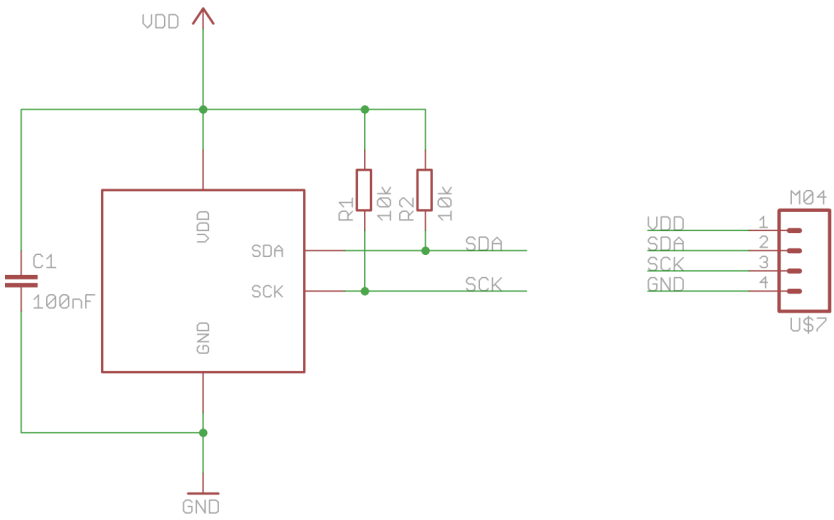


Figura 4.1: Esquemático MOD-1018.

V_{DD}	Alimentación ($2{,}1V$ a $3{,}6V$)
SDA	Bus de datos $I2C$
SCK	Bus de reloj $I2C$
GND	Conexión a tierra

Tabla 4.5: Conector del MOD-1018

Cabe destacar que este sensor tiene dirección $I2C$ fija $b'1000000$.

Para el TMP75C se diseñó un PCB auxiliar que contiene los elementos que necesita para su funcionamiento, según indica el fabricante en su hoja de datos. El sensor además cuenta con 3 pines (A_0, A_1 y A_2) mediante los cuales se puede configurar su dirección $I2C$ como se desee, conectando dichos pines a V_{CC} o GND.

[4]Resistencia entre la señal y la alimentación, generando un 1 en estado inactivo del bus.

Este sensor cuenta con un pin de alerta (ALERT), no utilizado en el diseño, mediante el cual da una indicación en caso de que la temperatura sobrepase un límite configurado.

El circuito diseñado contiene resistencias de pull-up en el bus $I2C$, resistencia de pull-up en el pin de alerta, y la configuración de la dirección $I2C$ del sensor. A este sensor se le fija su dirección de $I2C$ mediante hardware utilizando los pines A_0, A_1, A_2, los cuales se conectaron a GND y por lo tanto su dirección quedó fijada en $b'1001000$. El diseño deja a disposición las señales del bus $I2C$, alimentación y V_{CC}, distribuídas de manera análoga al módulo MOD-1018.

En la Figura 4.2 se aprecia el diseño del PCB auxiliar utilizado para conectar el sensor de temperatura al nodo.

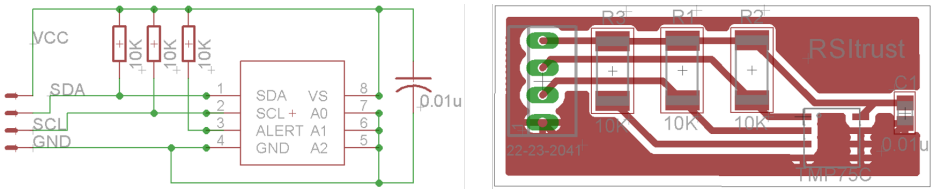
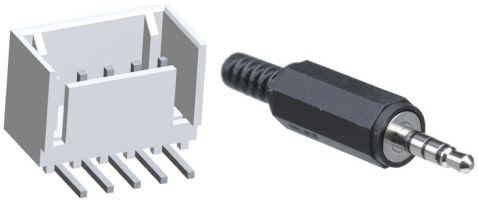


Figura 4.2: PCB para el sensor TMP75C

Para la conexión de los sensores al PCB, en un principio se pensó utilizar conectores de 2,54mm con lock (conector muy común en el mercado), el cual se ejemplifica en la Figura 4.3a[5] de forma de asegurar una buena conexión al PCB, y de este modo evitar problemas por una mala conexión. Debido a que la placa se va a instalar al aire libre, se debe utilizar una caja y prensacables para proteger la placa (ver sección 4.4). Dichos prensacables deben tener un diámetro tal que dejen pasar al conector pero que puedan apretar el cable. Utilizando los conectores considerados esto no es posible, por lo que se decidió utilizar conectores de 3,5mm. En la Figura 4.3b se aprecia un conector de este tipo, portador de 4 señales[6]. Este tipo de conector también asegura una buena conexión, siendo también de un tipo muy común en el mercado. El sensor EC-05 cuenta con un conector de este tipo, de 3 señales. Se decidió utilizar estos conectores, eligiendo un modelo que tiene un sensor de presencia mecánico, indicando con un contacto seco NC si se tiene un sensor conectado o no. El mismo puede ser utilizado para detectar si no hay sensor conectado, pudiendo generar un mensaje de error. En resumen, el sensor de temperatura TMP75C y el sensor de humedad del aire SHT75 utilizan un conector de 3,5mm de 4 señales, que se describen en la tabla 4.5 conectado a un conector hembra que cuenta con sensor de presencia, mientras que para el sensor de humedad del suelo EC-05, el cual tiene incorporado de fábrica un conector de 3,5mm

[5]Imagen extraída de la página web del distribuidor `http://es.rs-online.com/web/`
[6]Imagen obtenida de la página web del distribuidor `http://www.showmecables.com/`

de 3 señales (alimentación del sensor, GND y señal analógica de salida del sensor), se utilizó un conector hembra con sensor de presencia.



(a) Conector 2,54mm (b) Conector 3,5mm

Figura 4.3: Conectores para sensores

4.2.3. Diseño de alimentación

El diseño de la etapa de alimentación al circuito fue un paso importante del diseño. Como fuente de tensión se decidió utilizar dos pilas AA de ion de litio en serie, obteniendo $3V$ de alimentación nominales. La elección de pilas de litio en lugar de alcalinas se debe a que estas tienen una mayor capacidad. Este tipo de pilas por un mayor periodo de tiempo provee un voltaje de $1{,}5V$ pasando en forma abrupta a cero al finalizar su vida útil. En cambio, las pilas alcalinas van disminuyendo el voltaje que entregan con el paso del tiempo. De esta manera, la tensión entregada por las mismas descenderá por debajo del mínimo necesario para el circuito acortando su vida útil. En la Figura 4.4 se puede apreciar la gráfica de descarga de las pilas de ion de litio [24], mientras que en la Figura 4.5 se aprecia la de las pilas alcalinas [53]. En ambas gráficas se aprecia el voltaje en función del tiempo al alimentar una carga de 24Ω, lo que equivale a una corriente de $6{,}25mA$. Comparando ambas gráficas se observa la ventaja de las pilas de ion de litio frente a las pilas alcalinas como se describió.

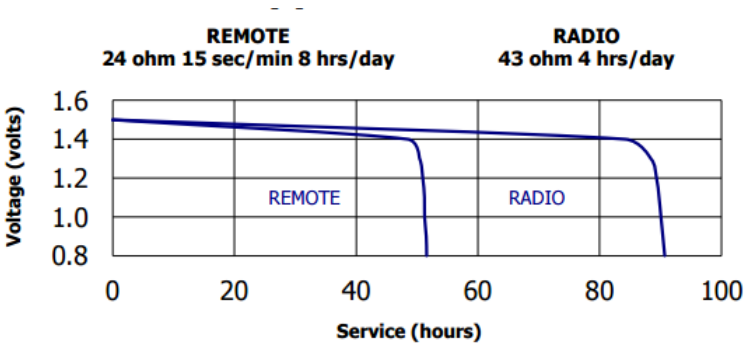


Figura 4.4: Descarga Pila de Ion de Litio.

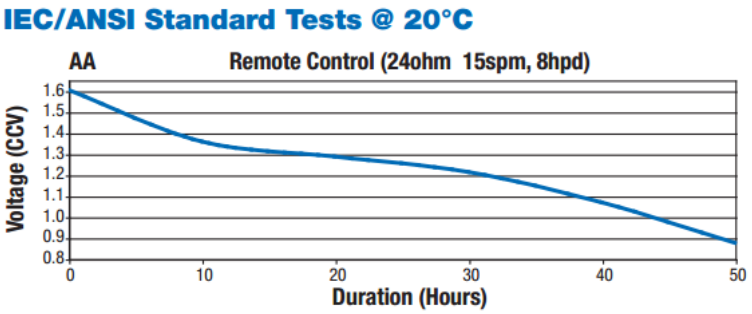


Figura 4.5: Descarga Pila Alcalina.

Con el objetivo de minimizar el consumo energético del sistema, se diseñó esta etapa minimizando la tensión a la que se alimenta el nodo. Para esto se buscó un conversor de tensión que redujera la tensión de alimentación al valor mínimo que asegure el correcto funcionamiento de los elementos del circuito.

Al tener $3V$ de alimentación, se debió diseñar una etapa de reducción de alimentación, la cual provea una tensión regulada y estable. Para esto se evaluaron diferentes tipos de dispositivos reductores de tensión. En primera instancia podría utilizarse un conversor lineal de tensión, o un conversor DC/DC. Debido al requerimiento de minimizar el consumo del sistema se debió analizar cuál de estos dispositivos es conveniente utilizar.

En el documento "*Using a DC-DC Converter to Reduce Power (Current) Consumption In CC430 Systems*" [41] de Texas Instruments se presenta una comparación entre la eficiencia de un conversor lineal y un conversor DC/DC. En un conversor lineal la corriente de salida es igual a la corriente de entrada, de esta manera la eficiencia de un conversor lineal está dada por la ecuación 4.1. La eficiencia energética en este caso está dada por la relación que hay entre la tensión de entrada al conversor y su salida. De esta manera, si aumenta la tensión de entrada o disminuye la tensión de salida la eficiencia será menor.

$$\eta = \frac{P_{Out}}{P_{In}} = \frac{V_{Out}.I_{Out}}{V_{In}.I_{In}} = \frac{V_{Out}}{V_{In}} \tag{4.1}$$

Para el caso de los conversores del tipo DC/DC la corriente de salida es diferente a la de entrada. La eficiencia está dada por la relación entre la potencia de entrada y de salida, la cual es generalmente de un 80 % - 90 %. Por este motivo se decidió utilizar un conversor DC/DC para la etapa de alimentación del sistema.

Conversor DC/DC

La tensión de todos los elementos del nodo se fijó en principio en 2,5V ya que es la tensión de alimentación a la que está calibrado el EC-05, y es un voltaje con el cual el resto de los elementos opera correctamente. Se evaluó alimentar los sensores a 2,5V y el módulo a 2,1V, pero esto traería la complicación de tener elementos digitales comunicándose utilizando diferentes niveles de referencia de tensión. Si se quisiera realizar de esta manera, se deberían incluir circuitos auxiliares de adaptación de niveles, aumentando la complejidad del diseño.

Esta sección presenta el proceso de selección de un conversor que reduzca la tensión de alimentación a 2,5V, aunque finalmente se encontró un conversor con el cual se puede seleccionar en tiempo de ejecución la tensión de alimentación, siendo 2,5V al tomar una medida de humedad del suelo con el EC-05, y $2,1V$ el resto del tiempo.

A la hora de elegir el conversor, se buscó que tuviera la capacidad de entregar la corriente demandada por el sistema y que el consumo del conversor sea despreciable frente al resto del circuito. El conversor debe tener una buena eficiencia a corrientes bajas, y debe poseer una señal de *byPass* que permita deshabilitarlo mientras el sistema esté inactivo, para evitar así su consumo estático.

En la Tabla 4.6 se muestra el consumo demandado por el nodo en el estado de mayor consumo, donde el MCU y radio están activos y los sensores están midiendo. Por otro lado, el estado inactivo del sistema implica que los sensores están apagados, el MCU en modo de bajo consumo y la radio inactiva. En este estado el consumo del MCU es de 1,7μA según indica el fabricante.

Dispositivo	**Consumo (mA)**
EMBIT (CC2538)	166
TMP75C	0,025
EC-05	10
SHT25	0,3
Total	176,325

Tabla 4.6: Consumo máximo del sistema

En un principio se optó por el conversor Torex XCL209 [47], al cual se le configura mediante un circuito externo la tensión de salida deseada. Este conversor además cuenta con un pin de entrada habilitador, mediante el cual se puede desactivar el conversor para disminuir su consumo cuando el nodo esté inactivo. Sin embargo, se descartó al detectar que la corriente de *byPass* del conversor es comparable a la corriente consumida por el resto del circuito en estado inactivo, además

de que no tiene una buena eficiencia a bajas corrientes. Luego, buscando nuevamente los dispositivos disponibles en el mercado, se optó por utilizar el TPS62740 de Texas Instruments [40]. Este conversor step down tiene tensión de salida fija configurable en saltos de $100mV$ desde $1{,}8V$ a $3V$ y soporta una corriente de salida de $300mA$ (suficiente para alimentar al nodo, como se ve en la tabla 4.6). Este conversor no cuenta con *byPass*, pero su consumo estático es muy bajo ($360nA$), siendo despreciable respecto al consumo del sistema incluso en su estado inactivo, y tiene una eficiencia mayor al 90 %.

La tensión de salida del conversor se fija con 4 bits que se conectan a V_{cc} o GND según su hoja de datos. La tabla 4.7 resume las configuraciones para las tensiones $2{,}1V$ y $2{,}5V$.

$V_{OUT}(V)$	V_{SEL4}	V_{SEL3}	V_{SEL2}	V_{SEL1}
2.1	0	0	1	1
2.5	0	1	1	1

Tabla 4.7: Configuración de tensión de salida del conversor

Se observa que la configuración de la tensión de salida del conversor, para $2{,}5V$ y $2{,}1V$ difieren únicamente en el estado del pin V_{SEL3}. En el diseño se colocó un *jumper* que permite conectar este pin a V_{CC}, GND o a un puerto de uso general (GPIO del inglés: General Purpose Input Ouput) del MCU. De esta manera, se puede seleccionar la tensión de salida en $2{,}5V$ o $2{,}1V$ de manera fija usando el *jumper*, o pudiendo configurarlo en tiempo de ejecución. Para esto último se coloca el *jumper* en la posición del GPIO para que el MCU pueda seleccionar por software la alimentación del circuito en tiempo de ejecución, alimentando a $2{,}5V$ cuando se desee leer el sensor de humedad EC-05 (tensión a la que se recomienda alimentar a la hora de tomar una medida), y a $2{,}1V$ el resto del tiempo. Esta solución es aún mejor que la prevista al inicio (alimentar al circuito todo el tiempo a $2,5V$). Esta cualidad del sistema no fue probada en este proyecto por problemas en la construcción del nodo como se explica en la sección 4.5, queda prevista para futuras utilizaciones del diseño.

4.2.4. Elementos auxiliares y conectores

Switches habilitadores

Para reducir el consumo de los sensores, se decidió apagarlos mientras no se estén utilizando. Para esto se definió utilizar switches FPF2004 [55] que habilitan o no la alimentación de los sensores. Éstos poseen una entrada ON, la cual es manejada mediante diferentes GPIO del MCU. Al estar esta entrada en '1' lógico la salida del switch queda conectada a tensión V_{cc}, permitiendo así alimentar

los sensores sólo a la hora de medir. Los switch FPF2004 pueden entregar una corriente de hasta $100mA$, más que suficiente para alimentar a los sensores cuyos consumos se resumen en la tabla 4.6. También se evaluó alimentar directamente los sensores desde GPIO del MCU, pero por seguridad se decidió la opción de los switches. Esto evita que ante una falla o alto consumo de corriente por parte del sensor se supere la corriente máxima soportada por el pin utilizado para alimentar el sensor, pudiendo causarle daños. Si se prefiere optar por no utilizar estas protecciones, como puede ser para abaratar costos, se prevé en el diseño sustituir el switch por una resistencia de 0Ω como puente entre su habilitación y la salida. De esta manera, el MCU alimenta el sensor a través de la habilitación.

JTAG y UART

A la hora de programar el MCU, se cuenta con dos protocolos, el Joint Test Action Group (JTAG) y el Universal Asynchronous Receiver-Transmitter (UART). Este último permite realizar la programación mediante el Bootstrap Loader (BSL).

El protocolo UART es un protocolo serie, el cual puede utilizarse para observar a través de un puerto de la PC impresiones en una consola, que puede realizar un programa, tanto como parte de la aplicación como para realizar pruebas de ejecución. Además, UART se emplea para programar aplicaciones en el MCU. El método para programar a través del puerto UART es el BSL. BSL es un programa precargado en un espacio reservado de la memoria Flash, el cual al ejecutarse graba en la memoria Flash lo que se le envía vía UART. Para que este programa se ejecute se debe habilitar configurando un bit de un registro del MCU. Luego de un reset el MCU lee el estado de un GPIO que le indica si se debe ejecutar el BSL o no. Este GPIO debe ser configurado, indicando el nivel lógico con el cual se realiza la habilitación de la ejecución del BSL. Si el programa BSL no viniera precargado en el MCU, este debe cargarse via JTAG, mecanismo que se explica más adelante[7]. Se previó en el diseño dejar disponible el puerto UART para este propósito, aunque en RSItrust se utilizó únicamente como medio de depuración de la aplicación mediante impresiones en consola, **BSL no fue probado en este proyecto**.

Para la comunicación UART se decidió utilizar conectores de $2{,}54mm$ para la comunicación, como el que se muestra en la Figura 4.6a.

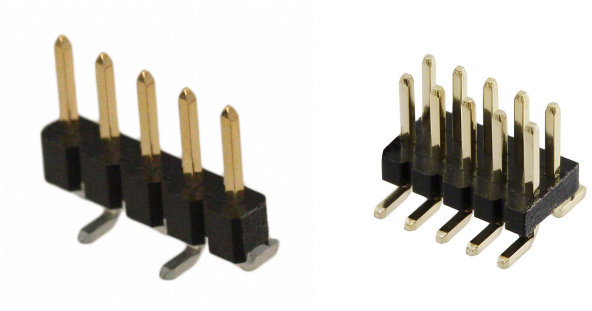
Un protocolo serie más avanzado es el JTAG. Este protocolo no solo permite grabar programas en un MCU, sino que también posibilita el test del diseño de un PCB, y depuración de programas en hardware pudiendo corregir errores tanto de código como de lógica del sistema. Este protocolo permite grabar directamente programas en la memoria Flash del MCU[8]. Para este propósito se dejó disponible

[7] Se puede profundizar en la utilización de BSL en la guía para el usuario [37]

[8] `https://en.wikipedia.org/wiki/Joint_Test_Action_Group/`.

el puerto para comunicación JTAG mediante un conector de 1,27mm el cual se muestra en la Figura 4.6b, dado que este conector es común para los programadores de este tipo.

Las imagenes de los conectores que se muestran en la Figura 4.6 fueron extraídas de la página web del distribuidor DigiKey[9]



(a) Conector UART (b) Conector JTAG

Figura 4.6: Conectores para comunicación serial

Elementos de uso general

Se incluyeron tres botones en el PCB. Se colocó un botón de Reset el cual se conecta al pin Reset del MCU, un botón llamado Select conectado a un GPIO del MCU, el cual se puede utilizar para indicar que se desea programar el MCU mediante BSL, método ya mencionado en la sección 4.2.4, y un botón adicional disponible para realizar pruebas (o para utilizar como parte de otra aplicación) el cual va conectado entre un GPIO del MCU y GND.

El PCB cuenta con un LED [51] indicador de que el circuito está alimentado, conectado en serie a través de un *jumper* para permitir decidir su utilización o no. Si el *jumper* no es colocado se logra una disminución del consumo ya que de esta manera el LED queda desconectado, aunque en primera instancia colocarlo y utilizar el LED indicador es conveniente para verificar el correcto funcionamiento del circuito. Se cuenta con otro LED para uso general que está conectado a un GPIO del MCU. A la hora de dimensionar los LEDs, se observaron las corrientes que entregan los puertos GPIO del MCU. Como la mayoría de ellos entrega $4mA$ y el resto $20mA$, para cubrir el peor caso se calculó la resistencia que va en serie con el LED para que sólo consuma $4mA$. Debido a esto, a ambos LEDs se le colocó una resistencia de 125Ω en serie, provocando en condiciones de funcionamiento una caída de $2V$ si se les hace pasar una corriente de $4mA$. El valor de

[9] `http://media.digikey.com/`

dicha resistencia resulta de la ecuación 4.2.

$$R = \frac{V_{CC} - V_{LED}}{I_{LED}} = \frac{2{,}5V - 2V}{4mA} = 125\Omega \tag{4.2}$$

4.2.5. Diseño final

La Figura 4.7 muestra el esquemático del circuito diseñado. Vale aclarar que luego de las pruebas realizadas en un prototipo que se fabricó este esquemático sufrió ligeras correcciones, como se detalla en la sección 4.5

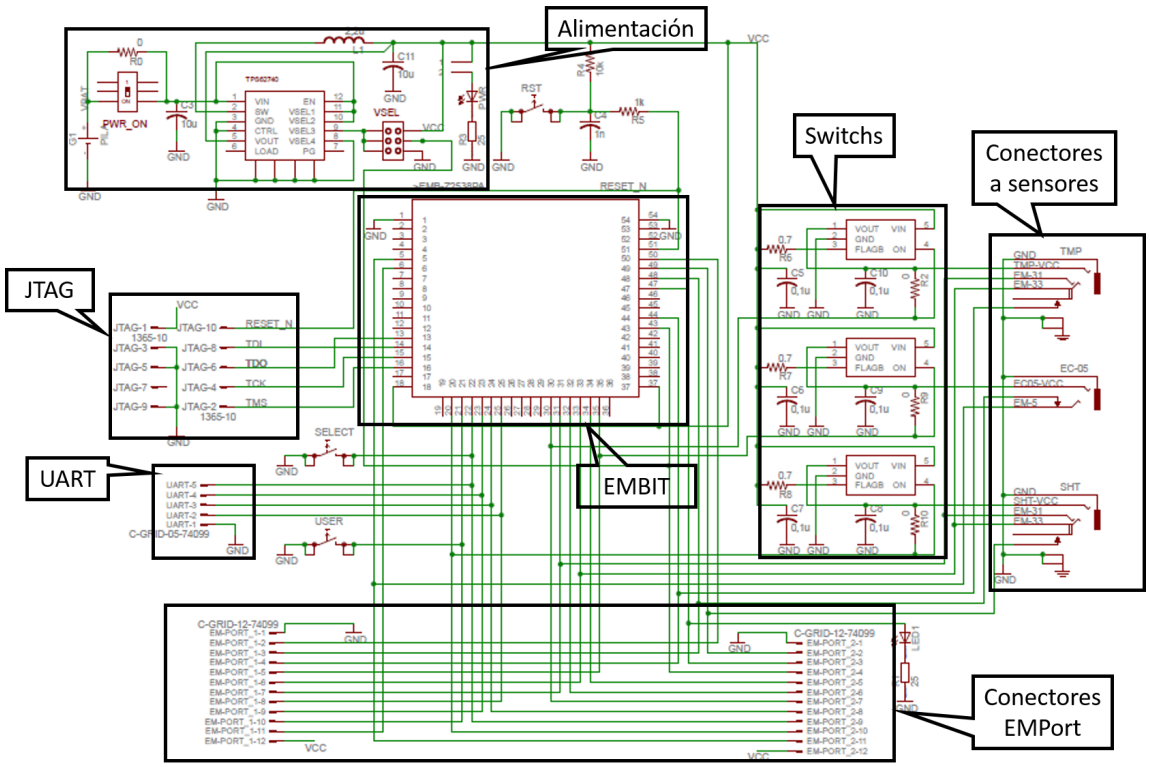


Figura 4.7: Esquemático del diseño

4.3. Diseño del PCB

Se diseñó un circuito con montaje superficial (aunque algunos elementos son *through hole*), permitiendo incorporar el módulo EMBIT y el conversor DC/DC, ambos de montaje superficial. De esta manera además se ocupa menos espacio. El PCB se realizó en doble capa, dado que no fue práctico realizarlo utilizando simple faz, debido a la cantidad de pistas del circuito.

Se tuvo en cuenta las sugerencias dadas para cada circuito integrado en sus respectivas hojas de datos.

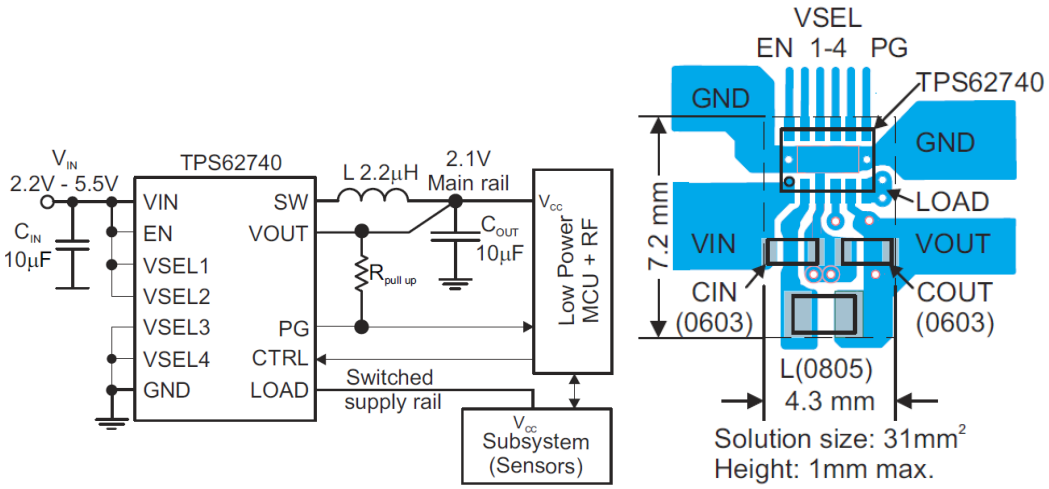
Para el módulo EMBIT el fabricante sugiere:

- El área debajo del módulo debe mantenerse libre de componentes.
- La capa superior de la placa debe estar libre de pistas, planos de potencia, vías. La capa inferior debe ser plano de tierra.
- La alimentación debe estar desacoplada mediante un capacitor cerámico, que debe estar lo más cerca posible del módulo.
- Elementos electrónicos ruidosos deben colocarse tan lejos como se pueda y deben estar correctamente desacoplados.
- Los pines de tierra deben estar conectados a un plano de tierra.
- Se debe mantener la antena libre de elementos metálicos.
- La placa no se debe alojar en recintos metálicos para evitar degradación de la señal de radio.

El fabricante de conversor TPS62740 da las siguientes pautas de diseño:

- El condensador de entrada debe colocarse lo más cerca posible a los pines de alimentación y tierra.
- El condensador de salida se debe colocar cerca entre pines V_{OUT} y GND.
- V_{OUT} debe estar conectado a la salida del condensador y debe routearse lejos de componentes ruidosos.
- El *Thermal Pad* del dispositivo debe estar conectado a tierra.

La Figura 4.8 muestra un ejemplo de aplicación, junto con el layout que se recomienda para el conversor.



(a) Circuito aplicación (b) Layout recomendado

Figura 4.8: Implementación de conversor TPS62740

Los conectores de los sensores fueron colocados de tal manera que, al conectar los sensores a los mismos, sus conectores queden por fuera de la placa, contra su borde inferior. Esta opción parece ser la manera más cómoda de integrar los sensores al PCB. Sin embargo, como se menciona en la sección 4.4, el tamaño de la placa debió ser modificado, quedando estos conectores sobre el área interior de la misma.

Si bien se realizó un diseño para cumplir con las especificaciones de este proyecto, fue contemplado que el mismo pueda ser utilizado para otras aplicaciones, como puede ser otros grupos de proyecto. Para ello se incluyeron conectores que dejan accesibles todos los puertos que tiene disponible el EMBIT. Para mantener la placa de tamaño reducido, en el diseño estos conectores ocupan el mismo espacio físico que un porta pilas, el cual es utilizado por RSItrust para alimentar al nodo. Esto último asumiendo que una aplicación que usa estos conectores utilizará una fuente de alimentación externa.

Para la distribución de las señales en los conectores se buscó minimizar la cantidad de cruces de pistas en el PCB y simplificar el ruteo. La Tabla 4.8 y la Tabla 4.9 muestran el mapa resultante de los pines disponibles del módulo EMBIT a los conectores genéricos incluidos, los cuales fueron llamados $EM-Port\_1$ y $EM-Port\_2$ respectivamente.

Embit	$EM - Port\_1$
18; 37 (V_{CC})	12
6	11
21	10
23	9
25	8
31	7
33	6
35	5
44	4
48	3
50	2
1; 54 (GND)	1

Tabla 4.8: Mapeo EMBIT - $EM - Port\_1$

Embit	$EM - Port\_2$
18; 37 (V_{CC})	12
5	11
20	10
22	9
24	8
30	7
32	6
34	5
43	4
47	3
49	2
1; 54 (GND)	1

Tabla 4.9: Mapeo EMBIT - $EM - Port\_2$

Los componentes extra a utilizar (resistencias, capacitores, inductores, LEDs, botones), se procuró que tuvieran un tamaño suficientemente grande para que se puedan soldar sin mayores complicaciones.

La Tabla 4.10 muestra la asignación de pines del EMBIT al circuito, indicando el número de pin del integrado CC2538.

Se utilizaron resistencias de 0Ω, para agregar versatilidad al diseño. Estas fueron colocadas, para cada switch como ya se mencionó en la sección 4.2.4, entre la

CC2538		Circuito		
Puerto	**Pin**	**Pin EMBIT**	**Señal**	**Elemento**
PA3	19	22	RTS; SELECT	BSL
PA2	18	23	CTS	
PA1	17	24	RX	
PA0	16	25	TX	
PB7	48	13	Data Output	JTAG
PB6	49	14	Data Input	
JTAG-TCK	47	15	Clock	
JTAG-TMS	46	16	Mode Select	
PD2	27	30	TMP Enable	TMP
PC1	13	31	SDA	
PD1	26	33	SCL	
PA5	21	44	Sensor Presencia	
PC0	14	35	EC Enable	EC-05
PA7	23	5	EC Output	
PA6	22	48	Sensor Presencia	
PD2	27	30	SHT Enable	SHT
PC1	13	31	SDA	
PD1	26	33	SCL	
PA5	21	44	Sensor Presencia	
RESET	28	51	RESET	RESET
PA4	20	43	Chip Enable	DC/DC
PC6	7	21	Boton Usuario	Boton Usuario
PC0	14	47	LED Usuario	LED Usuario

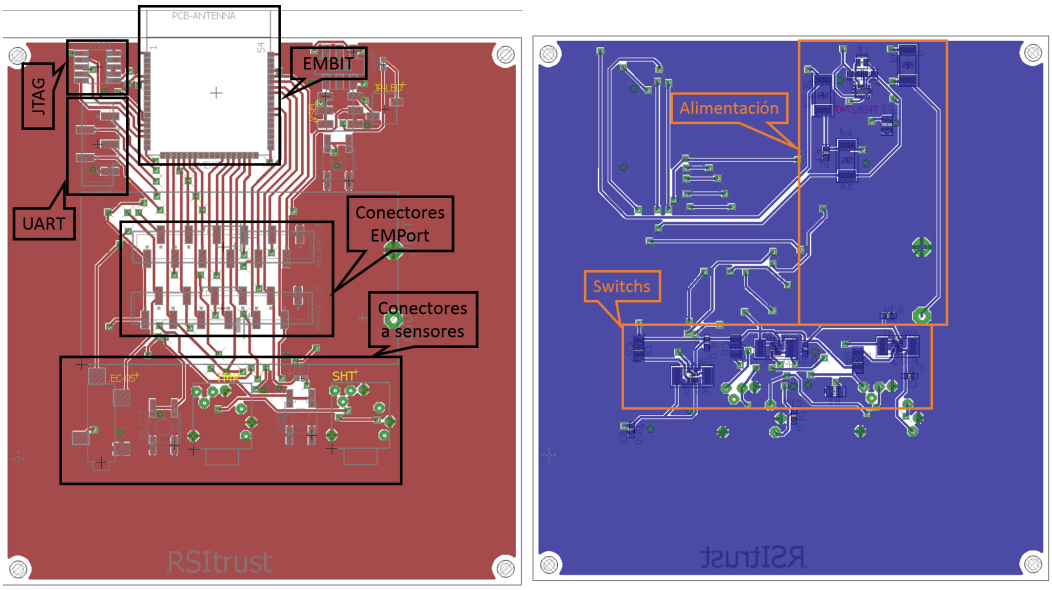
Tabla 4.10: Pines utilizados del EMBIT por RSItrust

habilitación y la salida, para tener la posibilidad de no usar los switch a la hora de utilizar estos puertos.

La Figura 4.9 es una imagen del layout del PCB diseñado. Se muestra tanto la cara superior como la inferior, indicando las diferentes etapas del circuito.

4.4. Montaje mecánico

Considerando que el nodo estará a la intemperie, se debe tener en cuenta la utilización de un recinto que aloje al nodo y lo proteja de las inclemencias climáticas. Sin embargo, el sensor de humedad del suelo deberá ser colocado bajo tierra, y a su vez, tanto el sensor de humedad como el sensor de temperatura del aire de-



(a) Cara superior (b) Cara inferior

Figura 4.9: Layout del PCB diseñado

berán ser instalados al aire libre, bajo una protección atmosférica. La Figura 4.10 muestra un ejemplo de estos protectores atmosféricos[10]. Debido a esto, los sensores deben instalarse alejados del PCB, y por lo tanto se deben realizar perforaciones en el gabinete que lo protegerá, para poder colocar los cables que se utilizarán para conectar los sensores. Para no perder la protección del circuito al perforar la caja hubo que tener en cuenta la utilización de prensacables.

Figura 4.10: Protección atmosférica para sensores

[10]Imagen extraída del fabricante OTT Hydromet `http://www.ott.com/`

Los prensacables elegidos fueron de $2{,}5mm \sim 6{,}5mm$ de diámetro[11].

Finalizado el diseño, se prosiguió con la elección de un recinto para proteger el hardware, teniendo en cuenta que el nodo estará a la intemperie. Para ello, se buscó un recinto con protección al menos del tipo IP55, la cual garantiza protección contra contacto, protección contra penetración de polvo y contra chorros de agua (desde todas las direcciones). Se optó por el modelo A9413341 de la serie G 190 de OKW Enclosures [23] con el kit de sellado para este modelo[12], garantizando una protección IP65 (protección mayor a la propuesta inicialmente) cuyas medidas son de $138mm$ x $190mm$. Esta caja es grande para las medidas del PCB, pero la siguiente menor ofrecida por el fabricante (modelo G155 [22]) no sería suficiente para poder colocar la placa en su interior. Cabe destacar que a futuro se planea fabricar recintos a medida para la protección del sistema.

Dentro de la serie G 190 de OKW Enclosures hay varios modelos según la profundidad de la caja. La elección del modelo A9413341, se debe a que tiene $68mm$ de profundidad, lo que permite la instalación de los prensacables seleccionados.

Los nodos deben ser montados a una altura de $1{,}80m$ cuando los mismos se ubican a $100m$ de distancia, debido a la Elipse de Fresnel explicada en la sección 6.2.1. Para esto, la caja debe ser colocada sobre un soporte de esta altura, contando, por ejemplo, con una planchuela de metal soldada verticalmente. Para la fijación de la caja al soporte se seleccionaron piezas de sujeción del mismo fabricante[13] diseñadas para este propósito, y el conjunto de tornillos necesarios[14].

4.5. Fabricación y test de hardware

Se fabricó un prototipo del PCB diseñado, que integra el módulo con los sensores, como se describe en la sección 4 para comprobar su funcionamiento. Este prototipo se encargó a agentes externos al grupo, tanto el impreso como la soldadura de componentes. Debido a la escasez de tiempo disponible, y teniendo en cuenta el tiempo de demora resultante al encargar en el exterior la fabricación de varias placas, se buscaron otras alternativas, procurando un precio conveniente y plazos cortos de entrega. Se pudo realizar la fabricación de una unidad en el mercado local. Se realizó una única placa dado que este proveedor no realiza perforaciones metalizadas ni máscara antisoldante, y el precio no era conveniente para realizar varias.

En el prototipo fabricado las vías fueron perforaciones no metalizadas, a las cuales se les soldaron cables haciendo contacto de un lado al otro de la placa. Si

[11] Medida mínima y máxima del diámetro interno del prensacable

[12] `http://www.okw.com/en/Shell-Type-Cases/A9113630.htm`

[13] `http://www.okw.com/en/Wall-mounting/A9204108.htm`

[14] `http://www.okw.com/en/Screws/A0308031.htm`

bien esto en principio se pensó que podía traer problemas, fue útil para realizar el test, siendo de ayuda para poder aislar las fallas detectadas al poder quitar las vías.

Durante la fabricación del prototipo se dañó el conversor de tensión, así como las pistas donde debería ir colocado. Por lo tanto **no se pudo comprobar el correcto funcionamiento de la etapa de conversión de tensión**. Además, se detectó un cortocircuito entre la alimentación del circuito y GND. Debido a ello, se aisló la etapa de alimentación del resto del circuito, debiendo alimentar directamente de una fuente externa.

4.5.1. Errores de diseño

Se inició el test midiendo continuidad, donde se detectaron cortocircuitos en el PCB. Cortando pistas y desoldando vías, se fueron aislando las diferentes partes del circuito para investigar la falla. En lo que sigue de este capítulo se explica cuáles fueron los errores detectados en la etapa de test y cómo repararlos.

Cortocircuitos en pistas

Se encontraron errores de ruteo de pistas, algunas de ellas quedaron en cortocircuito. No fueron errores conceptuales que se hayan generado por una mala interconexión de componentes en el esquemático, sino que fueron errores en el layout. En la Figura 4.11 se indica cómo reparar estos errores en la capa superior del PCB, y en la Figura 4.12 los correspondientes a la capa inferior, cortando pistas y realizando los puentes que se indican en las mismas. Notar que uno de estos errores se dio sobre el conversor DC/DC, el cual es muy pequeño y por lo tanto la reparación en ese punto no es posible.

Botones

El PCB consta de 3 botones, la Figura 4.13 muestra su ubicación.

La herramienta Eagle utilizada para realizar el PCB no cuenta con el package del botón seleccionado en sus librerías. Por lo tanto, el footprint de los botones se tuvo que diseñar. Al realizar este diseño se cometió el error de asignar los pines de manera incorrecta, provocando que al colocar los botones como se indica en el diseño del PCB, estos botones tienen sus 4 pines conectados entre sí, lo que provoca un cortocircuito permanente.

La Figura 4.14 indica cómo colocar los botones para que funcionen como es deseado. La imagen muestra una comparación de cómo se deberían colocar los botones según el diseño, y cómo se deben colocar para enmendar error mencionado, girando el botón 90° y debiendo soldar a la placa solamente los pines que quedan

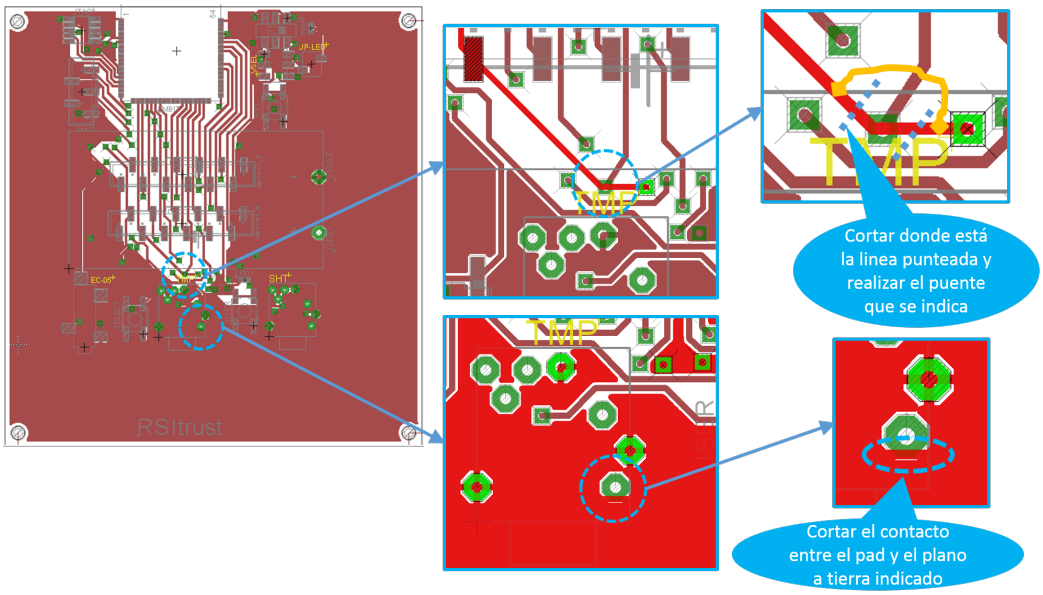


Figura 4.11: Modificaciones de pistas en capa superior.

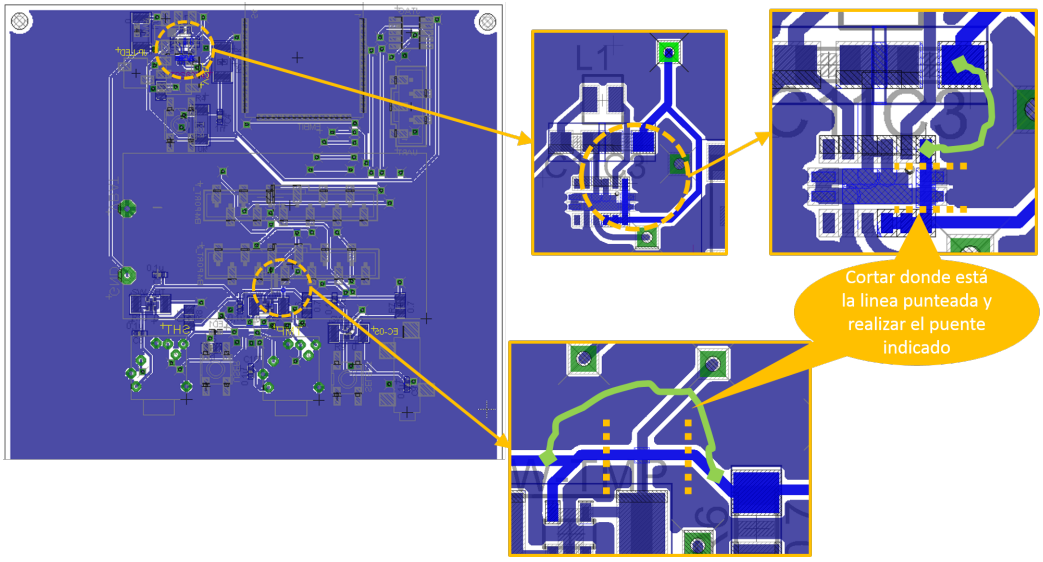


Figura 4.12: Modificaciones de pistas en capa inferior.

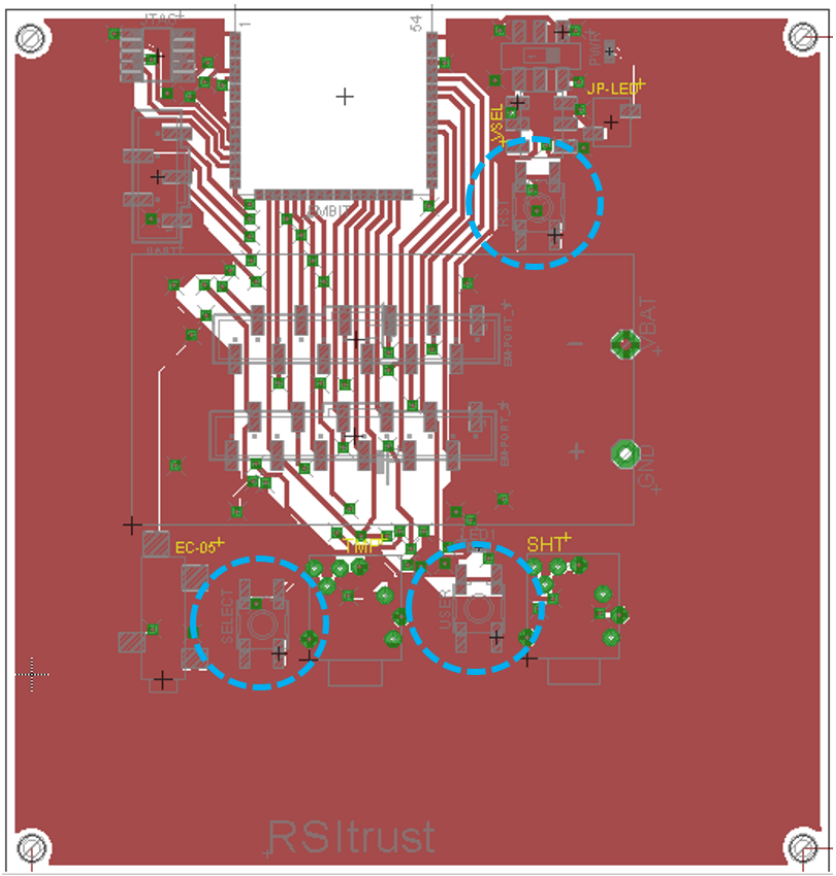


Figura 4.13: Ubicación de botones.

a la izquierda, cuidando que los pines sin soldar no hagan contacto con pistas.

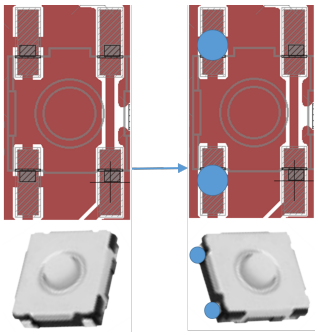


Figura 4.14: Corrección de botones.

Conector EC-05

Originalmente el diseño fue pensado de manera que el conector de los sensores, al estar conectados queden fuera de la placa (colocando el conector hembra al borde de la misma). Debido a que el gabinete que se consiguió para colocar la placa del nodo es más grande de lo deseado y que se diseñó el PCB para que el circuito ocupe el menor área posible, fue necesario hacer la placa más grande para poder colocar los tornillos de fijación. Para no modificar el ruteo se agrandó el PCB quedando área libre alrededor del circuito. Debido a esto, los conectores hembra de los sensores quedaron dentro de la placa, y no en el borde como se pensó en un principio.

No se previó que la parte plástica de los conectores de los sensores podrían no ser lo suficientemente finas como para poder conectarlos en esta nueva situación, el conector del EC-05 es grueso y la placa no permite que se conecte[15]. Para solucionarlo, se debe calar la placa en el espacio donde va el conector. Esto se muestra en la Figura 4.15.

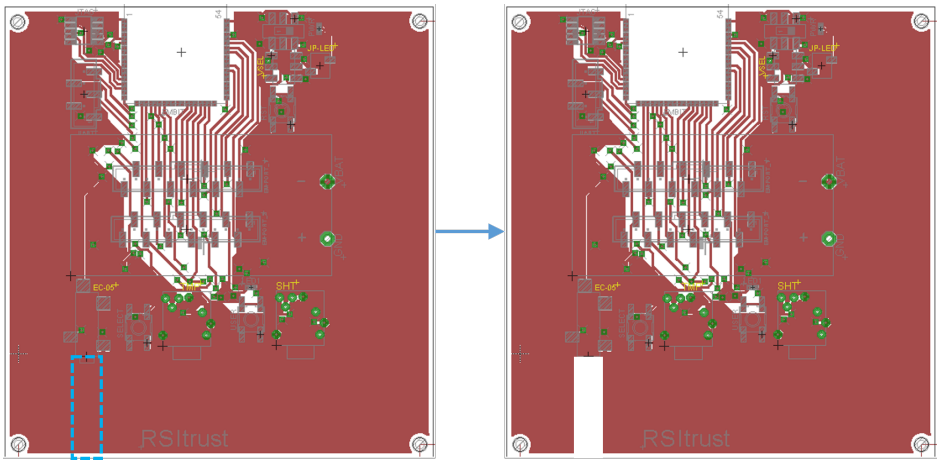


Figura 4.15: Calado para conector del EC05.

Otro error detectado fue que se conectó la alimentación del sensor EC-05 en el pin de su señal analógica, y viceversa. Para solucionar este problema, nuevamente hay que realizar puentes y cortar algunas pistas. Esto se detalla en la Figura 4.16.

LED de uso general

Si se observa en la tabla 4.10 se puede ver que el sensor de presencia del EC-05 y el LED de uso general están conectados al mismo puerto PC0 del MCU. Esto

[15]La hoja de datos de este sensor no da información sobre el diámetro del conector.

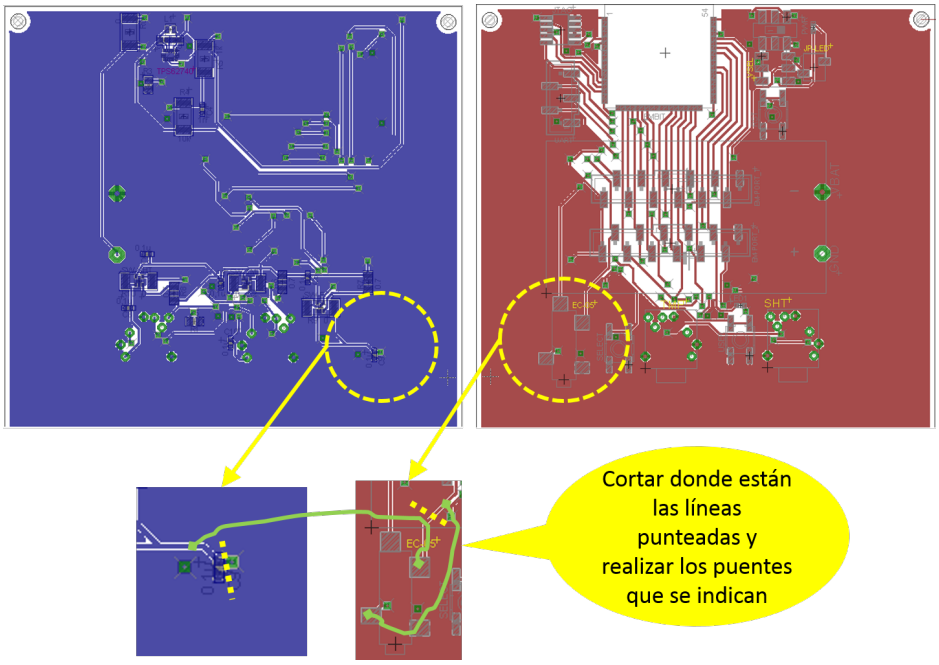


Figura 4.16: Reparación Señales EC05.

se debe a que internamente al EMBIT, dicho puerto está replicado en dos salidas diferentes del módulo. Este error se puede reparar como se muestra en la Figura 4.17. De esta manera, el LED queda conectado al puerto PD6 del MCU.

4.5.2. Test de funcionalidades

Luego de haber reparado el prototipo se procedió a realizar pruebas funcionales del mismo. Se comenzó por grabar un programa, para luego validar el funcionamiento de los restantes elementos.

Programación JTAG

Para la programación del nodo se utilizó el emulador TI XDS100v3 Emulator [42] embebido en la plataforma de desarrollo SmartRF06 utilizada. Este es un emulador de JTAG, el cual permite utilizar este protocolo a través de un puerto USB de la PC facilitando la programación y depuración de programas, tanto en el módulo de evaluación CC2538EM como en placas externas a través de un puerto disponible en la plataforma SmartRF06. Los pasos a seguir para la programación por JTAG se describen en el anexo B.3. La prueba inicial consistió en una aplicación sencilla que enciende el LED previsto en el diseño, resultando esta prueba exitosa.

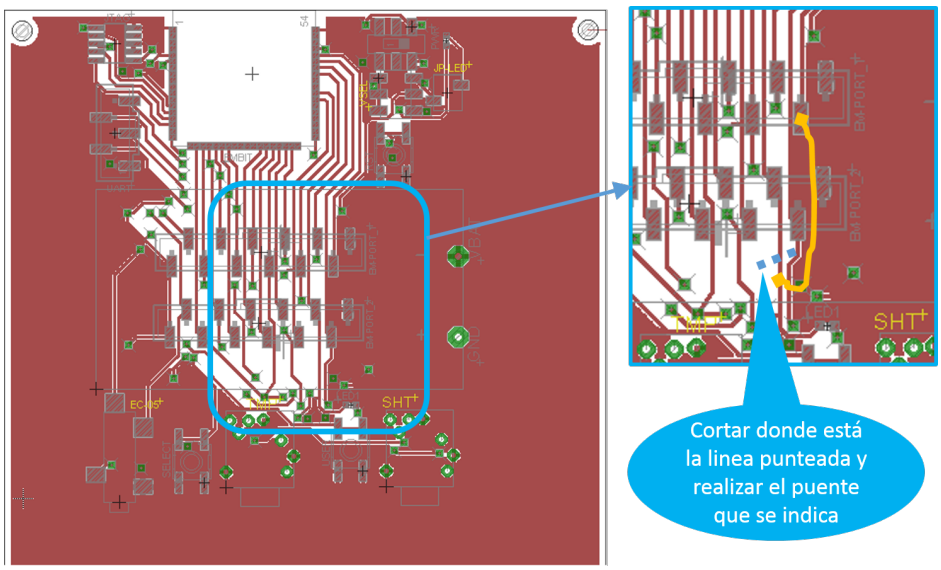


Figura 4.17: Recableado del LED.

Prueba de switches

Se le programó al MCU una aplicación de prueba, la cual de forma cíclica habilita y deshabilita los switches de alimentación a los sensores. Se observó su salida utilizando un osciloscopio, apreciando que las salidas eran habilitadas correctamente.

Puerto UART

Para verificar el funcionamiento del puerto UART se utilizó un adaptador USB-UART, el cual permite realizar la comunicación UART con el MCU mediante un puerto USB. Realizando la conexión a través de este puerto, se pudieron observar en el PC impresiones en consola realizadas con printf por una aplicación programada en el MCU. Esta función además fue útil a la hora de depurar la aplicación RSItrust.

Sensores de presencia

Utilizando impresiones en consola a través del puerto UART, se implementó una aplicación que, según lo interpretado en el GPIO donde está conectado el sensor de presencia del EC-05 indica si hay o no un sensor conectado. Esta prueba dio los resultados esperados, pudiendo distinguir si hay sensor conectado, función que se utiliza en la aplicación desarrollada en el marco de este proyecto como se explica en el capítulo 5.

Diseño final

Se realizó un nuevo diseño, donde se corrigieron los errores cometidos y detectados en el prototipo fabricado. Se le agregó un pin conectado a la alimentación en el conector para comunicación UART, pudiendo de esta manera alimentar al circuito a través del mismo mientras se esté utilizando durante el desarrollo. Se agregó además un punto de test a la salida del conversor para poder medir fácilmente la tensión a la cual se está alimentando el circuito. Se modificó el tamaño de la placa ajustándolo al área utilizada para los componentes y ruteo de pistas.Tener en cuenta que el tamaño de la placa fue cambiado sin afectar al layout para colocarlo dentro de la caja seleccionada, pensando en la realización de recintos a medida para su protección en aplicaciones futuras. Naturalmente, al realizar cajas a medida esta placa podrá tener nuevas modificaciones ajustándose a requerimientos del proveedor.

La Figura 4.18 muestra el circuito final, mientras que la Figura 4.19 muestra el PCB diseñado.

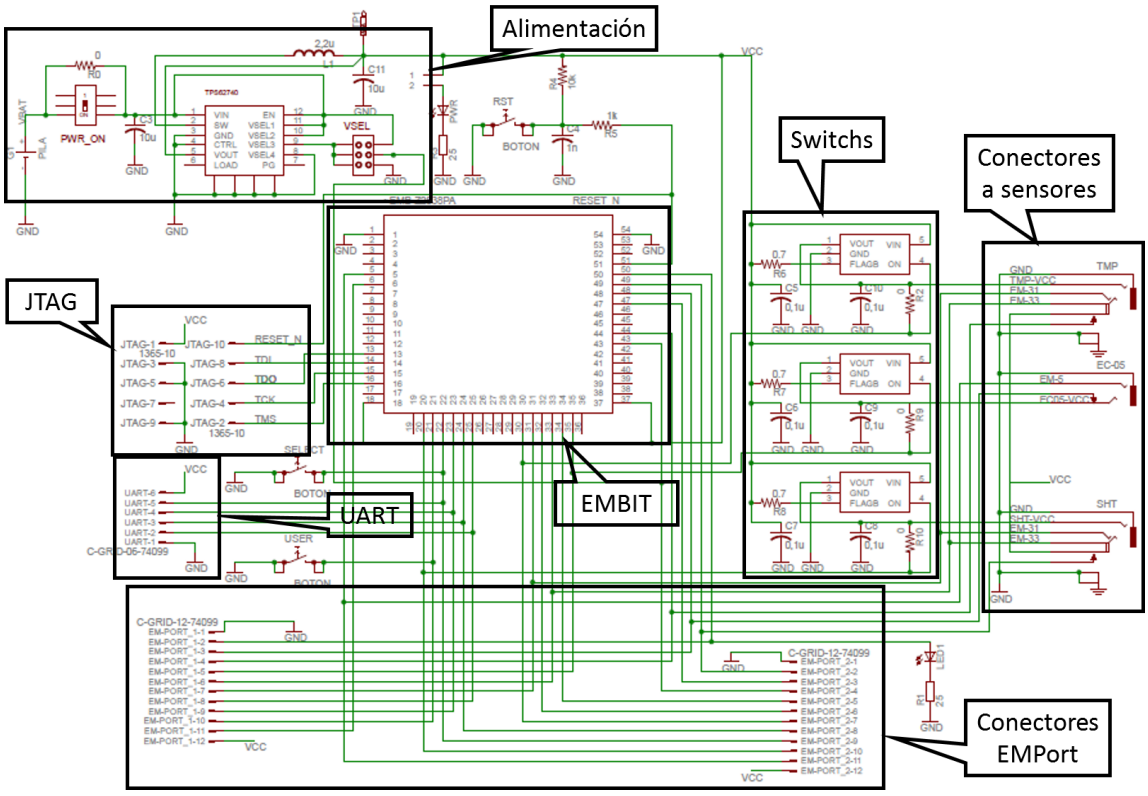
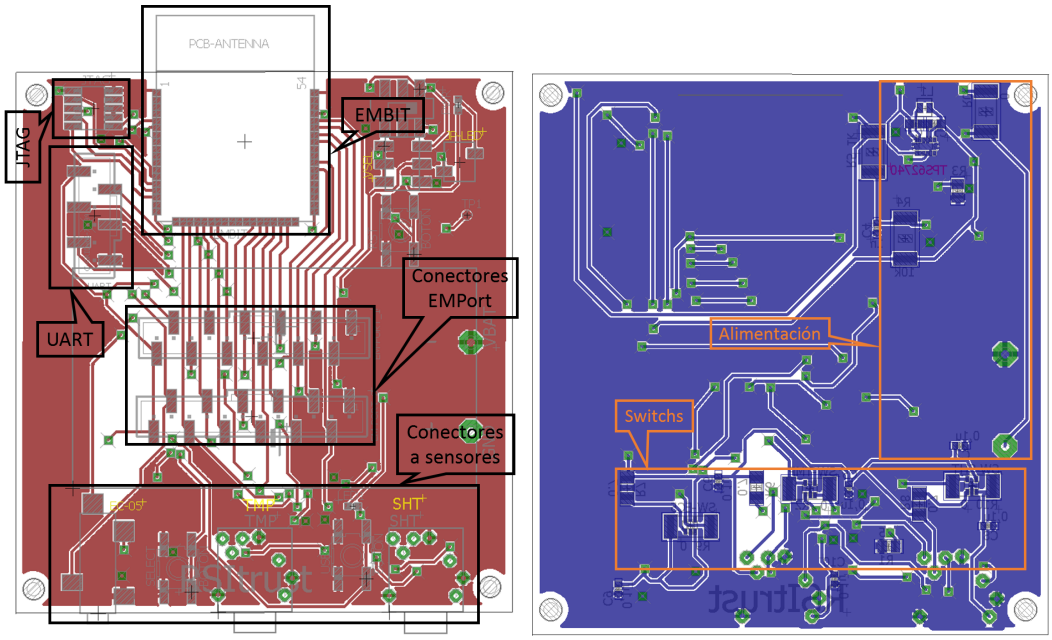


Figura 4.18: Circuito del nodo.

En el anexo B.2.1 se detalla una lista con los componentes necesarios para la construcción de un nodo, así como su correspondiente número de parte para



(a) Cara superior (b) Cara inferior

Figura 4.19: PCB diseñado

realizar la compra en Digi Key `http://www.digikey.com/`. También se dan indicaciones para el proceso de construcción, desde la compra de componentes y fabricación de impreso, soldadura de componentes y pruebas de hardware.

4.6. Conclusiones

Se logró el diseño de un nodo, el cual cumple con los objetivos principales, obteniendo un diseño que aporta a la reducción de consumo del sistema.

Debido a la implementación de switches que habilitan la alimentación de los sensores se logra proteger el MCU, pieza fundamental del circuito, ante posibles fallas provenientes de los sensores.

Si bien no se pudo comprobar el correcto funcionamiento del conversor DC/DC en el diseño propuesto, este posibilita alimentar el circuito a diferentes tensiones fijas y estables, adaptándose así a cada aplicación en particular logrando una reducción de consumo energético al poder disminuir la tensión de alimentación. A esto se agrega la posibilidad de cambiar la tensión de alimentación en tiempo de ejecución.

Utilizando los sensores de presencia con los que cuentan los conectores seleccionados para la integración de los sensores al nodo, es posible generar una señal de falla en caso de que no haya un sensor conectado.

El diseño prevé elementos extra a los utilizados en RSItrust, los cuales se mencionan a lo largo del capítulo; de esta manera el diseño es fácilmente adaptable para su utilización en otras aplicaciones, o como parte de la realización de pruebas durante su desarrollo.

El lector puede contar con los elementos necesarios para la construcción del nodo, utilizando como guía el anexo B.2.1. De esta manera, el hardware desarrollado en este proyecto es fácilmente reproducible y adaptable a las necesidades particulares de nuevas aplicaciones que lo necesiten.

Capítulo 5

Software

5.1. Introducción

Para lograr los objetivos del proyecto fue necesario contar con un sistema operativo y un *stack* de protocolos que permitieran desarrollar las funcionalidades requeridas usando de forma eficiente los recursos del nodo. Para ello se utilizó el sistema operativo Contiki y los protocolos descritos en la sección 3.4.

La aplicación desarrollada está compuesta por un módulo principal (*rsitrust-server*), módulo que contiene un proceso (*process_medir*) y seis recursos de CoAP (*res-conf, res-energest, res-vecinos, res-humS, res-humA, res-temp*). El módulo principal se encarga de inicializar el motor de CoAP, activar los recursos, configurar los puertos usados y dar comienzo al proceso *process_medir*. Este proceso es el encargado de tomar medidas periódicamente de los tres sensores y almacenar la última medida en la memoria del MCU. Los recursos permiten al usuario configurar diferentes parámetros de los nodos sensores, obtener las medidas tomadas y acceder a algunos datos que brindan información sobre el nodo.

En este capítulo se presentará una descripción funcional de la aplicación, se mostrará su diseño e implementación y finalmente se describirán las pruebas realizadas para verificar el cumplimiento de los requerimientos funcionales.

5.2. Descripción funcional de la aplicación

La red se forma con varios nodos sensores que actúan como servidores[1], y un nodo base o *root* que funciona como *gateway* conectado a una PC que actúa como cliente[2]. Esta PC permite que el usuario envíe comandos de configuración a los

[1]El servidor es un nodo que almacena información y la envía al cliente a través de la red.

[2]El cliente le solicita al servidor información que éste almacena.

nodos de la red utilizando el protocolo CoAP. A su vez, puede solicitar el valor de los parámetros configurados y el estado de algunas variables que almacenan información sobre el estado de la red y los nodos.

Los nodos sensores toman periódicamente medidas de cada uno de los tres sensores y la almacenan en memoria junto con una marca temporal o *timestap* que indica en qué momento fue tomada la medida. Esta información es enviada al cliente cuando éste lo solicita. Por ejemplo, si el usuario quiere saber cuál es la última medida de humedad del suelo registrada en uno de los nodos, basta con enviar una solicitud de esa variable al nodo para que éste le responda con la medida junto con la marca temporal. De forma similar, el cliente puede solicitar que se le envíe la medida de forma periódica, sin necesidad de solicitarlo en cada vez.

5.3. Diseño

Inicialmente se probaron los ejemplos de CoAP para Contiki utilizando la plataforma Tmote Sky aprovechando que esta plataforma es soportada por el simulador de red Cooja [5]. Posteriormente se comenzó la implementación de la aplicación desarrollando el código para esta plataforma, para finalmente migrar al CC2538.

Para poder solicitar las medidas, información de la red y enviar comandos desde el nodo base, se utilizaron diferentes recursos de CoAP que permiten realizar este tipo de tareas de forma sencilla.

Durante la etapa de diseño de la aplicación se decidió tomar las medidas de los sensores usando un proceso de Contiki. En un principio se había optado por tomar la medida del sensor dentro del *callback* o *handler* del recurso correspondiente, sin utilizar procesos (ver sección 3.3.2). Es decir, un recurso leía un registro del sensor, lo procesaba para obtener una medida y luego enviaba la medida a través de la radio. Para hacer esto de forma eficiente desde el punto de vista energético se debían alimentar los sensores en el momento que el nodo recibía la solicitud de la medida a través de la radio. Una vez alimentado el sensor es necesario esperar un tiempo determinado para que se pueda tomar una lectura del sensor. La forma de esperar este tiempo sin usar un proceso implica mantener ocupado el procesador durante ese tiempo. En la práctica se observó que esto afectaba la comunicación. Lo que sucede es que, al permanecer el procesador ocupado por mucho tiempo, no se puede manejar debidamente el control de la radio. Por este motivo se decidió utilizar un proceso para tomar las medidas con dos temporizadores *etimer* (ver sección 3.3.5). Uno de ellos se configura con el período de muestreo y el otro con el tiempo de espera entre el encendido de los sensores y su lectura. Periódicamente el proceso toma medidas de los sensores de manera secuencial. Para ello primero enciende la alimentación de los tres sensores. Luego espera un tiempo fijo determinado por el temporizador de espera. A continuación, toma las medidas del sensor TMP75C, luego del SHT25 y posteriormente del EC-05. Finalmente deshabilita

la alimentación de los tres sensores y setea el temporizador del tiempo de muestreo. En la Figura 5.1 se muestra un diagrama de flujo del proceso *process_medir* implementado. Dado que el prototipo de la placa fabricado no cuenta con el conversor DC/DC ni los conectores con sensor de presencia para los sensores SHT25 y TMP75C, las líneas de código relacionadas con sus pines se dejaron comentadas y no fueron probadas, como se explica más adelante.

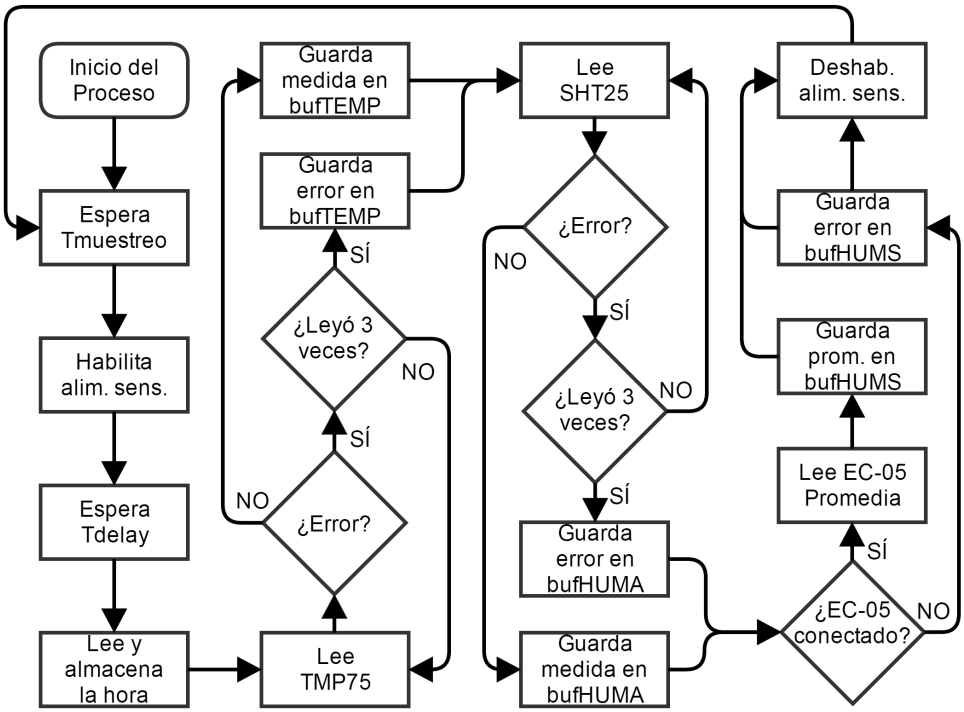


Figura 5.1: Diagrama de flujo del proceso *process_medir*

El recurso *res-conf* es el que permite configurar parámetros utilizando los métodos de CoAP PUT y POST los cuales se describen en la sección 3.4.7. Los parámetros que se pueden configurar en cada nodo son: el período de muestreo de los sensores, el período de observación de los recursos de los sensores, la hora actual y rangos de temperatura y humedad. Aprovechando que CoAP soporta la utilización de *queries*[3], para configurar los parámetros se definió que el nodo reciba en el campo *query* del mensaje, cuál es el parámetro que se desea configurar y en el payload el valor a ser configurado. Esto se hizo así siguiendo el ejemplo res-leds (ubicado en *contiki/examples/er-rest-example/resources*) que permite encender o apagar diferentes LEDs de la plataforma a partir del texto enviado en el campo *query* del mensaje. Además el recurso responde a solicitudes realizadas mediante

[3]Es la parte de una URL que contiene los datos que deben pasar a aplicaciones web.

el método GET, enviando en el payload del mensaje los valores configurados de hora, período de muestreo y períodos de observación de los tres sensores.

Para brindarle al usuario información relevante sobre el estado del nodo se implementó el recurso *res-energest*. Éste responde al método GET con los valores de los tiempos registrados por la herramienta Energest de Contiki descritos en la sección 3.3.6. Otro recurso que responde al método GET con información del nodo es *res-vecinos*. Este recurso, al recibir una solicitud, envía en el *payload* de la respuesta las direcciones IPv6 de sus vecinos de la red y su tabla de rutas.

Los recursos *res-humS*, *res-humA*, *res-temp* son los responsables de enviar las últimas medidas tomadas por los sensores. Estos recursos responden a solicitudes GET y OBSERVE con la última medida almacenada y una marca temporal de la hora a la cual fue tomada la medida. Si hubo un error en la medida el nodo sensor envía un mensaje con el estado *SERVICE UNAVAILABLE* de CoAP y el *string ERROR SENSOR* en el *payload* del mensaje.

Para comunicarse con los nodos sensores se utilizó un nodo conectado a la PC. Este nodo se programa con la *app*[4] *rpl-border-router* que se encuentra en *contiki/examples/ipv6/rpl-border-router* y permite la comunicación entre la RSI (con el *stack* de protocolos descrito en el capítulo 3) y la PC.

En la Figura 5.2 se muestran diferentes ejemplos de comunicación. La PC, a través del border router, realiza una solicitud de la temperatura al Nodo 1 y éste responde con dicha medida junto con la hora de la medición. Otro ejemplo que se muestra es la configuración de la hora al Nodo 2 utilizando el método PUT. Finalmente se ejemplifica cómo el Nodo n envía periódicamente las medidas al nodo base al ser solicitado mediante el método OBSERVE.

5.4. Implementación

La aplicación RSItrust está ubicada en *contiki/examples/rsitrust*. Sus módulos principales están distribuidos como se indica en la Figura 5.3. Además, se debieron implementar y modificar archivos indicados en la Figura 5.4. Para el desarrollo de la aplicación se tomó como punto de partida el ejemplo *er-rest-example* que se encuentra en *contiki/examples/er-rest-example*. En lo que sigue del capítulo se explica detalladamente cómo se implementó la aplicación.

CoAP está implementado en Contiki mediante el motor llamado *erbium*. Sus archivos principales se encuentran en el directorio *contiki/apps/er-coap*. La implementación de CoAP está formada por varios módulos como por ejemplo *er-coap*,

[4]En este texto se le llama aplicación a la desarrollada por el grupo del proyecto y *app* a las aplicaciones propias de Contiki.

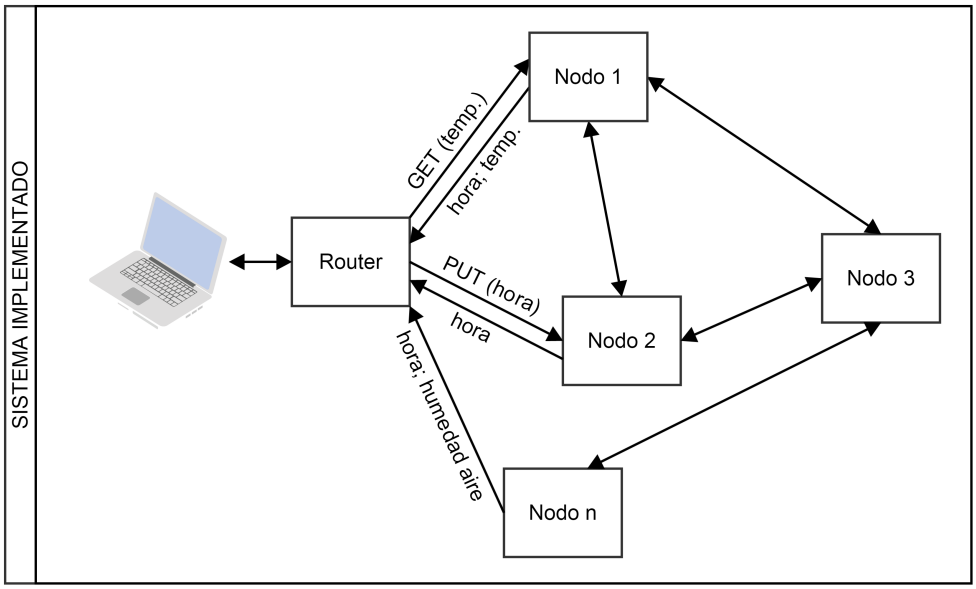


Figura 5.2: Ejemplo de comunicación con GET, PUT y OBSERVE.

er-coap-engine y *er-coap-observe*. Para crear una aplicación que utilice CoAP es necesario incluir esta aplicación en el *Makefile*. Para la implementación de las aplicaciones que usan CoAP se suele crear un archivo *res-nombre.c* por cada recurso. Además es recomendable, para mantener el código ordenado, crear una carpeta dentro de la carpeta del proyecto para guardar todos los recursos (separados del archivo principal de la aplicación).

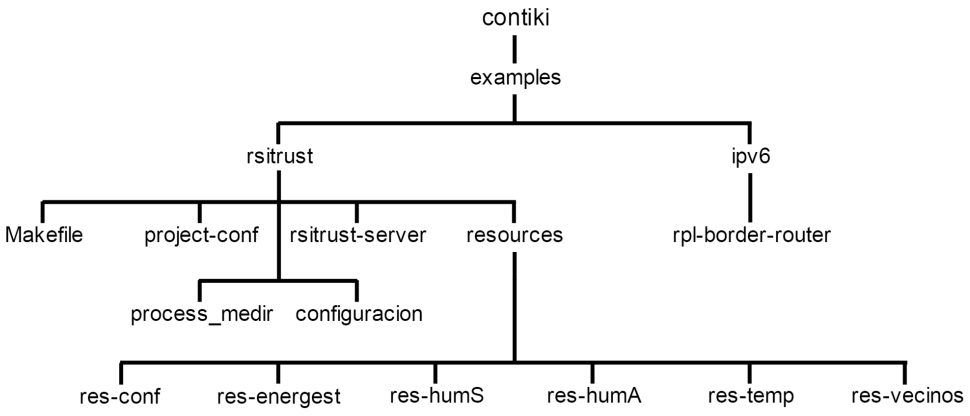


Figura 5.3: Principales archivos de la aplicación.

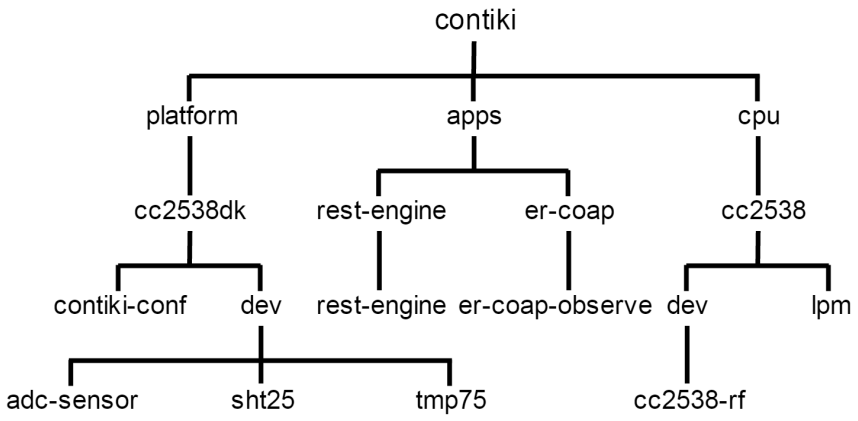


Figura 5.4: Archivos secundarios de la aplicación.

5.4.1. Makefile, project-conf y proceso principal

En el archivo *Makefile* del proyecto se le indica al compilador cómo compilar y generar las dependencias del programa. En el caso de la aplicación RSItrust se indica que se compile el archivo *rsitrust-server.c* y los archivos que se encuentran en la carpeta *resources* debajo de la carpeta del proyecto. Además, como se explicó en la sección 5.4, es necesario incluir la aplicación de CoAP. Esto se hace con las siguientes líneas:

APPS += er-coap
APPS += rest-engine

En el archivo *project-conf.h* se definen algunos parámetros de configuración general del proyecto. Por ejemplo, se modifican los parámetros de los protocolos del *stack* explicados en el capítulo 6:

#undef NETSTACK_CONF_RDC_CHANNEL_CHECK_RATE
#define NETSTACK_CONF_RDC_CHANNEL_CHECK_RATE 4

#undefRPL_DIO_INTERVAL_DOUBLINGS
#define RPL_DIO_INTERVAL_DOUBLINGS 20

Finalmente, en el archivo *rsitrust-server.c* donde reside el proceso principal de la aplicación se activan los recursos llamando a la siguiente función:

*void rest_activate_resource(resource_t *resource, char *path);*

Donde el primer parámetro es el recurso declarado en el archivo correspon-

diente al recurso y el segundo el *string* que define la ubicación del recurso en el servidor. En este archivo también se setean los pines GPIO necesarios para habilitar los switches conectados a la alimentación de los sensores y los pines sensores de presencia de los conectores de 3,5mm (ver capítulo 4). Posteriormente, dentro del proceso principal, se inicia el proceso *process_medir* y se ingresa en un loop a la espera de algún evento del sistema:

```
/* Se inicia el proceso que toma las medidas periódicamente */
process_start(&process_medir, NULL);
while(1) {
PROCESS_WAIT_EVENT();
}
```

5.4.2. El proceso process_medir

Como se mencionó anteriormente este proceso es el encargado de tomar las medidas de los sensores de forma periódica. Al comienzo del proceso se setea un *etimer* (ver sección 3.3.5) con el valor guardado en la variable *T_muestreo*, configurada mediante el recurso *res-conf* como se explica en la sección 5.4.3. Esta variable representa el tiempo en minutos que transcurre entre medidas, durante el cual se libera el procesador para realizar otras tareas (como por ejemplo manejar la radio). Como se explica en la sección 5.3 fue fundamental utilizar un proceso y un temporizador para poder liberar el procesador. Luego de seteado el temporizador se espera a que expire para continuar con la ejecución del proceso:

```
/* Se espera durante un tiempo T_muestreo para realizar las medidas */
etimer_set(&muestreo, (T_muestreo * CLOCK_SECOND * 60));
PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&muestreo));
```

Luego se alimentan los sensores activando los switches, se esperan 30ms y se almacena la hora actual del nodo usando la función clock_seconds() y sumando la hora inicial T_inicio configurada utilizando el recurso *res-conf* (ver sección 5.4.3). Este retardo de 30ms es necesario para que el sensor SHT25 (el más restrictivo) tenga sus registros disponibles para ser leídos por el MCU.

Posteriormente se lee el pin de presencia del conector del sensor TMP75C. En caso de estar conectado se activa el sensor TMP75C utilizando la función *SENSORS_ACTIVATE(tmp75)* definida en el archivo *tmp75.c* como se explica en la sección 5.4.5. Posteriormente se lee el valor de temperatura del sensor usando la función *tmp75.value(TMP75_VAL_TEMP)*, se convierte a string y se almacena en el arreglo *bufTemp*. Si se detecta un error en la comunicación con el sensor se vuelve a leer. Si hubo error en la lectura tres veces consecutivas se guarda el *string ERROR SENSOR*. Luego de guardado el valor de temperatura se desactiva el sensor.

De forma similar se activa y se lee el sensor SHT25, guardando el valor leído en la variable *bufHUMA*. Como el sensor SHT25 también permite medir la temperatura ambiente, se dejó previsto un código comentado para leer la temperatura con este sensor.

Finalmente se lee y almacena la medida de humedad del suelo mediante el sensor EC-05. En este caso, como el sensor necesita 2,5V para funcionar, primero se aumentaría la tensión seteando el pin V_{SEL3} del conversor DC/DC. Como la interfaz del sensor es analógica, la lectura se hace 8 veces usando la función *adc_sensor.value(ADC_SENSOR_HUM)* y luego se hace un promedio. Se decidió hacer la medida 8 veces porque dividir entre un múltiplo de 2 equivale a hacer un desplazamiento de bits hacia la derecha, requiriendo menor cantidad de ciclos de reloj que una división entre otro número. Luego de tomado el valor promediado de la medida del sensor se realiza su conversión para obtener la humedad relativa mediante la ecuación de la hoja de datos del sensor.

Una vez medidos los tres sensores y almacenados los valores leídos (o el string *ERROR SENSOR* si hubo algún error) se deshabilitan los pines GPIO de alimentación de los sensores, se resetearía el pin V_{SEL3} del conversor DC/DC y se vuelve a iniciar el contador T_muestreo. El ciclo anterior se repite indefinidamente ya que se encuentra dentro de la instrucción *while(1)*.

En un principio se consideró la posibilidad de encender los sensores de a uno. Se probó encender un sensor, esperar el tiempo que necesita para tener una medida válida, adquirir y procesar su medida, apagarlo y encender el siguiente. Sin embargo, se encontró la complicación de que el sensor TMP75C enviara una temperatura erronea de manera aleatoria. Se probó incrementar el tiempo de espera luego de encender dicho sensor, disminuyendo así las medidas erróneas, pero no logrando solucionar el problema en un 100 %. Debido a esto se optó por encender los sensores al mismo tiempo, obteniendo medidas correctas en todo momento. Una mejora a futuro que no se realizó por motivos de tiempo, sería separar cada sensor en un proceso distinto. En este caso, un proceso encendería un sensor y mientras se espera el tiempo que el sensor necesita para obtener una medida válida el MCU iría a otro proceso donde podría por ejemplo encender otro sensor. En este caso podrían llegar a estar los tres sensores encendidos a la vez. Esto agregaría versatilidad, pudiendo medir cada sensor con un período distinto dependiendo de las variaciones de la medida o de la criticidad se la misma.

Debido a que el prototipo de la placa fabricado no cuenta con el conversor DC/DC ni los conectores con sensor de presencia para los sensores SHT25 y TMP75C las líneas de código relacionadas con los pines asociados a estos componentes se dejaron comentadas.

5.4.3. Recursos de CoAP

Un recurso de CoAP se define utilizando una variable de tipo *resource_t*. Esta estructura está formada principalmente por punteros a funciones. Algunas de estas funciones son *get_handler, post_handler, put_handler, delete_handler* y se deben implementar aquellos métodos soportados por el recurso (GET, POST, PUT, etc.). La inicialización de esta estructura se realiza utilizando la función *rest_activate_resource(resource_t *resource, char *path)*, donde el primer parámetro es el recurso declarado y el segundo es el *string* que define la ubicación del recurso en el servidor.

Configuración

Para realizar la configuración de parámetros del nodo a través de la PC se crea en los nodos sensores el recurso *res-conf*. Las funciones auxiliares de este recurso se encuentran en el módulo *configuración*. El recurso responde a los métodos GET, POST y PUT. Para esto fue necesario implementar los respectivos *handlers*.

El *handler res_get_handler* se encarga de generar una respuesta con los valores de los parámetros que tiene configurados el nodo. La respuesta contiene la hora configurada seguida de los valores de período de observación de los tres sensores y el período de muestreo. Luego, dentro del *handler*, se indica que la respuesta contiene texto plano[5] y se guarda en el *payload* del mensaje la respuesta creada.

Para poder realizar la configuración de parámetros se implementó el *handler res_post_put_handler*. El nodo recibe un paquete que en su campo *query* indica qué parámetro se quiere configurar y en su *payload* contiene el valor que se desea cargar en ese parámetro. Usando esta información se escribe la variable asociada al parámetro indicado. Por ejemplo, si el campo *query* del mensaje contiene el *string* "conf=periodoM" y el *payload* "value=15" se va a configurar el período de muestreo con el valor de 15 minutos. Los parámetros que permite configurar este recurso son:

- Período de muestreo de los sensores (cada cuántos minutos se toma una medida de los 3 sensores).
- Período observación de la humedad del aire (cada cuántos minutos se envía un mensaje a los suscriptores del recurso humedad del aire).
- Período observación de la humedad del suelo.
- Período observación de la temperatura.

Además de los parámetros anteriores se pueden configurar máximos y mínimos para los valores de temperatura y humedad. Si bien en el caso de la aplicación

[5]Compuesto únicamente por texto sin formato, sólo caracteres.

RSItrust estos valores no son usados pueden ser utilizados en futuras aplicaciones para disparar alarmas en caso de que la medida quede por fuera de dicho rango.

Todos los parámetros configurados, exceptuando la hora, se almacenan en variables tal cual son recibidos en el *payload*. La hora configurada se almacena con la siguiente ecuación:

T_inicio = T_unix - clock_seconds();

Donde T_unix representa la hora en formato Unix[6] enviada en el *payload* del mensaje y *clock_seconds()* es una función que obtiene los segundos transcurridos desde que se encendió el MCU. Al almacenar este valor se puede obtener la hora actual del nodo en cualquier instante posterior a la configuración a partir de la ecuación:

hora = T_inicio + clock_seconds()

Esta ecuación es la usada en el proceso *process_medir* para obtener la hora actual. En el anexo B.4 se detalla cómo configurar cada uno de los parámetros y cuál es el query utilizado en cada caso.

Energest

Con la finalidad de usar la herramienta Energest de forma remota se implementó el recurso *res-energest*. Esta herramienta devuelve en *ticks* del *rtimer* las variables explicadas en la sección 3.3.6. Dentro del recurso, estos parámetros se convierten a segundos dividiendo por la variable de Contiki *RTIMER_ARCH_SECOND*. Dicha variable varía dependiendo del MCU, siendo para el CC2538 $RTIMER\_ARCH\_SECOND = 32768$. El recurso *res-energest* simplemente responde al método GET de forma similar a los otros recursos pero con los valores en segundos. Esta conversión a segundos no solo facilita la interpretación de los resultados, si no que permite adquirir valores durante más tiempo sin llegar al overflow. Las variables donde se guardan estos tiempos son de tipo *unsigned long* , las cuales son de $32bits$. Eso significa que se puede registrar hasta $2^{32} - 1$ segundos, que es equivalente a 136 años. Si se hubieran registrado los valores en *ticks* del *rtimer*, se llegaría al *overflow* a las 36 horas, lo cual no es útil para la aplicación.

Vecinos

Otro dato que puede ser relevante al formar una RSI es la tabla de rutas y los vecinos de los nodos. Para obtener esta información y brindarle a la aplicación una

[6]Se define como la cantidad de segundos transcurridos desde la medianoche del tiempo universal coordinado del 1 de enero de 1970.

herramienta útil para la puesta a punto de la RSI se creó el recurso *res-vecinos*. En este recurso se implementó el *handler res_get_handler* para que envíe todos los vecinos y tablas de rutas que tiene almacenados el nodo sensor. A diferencia de los recursos explicados anteriormente en este caso la información no puede ser enviada en el *payload* de un solo mensaje debido a la longitud del mensaje, por lo tanto debe enviarse dividida en varios mensajes. Esto se logra gracias al campo *offset* del *handler*, el cual permite indicar cuándo hay más información para enviar. Para implementar este recurso se tomó como referencia el ejemplo *res-chunks* que se encuentra en *contiki/examples/er-rest-example/resources*.

Para poder enviar las direcciones de los vecinos y las tablas de rutas es necesario incluir en el recurso el módulo *uip-ds6* ubicado en *contiki/net/ipv6/* donde se definen las estructuras *uip_ds6_route_t* y *uip_ds6_nbr_t*. En estas estructuras están definidas la tabla de rutas y las direcciones de los vecinos del nodo. Dentro del recurso se crean variables de estos dos tipos de estructuras y se forma un *string* con la información que se desea enviar en el *payload* del mensaje.

Sensores

Como se mencionó anteriormente, para enviar las últimas medidas tomadas por los sensores se implementaron los recursos *res-humS, res-humA y res-temp*. Estos recursos responden a los métodos GET y OBSERVE con la última medida almacenada y una marca temporal de la hora a la cual fue tomada la medida. A diferencia de los otros recursos implementados, éstos son del tipo *periodic_resource_t*. Por ejemplo, el recurso *res-temp* se define con el siguiente macro:

```
PERIODIC_RESOURCE(res_temp,
"title=" Sensor Temperatura ";rt= "res-temp, GET, OBS "",
res_get_handler,
NULL,
NULL,
NULL,
60 * CLOCK_SECOND,
res_periodic_handler);
```

En el caso de los recursos periódicos es necesario indicar en uno de los miembros el período de observación para el envío de los mensajes de respuesta a las solicitudes realizadas mediante OBSERVE. En este caso, como puede observarse en el macro anterior, este tiempo se configuró en un tiempo fijo de un minuto. En este macro se indica también que el recurso responde a GET y OBSERVE. En la sección 5.4.4 se explica cuáles fueron las modificaciones que se hicieron en la implementación de CoAP para lograr que este parámetro se pueda modificar en tiempo de ejecución.

El *handler res_get_handler* es el encargado de formar el mensaje de respuesta

al mensaje del tipo GET u OBSERVE. Este mensaje contiene una marca temporal seguida de la última medida almacenada en el MCU. Si la última no fue almacenada correctamente esto se indica en el *payload* del mensaje con el string *ERROR SENSOR* y se envía el mensaje con estado *SERVICE UNAVAILABLE*. Esto último fue pensado para que no sea necesario procesar el contenido del *payload* en el nodo base para saber si el mensaje contiene una medida válida.

Finalmente el *handler res_periodic_handler* simplemente notifica periódicamente a todos los nodos suscriptos a la observación del recurso (ver anexo C.6).

5.4.4. Modificaciones a la implementación de CoAP en Contiki

Para que la aplicación sea confiable y el período de observación de las medidas se pueda modificar desde la PC, se debieron hacer modificaciones sobre la implementación de CoAP en Contiki.

En el archivo *rest-engine.c* ubicado en *contiki/apps/rest-engine/* se modificó la línea *etimer_reset(&periodic_resource->periodic_timer)* por *etimer_set(&periodic_resource->periodic_timer, periodic_resource->period)*. Haciendo este cambio se logra que el período de observación de los recursos observables se pueda modificar durante la ejecución del programa ya que se setea el temporizador en lugar de resetearlo con el valor que ya estaba configurado.

Una característica importante de CoAP es que permite agregar confiabilidad a la comunicación si se desea. Esto se logra indicando en el mensaje enviado que éste requiere confirmación por parte del receptor. El archivo *er-coap-observe.c* que se encuentra en *contiki/apps/er-coap* fue modificado para agregar confiabilidad a la comunicación de los recursos observables. Esta modificación fue implementada modificando en la inicialización del mensaje *COAP_TYPE_NON* (no confirmable) por *COAP_TYPE_CON* (confirmable).

5.4.5. Drivers de los sensores

Como se mencionó anteriormente el sistema implementado cuenta con tres sensores: el EC-05 para medir la humedad del suelo, el SHT25 para la humedad del aire y el TMP75C que mide la temperatura. El primero es un sensor de interfaz analógica mientras que los otros dos son de interfaz digital y se comunican con el MCU mediante el bus de comunicación $I2C$.

Los drivers de los tres sensores fueron implementados aprovechando las estructuras definidas en el módulo *sensors* ubicado en *contiki/core/lib* para tener mayor versatilidad en los códigos fuente. En este módulo se define la estructura *sensors_sensor* para utilizar de forma ordenada las funciones encargadas de activar

y obtener la medida de los sensores. Al definir los diferentes sensores utilizando esta estructura es necesario implementar las funciones de la estructura para cada uno de ellos.

Los sensores se activan utilizando el macro *SENSORS_ACTIVATE(sensor)*. En el caso del SHT25 y el TMP75C esto habilita la comunicación $I2C$ mientras que en el EC-05 inicializa los puertos del conversor ADC.

La implementación de la obtención de las medidas de los distintos sensores se detalla a continuación en la sección correspondiente a cada sensor.

Drivers del sensor EC-05

Para realizar la lectura del sensor EC-05 se modificaron los archivos *adc-sensor.h* y *adc-sensor.c* ubicados en *contiki/platform/cc2538dk/dev*. El sensor EC-05 se conectó a un GPIO del MCU, el cual se configuró como conversor ADC para leer la medida. A este conversor se le puede configurar el pin del cual se lee el voltaje a convertir, la cantidad de bits de la medida convertida a digital y el rango de tensión de referencia, el cual puede ser externo o interno. Para tener mayor resolución se configuró la cantidad de bits de la medida convertida en 12 bits, siendo esta la máxima cantidad de bits a la que el conversor ADC puede convertir. Además, para simplificar el diseño del hardware se decidió utilizar una referencia de tensión interna. Las configuraciones anteriores se implementaron dentro de las funciones *value* y *configure* en el archivo *adc-sensor.c*.

A partir de la hoja de datos del EC-05 [19] se obtuvo que necesita una alimentación entre $2{,}5VDC$ y $3{,}6VDC$ consumiendo $10mA$ y presenta una salida del 10 % al 40 % de la tensión de excitación. Como el sensor se alimenta a $2{,}5VDC$ la salida del sensor va a variar entre $0{,}25V$ (equivalente a 10 %de la alimentacion) y $1V$ (equivalente al 40 %). Teniendo en cuenta lo anterior, se realizó una regla de tres para poder obtener a la salida del conversor ADC la medida correcta [19]. Por otro lado, la ecuación de calibración depende en dónde se va a utilizar el sensor, suelo mineral, tierra para macetas o sustratos de cultivo. En el caso de RSItrust se utilizó la calibración para suelo mineral cuando se alimenta el sensor a $2{,}5V$, utilizando la ecuación 5.1 que se obtiene de la hoja de datos del fabricante.

$$Humedad = 11,9 * 10^{-4} * V - 0,401 \tag{5.1}$$

donde V es la señal de salida del sensor expresada en mV. Como se configura el ADC en $12bits$, el 100 % de humedad a la salida del conversor es 4095, por lo cual se obtiene la siguiente regla de 3:

$4095 - > 1000mV(100\,\%dehumedad)$
$OUT\_ADC - > mV(x\,\%dehumedad)$

$\Rightarrow mV = (1000 * OUT\_ADC)/4095$

Combinando con la ecuación de humedad 5.1 se tiene:

$$Humedad = (1,19/4095) * OUT\_ADC - 0,401 \tag{5.2}$$

Esta regla de tres es la que se hace en el proceso *process_medir* luego de obtener la medida del sensor utilizando la función *adc_sensor.value(ADC_SENSOR_HUM)*. Además, para realizar operaciones con números enteros las ecuaciones anteriores se implementaron en el código como se indica en las ecuaciones 5.3 y 5.5. Realizar operaciones con números enteros en lugar de punto flotante implica un ahorro en memoria y procesamiento del MCU.

$$ParteEntera = (OUT\_ADC * 119 * 10 - 4095 * 401)/(4095 * 1000); \tag{5.3}$$

La cual es la ecuación equivalente a:

$$OUT\_ADC * (11{,}9 * 10^{-4}/4095) - 0{,}401 \tag{5.4}$$

$$ParteDecimal = ((OUT\_ADC * 119 * 1000/(100 * 127) - 401)/10)\,\%100; \tag{5.5}$$

Comunicación de los sensores SHT25 y TMP75C

Como se mencionó anteriormente la comunicación de los sensores SHT25 y TMP75C se realiza mediante el bus $I2C$. La principal característica de $I2C$ es que utiliza dos líneas para transmitir la información; una para los datos y otra para la señal de reloj, una línea de referencia a tierra y otra para la tensión de alimentación. Las líneas se nombran de la siguiente forma:

- SDA: datos
- SCL: reloj
- GND: tierra
- VCC: alimentación

Cada dispositivo conectado al bus $I2C$ tiene una única dirección y puede ser configurado como maestro o esclavo. El dispositivo maestro inicia la transferencia de datos y genera la señal de reloj. No es necesario que el maestro sea siempre el mismo dispositivo; esta característica puede ser asignada a cualquiera de los dispositivos que tengan esa capacidad. Es por esto que al bus $I2C$ se lo denomina bus multimaestro. [16]

Cada transmisión comienza con un START (bajada del SDA mientras el SCA está en alto) y termina con un STOP (subida del SDA mientras el SCA está en alto).

Luego del START se envía la dirección del sensor con el cual se quiere comunicar (los primeros 7 bits), un bit para indicar si es lectura o escritura (1 y 0 respectivamente) y luego se recibe un reconocimiento por parte del sensor. Luego se manda un dato (el cual el sensor interpreta) y por último un STOP. Esto se puede ver en la Figura 5.5.

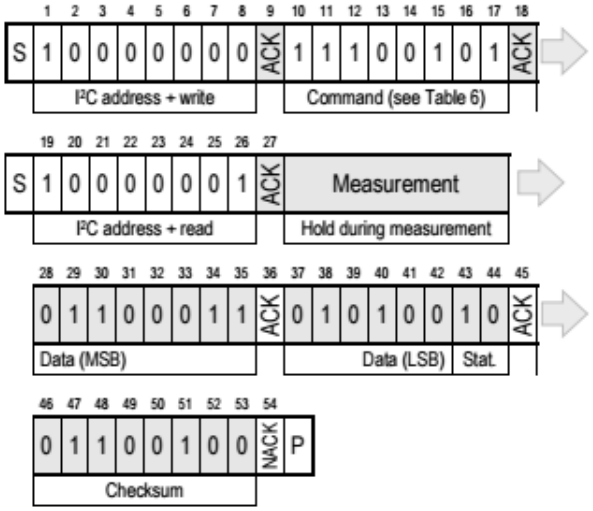


Figura 5.5: Escritura y Lectura $I2C$.

La implementación de $I2C$ para el MCU CC2538 en Contiki está compuesta por los archivos *i2c.h* e *i2c.c* y se encuentra en *contiki/cpu/cc2538/dev*[7]. Cada sensor tiene asignada una dirección fija determinada por hardware como se explica en el capítulo 4. Como se puede distinguir por software cuál de los sensores se quiere leer y no habían más GPIO libres para este propósito, ambos se conectaron al mismo puerto del MCU.

Una característica importante de los drivers utilizados para la comunicación es que permiten configurar fácilmente la velocidad. Este tipo de comunicación fue desarrollada para ser usada en distancias cortas (del orden de unos pocos centímetros) pero puede aumentarse la distancia si se reduce la frecuencia utilizada. A partir de pruebas realizadas se observó que es imposible la comunicación a alta velocidad ($400kHz$) con el sensor SHT25 conectado al MCU a través del cable seleccionado de 1.5m. Siguiendo la sugerencia en su hoja de datos se bajó la frecuencia a $100kHz$ logrando que el sensor se comunique a pesar de estar a una

[7]Estos drivers fueron publicados en github.com/contiki-os/contiki en marzo de 2015.

distancia mayor a la utilizada normalmente en la comunicación $I2C$.

Una limitante importante de los drivers $I2C$ implementados en Contiki que se observó es que, al no tener un sensor conectado, el MCU se resetea. Esto sucede porque los drivers usan un *busy-wait*[8] para determinar si el bus de comunicación está ocupado. Al no tener un sensor conectado o al perder comunicación con el sensor (por ejemplo, al cortarse el cable), el *watchdog-timer*[9] expira haciendo que se resetee el MCU. Esto se podría solucionar utilizando temporizadores y procesos en la implementación de los drivers $I2C$. De esta manera al momento de chequear que el bus está ocupado el MCU no se va a quedar trancado esperando que se libere, si no que va a setear un temporizador y va a ir a atender otros procesos. Cuando dicho temporizador llegue a cero el MCU agregará en la cola de atención el evento relacionado y cuando le llegue su turno el MCU volverá al proceso del *driver* y chequeará nuevamente la condición de si el bus está libre. En el peor caso, en el cual no hay sensor conectado en ningún momento, el MCU periódicamente volverá a verificar la condición y al no cumplirse seguirá con otros procesos no trancando el sistema y por ende el *watchdog-timer* no expirará.

SHT25

Al encender el sensor SHT25, el mismo necesita al menos $15ms$ con el SCL en alto para alcanzar el estado de reposo, siendo el consumo máximo durante el arranque de $350\mu A$.

En la Figura 5.6 se pueden ver los diferentes comandos para comunicarse por $I2C$. A su vez hay dos modos distintos de comunicación con el sensor, *Hold Master mode* o *No Hold Master mode*. En el primer caso la línea SCL es bloqueada, siendo controlada por el sensor durante el proceso de medida; mientras que en el segundo caso el SCL queda libre para realizar otra comunicación mientras se realiza la medida por parte de un sensor. *No hold master mode* permite procesar otras comunicaciones $I2C$ en el mismo bus. Dichos modos son indicados con los comandos de la Figura 5.6. El comando *soft reset* reinicia el sistema y el mismo comienza a funcionar con la configuración por defecto. Este Reset tarda menos de $15ms$.

Siguiendo los pasos de comunicación, se inicia con un START, se manda la dirección del sensor con el cual se quiere comunicar (los primeros 7 bits), un bit para indicar si es lectura o escritura (1 y 0 respectivamente) y luego se recibe un ACK por parte del sensor. Luego se manda el comando de la tabla el cual indica el modo y el registro a leer o escribir (temperatura o humedad), y por último un STOP. Esto también se puede ver en la Figura 5.5.

[8] Chequea constantemente si una condición es verdadera.

[9] Es un temporizador que se resetea durante la ejecución normal del programa y si expira hace que se resetee el MCU.

Command	Comment	Code
Trigger T measurement	hold master	1110'0011
Trigger RH measurement	hold master	1110'0101
Trigger T measurement	no hold master	1111'0011
Trigger RH measurement	no hold master	1111'0101
Write user register		1110'0110
Read user register		1110'0111
Soft reset		1111'1110

Figura 5.6: Comandos I2C, donde T= temperatura y RH (Humedad relativa).

La configuración del sensor se realiza mediante sus registros, pudiendo ver el significado de cada bit en la Figura 5.7. Se configuraron de tal manera de tener menor resolución, lo que es equivalente a menor consumo y se utiliza el modo *hold master* el cual venía configurado por defecto. Para RSItrust utilizar *no hold master* no aporta al sistema, ya que las solicitudes a los sensores se hacen todas en el mismo proceso y el MCU procesa la medida antes de comunicarse con otro sensor. En el archivo *contiki/platform/cc2538/dev/sht25.h* se define la configuración que se envía al iniciar una comunicación $I2C$.

Bit	# Bits	Description / Coding	Default
7, 0	2	Measurement resolution '00': RH 12 bit, T 14 bit '01': RH 8 bit, T 12 bit '10': RH 10 bit, T 13 bit '11': RH 11 bit, T 11 bit	'00'
6	1	Status: End of battery[14] '0': VDD > 2.25V '1': VDD < 2.25V	'0'
3, 4, 5	3	Reserved	
2	1	Enable on-chip heater	'0'
1	1	Disable OTP Reload	'1'

Figura 5.7: Configuración del sensor.

La implementación de los drivers del sensor SHT25 se hizo a partir de los drivers desarrollados por el fabricante Zolertia para la plataforma Z1. Esta plataforma cuenta con un MCU MSP430 por lo que estos códigos debieron ser modificados para utilizar las funciones correspondientes a los drivers de $I2C$ para el MCU

CC2538. Además, como el sensor cuenta con la posibilidad de hacer un chequeo de datos mediante comprobación de redundancia cíclica (CRC), se decidió implementar este chequeo en los drivers.

La configuración del sensor está implementada en el archivo *sht25.h*. Los parámetros más importantes configurados son los pines del MCU, el polinomio generador del código de redundancia cíclica y la dirección del sensor:

```
#define I2C_SDA_PORT                GPIO_C_NUM
#define I2C_SDA_PIN                 1
#define I2C_SCL_PORT                GPIO_D_NUM
#define I2C_SCL_PIN                 1
#define SHT25_CRC_POLYNOMIAL        0x131 // Polinomio generador: x8 +
x5 + x4 +1
#define SHT25_ADDR                  0x40
```

Para obtener la medida de temperatura primero se leen mediante la comunicación $I2C$ 3 bytes del registro de humedad. Dos de estos bytes representan el valor de humedad y el tercero representa el código para chequeo de error. Posteriormente se comprueba que no haya habido error de comunicación y se prosigue a transformar estos bytes a un valor de humedad.

A partir de la siguiente ecuación se obtiene la humedad relativa:

$$RH = -6 + 125 * \frac{S_{RH}}{2^{16}} \tag{5.6}$$

Donde S_{RH} es el valor leído del registro de humedad del sensor. Este valor de humedad (RH) corresponde a la humedad relativa sobre agua líquida. Para obtener la humedad relativa sobre hielo, el valor necesita ser transformado a humedad relativa sobre agua a una temperatura t:

$$RH_t = RH_\omega * \frac{e^{\frac{\beta_\omega t}{\gamma_\omega + t}}}{e^{\frac{\beta_i t}{\gamma_i + t}}} \tag{5.7}$$

Las unidades son %RH (porcentaje de humedad relativa) para la humedad y ^oC para la temperatura; y los coeficientes tienen los siguientes valores: $\beta_\omega = 17{,}62$ $\gamma_\omega = 243{,}12^oC$ $\beta_i = 22{,}46$ $\gamma_i = 272{,}62^oC$.

Si se quisiera medir la temperatura se debería leer el registro de temperatura y al valor obtenido del registro donde se guarda la medida, debería ser convertido utilizando la siguiente ecuación:

$$RT = -46{,}85 + 175{,}72 * \frac{S_T}{2^{16}} \tag{5.8}$$

TMP75C

Los drivers del sensor TMP75C son similares a los del SHT25 y fueron desarrollados a partir de éstos. En cuanto a su configuración su única diferencia es la dirección $I2C$. Por otro lado, el sensor TMP75C no cuenta con chequeo de error. Es por esto que sólo se necesitan leer 2 bytes del registro de temperatura. La temperatura se calcula a partir de los 12 bits más significativos de esos 2 bytes. Para realizar el cálculo primero se detecta si se trata de una temperatura negativa (representada en complemento a 2) o positiva y luego se realiza la conversión.

Por ejemplo:

$$001100100000 = 320h = 800 \Longrightarrow 800 * (0{,}0625^oC/LSB) = +50^oC \tag{5.9}$$

1110 0111 0000 en como complemento a 2: 0001 1001 0000, luego:

$$000110010000 = 190h = 400 \Longrightarrow 400 * (0{,}0625^oC/LSB) = 25^oC = (|{-}25^oC|) \tag{5.10}$$

5.4.6. Drivers de la radio

Para poder utilizar el PA/LNA de radio CC2592 descrito en el capítulo 4 se debió modificar el módulo que se encarga de los drivers de la radio, *cc2538-rf*, ubicado en *contiki/cpu* y el archivo *lpm.c* que se encuentra en *contiki/cpu/cc2538*. Estas modificaciones fueron realizadas por Javier Schandy co-tutor del presente proyecto. El amplificador aporta ganancia de transmisión obteniendo una potencia de transmisión de hasta $22dBm$ y una ganancia de recepción de $+11dB$ o $+6dB$, que se puede seleccionar desde el MCU. Existen tres pines de control en el chip CC2592 que controlan en qué estado se encuentra (*Power Down, Low Gain Mode, High Gain Mode, TX*). Manipulando adecuadamente estos pines y configurando los registros necesarios del CC2538 se logra controlar el PA/LNA de la radio. Dentro de las modificaciones realizadas en los archivos mencionados se encuentran la definición de parámetros de ganancia, escritura de registros, seteo de pines e inicialización del amplificador.

5.5. Pruebas

En esta sección se describen las pruebas realizadas para la implementación de los drivers de los sensores y posteriormente las pruebas realizadas para verificar los requerimientos funcionales de la aplicación.

5.5.1. Sensores

Dado que el sensor EC-05 tiene interfaz analógica, para probar su funcionamiento simplemente se imprimió en consola el valor leído por el MCU y su valor

convertido a humedad del suelo. Además se midió con un multímetro la tensión de la señal y se verificó con el valor desplegado en consola. Una vez logrado el correcto funcionamiento del sensor para un mismo valor de humedad se procedió a introducirlo en tierra y mojarla para observar que la medida aumentaba acorde a la tensión.

Los sensores SHT25 y TMP75C fueron probados por separado y luego se conectaron simultáneamente compartiendo el bus de datos para verificar que se pudieran comunicar como se había diseñado. Para las pruebas se utilizaron los códigos que se encuentran en las carpetas *sht25*, *tmp75* y *test-sensoresi2c*. Los códigos de prueba obtienen las medidas de los sensores cada $5s$ y las imprimen en consola.

El primer problema que se detectó con el sensor SHT25 fue que, al conectar el sensor utilizando el cable de $1{,}5m$, éste no se comunicaba correctamente con el MCU (daba error al calcular el CRC). Luego de realizar varias pruebas para eliminar problemas en el cable o la alimentación, esto se solucionó bajando la frecuencia de la comunicación a $100kHz$ como se recomienda para cables largos.

Finalmente, luego de implementados los drivers de los sensores, se conectaron ambos simultáneamente y se probaron imprimiendo en consola los valores medidos cada $5s$ y variando la humedad y temperatura. Estos valores se compararon con los medidos por una estación meteorológica de uso doméstico verificando su correcto funcionamiento.

5.5.2. Pruebas de funcionalidades de la aplicación

Para verificar el cumplimiento de los requerimientos funcionales de la aplicación se utilizó la extensión de *Mozilla Firefox*[10] llamada *Copper* (ver anexo C.7) que es un cliente de CoAP. Al probar la aplicación utilizando el nodo fabricado en una red *multihop* también se probó el correcto funcionamiento de la radio.

Durante las primeras pruebas se utilizó un MCU Tmote Sky como border router ya que resultaba más práctico al no contar por momentos con dos SmartRF06. Se notó que, aproximadamente a los 10 minutos, el nodo sensor se resetaba si se estaba observando un recurso con un período de 1 minuto y con el MCU Tmote Sky como border router. Se observó que esto también sucedía con el ejemplo original *er-rest-example* y el recurso *res-chunks*. Se probó utilizando un MCU CC2538 como border router y la aplicación funcionó correctamente por lo que se concluyó que hay algún tipo de incompatibilidad en la implementación de Contiki entre estas plataformas. Como en el presente proyecto la utilización del Tmote Sky como *border router* fue para poder contar con un nodo más durante el desarrollo y no como plataforma final, no se investigó el motivo por el cual se reseteaba el MCU.

[10]Mozilla Firefox es un navegador web libre y de código abierto.

Para probar el recurso *res-conf* se configuraron los períodos de observación, la hora y el período de muestreo enviando mensajes del tipo PUT a través del border router. En la Figura 5.8 se observa cómo configurar la hora. Notar que en el campo de la URL se escribe "?conf=hora" y en el recuadro inferior se indica el valor a configurar precedido del string "value="(esto se explica detalladamente en el manual B.3). En la Figura 5.9 se muestra la respuesta del nodo sensor confirmando que la hora fue configurada correctamente.

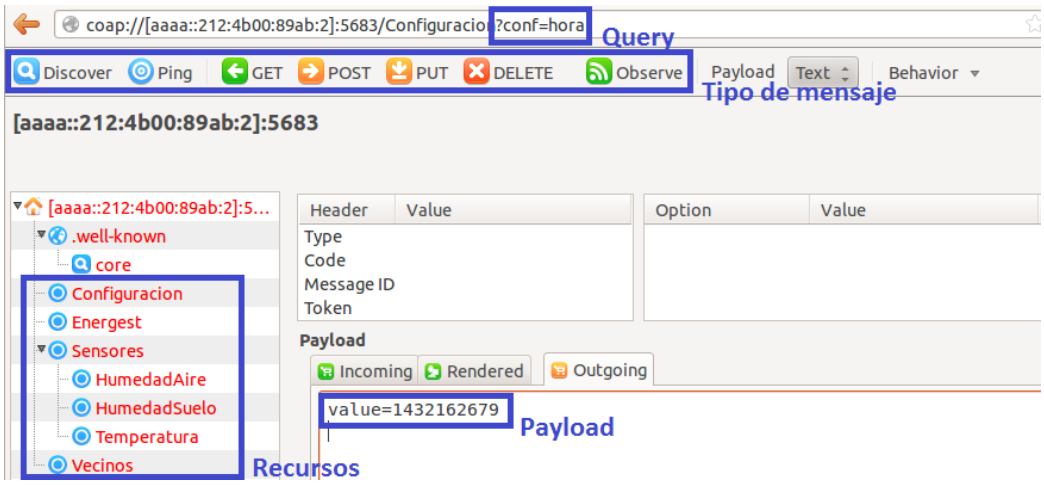


Figura 5.8: PUT (hora) al recurso Configuración.

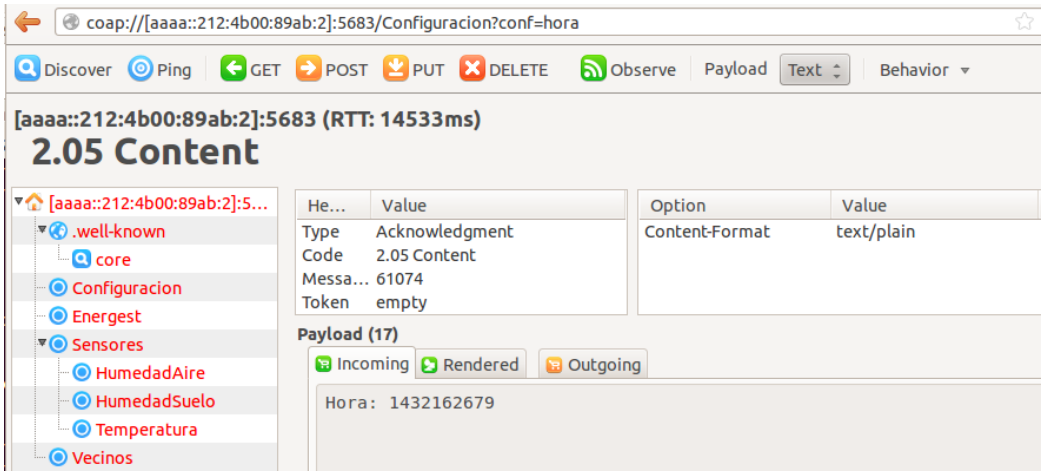


Figura 5.9: Respuesta al PUT (hora) del recurso Configuración.

El recurso *res-conf* permite también saber cuál es la configuración de los parámetros almacenada en el nodo sensor. La Figura B.12 muestra la respuesta a

una solicitud de los parámetros mediante un mensaje de tipo GET. Los parámetros enviados son la hora (en tiempo Unix), los períodos de observación y el período de muestreo.

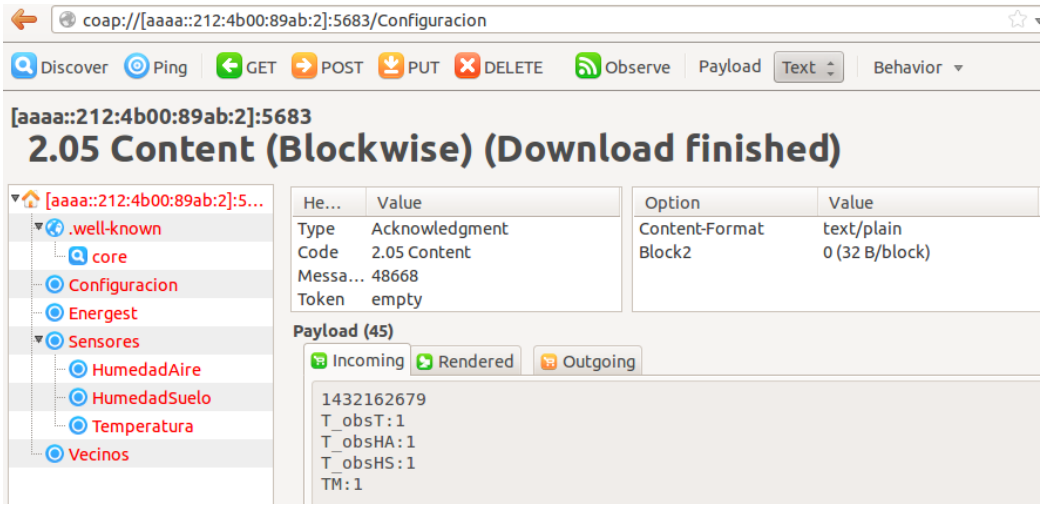


Figura 5.10: GET al recurso Configuración.

Para obtener los valores de las variables de la herramienta Energest de Contiki, basta con enviar un paquete GET al nodo sensor, el cual responde como se muestra en la Figura 5.11 con los tiempos en segundos correspondientes a CPU, LPM, LISTEN y TRANSMIT respectivamente.

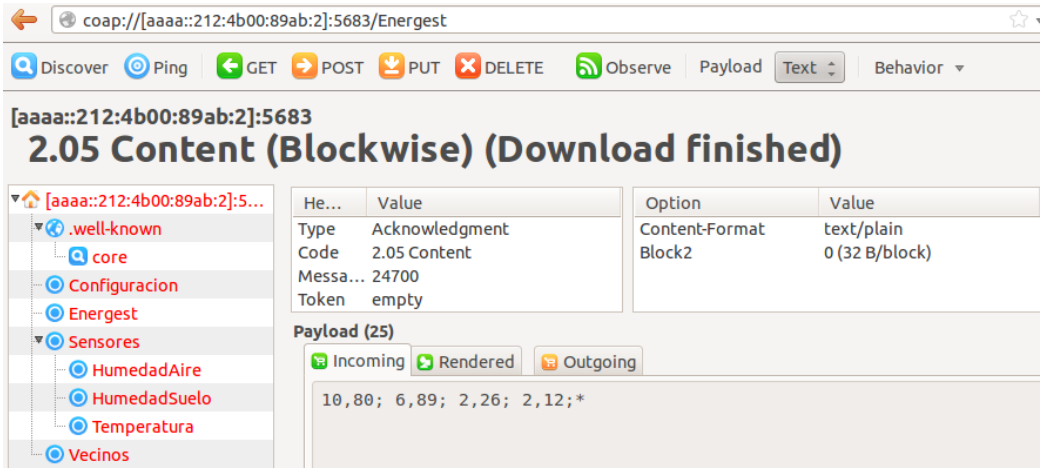


Figura 5.11: Respuesta a un GET del recurso Energest.

En cuanto a los recursos encargados de enviar las medidas, estos responden con

la medida y una marca temporal como se explicó anteriormente. En la Figura 5.12 se observa una medida tomada correctamente, mientras que en la Figura 5.13 se muestra el mensaje enviado cuando hay un error en la comunicación con el sensor.

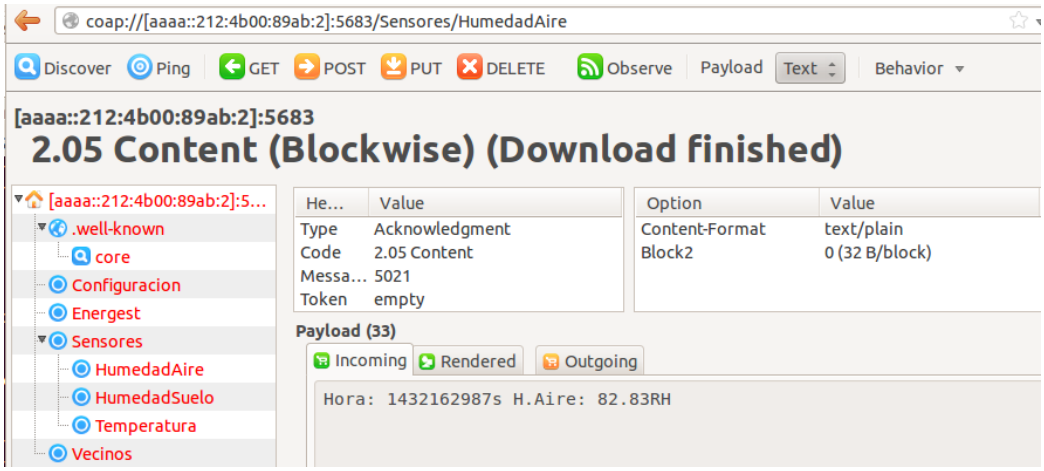


Figura 5.12: Respuesta a un GET del recurso Humedad del Aire.

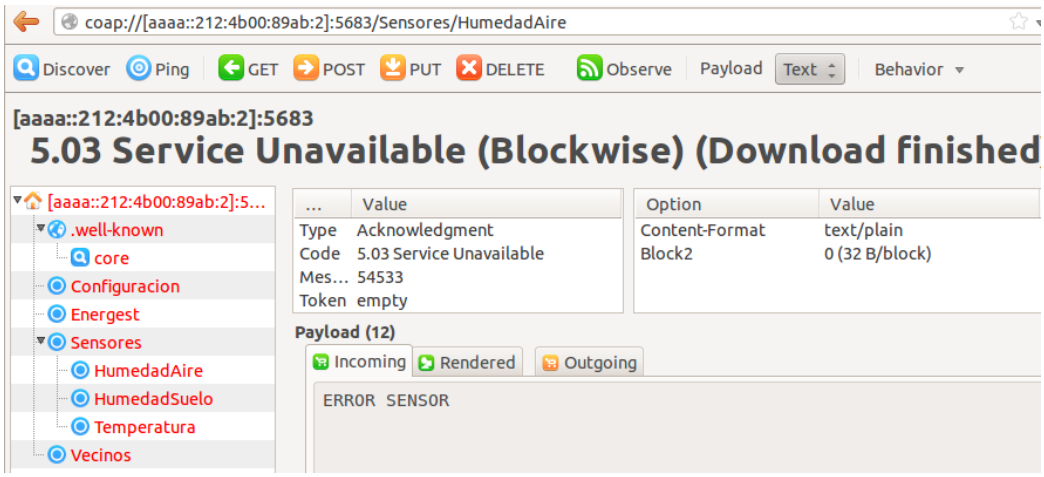


Figura 5.13: Respuesta a un GET del recurso Humedad del Aire con error.

Finalmente, como se puede ver en la Figura 5.14, un mensaje GET enviado al recurso *res-vecinos* permite conocer los vecinos de la red y rutas almacenadas en el nodo sensor.

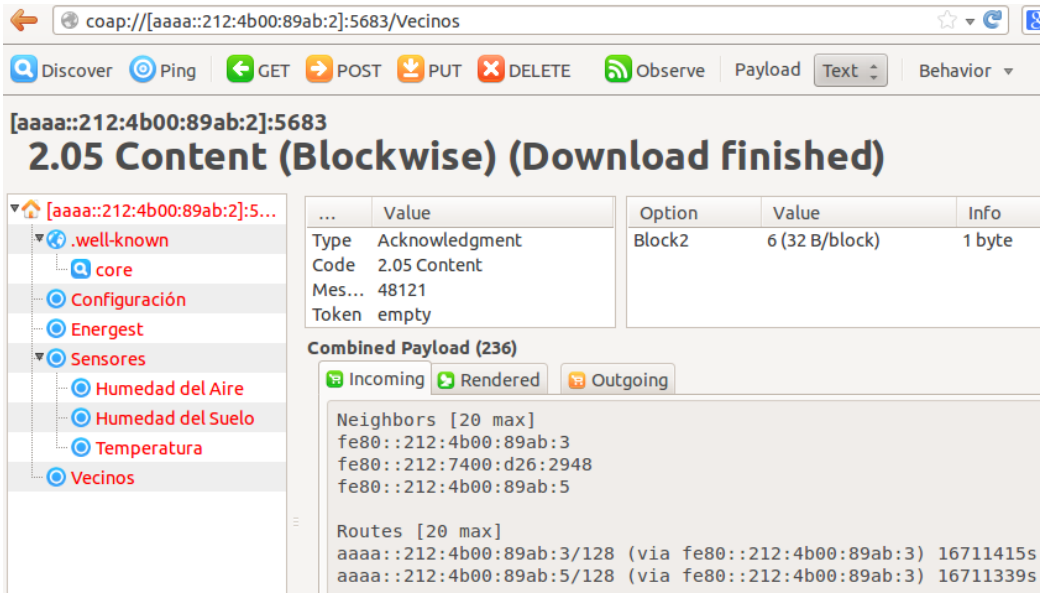


Figura 5.14: Respuesta a un GET del recurso Vecinos.

5.6. Conclusiones

Se logró implementar una aplicación en Contiki utilizando el *stack* de protocolos elegido y cumpliendo con los objetivos planteados.

A su vez, la aplicación fue desarrollada de forma de poder agregar, modificar o quitar funcionalidades fácilmente.

Se lograron integrar al nodo en forma exitosa los tres sensores a utilizar, implementando los *drivers* de los sensores SHT25 y TMP75C a partir de los *drivers* desarrollados por el fabricante Zolertia para la plataforma Z1, los cuales fueron debidamente modificados para utilizar en el CC2538. Además, se configuró el conversor ADC del CC2538 para poder convertir de analógico a digital la medida obtenida del sensor EC-05.

Al desarrollar la aplicación se estudió y se le hicieron modificaciones al sistema operativo Contiki OS, modificando tanto parámetros de los protocolos como los archivos de la radio del CC2538.

Los códigos fueron debidamente comentados para poder ser comprendidos y modificados en aplicaciones futuras. Se crearon manuales para que la aplicación pueda ser cargada y probada sin necesidad de recurrir a otros manuales o documentos.

Capítulo 6

Optimización de Protocolos

6.1. Introducción

Como se menciona en la especificación del proyecto en el capitulo 1, uno de los principales objetivos del proyecto es minimizar el consumo energético, para poder obtener una mayor vida útil de los nodos alimentando con baterías. En lo que resta del capítulo se detallan las modificaciones, pruebas y simulaciones realizadas sobre los protocolos de comunicación para cumplir con este objetivo.

Basándose en el *stack* de protocolos utilizado, se investigó qué variables de los mismos se podían modificar para reducir el consumo. Como el componente del sistema de mayor consumo es la radio, se procedió a analizar cómo varía el consumo al variar los parámetros de los protocolos de red. Bajo el supuesto de que al modificar los protocolos de red solo se afecta el comportamiento de la radio, se calculó sólo el consumo de la misma. Para ello, se simuló una red de sensores inalámbricos *multihop*, variando los parámetros seleccionados. Utilizando la herramienta Energest, introducida en el capítulo 3, se analizó cómo variaron los tiempos de uso de la radio al variar algunos parámetros de los protocolos de red. Se realizó la simulación para el Tmote Sky, dado que el simulador Cooja no soporta el MCU CC2538. Para intentar que las simulaciones reflejaran lo mejor posible el comportamiento del CC2538, se utilizó una aplicación de Texas Instruments en un kit de desarrollo basado en el CC2538 para poder tener un modelo de las pérdidas del SoC. Con los resultados obtenidos se modeló la interferencia en el simulador y se variaron los parámetros de los protocolos. Al encontrar el punto óptimo en donde el tiempo que permanecía encendida la radio era menor, se pasó a probar esas modificaciones de las variables de los protocolos en el CC2538 con la aplicación RSItrust, formando una red con cuatro módulos EM y un Tmote Sky como *border router*.

6.2. Simulación

Simular el sistema minimiza el tiempo de pruebas frente a realizar las pruebas directamente en la plataforma de hardware. Cooja permite simular al 1000 % de velocidad permitiendo hacer mayor cantidad de pruebas en menor tiempo. Además, cambiar los parámetros compilar y cargar en cada nodo físico toma mucho más tiempo que cambiar los parámetros y recargar la simulación.

La simulación se realizó en Cooja, que como se vio, no permite simular la plataforma CC2538. Debido a esto, se utilizó la aplicación desarrollada para la plataforma Tmote Sky. Como se simuló en una plataforma diferente a la utilizada para el sistema, se debieron variar parámetros de simulación para lograr un modelo que se asemejara al sistema real. Se utilizó el modelo de simulación *Multi-path Ray-tracer Medium* (MRM) [15], el cual permite modificar los parámetros de la radio (potencia de transmisión, frecuencia del canal) y del entorno (potencia del ruido) configurando una simulación que se asemeja a la realidad.

6.2.1. Pruebas preliminares

Para poder hacer una simulación de un sistema real se realizaron pruebas preliminares en campo, para las cuales se programaron dos módulos CC2538EM con la aplicación *Packet Error Rate Test* (PER-test) de Texas Instrument [28], utilizándolos conectados a las placas de desarrollo SmartRF06EB del kit CC2538DK. Este software consiste en un test genérico, que permite conocer la tasa de paquetes recibidos con error. Esta aplicación permite variar la potencia de transmisión ($22dBm$ a $7dBm$) y la ganancia de recepción (alta o baja), así como también se configura la cantidad y la tasa de paquetes a enviar, de un paquete por segundo a 1000 paquetes por segundo y el canal de comunicación.

Como se detalló en el capítulo 3, el protocolo de capa física a utilizar es el IEEE 802.15.4, el cual utiliza la banda de frecuencia de $2{,}4GHz$. En esta banda de frecuencia se especifican 16 canales del 11 al 26, solapados cada $5MHz$ desde 2405 a 2480 MHz [14]. En las pruebas realizadas se utilizó el canal de comunicación 26 (2480 MHz) ya que es un canal con poca interferencia WiFi (banda de $2{,}4GHz$), como se puede observar en la Figura 6.1 [49].

Como los nodos en el campo de cítricos estarían separados $100m$ de distancia entre ellos, se buscó a esa distancia la potencia necesaria para obtener un $PacketErrorRate$ (PER) máximo de 10 %, siendo el PER el porcentaje de paquetes perdidos frente al total de paquetes enviados. Un PER máximo de 10 % se consideró como una pérdida razonable de paquetes para no afectar el funcionamiento de la red.

De esta manera se encontró que la mínima potencia de transmisión necesaria es de $20dBm$ y que alcanza con baja ganancia de recepción para obtener un PER

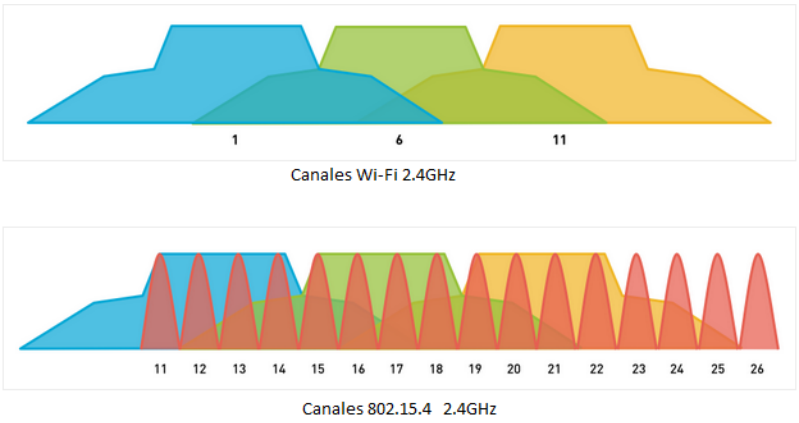


Figura 6.1: Bandas de frecuencia

menor al 10 %. Si se usa una potencia menor ($16dBm$) el PER obtenido es de 13 % superando el límite, y si se usa $20dBm$ de potencia de transmisión y alta ganancia de recepción se obtiene un mejor resultado pero también un mayor consumo, por lo cual no se eligió. Cada prueba se repitió varias veces en el predio de facultad el cual cuenta con árboles y edificios cercanos, obteniendo con los parámetros elegidos un PER promedio de 5 % como muestra la Tabla 6.1.

Transmisor			**Receptor**	
Potencia	**Cadencia**	**Paquetes enviados**	**Ganancia**	**%PER Promedio**
22dBm	1p/s	300	Alta	2.09
20dBm	1p/s	300	Alta	4
20dBm	1p/s	300	Baja	5
16dBm	1p/s	300	Alta	13.98

Tabla 6.1: Resultados de las pruebas preliminares

Algo importante a tener en cuenta en las pruebas, es la altura respecto al piso a la cual se colocan los nodos. Ésta se calcula con la ecuación de Fresnel (ecuación 6.1), donde se llama zona de Fresnel al volumen de espacio entre el emisor de una onda y un receptor, donde el desfasaje de la onda no supere los 180º [10].

$$r_n = \sqrt[2]{\frac{n\lambda d_1 d_2}{d_1 + d_2}} \tag{6.1}$$

Donde:

- r_n = radio en metros de la elipsoide de la zona n en un punto dado, (n=1,2,3...).

- d_1 = distancia desde el transmisor al centro del elipsoide en metros.
- d_2 = distancia desde el centro del elipsoide al receptor en metros.
- λ = longitud de onda de la señal transmitida en metros.

En la Figura 6.2 se pueden ver los parámetros. Aplicando la fórmula de la primera zona de Fresnel (r_1) se obtiene:

$$r_1 = 8,657\sqrt{\frac{D}{f}} \tag{6.2}$$

Donde

- r_1 = radio en metros (m).
- D = distancia en kilómetros (km) ($d_1 = d_2, D = d_1 + d_2$).
- f = frecuencia de la transmisión en GHz ($\lambda = \frac{c}{f}$)

Considerando los parámetros de este proyecto, $D = 0{,}100km$, $f = 2{,}480GHz$, se obtiene una altura mínima de colocación de los nodos de $1{,}74m$. En las pruebas se utilizó una altura de $1{,}75m$ aproximadamente.

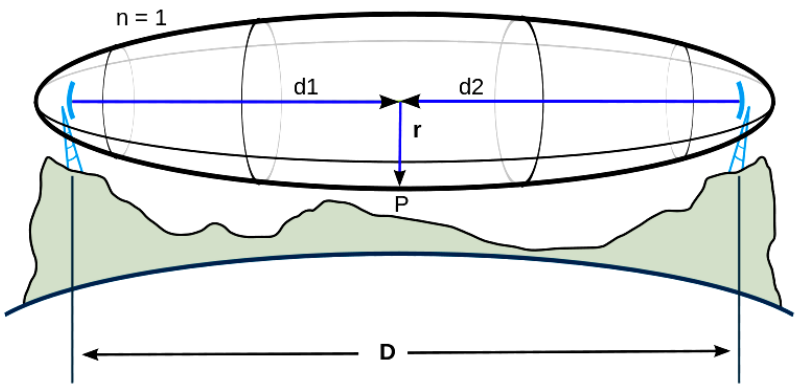


Figura 6.2: Zona de Fresnel

6.2.2. Simulación en Cooja

Trabajando con la aplicación para el Tmote Sky, se modificaron los parámetros del modelo de simulación MRM, para que la radio tuviera las mismas propiedades que la del CC2538 y el sistema se pareciera al probado en campo. Se fijó la frecuencia del canal que se usó ($2480MHz$), la potencia de transmisión de $20dBm$ y la sensibilidad que según la hoja de datos es $-105dBm$. Con los parámetros

anteriores fijos, se varió el umbral de recepción del SNR[1] para obtener un PER del 5 % al colocar los nodos a $100m$ de distancia entre ellos, y de esta manera simular utilizando los mismos parámetros que en las pruebas preliminares. Se llegó a esta condición con un umbral de $15{,}4dB$. La simulación se realizó durante 2 horas observando una medida cada 15 minutos al igual que en la prueba real.

Energest

Para poder estimar el consumo de cada estado del MCU se utilizó la herramienta de Contiki, Energest.

Como se describe en el capítulo 3, Energest permite saber cuánto tiempo permanece el sistema en determinados estados. Para obtener dicho tiempo en segundos, se divide por la variable de *Contiki* $RTIMER\_ARCH\_SECOND$ que para el MCU CC2538 es igual a 32768 (2^{15}).

Se incluyó esta herramienta en la aplicación de RSItrust para poder ver en consola cuánto tiempo estaba transmitiendo y recibiendo el nodo, y de esta manera modificar los parámetros de los protocolos de comunicación para disminuir dichos tiempos. Disminuir estos tiempos permiten disminuir el consumo del sistema. Con los tiempos obtenidos se calcula el consumo utilizando los parámetros de la hoja de datos del EMBIT [36] como se explica más adelante en este capítulo.

Modificación de Protocolos

El primer parámetro modificado fue el CCR de ContikiMAC. Esta variable indica la frecuencia con la cual se despierta el nodo para escuchar el canal de radio. Por defecto el CCR está configurado en $8Hz$, por lo que se tenían periodos de escucha del canal cada $125ms$. Se realizó la simulación con este valor original y se guardaron los valores que brinda Energest para luego compararlos y obtener un posible valor óptimo.

Luego se modificó el CCR a $4Hz$, obteniendo un periodo de escucha de $250ms$ y tiempos de uso de la radio menores. Este tiempo era menor, pero el tiempo de transmisión (TX) aumentó al doble, mientras que el de recepción (RX) disminuyó a la mitad. El tiempo TX aumentó debido a que el nodo al querer enviar un paquete, envía durante todo un periodo para asegurarse que el receptor lo reciba. Como el periodo aumentó al doble el tiempo TX también aumentó en igual proporción. Por otro lado, la disminución de RX, se debe a que el nodo se encendía la mitad de veces. Como la reducción de RX es más significativa, la suma de ambos disminuyó. Cabe destacar que la mayoría de los paquetes que se envían en el transitorio del

[1]SNR: Relación señal a ruido

sistema son broadcast, teniendo mayor peso durante el transitorio que las solicitudes de medidas, las cuales se hacen cada 15 minutos. Estos mensajes broadcast se deben al protocolo RPL, donde los nodos periódicamente envían mensajes DIOs para mantener actualizada la tabla de rutas. Gracias al algoritmo Trickle el envío de DIOs va disminuyendo al estabilizarse la red, como se explica más adelante en esta sección. La reducción más significativa del RX se debe a que al estabilizarse la red el nodo continúa despertando cada $\frac{1}{CCR}ms$ para chequear si algunos de sus nodos vecinos envió algún paquete. De esta manera un aumento en el periodo de escucha genera que el nodo se despierte menos, disminuyendo el consumo. Por otro lado las transmisiones se atenúan automáticamente con el algoritmo Trickle logrando alcanzar períodos de transmisión superiores a los 15 minutos.

Al bajar el CCR a $2Hz$, el tiempo TX se cuadruplicó, mientras que RX disminuyó a la cuarta parte. Todas estas variaciones se hicieron dejando los demás parámetros de los protocolos sin modificar.

Luego se modificaron los parámetros del protocolo RPL, el cual utiliza el algoritmo Trickle para la difusión de los mensajes DIOs. Estos mensajes son utilizados para mantener actualizadas las tablas de rutas en los nodos de la red, como se vio en la sección 3.4.5. El algoritmo Trickle chequea si la versión del mensaje que se recibe es más vieja, nueva o igual a la que se tiene registrada. Si la versión es más vieja o igual la recepción se considera consistente, y el periodo de envío de mensajes DIO aumenta, disminuyendo la carga de paquetes en la red. Este algoritmo evita el reenvío de paquetes con información repetida cuando la red está estable y todos sus nodos tienen la misma información.

El intervalo $[I_{min}, I_{max}]$ de TRICKLE introducido en la sección 3.4.5 está definido en los archivos de RPL de Contiki como [T_{min} y T_{max}].

Se estudió que los parámetros útiles para lograr disminuir el consumo de la red, son los de los intervalos con que se envían DIO (T_{min} y T_{max}), y el tiempo de vida. El tiempo de vida es el tiempo máximo que el nodo espera recibir un mensaje DIO antes de que considere que perdió a su padre y envíe un mensaje DIS de solicitud de información.

Si no se pierden paquetes y la red es estable, los tiempos de envío de mensajes DIO se van duplicando hasta llegar a T_{max}. De esta manera el tráfico promedio de paquetes disminuye. Para tener menos tráfico de paquetes al estabilizarse la red, se aumentó el valor máximo de intervalo T_{max}. Para evitar que T_{max} supere el tiempo de vida, se consideró aumentar el tiempo de vida de RPL. De esta manera, cuando las versiones de los DIOs son obsoletas, el intervalo de tiempo comenzará a aumentar hasta llegar a T_{max} sin llegar a superar el tiempo de vida. Si este último llegara a superarse, el nodo consideraría que perdió a su padre y comenzaría a mandar mensajes DIS en busca de un nuevo camino hacia el nodo raíz, lo cual no es lo que se desea.

La configuración por defecto del protocolo RPL de Contiki, tiene un tiempo de vida de 193 días. Por lo cual, un nodo que pierde a su padre manda un mensaje DIS cada 193 días, lo que sumaría un consumo insignificante al sistema. De esta manera, se dejó el tiempo de vida por defecto y se comenzó a agrandar el $Tmax$ de RPL, dejando fijo el resto de los parámetros del protocolo RPL, obteniendo así menor consumo.

Las variables que se encontraron en $contiki/core/net/rpl/rpl-conf.h$ son:

- $RPL\_DIO\_INTERVAL\_MIN = 12$ (valor por defecto)
- $RPL\_DIO\_INTERVAL\_DOUBLINGS = 8$ (valor por defecto)

Con estas dos variables se calcula:

$T_{min}[ms] = 2^{RPL\_DIO\_INTERVAL\_MIN} = 2^{12} = 4096ms$ (4s)
$T_{max}[ms] = T_{min} * 2^{RPL\_DIO\_INTERVAL\_DOUBLINGS} = 4096 * 2^8 = 1048576ms$
$(T_{max}[ms] = 17{,}5min)$

Lo que se hizo fue incrementar el valor *RPL_DIO_INTERVAL_DOUBLINGS* para lograr que al estabilizarse la red, la tasa de paquetes disminuyera, como se explicó anteriormente.

Con los parámetros $CCR = 2Hz$ y *RPL_DIO_INTERVAL_DOUBLINGS = 8* se obtuvo el menor tiempo de uso de la radio. Esto se puede apreciar en la gráfica de la Figura 6.3, y en la Tabla 6.2, las cuales muestran cómo varían los tiempos de uso de la radio al modificar los dos parámetros. En la Figura 6.3 se indica con un punto rojo el mínimo tiempo de uso de la radio, el cual corresponde a los parámetros mencionados. En la Tabla 6.2 se puede apreciar que al mantener el CCR constante, las variaciones del tiempo no cambian significativamente al variar el parámetro *RPL_DIO_INTERVAL_DOUBLINGS*. Esto se debe a que el tiempo de simulación es muy corto para que este parámetro influya en los resultados. Realizando nuevamente el cálculo de T_{max} con *RPL_DIO_INTERVAL_DOUBLINGS =8* se obtiene un tiempo de $T_{max} = T_{min} * 2^{RPL\_DIO\_INTERVAL\_DOUBLINGS} = 4096 * 2^8 = 17$ minutos. En una red ideal con estos parámetros, se deberían esperar 35 minutos hasta que la red se estabilice, llegando al intervalo máximo T_{max}. Luego los mensajes DIOs se enviarían cada 17 minutos. En una red real demoraría mucho más en llegar al T_{max} ya que al haber cambios en la red se envían DIOs con información nueva, generando que el tiempo entre DIOs se resetee a T_{min}. Si se extrapola este resultado a una red real, para minimizar el envío de paquetes DIOs al estabilizarse la red, el T_{max} debería de ser lo más grande posible. Como contraparte, el sistema va a ser lento para detectar cambios en la red, ya que va a demorar como mínimo T_{max} en detectar el cambio. Teniendo en cuenta que T_{max} no supere el tiempo de vida, el RPL_DIO_INTERVAL_DOUBLINGS máximo a utilizar es 20. De esta manera, los parámetros elegidos para realizar las pruebas de campo con la aplicación RSItrust

en el CC2538DK fueron $CCR = 2Hz$ y $RPL\_DIO\_INTERVAL\_DOUBLINGS = 20$.

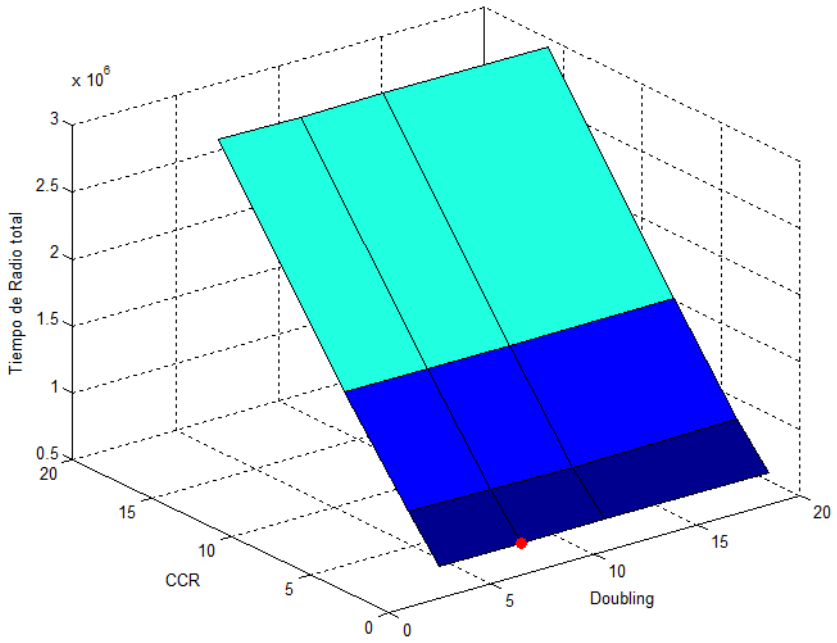


Figura 6.3: Tiempo de uso de radio al variar CCR y Doubling.

6.3. Prueba de campo

A la hora de desplegar la red de 5 nodos, pareció conveniente disminuir la distancia entre nodos para poder realizar la prueba de manera más sencilla. Para ello, se realizó otra prueba con el ejemplo PER-Test para poder realizar la prueba de lo simulado a una distancia equivalente a $100m$ entre nodos y $20dBm$ de potencia de transmisión, manteniendo un PER de 5 %.

Se encontró que para obtener un PER de 5 %, los nodos se deben colocar a $20m$ de distancia entre ellos y $0dBm$ de potencia de transmisión. Con esta nueva distancia de $20m$ se utilizó la ecuación de Fresnel para recalcular la altura a la cual había que posicionar a los nodos, la cual fue de $0{,}8m$.

La red *multihop* se formó con la aplicación *border-router* cargada en un Tmote Sky y la aplicación RSItrust cargada en 4 plataformas CC2538EM con el MCU

Caso de estudio	Tiempo de uso de la radio en 2hs de prueba (s)
CCR 2, Doubling 4	17.124
CCR 2, Doubling 8	17.007
CCR 2, Doubling 12	17.184
CCR 2, Doubling 20	17.020
CCR 4, Doubling 4	26.111
CCR 4, Doubling 8	25.746
CCR 4, Doubling 12	25.520
CCR 4, Doubling 20	25.786
CCR 8, Doubling 4	46.463
CCR 8, Doubling 8	46.265
CCR 8, Doubling 12	46.223
CCR 8, Doubling 20	46.150
CCR 16, Doubling 4	89.871
CCR 16, Doubling 8	89.589
CCR 16, Doubling 12	89.879
CCR 16, Doubling 20	89.701

Tabla 6.2: Resultados de simulación

CC2538. Los mismos se ubicaron en el predio de facultad a $20m$ de distancia con una potencia de transmisión de $0dBm$.

Primero se realizó la prueba con los parámetros originales ($CCR = 8Hz$ y $RPL\_DIO\_INTERVAL\_DOUBLINGS = 8$) y luego se cambiaron por los elegidos en la simulación ($CCR = 2Hz$ y $RPL\_DIO\_INTERVAL\_DOUBLINGS = 20$).

Al comenzar se tomó en cuenta el orden para encender los nodos. Empezando desde el *border router*, se encendió primero el nodo más cercano y luego de que el mismo enviara un mensaje DIO se encendió el siguiente, y así sucesivamente. Esto se hizo así para evitar que al armar la tabla de ruteo un nodo lejano fuera elegido como padre preferido. Esto puede ser un problema ya que el descarte de dicho padre no es inmediato, por lo cual podría generar mala comunicación al comienzo de la prueba. Para tener más información se colocó un *sniffer*[2] fabricado por Atmel [12] a mitad de camino entre el *border router* y el nodo más lejano para poder visualizar el tráfico de paquetes y el mensaje DIO inicial de cada nodo.

A continuación se envió un mensaje PING a cada nodo para verificar la comunicación, ya que el mensaje PING manda una solicitud de eco ICMP para comprobar si existe una ruta entre el host local con cada uno de los equipos re-

[2]Sniffer: Hardware encargado de captar paquetes en una red

motos de una red. Luego de confirmar que existía conectividad entre los nodos se prosiguió a observar uno de los recursos de la aplicación del nodo más cercano al *border router*. La prueba se hizo durante 2 horas observando cada 15 minutos. Al finalizar las 2 horas se hizo un GET al recurso res-energest (recurso que se explica en la sección 5.4.3) de todos los nodos, el cual devuelve en el payload los tiempos en cada estado del MCU.

Luego se configuraron los parámetros en: $CCR = 2Hz$ y $RPL\_DIO\_INTERVAL\_DOUBLINGS = 20$, se reprogramaron los nodos y se repitió la misma prueba.

Al analizar los datos obtenidos se observó que el consumo con el $CCR = 2Hz$ era mayor al consumo con el $CCR = 8$ (valor por defecto). Esto no era lo esperado ya que el resultado en la simulación fue el opuesto. Este problema se debió a que se supuso que el consumo del CC2538 durante la transmisión era igual al de recepción. Como el consumo se calcula con ecuación 6.3 se consideró que el menor tiempo de uso de la radio era equivalente al menor consumo. Esto no es correcto, en la Tabla 6.3 se puede apreciar que la transmisión tiene mayor consumo que la recepción. Estos parámetros fueron extraídos de la hoja de datos del EMBIT [36]. Calculando los consumos para los distintos casos de estudios, para una red con nodos ubicados a $100m$ y $20dbm$ de potencia de transmisión, se obtuvo que el menor consumo se conseguía con $CCR = 4Hz$ y no $CCR = 2Hz$. Recordar que con un sistema con nodos ubicados a $20m$ y $0dbm$ de potencia de transmisión, se obtenía el mismo PER que con nodos ubicados a $100m$ y $20dbm$ de potencia de transmisión. De esta manera se extrapoló el resultado al sistema real. En la tabla 6.4 se pueden apreciar dichos consumos.

Etapa	**Consumo (mA) a 20dBm**
LPM	$1{,}7x10^{-3}$
CPU - RX - TX	15.3
Rx	34.5
Tx	166

Tabla 6.3: Consumos obtenidos de la hoja de datos del Embit

$$Q = I_{TX} * \%TX + I_{RX} * \%RX \tag{6.3}$$

Donde % TX es el tiempo TX dividido el tiempo total de simulación y % RX es el tiempo RX dividido el tiempo total de simulación.

Caso de estudio	Consumo de la radio en 2hs de prueba (mAh)
CCR 2, Doubling 4	$1{,}80929x10^{-}4$
CCR 2, Doubling 8	$1{,}800x10^{-}4$
CCR 2, Doubling 12	$1{,}813x10^{-}4$
CCR 2, Doubling 20	$1{,}801x10^{-}4$
CCR 4, Doubling 4	$1{,}861x10^{-}4$
CCR 4, Doubling 8	$1{,}780x10^{-}4$
CCR 4, Doubling 12	$1{,}761x10^{-}4$
CCR 4, Doubling 20	$1{,}784x10^{-}4$
CCR 8, Doubling 4	$2{,}522x10^{-}4$
CCR 8, Doubling 8	$2{,}507x10^{-}4$
CCR 8, Doubling 12	$2{,}490x10^{-}4$
CCR 8, Doubling 20	$2{,}485x10^{-}4$
CCR 16, Doubling 4	$4{,}471x10^{-}4$
CCR 16, Doubling 8	$4{,}450x10^{-}4$
CCR 16, Doubling 12	$4{,}470x10^{-}4$
CCR 16, Doubling 20	$4{,}456x10^{-}4$

Tabla 6.4: Consumos obtenidos de la simulación

Se rehicieron las pruebas de $2hs$ pero esta vez con $CCR = 4Hz$, obteniendo un menor consumo y una mayor vida útil.

Los parámetros obtenidos de las pruebas se muestran en la Tabla 6.5, donde en $Ttotal$ se indica el tiempo total de simulación en segundos.

Parámetro	Tiempo(s)		
	CCR 2	CCR 4	CCR 8
LPM	7439.1718	7489.2953	7078.0511
CPU	61.3731	78.8227	134.6952
Rx	13.5250	21.6829	39.8707
Tx	28.9003	10.6566	7.1583
Ttotal	7500.5449	7568.1180	7212.7463

Tabla 6.5: Parámetros de tiempo obtenidos con energest

Como se puede ver en los parámetros de la Tabla 6.5 se verifica el modelo simplificado de la red y lo visto en la simulación. Al bajar CCR de 8 Hz a 4 Hz, el tiempo de TX se duplica y el tiempo de RX baja a la mitad. Mientras que al pasar de CCR 8 Hz a 2 Hz el TX se cuadruplica y el RX pasa a ser la cuarta parte. Las

gráficas circulares 6.4, muestran el porcentaje de tiempos sobre el total de 2 horas al variar el parámetro CCR. Como se puede apreciar la mayor parte del tiempo el MCU está en modo bajo consumo, mientras que al variar los parámetros se puede notar el aumento y disminución de TX y RX como se explicó anteriormente.

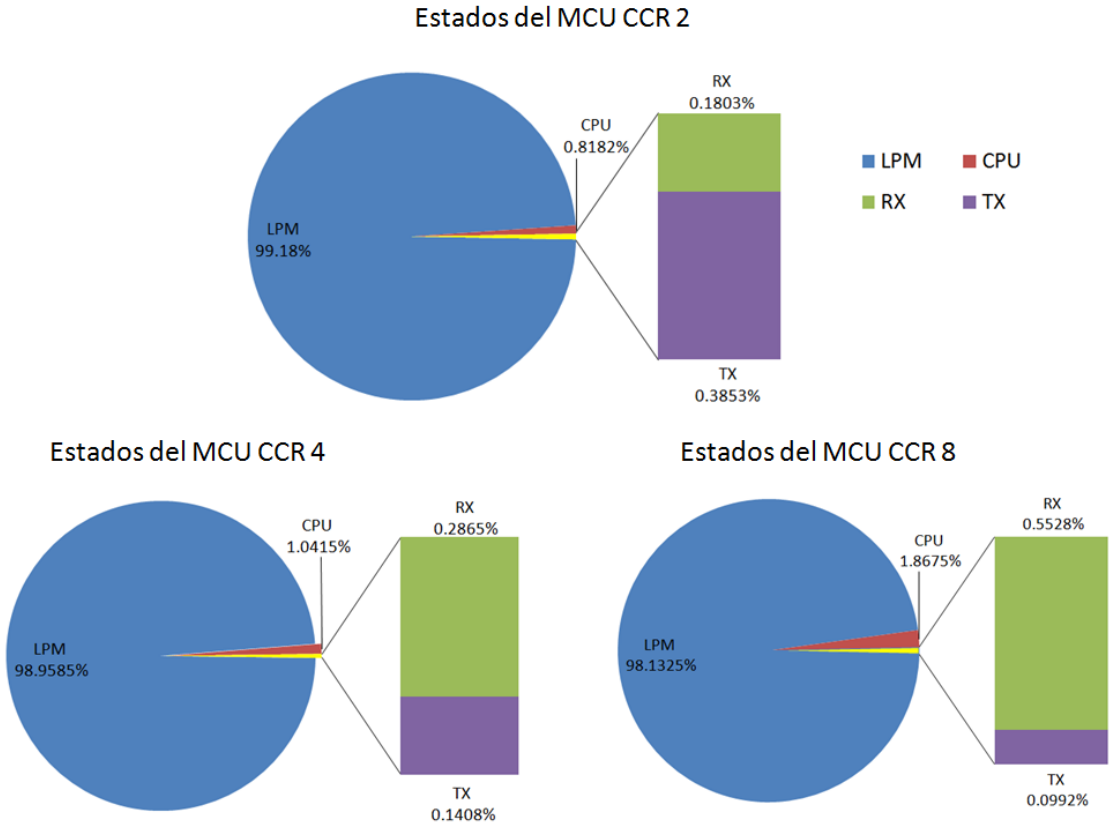


Figura 6.4: Gráfica circular de los tiempos del MCU al variar CCR

Para el cálculo de tiempo de vida se utilizó la ecuación 6.4.

$$Q_{pila} = 2800mAh = Tt(Ie1 * \%te1 + Ie2 * \%te2 + \ldots\ldots +) \qquad (6.4)$$

Donde $\%te1$ representa el cociente entre el tiempo de la etapa 1 (LPM) dividido el tiempo total e $Ie1$ es la corriente consumida en dicha etapa. De esta manera, conociendo que la carga de las pilas a utilizar es de $2800mAh$ y los consumos de la Tabla 6.3, se estima el Tt (tiempo total de vida).

Observando la Tabla 6.6 se puede ver que al configurar el protocolo con $CCR = 4Hz$ el tiempo de vida del nodo es mayor que con los parámetros originales, quedando esta configuración como la elegida para el sistema.

Caso de estudio	Tiempo de vida en meses
CCR 2	5
CCR 4	9
CCR 8	7

Tabla 6.6: Tiempo de vida útil del nodo

Si se repite el cálculo de tiempo de vida, pero para los consumos del MCU sin utilizar el PA/LNA, se pueden observar grandes cambios en la vida útil, superando el año de autonomía que se esperaba obtener al bajar el consumo. En la Tabla 6.7 se pueden ver los valores de consumo que cambiaron respecto a la Tabla 6.3, los cuales fueron obtenidos de la hoja de datos de CC2538 [30] y en la Tabla 6.8 se ven los nuevos resultados de tiempo de vida útil del sistema.

Etapa	Consumo (mA) a 7dBm
Rx	20
Tx	34

Tabla 6.7: Consumos obtenidos de la hoja de datos del CC2538

Caso de estudio	Tiempo de vida en meses
CCR 2	18
CCR 4	19
CCR 8	11

Tabla 6.8: Tiempo de vida útil del nodo

Como se puede apreciar en la Tabla 6.7 la potencia máxima obtenida con este sistema es de $7dBm$ en lugar de $22dBm$. Esto implica que si se tuviera un dispositivo similar al EMBIT sin PA/LNA se necesitaría una mayor cantidad de nodos a menor distancia para cubrir el área de estudio.

6.4. Conclusión

Siguiendo la metodología descripta se pudo aumentar un 29 % la vida útil del sistema con respecto a la obtenida con los parámetros originales. Ésto se logró al modificar los parámetros CCR de ContikiMAC y *RPL_DIO_INTERVAL_DOUBLINGS* de RPL.

Un detalle a tener en cuenta es que una prueba de dos horas podría no ser suficiente para salir del transitorio por lo cual se estaría estimando la vida útil con el sistema siempre en transitorio. Debido a esto la vida útil del sistema podría ser mayor en la realidad.

Con este resultado no se logró superar el objetivo inicial de un año de vida útil, pero si se implementara el sistema con otro módulo similar al EMBIT sin PA/LNA, se podría obtener un tiempo de vida de 19 meses. Este aumento de vida útil implicaría utilizar más nodos para cubrir el área de estudio, ya que la potencia de transmisión máxima es mucho menor a la utilizada. Ésto implicaría un notorio aumentando del costo del sistema.

Capítulo 7

Test de la Red

7.1. Introducción

Al finalizar la etapa de implementación de la aplicación de software, en la cual también se probó el diseño de hardware, se prosiguió con las pruebas de la red.

Uno de los principales objetivos del presente proyecto es formar una RSI robusta, por lo cual para este test se armó una red y se probó su funcionamiento, comprobando en primera instancia que se haya armado correctamente la red utilizando herramientas desarrolladas para este fin, para luego comprobar el correcto funcionamiento de la aplicación.

7.2. Pruebas de la red

En primera instancia se programó un CC2538EM con la aplicación *border-router*, y los restantes 3 CC2538EM y la placa diseñada durante el proyecto, se programaron con la aplicación RSItrust.

Estas plataformas configuradas con $0dBm$ de potencia de transmisión se colocaron a una distancia de $20m$ y una altura de $0{,}8m$ al igual que en las pruebas de campo explicadas en la sección 6.3. De esta manera se formó una red *multihop*.

Primero se probó la comunicación enviando mensajes PING a todos los nodos, y recibiendo la respuesta correspondiente. Como esta prueba funcionó correctamente, verificando la comunicación con los nodos, se probó la aplicación en la placa diseñada durante el proyecto, sabiendo que el funcionamiento en las demás placas es análogo.

Se procedió a solicitarle al nodo su tabla de rutas como se puede ver en la Figura 7.1. En dicha imagen se puede apreciar en la URL, la dirección IP del nodo, y el recurso vecinos el cual fue implementado para realizar esta prueba, y

para la hora de la puesta en marcha del sistema, y no como una funcionalidad de la aplicación. Éste permite que al enviarle una solicitud GET el nodo responda enviando su tabla de ruta.

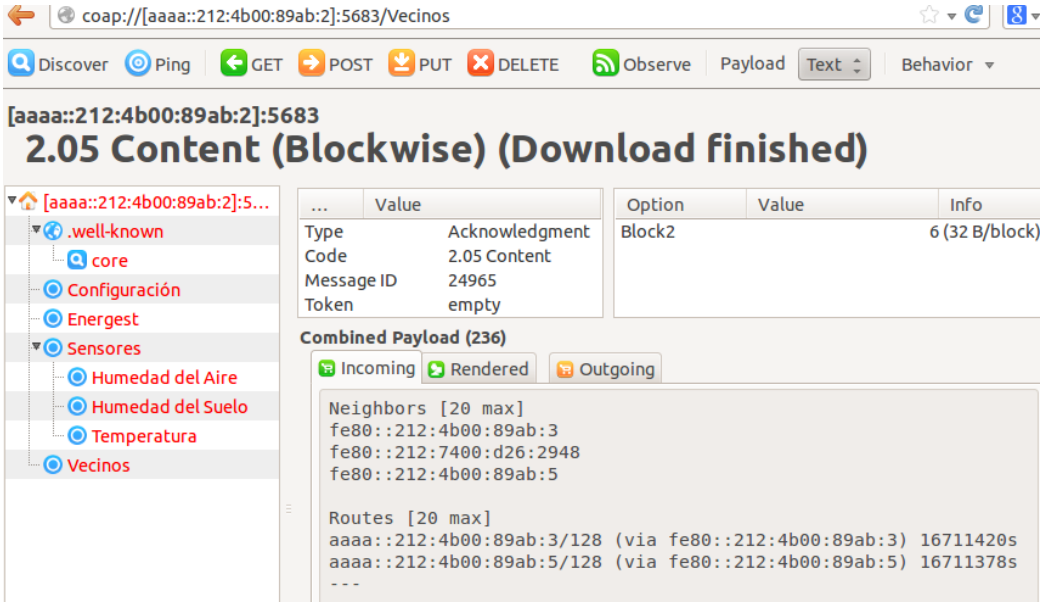


Figura 7.1: Tabla de ruta de un nodo

A continuación se procedió a configurar los parámetros del sistema. Estos son periodo de muestreo, período de observación de los distintos sensores y la hora. En la Figura 7.2 se puede apreciar la hora Unix configurada y en la Figura 7.3 se observa la respuesta del nodo ante una solicitud GET al recurso Configuración. En la pestaña *Incoming* de *Copper* se puede ver la hora, período de observación de la temperatura, humedad del aire y humedad del suelo y el período de muestreo TM configurados en el nodo.

Luego de la configuración se hicieron solicitudes GET a los recursos de los sensores de la placa fabricada en RSItrust, verificando el correcto funcionamiento. Desde *Copper* se pudieron observar las medidas, las cuales eran las esperadas comparando contra una miniestación meteorológica de uso doméstico. En las Figuras 7.4, 7.5 y 7.6 se pueden apreciar las medidas de los 3 sensores ante una solicitud GET. Además en la Figura 7.7 se observa el mensaje de error recibido ante un problema de comunicación *I2C* o ante la desconexión de alguno de los sensores. Notar también que, cuando el mensaje es correcto, es del tipo "*2.05 Content*", en donde el 2 indica que es una respuesta y el .05 que se cumplió la acción. Cuando el mensaje tiene un error es del tipo "*5.03 Service Unavailable*", en donde el 5 indica Error de respuesta del servidor y el .03 servicio no disponible.

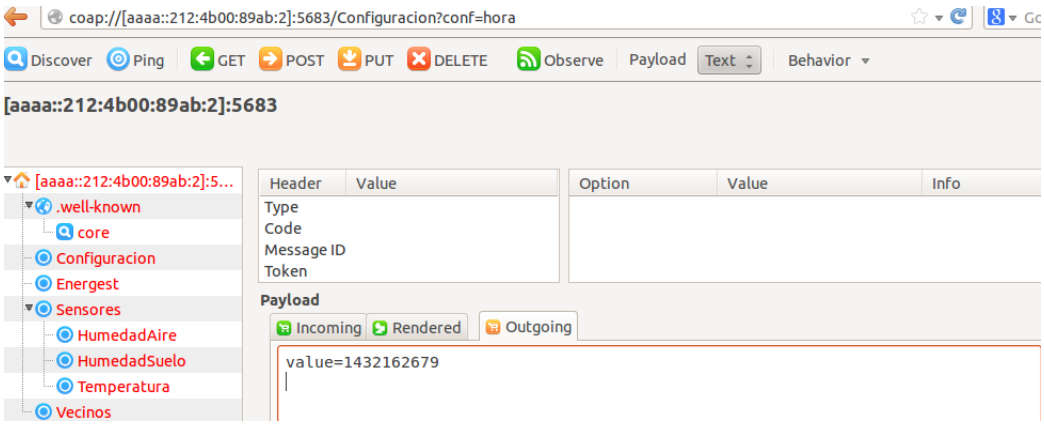


Figura 7.2: Configuración de hora de un nodo

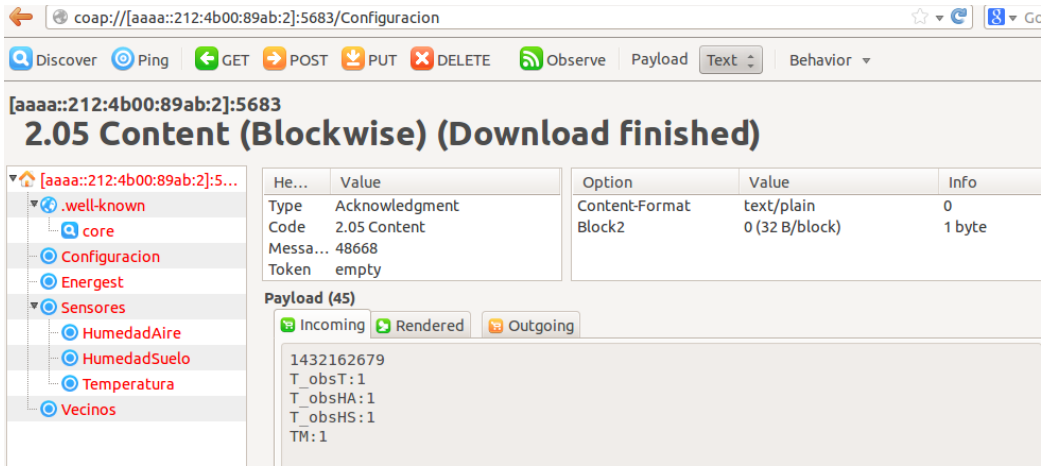


Figura 7.3: Configuración general de un nodo

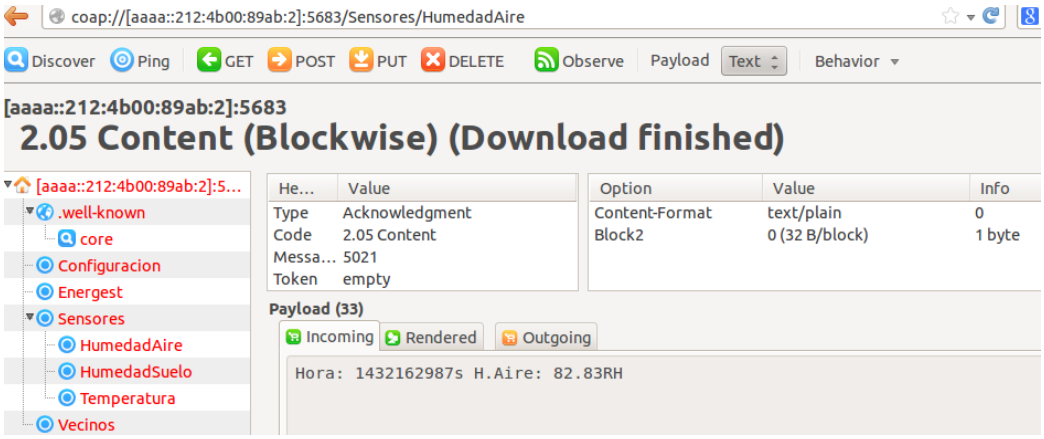


Figura 7.4: Respuesta a solicitud de Humedad del Aire.

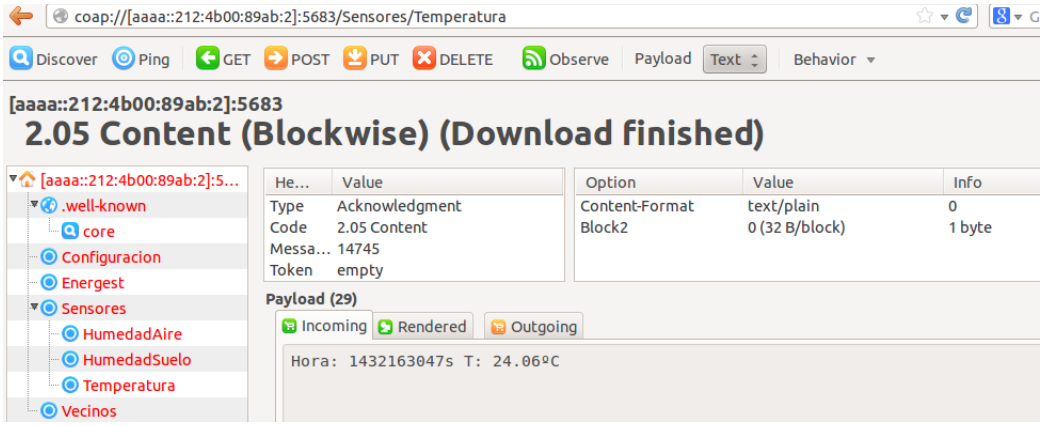


Figura 7.5: Respuesta a solicitud de Temperatura.

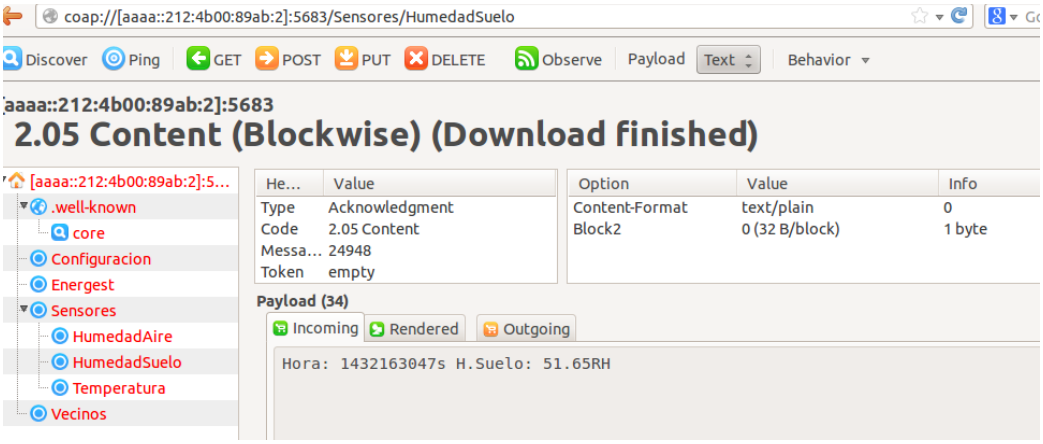


Figura 7.6: Respuesta a solicitud de Humedad del Suelo.

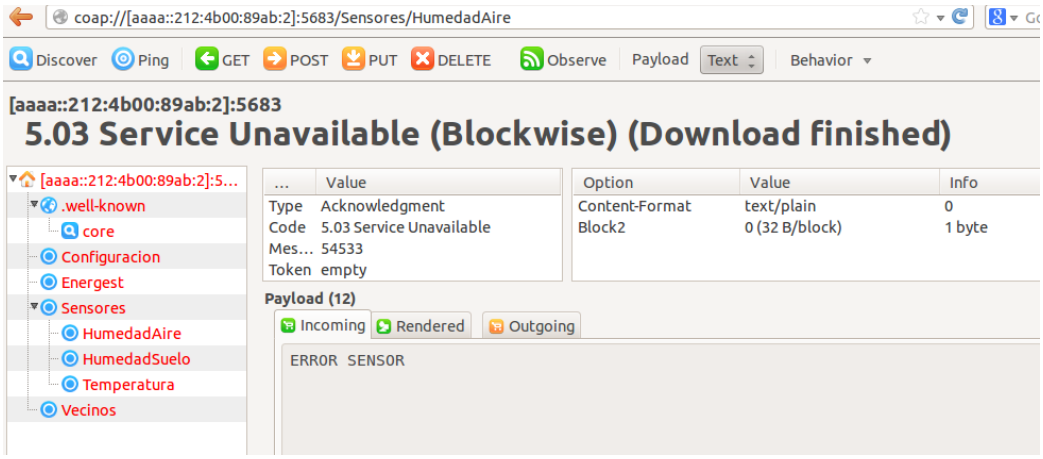


Figura 7.7: Respuesta de Error ante solicitudes GET.

Luego de probar los diferentes recursos de la aplicación se abrieron tres instancias de *Copper* y en cada una de ellas se hizo una subscripción Observe a cada uno de los sensores. Observando cada 15 minutos durante $25hs$ se probó la resistencia de la red y la comunicación. Al finalizar las $25hs$ se apreciaron las medidas de todos los sensores y al hacer una solicitud GET al recurso Energest se apreciaron los distintos períodos de tiempo del MCU, verificando que el sistema funcionó correctamente y el tiempo total de pruebas. En la Figura 7.8 se ve la respuesta a un GET al recurso *Energest*, con el mensaje: *88540,2; 1561,66; 538,61; 12,55;**. Esta información indica, respectivamente:

- LPM - Tiempo en segundos en que el MCU permaneció en modo de bajo consumo
- CPU - Tiempo en segundos en que el MCU permaneció con el CPU activo
- Rx - Tiempo en segundos en que se tuvo la recepción de la radio encendida
- Tx - Tiempo en segundos en que se tuvo la transmisión de la radio encendida

A partir de estos valores, se obtiene que el tiempo que estuvo el MCU encendido fue:

$$T_{encendido} = LPM + CPU = 88540,2s + 1561,66s = 90101{,}86s = 25hs \quad (7.1)$$

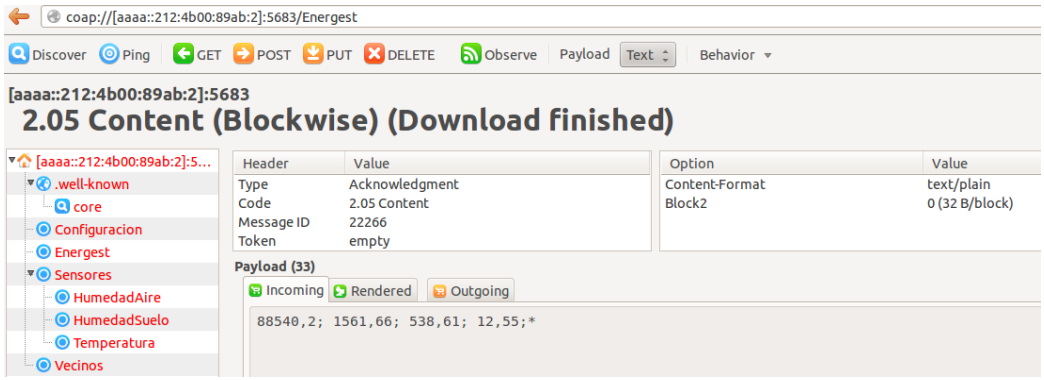


Figura 7.8: Respuesta a solicitud de Energest.

A continuación se repitió la prueba anterior durante 2hs pero observando cada 1 minuto. De esta manera se exigió el sistema, el cual respondió correctamente.

Otra prueba fundamental fue la prueba de alcance. Para verificar la misma se utilizó nuevamente la aplicación de PER-Test de Texas Instuments para poder visualizar en el display del CC2538DK el PER que se obtenía. La prueba consistía en

transmitir 1000 paquetes aumentando la distancia entre nodos. Dicha distancia se aumentaría mientras el PER no superara el 10 % estipulado. Además se utilizó la máxima potencia de transmisión ($22dBm$) y alta ganancia de recepción.

Se pudo aumentar la distancia hasta unos $500m$ obteniendo un PER de 1,1 %. Al aumentar aún más la distancia (hasta unos $560m$) se tuvo una gran pérdida de paquetes, obteniendo un porcentaje de pérdidas de 85 %, ya que no se contaba con la altura necesaria para no tener interferencia como se vio en la sección 6.2 en la zona de Fresnel. De todas maneras se puede apreciar que la distancia se puede aumentar considerablemente si se cuenta con la altura necesaria.

7.3. Conclusión

Se probó la red de sensores verificando el correcto funcionamiento de la aplicación y de la comunicación de la red.

También se le realizó una prueba de *stress*, verificando que el sistema y la comunicación es robusta.

Capítulo 8

Conclusiones

En el presente proyecto se logró diseñar e implementar una RSI, con el fin de obtener un producto confiable (PER $10\,\%$) y pensado para ser producido en mayor escala. Dicho sistema cuenta con sensores de humedad y temperatura del aire, y humedad del suelo, los cuales se lograron integrar al nodo en forma exitosa. Se implementaron *drivers* para los sensores SHT25 y TMP75C a partir de los *drivers* desarrollados por el fabricante Zolertia para la plataforma Z1, los cuales fueron debidamente modificados para utilizar en el CC2538. Además, se configuró un conversor ADC del CC2538 para adquirir la señal del sensor EC-05. Utilizando los sensores de presencia con los que cuentan los conectores seleccionados para la integración de los sensores al nodo, es posible generar una señal de falla en caso de que no haya un sensor conectado.

Se logró implementar una aplicación en Contiki utilizando el *stack* de protocolos elegido. Al desarrollar dicha aplicación se estudió y se le hicieron modificaciones al sistema operativo Contiki OS, modificando tanto parámetros de los protocolos como los archivos de la radio del CC2538.

Modificando parámetros de los protocolos de comunicación se logró aumentar un $29\,\%$ la vida útil del sistema con respecto a la obtenida con los parámetros originales, totalizando una vida útil estimada de 9 meses con pilas de capacidad útil de $2800mAh$. Si bien no se logró el objetivo de obtener una vida útil de más de un año, se estudió que si se contara con una plataforma similar al EMBIT pero sin el amplificador PA/LNA, con el sistema optimizado se podría contar con una autonomía de 19 meses, dado que se podrá utilizar una potencia máxima de transmisión de $+7dBm$. Al disminuir la potencia se obtiene un menor consumo, pero será necesario colocar los nodos más cerca, debiendo utilizar más nodos para cubrir la misma superficie. Esto muestra un compromiso entre aumentar la separación entre los nodos (necesitando menos nodos para cubrir la misma superficie) y aumentar la vida útil de las baterías.

El diseño de hardware cuenta con una caja de protección $IP65$, por lo cual supera la protección $IP55$ indicada en el criterio de éxito del proyecto. Esta pro-

tección $IP65$ protege al nodo del polvo y del agua en un 100 %.

La alimentación de los sensores se habilita mediante los switches implementados. De esta manera, se logra proteger el MCU, pieza fundamental del circuito, ante posibles fallas provenientes de los sensores.

El hardware diseñado permite reducir el consumo del sistema, agregando una etapa de reducción de alimentación utilizando un conversor DC/DC que tiene una eficiencia mayor al 90 %. Si bien no se pudo comprobar su correcto funcionamiento, el conversor posibilita alimentar el circuito a diferentes tensiones, adaptándose así a cada aplicación en particular logrando una reducción de consumo energético.

Se diseñó un sistema pensado en su reutilización, para esto el hardware prevé elementos extra a los utilizados en RSItrust, de esta manera el diseño es fácilmente adaptable para su utilización en otras aplicaciones, o como parte de la realización de pruebas durante su desarrollo. A su vez, la aplicación fue desarrollada de forma de poder agregar, modificar o quitar funcionalidades fácilmente.

El sistema diseñado puede ser reproducido, gracias a que se redactó un manual, en el cual se describe detalladamente cómo reproducir el producto final obtenido tanto para hardware como para software, y describe diferentes posibilidades para su adaptación a las necesidades particulares de nuevas aplicaciones según lo necesiten. Se indica también cómo utilizar los códigos desarrollados en Contiki, los cuales fueron debidamente comentados para poder ser comprendidos y modificados en aplicaciones futuras. El manual redactado describe además cómo cargar y probar la aplicación sin necesidad de recurrir a otros manuales o documentos.

8.1. Dificultades

En primer lugar, el supuesto de que se contaría desde un principio con sensores de humedad y temperatura del aire, de humedad del suelo y el MCU a utilizar no se cumplió. Al no tener definido el hardware a utilizar, se realizó un cambio en el orden de ejecución de tareas. Se comenzó por la programación de la aplicación en una plataforma que no sería la utilizada para la aplicación final, para luego migrar a la plataforma definitiva, y se avanzó en el estudio de los protocolos de comunicación.

Una vez definido el hardware a utilizar se procedió a la compra de kits de desarrollo para la plataforma seleccionada. Debido a ciertos reglamentos existentes en Aduana para las compras realizadas por entes públicos, y algunos cambios en dichos reglamentos, estos elementos fueron retenidos más tiempo del usual al ingresar al país, provocando nuevos retrasos en la ejecución del proyecto.

Otra dificultad enfrentada por el equipo de proyecto en esta etapa fue la subestimación de los tiempos que requirió el estudio del sistema operativo Contiki,

estudio de los protocolos de comunicación y su implementación.

Durante el diseño del hardware, agentes externos al grupo fueron influyentes debido a solicitudes de realización de cambios y agregados al circuito, lo que resultó en la necesidad de varias iteraciones de diseño requiriendo una mayor dedicación de tiempo por parte de los integrantes del grupo.

Una vez culminado el diseño del nodo se procedió a realizar la compra de los componentes necesarios para su construcción. Estas compras fueron retrasadas, ya que se esperó a realizarlas junto con las de otros grupos para reducir costos de envío.

Nuevos retrasos en Aduana y el hecho de que los elementos arribaron al país en una fecha desafortunada, donde personas involucradas en los trámites se encontraban de licencia, sumado esto a errores de facturación por parte de proveedores continuaron generando trabas en la ejecución del proyecto.

Se cometieron errores en la elección de los componentes comprados y se detectaron errores de diseño del PCB, esto derivó en nuevos contratiempos dado que se debió realizar modificaciones en el prototipo fabricado. Esto nos deja el aprendizaje de que es más conveniente realizar un prototipo más sencillo, de montaje pasante, para probarlo y, luego de validado el diseño enviar a fabricar el hardware definitivo.

8.2. Evaluación del Proyecto

Luego de finalizado el desarrollo del proyecto, se realizó una autocrítica por parte del equipo evaluando su ejecución. En esta sección se presenta una evaluación general de la ejecución del proyecto y aprendizajes, los cuales se espera que sean útiles para quien desee llevar adelante un proyecto de estas características.

8.2.1. Retrospectiva

Una tarea fundamental para poder llevar adelante un proyecto es una correcta planificación. Esta no es tarea sencilla, requiere experiencia, conocimiento del tiempo que pueden llevar las tareas o capacidad para estimarlo. En este caso, para todos los integrantes del equipo éste fue el primer proyecto llevado adelante desde su planificación inicial. Debido a la inexperiencia del equipo, se cometieron errores en la planificación, por lo cual se debió replanificar.

En primera instancia, se organizaron las tareas en un orden de ejecución poco eficiente en la planificación original. Se planificó el desarrollo de la aplicación para luego estudiar los protocolos de red a utilizar. Se debió invertir este orden, debido a que se debieron conocer bien los protocolos y su implementación en Contiki OS

para que sea posible desarrollar la aplicación.

Se subestimaron los tiempos de estudio. En la práctica esta tarea llevó bastante más tiempo que el estimado, retrasando el inicio de las siguientes tareas planificadas. Este es un punto importante al planificar un proyecto, el tiempo de las tareas se debe estimar bien, y es algo que se entrena con el tiempo, aunque es algo bastante difícil.

El desarrollo y construcción del hardware también fue subestimado. No se tuvo en cuenta que esta tarea podía llevar varias iteraciones de diseño y varios agregados que surgieron a lo que se pensó inicialmente, lo que resultó en atrasos de los tiempos establecidos. Otro aspecto que no se tuvo en cuenta para esta tarea fue la dependencia de agentes externos, dado que la compra de componentes y la fabricación del impreso se hicieron en el exterior, esto resulta en demoras por envíos y demoras en Aduana. Este punto no es para nada menor, dado que por ser compras a través de un ente público los elementos fueron retenidos necesitando trámites extra para poder obtenerlos.

Si bien se dieron diferentes contratiempos, realizando nuevas planificaciones y limitando el alcance del proyecto se logró llevar adelante este proyecto. Un punto importante fue la decisión de realizar un prototipo en el mercado local, el cual presentó limitaciones del punto de vista de la fabricación, pero permitió detectar fallas de diseño, repararlas y realizar pruebas de funcionamiento del nodo. Cabe destacar que los inconvenientes fueron previstos como posibles riesgos al inicio del proyecto y algunas de las acciones tomadas fueron planificadas desde el inicio para mitigarlos en caso de que sucedieran.

Una buena práctica durante el proyecto fue realizar reuniones periódicas con los tutores. Esto permitió una fluida comunicación, pudiendo evacuar dudas, recibir consejos, realizar propuestas en cuanto a la implementación de la aplicación, etc.

Otra buena costumbre adoptada por el equipo fue documentar durante el desarrollo de todo el proyecto, facilitando a la hora de realizar la documentación final.

8.2.2. Aprendizaje

Se resume aquí lo que se aprendió durante el proyecto. Más allá de conocimientos técnicos se obtuvo un aprendizaje en cuanto a la ejecución de un proyecto, comenzando por su planificación y durante el transcurso del mismo hasta su finalización y cierre.

En cuanto a la planificación del proyecto, tener en cuenta que las tareas pueden demorar más en llevarse adelante que lo estimado en un principio. Se debe

ser realista y no demasiado optimista al estimar estos tiempos. Además, se aconseja dejar previstos buffer de tiempo en cada tarea, de esta manera se cuenta con cierta flexibilidad de tiempo al llevarlas a cabo. Planificar la realización de tareas ejecutándose al mismo tiempo por diferentes integrantes del equipo aporta a la eficiencia de utilización de recursos y acorta el tiempo total del proyecto.

Para proyectos en los cuales se diseñe hardware con cierto nivel de complejidad, y sea necesario enviar a fabricar en el exterior, se recomienda realizar un prototipo más sencillo, utilizando componentes electrónicos pasantes (through en lugar de superficiales, si es el caso), para poder rápidamente realizar cambios de manera más sencilla. Teniendo en cuenta las demoras que se generan al realizar compras en el exterior, realizar cambios en el diseño puede llevar a que el arribo de nuevos componentes se retarde, generando nuevos atrasos. Luego de validado el diseño enviar a fabricar el hardware definitivo.

Como una buena práctica, documentar durante el desarrollo de todo el proyecto, facilita a la hora de realizar la documentación final. Optimizando tiempos y evitando el olvido de punto importantes a mencionar.

Algo importante a tener en cuenta es la comunicación entre los integrantes del equipo y con los tutores del proyecto. De esta manera todos están al tanto del estado del proyecto y se da lugar a la discusión en las decisiones a tomar, contando con la experiencia de los tutores.

8.3. Trabajos Futuros

En esta sección se proponen posibles trabajos futuros, partiendo del desarrollo del presente proyecto. Estas propuestas apuntan a mejoras de la aplicación, eficiencia y versatilidad de código, utilizando este sistema como base de sistemas más complejos, y aplicables a diferentes situaciones.

Como se mencionó en el capítulo 5, la aplicación permite configurar un rango de temperatura y humedades en el cual el usuario considere que estos parámetros no son perjudiciales para el cultivo. Una posible mejora es implementar un sistema de alarmas, a partir del cual si una medida está por fuera del rango definido el nodo envía inmediatamente un mensaje de alarma al cliente informando de lo sucedido. A su vez se podría implementar un Gateway que permita, a través de la red celular, enviar mensajes de alarma al usuario y subir los datos obtenidos a una base de datos online en donde se podrían graficar las medidas y procesar la información.

Otra posible mejora es referida a los pines sensores de presencia de los conectores de los sensores. Como el sistema es capaz de identificar si el sensor está conectado o no, se podría modificar el código de tal manera que si uno se los sensores no está conectado, el sistema no genere el recurso asociado. Esto permitiría uti-

lizar los nodos de forma eficiente sin alguno de los sensores conectados. Además, dentro del proceso medir, el cual enciende periódicamente los sensores, se podría condicionar el encendido observando si el sensor está conectado o no. A su vez definiendo palabras de configuración se podría indicar si el sistema cuenta o no con pines sensores de presencia, haciendo el sistema más versátil.

Para mejorar el funcionamiento del sistema se podrían implementar tres procesos, uno para cada sensor, en lugar de utilizar un único proceso como *process_medir*. Esto permitiría, por ejemplo, que al encender un sensor y esperar el tiempo que necesita el sensor para adquirir una medida, el MCU pueda ejecutar otro proceso y adquirir la medida de otro sensor. Esto agregaría versatilidad, pudiendo medir cada sensor con un período distinto dependiendo de las variaciones de la medida o de la criticidad se la misma.

Si se quisieran recibir las últimas n medidas en un único payload o un promedio de las mismas, se podría crear una cola circular por sensor, en donde se irían guardando las medidas cuando son tomadas.

Para que los elementos del circuito diseñado y su ubicación queden preestablecidos, se podría crear en Contiki una plataforma para el hardware diseñado.

Otra posible mejora es la utilización de paneles solares en lugar de pilas de ion de litio. De esta manera la vida útil del sistema se incrementaría considerablemente. Evitando la necesidad de cambiar las baterías periódicamente.

Actualmente, el sistema se resetea si no se conecta ningún sensor o si pierde comunicación con el sensor (por ejemplo, al cortarse el cable), tal como se explica en la sección 5.4.5. Esto se debe a los *drivers* de $I2C$, los cuales incluyen esperas *busy-wait*, utilizando un *while* a la espera de una condición que confirma la correcta comunicación con el sensor. Para mejorar esto se podrían implementar los *drivers* utilizando procesos y de esta manera al encontrar error de comunicación poner un $PROCESS\_WAIT\_EVENT$ permitiendo al MCU atender otros procesos y no quedarse ocupado esperando la condición. De esta manera al momento de chequear que el bus está ocupado el MCU va a setear un temporizador y va a ir a atender otros procesos. Cuando dicho temporizador llegue a cero el MCU agregará en la cola de atención el evento relacionado y cuando le llegue su turno el MCU volverá al proceso del *driver* y chequeará nuevamente la condición de si el bus está libre. En el peor caso, en el cual no hay sensor conectado en ningún momento, el MCU periódicamente volverá a verificar la condición y al no cumplirse seguirá con otros procesos no trancando el sistema y por ende el *watchdog-timer* no expirará.

Con estas mejores se podría obtener un sistema más versátil, eficiente a nivel de procesamiento y más atractivo para el usuario final.

Apéndice A

Plan de proyecto

A.1. Introducción

Como todo proyecto, RSItrust fue planificado marcando objetivos, delimitando su alcance, previendo la existencia de eventos que podrían poner en riesgo su normal ejecución según lo planificado. Se estudiaron antecedentes y se utilizaron nuevas tecnologías. Durante el transcurso del proyecto se enfrentaron diversas dificultades e imprevistos que forzaron a nuevas planificaciones y redefinición de alcance.

Este anexo resume cómo fue planificado el proyecto y las modificaciones que sufrió durante el transcurso del mismo.

A.2. Planificación inicial

A.2.1. Objetivo general

El objetivo del proyecto es diseñar e implementar una Red de Sensores Inalámbricos, con el fin de obtener un producto confiable (diseñado para ser tolerante a fallos, robusto y debidamente testeado) y pensado para ser producido en mayor escala.

A.2.2. Alcance original

El alcance del proyecto abarca los siguientes puntos:

1. Diseño de circuitos auxiliares para conexión e integración de sensores a cada nodo que conforma la red.

2. Diseño y construcción del montaje físico de protección de los dispositivos.

3. Diseño, análisis e implementación de la red. Este punto incluye la implementación de la aplicación y modificación de los parámetros necesarios de los protocolos de comunicación de red.

Queda fuera del alcance del proyecto:

1. Selección del microcontrolador y sensores a utilizar.

2. Diseño del módulo principal de los nodos de sensores inalámbricos (placa conteniendo MCU y radio).

3. Diseño hardware del nodo base.

4. Integración del nodo base a la red.

A.2.3. Supuestos

El proyecto se desarrolla bajo los siguientes supuestos:

1. Contiki OS cuenta con documentación suficiente (tutoriales, wiki, ejemplos y código comentado) que permita desarrollar las aplicaciones y luego poder realizar modificaciones en la implementación de los protocolos.

2. Se cuenta con los sensores de humedad y temperatura del aire, de humedad del suelo y el MCU.

3. Se cuenta con la financiación de INIA para la realización del Proyecto.

4. Se cuenta con instrumentos para realizar las medidas y apoyo para pruebas en campo.

A.2.4. Diagrama de Gannt

En la Figura A.1 se presenta el diagrama de Gannt que se planeó al iniciar el proyecto. Según esta planificación el proyecto terminaría en abril del año 2015.

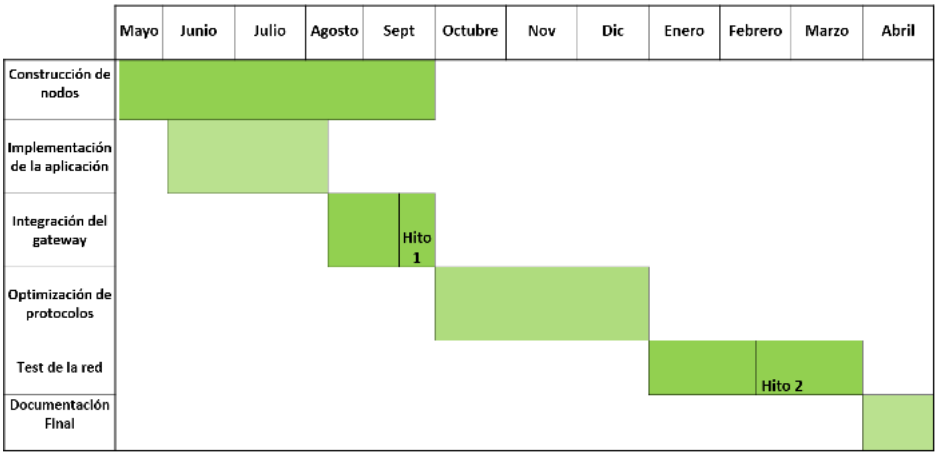


Figura A.1: Planificación original

A.3. Primer replanificación

Durante la primera etapa del proyecto surgieron dificultades, las cuales provocaron que no se haya podido seguir ejecutando según lo planificado.

En primer lugar, el supuesto de que se contaría desde un principio con sensores de humedad y temperatura del aire, de humedad del suelo y el MCU a utilizar no se cumplió. Al no tener definido el hardware a utilizar, se realizó un cambio en el orden de ejecución de tareas. Se comenzó por la programación de la aplicación en una plataforma que no sería la utilizada para la aplicación final, para luego migrar a la plataforma definitiva, y se avanzó en el estudio de los protocolos de comunicación.

Una vez definido el hardware a utilizar se procedió a la compra de kits de desarrollo para la plataforma seleccionada. Debido a ciertos reglamentos existentes en Aduana para las compras realizadas por entes públicos, y algunos cambios en dichos reglamentos, estos elementos fueron retenidos más tiempo del usual al ingresar al país, provocando nuevos retrasos en la ejecución del proyecto.

Otra dificultad enfrentada por el equipo de proyecto en esta etapa fue la subestimación de los tiempos que requirió el estudio del sistema operativo Contiki, estudio de los protocolos de comunicación y su implementación.

Estas dificultades encontradas resultaron en una replanificación, la cual se muestra en la Figura A.2. Allí se aprecia el cambio en el orden de las tareas, comenzando por la implementación de la aplicación, para luego continuar con el diseño de hardware y construcción de nodos.

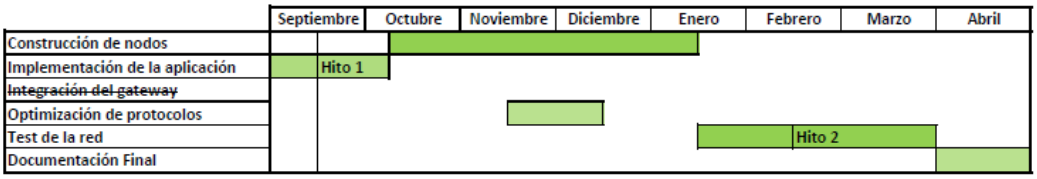


Figura A.2: Replanificación al hito 1

Obsérvese que tareas que debían ser ejecutadas en paralelo se debieron ejecutar individualmente, dedicándose a realizar cada tarea los 3 miembros del equipo. Esto provocó que sea necesario tomar la decisión de limitar el alcance del proyecto para poder mantener la fecha de finalización del mismo. Para este propósito, como se puede ver en la Figura A.2 se eliminó la integración de un gateway a la red. De todas formas, el gateway iba a ser desarrollado por un grupo de la asignatura Sistemas embebidos para tiempo real y se iba a integrar al proyecto RSItrust solo en caso de ser fácilmente integrable. Esto último no fue así, por lo tanto se quitó de la planificación.

A.4. Segunda replanificación

Luego del primer hito siguieron surgiendo dificultades a enfrentar por el equipo de proyecto. Según la primera replanificación, se continuó con el desarrollo de la aplicación, para luego realizar la construcción del hardware.

Durante el diseño de hardware, agentes externos al grupo fueron influyentes debido a solicitudes de realización de cambios y agregados al circuito, lo que resultó en la necesidad de varias iteraciones de diseño requiriendo una mayor dedicación de tiempo por parte de los integrantes del grupo.

Una vez culminado el diseño del nodo se procedió a realizar la compra de los componentes necesarios para su construcción. Estas compras fueron retrasadas, ya que se esperó a realizarlas junto con las de otros grupos para reducir costos de envío.

Nuevos retrasos en Aduana y el hecho de que los elementos arribaron al país en una fecha desafortunada, donde personas involucradas en los trámites se encontraban de licencia, sumado esto a errores de facturación por parte de proveedores continuaron generando trabas en la ejecución del proyecto.

Se realizó una nueva replanificación intentando mitigar estos trastiempos, la cual se muestra en la Figura A.3 . Se avanzó en el estudio de los protocolos a utilizar, esta tarea se extendió mas allá de lo planificado debido a la inexperiencia en la utilización del modelo de simulación que se utilizó como herramienta.

Figura A.3: Replanificación al hito 2

Apéndice B

Manual de usuario

B.1. Introducción

Este anexo explica cómo reproducir el producto logrado en este proyecto. Comienza por una guía de construcción de un nodo mostrando la flexibilidad del diseño presentando diferentes posibilidades de construcción, para luego mostrar cómo programar dicho nodo, y las pruebas necesarias para comprobar el correcto funcionamiento del sistema. Se asume que el lector posee los archivos necesarios para la fabricación y programación del nodo.

B.2. Fabricación de un nodo

Esta sección muestra cómo construir un nodo diseñado en el proyecto RSItrust a partir de los archivos disponibles. Se explica cómo crear la placa base y las placas auxiliares para la integración de los sensores. Se sugieren además proveedores para la fabricación de circuitos impresos y compra de componentes.

B.2.1. Construcción de placa base

Se debe contar con los archivos GERBER del PCB, los cuales se listan a continuación:

- .GTL: Top copper
- .GBL: Bottom copper
- .GTO: Top silkscreen
- .GBO: Bottom Silkscreen
- .GTS: Top soldermask

- .GBS: Bottom soldermask
- .GTP: Top paste
- .GBP: Bottom paste
- .TXT: Drill file

Los archivos GERBER disponibles para la placa base del nodo corresponden al PCB panelizado en un arreglo de 2x2 placas. En la Tabla B.1 se muestran las medidas de cada PCB y del arreglo total, información necesaria al enviar estos archivos al fabricante.

	PCB	Arreglo
$x(inch)$	2,8	5,65
$y(inch)$	3,05	6,15

Tabla B.1: Medidas GERBER

La Tabla B.2 es una lista de los elementos necesarios para la fabricación del PCB base. En ella se muestran los nombres correspondientes al PCB. Se incluye además su código de elemento de DigiKey[1], donde se pueden comprar dichos elementos. En el caso del módulo EMBIT, dirigirse a la página de su fabricante[2].

Los pasos a seguir para la construcción del PCB base del nodo son los siguientes:

- Enviar a fabricar el PCB utilizando los archivos GERBER realizados por RSItrust. Un posible proveedor es Advanced circuits[3].
- Soldar el conversor de tensión TPS62740 utilizando un horno, ya que este elemento es muy pequeño.
- Soldar el módulo EMBIT.
- Soldar el resto de los elementos, dejando por último los conectores, solo por comodidad.

[1] `http://www.digikey.com/`
[2] `http://www.embit.eu/products/wireless-modules/emb-z2538pa/`
[3] `http://www.4pcb.com/`

Elemento	**Valor**	**Package**	**Código DigiKey**
C3	$10\mu F$	C0603	490-3896-1-ND
C4	$1nF$	C1005	445-13480-1-ND
C5	$0{,}1\mu F$	C0402	1276-1210-1-ND
C6	$0{,}1\mu F$	C0402	1276-1210-1-ND
C7	$0{,}1\mu F$	C0402	1276-1210-1-ND
C8	$0{,}1\mu F$	C0402	1276-1210-1-ND
C9	$0{,}1\mu F$	C0402	1276-1210-1-ND
C10	$0{,}1\mu F$	C0402	1276-1210-1-ND
C11	$10\mu F$	C0603	490-3896-1-ND
EC-05	-	-	CP-3524SJTR-ND
EC-05	-	-	CP-3524SJTR-ND
$EM-PORT_1$	-	-	A116319CT-ND
$EM-PORT_2$	-	-	A116319CT-ND
G1 (portapilas)	-	-	BH2AA-PC-ND
JP-LED	-	-	952-1913-ND
JTAG	-	-	S9013E-05-ND
L1	$10\mu H$	No estándar	445-9630-1-ND
LED1	-	0603	160-1447-1-ND
PWR	-	0603	160-1447-1-ND
PWR_ON	-	-	401-2002-1-ND
R0	0Ω	R2512	RMCF2512ZT0R00CT-ND
R1	125Ω	M2012	Y1624125R000Q0W-ND
R2	0Ω	R2512	RMCF2512ZT0R00CT-ND
R3	25Ω	M2012	PAT25.0BCT-ND
R4	$10k\Omega$	R2512	PT10.0KAFCT-ND
R5	$1k\Omega$	R2512	PT1.0KXCT-ND
R6	$0{,}7\Omega$	R1206	RHM.70CATR-ND
R7	$0{,}7\Omega$	R1206	RHM.70CATR-ND
R8	$0{,}7\Omega$	R1206	RHM.70CATR-ND
R9	0Ω	R2512	RMCF2512ZT0R00CT-ND
R10	0Ω	R2512	RMCF2512ZT0R00CT-ND
RST	-	-	P12959SCT-ND
SELECT	-	-	P12959SCT-ND
SHT	-	-	CP-435107RS-ND
SW-EC5	-	-	FPF2004CT-ND
SW-SHT	-	-	FPF2004CT-ND
SW-TMP	-	-	FPF2004CT-ND
TPS62740	-	-	296-37134-1-ND
U$1 (EMBIT)	-	-	No disponible
UART	-	-	609-4398-1-ND
UART	-	-	609-4398-1-ND
USER	-	-	P12959SCT-ND
VSEL	-	-	609-3487-1-ND

Tabla B.2: Partlist placa principal

B.2.2. Placas auxiliares

También se cuenta los archivos GERBER para la realización del PCB auxiliar que permite la integración del sensor de temperatura TMP75c al nodo. Estos archivos se describen de la misma manera que para el PCB base en la sección B.2.1.

Este PCB no tiene mayores complicaciones, por lo tanto es posible realizarlo de forma manual, o utilizando equipamiento disponible en el IIE.

La Tabla B.3 es una lista de los componentes necesarios para la construcción de este PCB auxiliar. Cuenta también con su código correspondiente del distribuidor DigiKey, donde se pueden conseguir estos elementos. Se recomienda una vez más, soldar los componentes del circuito, dejando para el último el conector.

Elemento	**Valor**	**Package**	**Código DigiKey**
1 (conector)	-	-	3M9449-ND
C1	0,01μF	C0805	490-1664-1-ND
R1	10$k\Omega$	R2512	PT10.0KAFCT-ND
R2	10$k\Omega$	R2512	PT10.0KAFCT-ND
R3	10$k\Omega$	R2512	PT10.0KAFCT-ND
U$ (TMP75C)	-	-	296-37792-5-ND

Tabla B.3: Partlist placa TMP75C

Para el sensor de humedad SHT25 no se diseñó un PCB auxiliar, sino que se utilizó el módulo MOD-1018 de Embedded Adventures[4]. Si se desea se puede diseñar un PCB para la integración de este sensor al nodo utilizando el circuito que sugiere el fabricante en su hoja de datos.

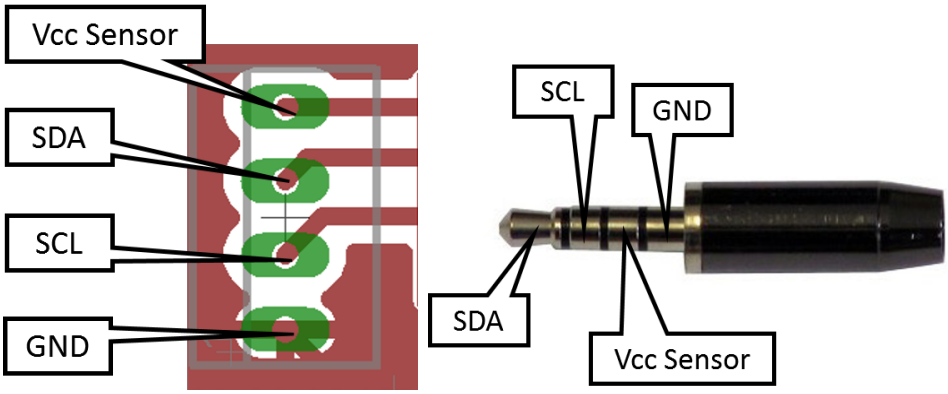
El sensor de humedad del suelo EC-05 no necesita una placa auxiliar, éste tiene incorporado un conector de 3,5mm, sin necesidad de agregar elementos extra. Un posible distribuidor donde se puede conseguir este sensor es AliExpress[5].

Una vez fabricadas las placas se deben conectar utilizando los conectores previstos, los cuales se listan en la Tabla B.4. Se debe soldar un cable, uniendo las señales correspondientes de dichos conectores. La Figura B.1 muestra los contactos correspondientes para cada señal en cada conector.

[4] `www.embeddedadventures.com`
[5] `http://www.aliexpress.com/`

Conector	Código DigiKey
3,5mm macho	839-1036-ND
2,54mm hembra	A34192-ND

Tabla B.4: Conectores para integración de sensores



(a) Conector 2,54mm (b) Conector 3,5mm

Figura B.1: Conectores para integración de sensores

B.2.3. Prueba de hardware

Luego de fabricado el nodo, se procede a realizar pruebas que determinen su correcta construcción.

Test de continuidad

Se debe comenzar realizando pruebas de continuidad, y de esta manera lograr determinar que se hayan conectado correctamente los elementos del circuito. En caso de detectar cortocircuitos, se deben aislar las diferentes partes del circuito, realizando cortes en pistas y quitando elementos de ser necesario. Repetir el proceso hasta detectar dónde está la falla, y si es posible enmendarla.

Test de alimentación

Luego de haber testeado la continuidad del circuito, se procede a alimentar el mismo. Para esto es recomendable utilizar una fuente externa de $3V$ con control de corriente en bornes del portapilas. De esta manera, en el caso de haber cortocircuitos no detectados previamente, los cuales puedan provocar un alto consumo de corriente, la fuente cortará la alimentación evitando posibles daños a los elementos del circuito. Si sucede esto, volver a realizar los test de continuidad buscando la

falla de cortocircuito.

Si el consumo de corriente es estable, se debe pasar a medir la alimentación del circuito, la cual es regulada por el conversor DC/DC. La tensión de alimentación puede seleccionarse mediante un jumper VSEL, y se puede medir utilizando un punto de test provisto para este propósito. La Figura B.2 muestra la ubicación de estos elementos.

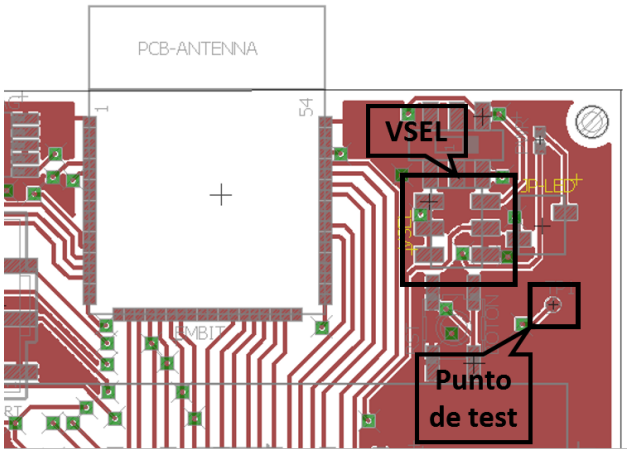
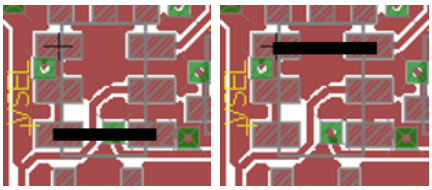


Figura B.2: Elementos para test de alimentación

El jumper VSEL puede colocarse de manera que el pin V_{SEL3} del conversor quede conectado a la alimentación de la fuente externa, a GND o a un GPIO del MCU. La Figura B.3a muestra la posición del jumper VSEL para alimentar el circuito a 2,1V, mientras que la Figura B.3b muestra su posición para alimentar a 2,5V. Se debe colocar el jumper en las distintas posiciones y medir en el punto de test la tensión, y verificar si es correcta.



(a) 2,1V (b) 2,5V

Figura B.3: Configuraciones de VSEL para distintas alimentaciones

Elementos restantes

Las pruebas de los restantes elementos del circuito se pueden realizar utilizando programas de test, que lean entradas, escriban salidas y envíen mensajes mediante UART. Para esto, Contiki OS cuenta con una gran variedad de ejemplos que se pueden utilizar. En la sección B.3 se explica cómo compilar y grabar un programa en el MCU del nodo.

De esta manera ya se cuenta con un nodo pronto para ser programado y utilizado en una red de sensores.

B.2.4. Elementos extra y opcionales

El diseño cuenta con algunos elementos que permiten decidir diferentes maneras de utilizar el PCB. De aquí en adelante se mostrarán las diferentes posibilidades que este diseño ofrece.

Selector de tensión

Está previsto en el diseño un jumper llamado VSEL cuya ubicación se muestra en la Figura B.2, mediante el cual se puede seleccionar la tensión a la que se alimentará el circuito. Este conecta el pin V_{SEL3} del conversor DC/DC a la fuente de alimentación externa[6], GND o un GPIO del MCU. El conversor tiene los pines V_{SEL1}, V_{SEL2} y V_{SEL4} conectados a la alimentación. La Tabla B.5 muestra la tensión de salida en los diferentes casos.

$V_{OUT}(V)$	V_{SEL4}	V_{SEL3}	V_{SEL2}	V_{SEL1}
2.1	0	0	1	1
2.5	0	1	1	1

Tabla B.5: Configuración de tensión de salida del conversor

De esta manera, se podrá seleccionar una tensión de alimentación al circuito de $2{,}1V$ o de $2{,}5V$. En la Figura B.4 se muestra el jumper con sus posibles configuraciones. Para obtener una alimentación de $2{,}5V$ se debe colocar como en la Figura B.4b. En la posición que se muestra en la Figura B.4a se logra una alimentación de $2{,}1V$, mientras que la posición de la Figura B.4c conecta V_{SEL3} al GPIO PA4 del MCU. RSItrust utiliza la opción de conectar V_{SEL3} al GPIO del MCU, para de esta manera poder seleccionar la tensión de alimentación en tiempo de ejecución,

[6]Para la aplicación RSItrust se utiliza el portapilas, el cual lleva 2 pilas AA de $1{,}5V$

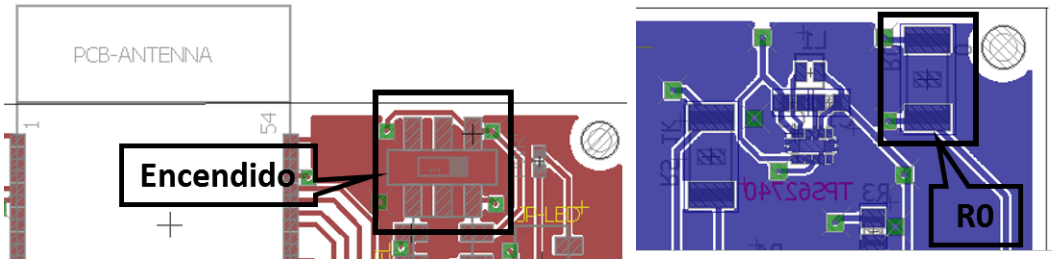
siendo 2,5V al Medir con el EC-05 y 2,1V el resto del tiempo.

(a) 2,1V (b) 2,5V (c) GPIO del MCU

Figura B.4: Selector de tensión VSEL

Switch de encendido

El diseño cuenta con un switch tipo *slide* para el encendido de la placa. En paralelo al mismo en el diseño hay una resistencia de 0Ω (R0), previsto para el caso en que se prefiera no utilizar el switch de encendido. Cuál opción tomar es a preferencia del usuario. En la Figura B.5 se indica la posición de cada uno de estos elementos.



(a) Posición del switch (b) Posición de R0

Figura B.5: Switch de encendido

Switches de habilitación de sensores

El hardware diseñado cuenta con Switches que habilitan la alimentación de los sensores. Estos pueden no ser utilizados si así se desea. Para eso, se deben colocar en su lugar resistencias de 0Ω, las cuales están previstas en el diseño (ocupando el mismo lugar que los switch), conectando la alimentación de los sensores directamente a los GPIO del MCU. La Figura B.6 muestra la ubicación de estos elementos.

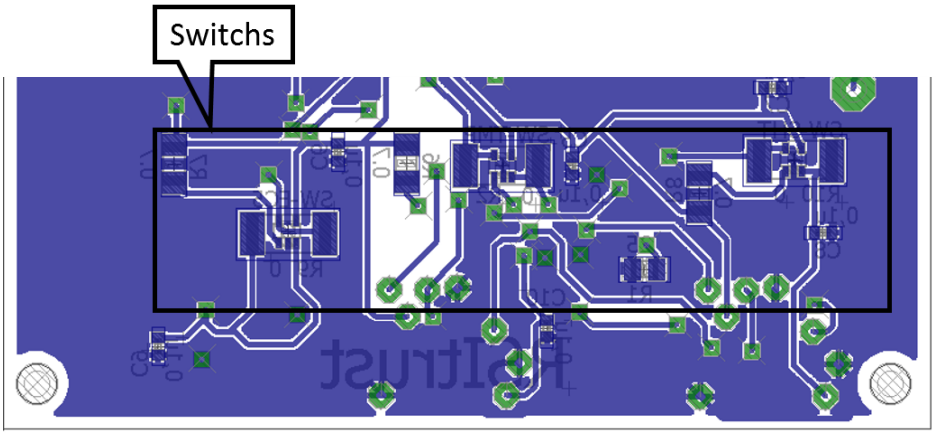


Figura B.6: Switches habilitadores de sensores

Conectores de acceso a puertos del MCU

El diseño cuenta con conectores llamados $EM - Port\_1$ y $EM - Port\_2$, los cuales dejan a disposición todos los pines que el módulo EMBIT tiene disponibles. Estos puertos están colocados en el mismo lugar físico que el portapilas utilizado en este proyecto, por lo tanto si se requiere el uso de estos conectores no se podrá colocar el portapilas, debiéndose utilizar una fuente externa de alimentación. Se puede ver su ubicación en la Figura B.7. De esta manera, otros grupos de proyecto podrán aprovechar el diseño utilizándolo en diferentes aplicaciones que puedan requerir el uso de los diferentes puertos del EMBIT.

La Tabla B.6 y la Tabla B.7 muestran el mapa resultante de los pines disponibles del módulo EMBIT a los conectores genéricos incluidos, los cuales fueron llamados $EM - Port\_1$ y $EM - Port\_2$ respectivamente.

LED indicador de alimentación

El diseño cuenta con un LED indicador de que el circuito está alimentado, el cual está conectado en serie con un jumper permitiendo decidir su utilización o no. Al colocar el jumper, el LED encenderá cuando el circuito esté alimentado. Si no se coloca, el LED queda inutilizado, permitiendo un mayor ahorro de energía. la Figura B.8 indica su ubicación en el PCB.

B.2.5. Utilización a futuro

Si se desea tomar el hardware diseñado como punto de partida para proyectos futuros, se debe contar con los siguientes archivos:

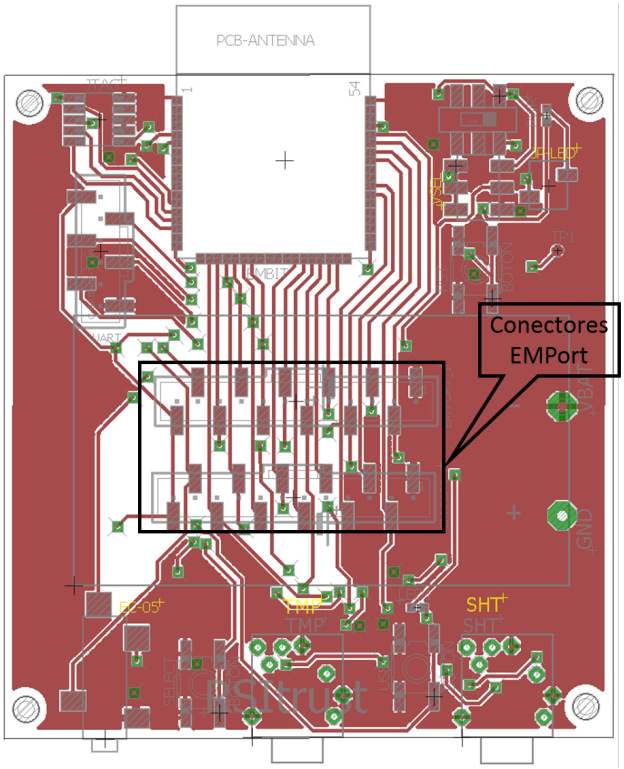


Figura B.7: Switches habilitadores de sensores

Embit	$EM-Port\_1$
18; 37 (V_{CC})	12
6	11
21	10
23	9
25	8
31	7
33	6
35	5
44	4
48	3
50	2
1; 54 (GND)	1

Tabla B.6: Mapeo EMBIT - $EM-Port\_1$

Embit	$EM - Port\_2$
18; 37 (V_{CC})	12
5	11
20	10
22	9
24	8
30	7
32	6
34	5
43	4
47	3
49	2
1; 54 (GND)	1

Tabla B.7: Mapeo EMBIT - $EM - Port\_2$

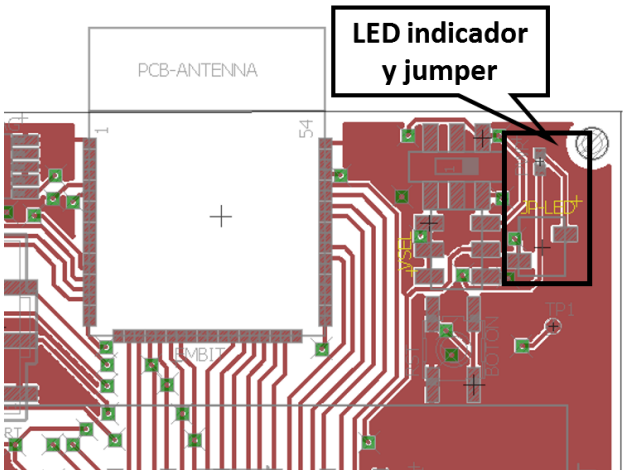


Figura B.8: LED indicador de alimentación

- Esquemático y PCB realizados en Eagle versión 7.2[7]
- Librería de elementos para Eagle "PFC.lbr" creada en este proyecto
- Archivo "Replicate_Board.ulp" utilizado para panelizar el PCB[8]
- Archivo gerb274x-PFC.cam para utilizar con la herramienta CAM procces-

[7]Eagle se puede descargar gratuitamente de la página web de CadSoft `http://www.cadsoftusa.com/download-eagle/`

[8]Más información sobre como panelizar en `http://www.cadsoftusa.com/downloads/file/replicate_board.rar`

sor del Eagle, la cual genera el GERBER del PCB[9]

La Tabla B.8 muestra la asignación de pines del proyecto RSItrust.

CC2538		**Circuito**		
Puerto	**Pin**	**Pin EMBIT**	**Señal**	**Elemento**
PA3	19	22	RTS; SELECT	BSL
PA2	18	23	CTS	
PA1	17	24	RX	
PA0	16	25	TX	
PB7	48	13	Data Output	JTAG
PB6	49	14	Data Input	
JTAG-TCK	47	15	Clock	
JTAG-TMS	46	16	Mode Select	
PD2	27	30	TMP Enable	TMP
PC1	13	31	SDA	
PD1	26	33	SCL	
PA5	21	44	Sensor Presencia	
PC0	14	35	EC Enable	EC-05
PA7	23	5	EC Output	
PA6	22	48	Sensor Presencia	
PD2	27	30	SHT Enable	SHT
PC1	13	31	SDA	
PD1	26	33	SCL	
PA5	21	44	Sensor Presencia	
RESET	28	51	RESET	RESET
PA4	20	43	Chip Enable	DC/DC
PC6	7	21	Boton Usuario	Boton Usuario
PD6	14	47	LED Usuario	LED Usuario

Tabla B.8: Pines utilizados del EMBIT por RSItrust

B.3. Programación de Hardware

En esta sección se explican los pasos necesarios para programar un nodo y probar la aplicación RSItrust. Primero se detallan los pasos para poder programar el CC2538 y luego se presenta una guía para probar las prestaciones de la aplicación. Para esto el lector debe contar con los archivos listados en la Tabla B.9 y ubicarlos

[9]Más información sobre el CAM Proccessor de eagle en esta entrada: `http://www.todopic.com.ar/foros/index.php?topic=24080.0`

en la ruta indicada.

B.3.1. Instant Contiki

Para comenzar a trabajar en Contiki es conveniente utilizar la máquina virtual Instant Contiki. [10] Este es un entorno de desarrollo completo para Contiki. Es una máquina virtual de Ubuntu que se ejecuta utilizando VMWare[11] y contiene todas las herramientas de compilación, compiladores y el simulador Cooja ya instalados.

B.3.2. Programación del CC2538DK

A continuación se describen los pasos a seguir para programar la plataforma CC2538DK utilizando la máquina virtual Instant Contiki. Previamente se recomienda crear el siguiente alias:

cc2538dk='sudo python /home/user/contiki/tools/cc2538-bsl/cc2538-bsl.py -e -w -v -p /dev/ttyUSB1'

1. **Instalar Uniflash para flashear el programa**
   Descargar la última versión para Linux de la herramienta Uniflash[12] y descomprimir el archivo. Luego abrir una terminal, cambiar los permisos de ejecución del instalador y ejecutarlo:

   $sudo\ chmod\ a+x\ uniflash\_setup\_3.1.0.00026.bin$
   $sudo\ ./uniflash\_setup\_3.1.0.00026.bin$

   Luego seguir los pasos del instalador seleccionando el tipo de instalación "custom" y soporte únicamente para los procesadores "Wireless Connectivity CCxxxx Cortex M Devices". No olvidar seleccionar también el soporte para el emulador JTAG.

2. **Instalar parche de Uniflash para CC2538**
   Descargar el parche para uniflash[13] y copiarlo en la ruta: /opt/ti/uniflashv3/ccs_base/DebugServer/bin.

3. **Instalar archivo de configuración de plataforma**
   Descargar el archivo de configuración de platafórma[14] y copiarlo en la ruta

[10] http://sourceforge.net/projects/contiki/files/Instant%20Contiki/
[11] https://my.vmware.com/web/vmware/info/slug/desktop_end_user_computing/vmware_workstation/11_0
[12] http://processors.wiki.ti.com/index.php/Category:CCS_UniFlash
[13] https://mega.co.nz/#!ks5XDICS!rbrLGtwWG7Y-SS_hKC31A_Y5uhkrXO2doIgjBIDKluM
[14] https://mega.co.nz/#!EgBjTKIb!uW13zMmQSq3Ax4CzZetKYuxyqZ-zZHY_ur0l59eEeYg

Archivos	**Observaciones**
rsitrust-server.c (contiki/example/rsitrust)	Servidor de la aplicación
Makefile (contiki/example/rsitrust)	Se configuraron los protocolos a usar
project-conf.h (contiki/example/rsitrust)	Se definieron banderas de configuración de protocolos
contiki-conf.h (contiki/platform/cc2538dk)	Se cambió el canal de comunicación
er-coap-observe.c (contiki/apps/er-coap)	Se modificó para hacer confirmable al Observe
rest-engine.c (contiki/apps/rest-engine)	Se modificó para poder configurar el periodo de Observación
cc2538-rf.c (contiki/cpu/cc2538/dev)	Se define la potencia de transmisión Se selecciona el uso del PA/LNA
i2c.c e i2c.h (contiki/cpu/cc2538/dev)	Drivers de la interfaz I2C
adc-sensor.c y adc-sensor.h (contiki/platform/cc2538/dev)	Drivers del conversor ADC
sht25.c y sht25.h (contiki/platform/cc2538/dev)	Drivers del sensor de humedad del aire
tmp75.c y tmp75.h (contiki/platform/cc2538/dev)	Drivers del sensor de temperatura
process_medir.c (contiki/example/rsitrust/resources)	Proceso encargado de tomar las medidas
res-temp.c (contiki/example/rsitrust/resources)	Recurso Observable de temperatura
res-humA.c (contiki/example/rsitrust/resources)	Recurso Observable de humedad del aire
res-humS.c (contiki/example/rsitrust/resources)	Recurso Observable de humedad del suelo
res-conf.c (contiki/example/rsitrust/resources)	Recurso configuración
configuracion.h (contiki/example/rsitrust/resources)	Archivos auxiliares para configuración
res-energest.c (contiki/example/rsitrust/resources)	Recurso Energest
res-vecinos.c (contiki/example/rsitrust/resources)	Recurso vecinos

Tabla B.9: Archivos de la aplicación RSItrust.

/opt/ti.

4. **Añadir el FTDI del SmartRF a la lista de dispositivos conocidos del SO**
   Cada vez que se reinicia la máquina virtual es necesario realizar los siguientes pasos antes de empezar a programar la placa. Con la placa conectada ejecutar los siguientes comandos en una terminal:

   $sudo\ su$
   $modprobe\ ftdi - sio$
   $echo\ 0403\ a6d1 > /sys/bus/usb - serial/drivers/ftdi\_sio/new\_id$
   $exit$

   Donde $0x403$ es el Vendor Id y $0xA6D1$ es el Product Id. Puede verificar estos datos con el comando $lsusb$.

5. **Compilar un programa**
   Para compilar un programa se debe abrir una consola nueva, dirigirse hasta la carpeta donde se encuentra el programa a compilar y ejecutar el siguiente comando:

   $make\ filename\ TARGET = cc2538dk\ NODEID = xx$

   En donde xx es un número entero correspondiente al ID que se desea asignar al nodo. Esto genera un archivo filename.bin.

6. **Flashear un programa**
   Se puede flashear tanto desde Windows como desde Ubuntu. Para flashear desde Windows se debe instalar el programa Flash Programmer 2 disponible en la página de Texas Instruments [29]. Conectar a la PC la Smart RF06 con el CC2538EM, buscar el dispositivo en el programa y elegir el archivo .bin a cargar. El programa permite borrar, programar y verificar el programa cargado. Es conveniente seleccionar estas tres opciones como se muestra en la Figura B.10. Luego de seleccionado el archivo y las opciones, presionar el botón play. El recuadro Status debería indicar que se comenzó a programar la placa y posterirmente confirmaría si el programa se verificó correctamente. Si la verificación no es correcta volver a presionar play.

   Si se desea programar en la placa de RSItrust, conectar la misma a la SmartRF06 como se muestra en la Figura B.9 y seguir el mismo procedimiento.

   Para flashear un programa desde Ubuntu a la placa primero se debe iniciar el 'boot loader'. Para iniciarlo presionar en la SmartRF06 el botón 'RESET' mientras se mantiene presionado el botón 'SELECT'. Luego, abrir una nueva

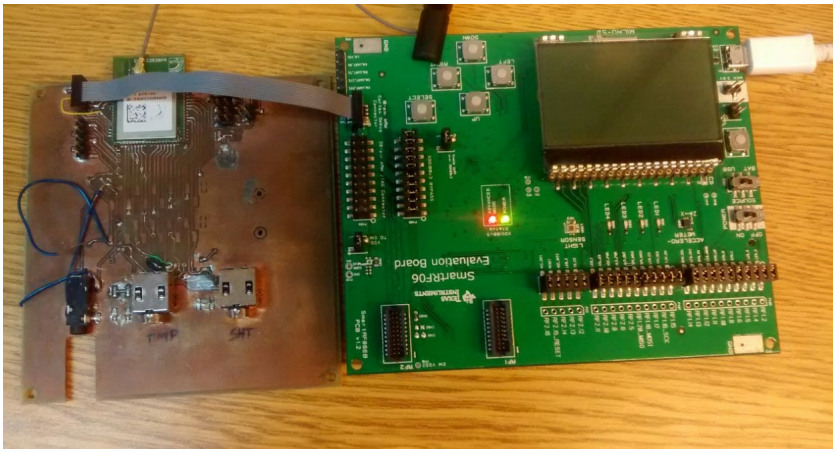


Figura B.9: Conexión placa RSItrust - SmartRF06.

terminal, dirigirse al directorio donde se encuentra el programa y ejecutar el siguiente comando (habiendo creado primero el alias que se describe al principio de la sección B.3.2):

$cc2538dk\ filename.bin$

Es muy importante asegurarse de que la verificación sea correcta ya que de lo contrario el programa puede haber quedado corrupto.

B.4. Guía para probar la aplicación RSItrust

Para probar la aplicación RSItrust se debe cargar un nodo con el programa border-router (el cual debe permanecer conectado a la PC) y al menos un nodo con el programa rsitrust-server. A continuación se describen los pasos necesarios para esto.

B.4.1. Programación de los nodos

1. Cargar rsitrust-server en un nodo:
   a) Abrir una terminal.
   b) Ejecutar:
   cd contiki/examples/rsitrust
   make TARGET=cc2538dk rsitrust-server NODEID=2
   c) Cargar el programa rsitrust-server en un nodo como se explica en la sección B.3.2
2. Cargar border-router en un nodo:

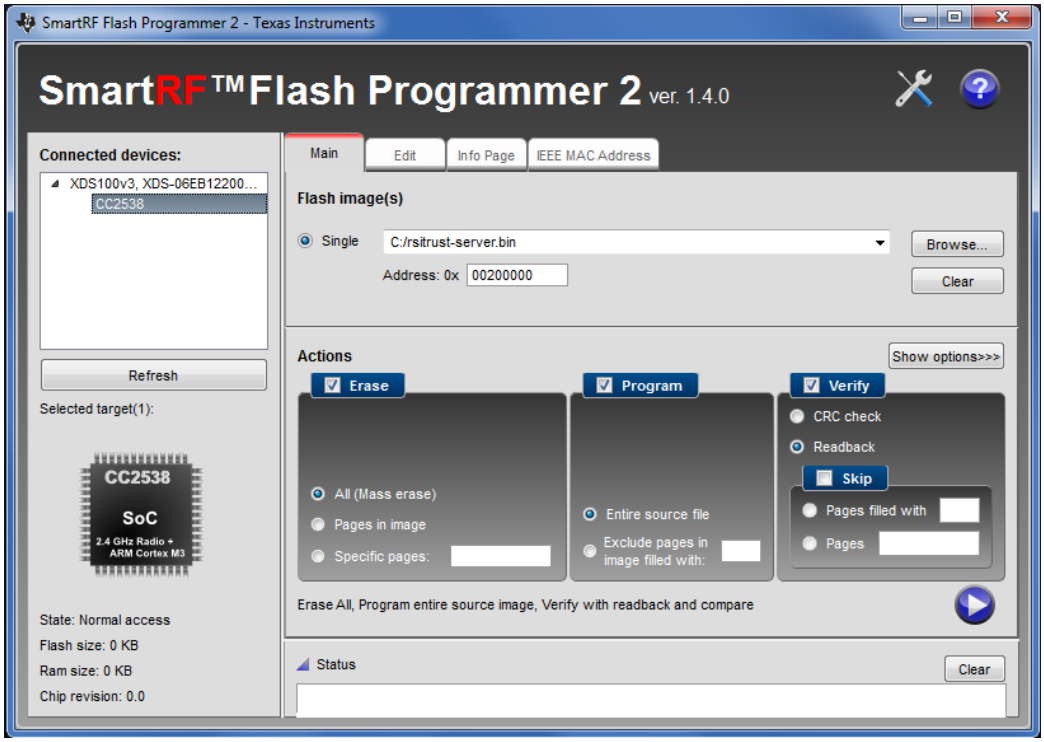


Figura B.10: Flash Programmer 2.

a) Abrir una nueva terminal.
b) Ejecutar:
cd contiki/examples/ipv6/rpl-border-router
make TARGET=cc2538dk border-router NODEID=1
c) Cargar el programa border-router en un nodo como se explica en la sección B.3.2

3. Conectar border-router:

a) Abrir una nueva terminal.
b) Ejecutar:
cd contiki/tools
make tunslip6
cd
sudo contiki/tools/tunslip6 -s /dev/ttyUSB1 aaaa::1/64
En la consola debería desplegarse la dirección IPv6 que se le asignó al nodo. Si esto no es así, resetear el nodo y volver a ejecutar el comando anterior.

4. Usar el navegador Firefox[15] para comunicarse con los nodos:

[15]El navegador Firefox instalado en la máquina virtual Instant Contiki cuenta con una extensión llamada Cooper que permite interactuar con los recursos de CoAP.

a) Escribir en la URL: coap://[aaaa::212:4b00:89ab:2]:5683
b) Presionar enter
c) Pulsar el botón Discover
Si los nodos se comunican correctamente al cabo de unos segundos deberían aparecer todos los recursos en la zona izquierda de la ventana de Firefox como se muestra en la Figura B.11.

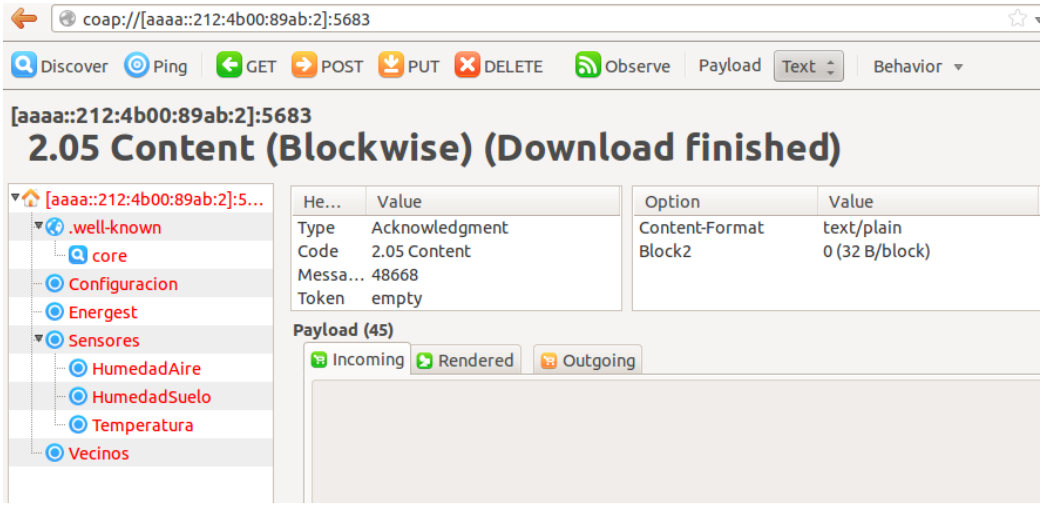


Figura B.11: Ventana de Copper.

B.4.2. Configuración de parámetros

Para configurar los diferentes parámetros del nodo se debe seleccionar 'Configuración' del árbol de recursos y agregar en la URL en Firefox '?conf=[parámetro a configurar]'. La URL debe quedar de siguiente manera:

coap://[aaaa::212:4b00:89ab:2]:5683/Configuracion?conf=[parámetro a configurar]

Donde [parámetro a configurar] debe sustituirse por alguna de las siguientes opciones:

1. hora: hora en formato unix.
2. tempM: temperatura máxima.
3. tempm: temperatura mínima.
4. humAM: humedad del aire máxima.
5. humAm: humedad del aire mínima.
6. humSM: humedad del suelo máxima.

7. humSm: humedad del suelo mínima.
8. periodoM: periodo de muestreo de los sensores.
9. periodoT: periodo de observación del recurso de temperatura.
10. periodoHA: periodo de observación del recurso de humedad del aire.
11. periodoHS: periodo de observación del recurso de humedad del suelo.

Por ejemplo si se quiere modificar el periodo de muestreo la URL queda:

coap://[aaaa::212:4b00:89ab:2]:5683/actuators/Configuracion?conf=periodoM

Luego se presiona ENTER y en la pestaña outgoing se escribe: 'value=15', donde 15 representa el periodo de muestreo de los sensores en minutos.

Configuración de temperatura

1. Seleccionar Configuración en el árbol.
2. Agregar '?conf=tempM' en el campo URL para modificar la temperatura máxima, o '?conf=tempm' para el mínimo
3. Presionar ENTER.
4. En la pestaña Outgoing escribir, por ejemplo, 'value=15'
   Siendo 15 la temperatura en ^oC a configurar.
5. Hacer clic en POST o PUT.
6. Si quedó correctamente configurado en la pestaña Incoming se puede ver la variable con el valor asignado.

Configuración de humedad del aire

1. Seleccionar Configuración en el árbol.
2. Agregar '?conf=humAM' en el campo URL para modificar la humedad máxima, o '?conf=humAm' para el mínimo
3. Presionar ENTER.
4. En la pestaña Outgoing escribir, por ejemplo, 'value=45'.
   Siendo 45 la humedad en % a configurar.
5. Hacer clic en POST o PUT.
6. Si quedó correctamente configurado en la pestaña Incoming se puede ver la variable con el valor asignado.

Configuración de humedad del suelo

1. Seleccionar Configuración en el árbol.
2. Agregar '?conf=humSM' en el campo URL para modificar la humedad máxima, o '?conf=humSm' para el mínimo
3. Presionar ENTER.
4. En la pestaña Outgoing escribir, por ejemplo, 'value=45'.
   Siendo 45 la humedad en % a configurar.
5. Hacer clic en POST o PUT.
6. Si quedó correctamente configurado en la pestaña Incoming se puede ver la variable con el valor asignado.

Configuración de Periodo de observación de Temperatura

1. Seleccionar Configuración en el árbol.
2. Agregar '?conf=periodoT' en el campo URL
3. Presionar ENTER.
4. En la pestaña Outgoing escribir, por ejemplo, 'value=20'.
   Cada 20 minutos el nodo va a enviar el valor de temperatura medido.
5. Hacer clic en POST o PUT.
6. Si quedó correctamente configurado en la pestaña Incoming se puede ver la variable con el valor asignado.

Configuración de Periodo de observación de Humedad de aire

1. Seleccionar config Configuración en el árbol.
2. Agregar config?conf=periodoHA en el campo URL
3. Presionar ENTER.
4. En la pestaña Outgoing escribir, por ejemplo, 'value=15'.
   Cada 15 minutos el nodo va a enviar el valor de humedad de aire medido.
5. Hacer clic en POST o PUT.
6. Si quedó correctamente configurado en la pestaña Incoming se puede ver la variable con el valor asignado.

Configuración de Periodo de observación de Humedad del suelo

1. Seleccionar Configuración en el árbol.

2. Agregar '?conf=periodoHS' en el campo URL

3. Presionar ENTER.

4. En la pestaña Outgoing escribir, por ejemplo, 'value=15'.
   Cada 15 minutos el nodo va a enviar el valor de humedad de aire medido.

5. Hacer clic en POST o PUT.

6. Si quedó correctamente configurado en la pestaña Incoming se puede ver la variable con el valor asignado.

Configuración de la Hora

1. Seleccionar Configuración en el árbol.

2. Agregar '?conf=hora' en el campo URL

3. Presionar ENTER.

4. En la pestaña Outgoing escribir, por ejemplo, 'value=1432162679'.
   Ese valor es la hora Unix del momento que se configura la hora.

5. Hacer clic en POST o PUT.

6. Si quedó correctamente configurado en la pestaña Incoming se puede ver la variable con el valor asignado.

Obtención de los tiempos configurados

1. Seleccionar Configuración en el árbol.

2. Presionar ENTER.

3. Hacer clic en GET.

4. En la pestaña Incoming deberían visualizarse los parámetros de tiempo configurados como se observa en la Figura B.12.

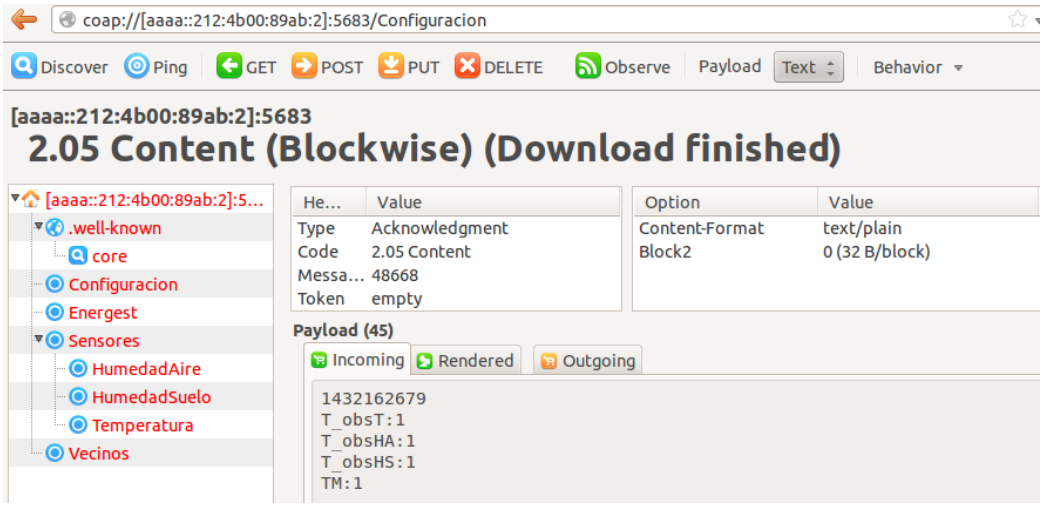


Figura B.12: Respuesta con los tiempos configurados.

B.4.3. Solicitud de Medidas

Temperatura

1. Seleccionar Temperatura en el árbol.
2. Hacer clic en Observe.
3. En la pestaña Incoming se visualizará la hora en que se realizó la medida y la temperatura. Este paquete llega cada periodoT minutos.
4. Si se desea solo una medida y que la misma no sea periódica se puede hacer clic en GET en lugar de un Observe.

Humedad del Aire

1. Seleccionar HumedadAire en el árbol.
2. Hacer clic en Observe.
3. En la pestaña Incoming se visualizará la hora cuando se realizó la medida y la humedad. Este paquete llega cada periodoHA minutos.
4. Si se desea solo una medida y que la misma no sea periódica se puede hacer clic en GET en lugar de un Observe.

Humedad del Suelo

1. Seleccionar HumedadSuelo en el árbol.
2. Hacer clic en Observe.

3. En la pestaña Incoming se visualizará la hora cuando se realizó la medida y la humedad. Este paquete llega cada periodoHA minutos.

4. Si se desea solo una medida y que la misma no sea periódica se puede hacer clic en GET en lugar de un Observe.

B.4.4. Solicitud de Información General

Tiempos del MCU

1. Seleccionar Energest en el árbol.

2. Hacer clic en GET.

3. En la pestaña Incoming se visualizarán los tiempos del MCU en el siguiente orden: LPM, CPU, LISTEN, TRANSMIT, MAX.

Tablas de ruta del nodo sensor

1. Seleccionar Vecinos en el árbol.

2. Hacer clic en GET.

3. En la pestaña Incoming se visualizarán los vecinos y la tabla de rutas del nodo.

Apéndice C

Protocolo CoAP

C.1. Introducción

Constrained Application Protocol (CoAP) es un protocolo de capa de aplicación usado por dispositivos electrónicos simples, como sensores de bajo consumo. CoAP adopta de HTTP la abstracción de recursos, URIs (Uniform Resource identifier), RESTful y opción de encabezados extensibles. Con este protocolo se puede duplicar la vida útil de las baterías, y al utilizar UDP en lugar de TCP se reduce a hasta 10 veces la transmisión de datos. La información brindada en este capítulo es extraída del draft de CoAP [61]

C.2. Problemas de HTTP

HTTP en sistemas de recursos limitados tiene el problema, además del consumo del protocolo en sí, de falta de apoyo a los mensajes de difusión y mecanismos de suscripción.

Para superar estas limitantes se crea CoAP que usa funcionalidades de HTTP, las cuales se rediseñaron para sistemas de recursos limitados y añade otras capacidades para atender las necesidades de la comunicación Machine to Machine M2M y el internet de las cosas IoT. La desventaja de HTTP frente a CoAP es la cantidad de mensajes que se intercambian para realizar una conexión, lo que provoca un mayor consumo de energía del sistema. En la Figura C.1 se puede apreciar la diferencia en la cantidad de mensajes necesarios para enviar un dato entre una comunicación que utiliza HTTP y una que utiliza el protocolo CoAP.

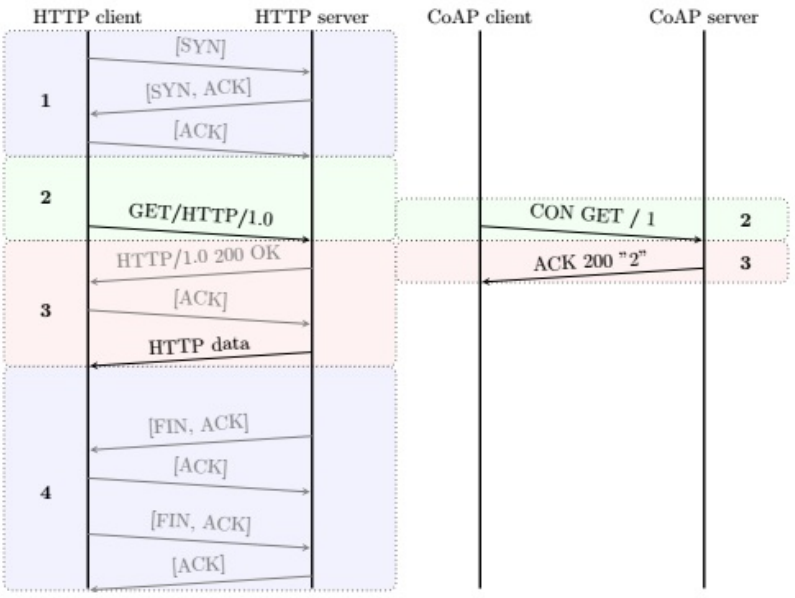


Figura C.1: Comunicación HTTP y CoAP.

C.3. REST

Los servidores de redes de bajo consumo no poseen memoria, por ende no retienen ninguna de las solicitudes previas. Los recursos son accedidos y manipulados utilizando protocolos de capa de aplicación basados en diálogos cliente-servidor y solicitud-respuesta.

El modelo REST[1] no especifica el protocolo a usar pero solicita que el mismo cuente con los comandos:

- GET: Comando con el cual se recupera una representación de la información que corresponde al recurso identificado por la URI.
- POST: Envía una acción al recurso. Esta acción puede ser crear, modificar o eliminar un recurso.
- PUT: Solicita que el recurso identificado por la URI se actualice o se cree.
- DELETE: Elimina el recurso identificado por la URI.

Las aplicaciones que utilizan los principios REST se les conoce como RESTful, y en el modelo REST aplicado a IoT o M2M los sensores y nodos son identificados por URIs.

[1]REST (Representational State Transfer) es un estilo de arquitectura de software

C.4. Low Power and Lossy Networks

En las redes LLN se utiliza UDP en lugar de TCP dado que este tipo de redes no considera las retransmisiones, el control de flujo ni otros encabezados TCP (el cual es un protocolo orientado a conexión) reduciendo de esta manera el consumo energético de la comunicación al disminuir notablemente el tráfico de paquetes necesarios para la comunicación. Si la aplicación lo requiere, se debe asegurar que los mensajes lleguen a destino, para esto CoAP implementa un sistema de reconocimiento de mensajes.

C.4.1. CoAP sobre UDP

UDP es un protocolo que agrega únicamente los puertos de origen y destino del paquete al mensaje. TCP además agrega control de flujo y manejo de errores, lo que sobrecarga la comunicación.

UDP no detecta errores y no retransmite mensajes. Debido a esto, utilizar UDP no asegura que los mensajes lleguen a destino. Para resolver este problema CoAP implementa un mecanismo opcional de retransmisión, a nivel de capa de aplicación.

En la Figura C.2 se observan las diferentes capas del stack. Las capas Request/Response y Transaction forman CoAP. La primera es una subcapa en la cual se da la comunicación RESTful, y la segunda es la transaction layer. En esta última, se da el envío de mensajes individuales entre nodos.

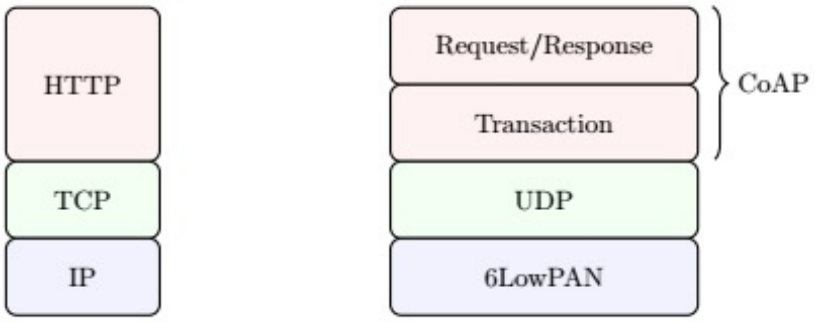


Figura C.2: Stack de protocolos.

En CoAP los mensajes pueden ser de cuatro tipos:

1. Confirmable: se requiere reconocimiento.
2. No confirmable: no se requiere reconocimiento.

3. Acknowlegement: reconocimiento (confirmación) de mensaje recibido.

4. Reset: se envía cuando se descarta un mensaje de confirmación.

De esta manera, a través del sistema de reconocimiento de mensajes, CoAP resuelve a nivel de aplicación lo que no está implementado en UDP, para realizar una comunicación robusta utilizando un sistema de reconocimiento de mensajes.

C.5. Formato del mensaje

En la Figura C.3 se observan los diferentes campos del mensaje CoAP.

1. Ver: La versión es la uno (01), otros valores de este campo son reservados para versiones futuras. Está representada por dos bits.

2. T: Tipo de mensaje (2bits).

3. TKL: Token length, expresa el largo del campo Token en bytes (4bits).

4. Code: Es un campo de 8 bits, los cuales se dividen en clase y detalles.

   a) Clase: son los 3 bits más significativos: En una solicitud son 000 y en una respuesta pueden ser 2 (respuesta exitosa), 4 (error de respuesta del cliente), 5 (error de solicitud del servidor).

   b) Detalle: En una solicitud indica si es un GET (00001), POST (00010), PUT (00011), DELETE (00100). En una respuesta se indica si fue exitosa la solicitud, por ejemplo: 05 = content, 03= valid, 01=creado, 04=no se modificó, etc.

   El campo Code se representa como code.detalle (e.g.: 2.05).

5. Mensaje ID: Se usa para detección de mensajes duplicados, y para corresponder mensajes del tipo acknowlegement/reste con mensajes Confirmable/No-Confirmable. Este campo es de 16bits.

6. Token: Es un campo del largo indicado en el TKL. Es usado para corresponder respuestas con solicitudes.

7. Opciones: Se pueden usar en solicitudes o en respuestas para indicar por ejemplo el formato del contenido, la URI del recurso requerido, etc.

8. FF: Se utiliza para indicar la finalización de las opciones y el comienzo del payload. Si no hay opciones, el FF no va, ya que se sabe que luego de los TKL bytes de token viene el payload.

La existencia del Token y el Mensaje ID se debe a que un recurso puede ser observado. Esto implica que la solicitud no va a recibir solo una respuesta, sino que recibe una cada vez que el recurso se modifica. Todas estas respuestas tienen el mismo Token que la solicitud inicial, pero su mensaje ID es diferente.

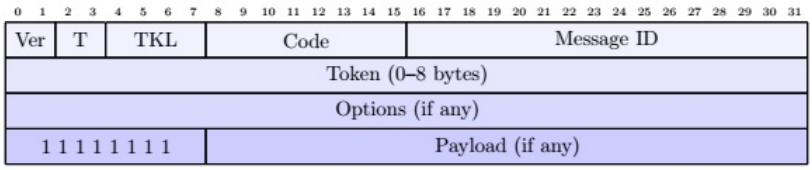


Figura C.3: Fromato mensajes CoAP.

C.5.1. Opciones

Las opciones tienen el formato mostrado en la Figura C.4. Las posibles opciones se encuentran numeradas en la tabla de la Figura C.5.

Siempre se parte de la posición cero de la tabla y con deltas se va accediendo a la opción que se desee. En la tabla se encuentran lugares libres en los cuales se pueden crear opciones nuevas como se verá más adelante.

Luego del mensaje ID y de los TKL bytes de Token, los siguientes 4 bits indican el delta (desfasaje) entre el cero y la opción deseada de la tabla. Los siguientes 4 bits indican cuantos bytes va a tener el valor de la opción y los bytes siguientes son los del valor. Si no aparece FF, lo que sigue son otros 4 bits de delta que indica que debe trasladarse en la tabla desde la opción seleccionada antes hasta la nueva opción. Y allí comienza nuevamente el razonamiento hasta FF.

C.5.2. Tipo de opción

1. Uri-Host, Uri-Post, Uri-Path, Uri-Query: Son usados para especificar el recurso destino de una solicitud a un servidor CoAP.

    a) Uri-Host: especifica el host de internet de los recursos que se solicitan. El valor por defecto es la IP destino del mensaje de solicitud.

    b) Uri-Post: especifica el número de puerto de la capa de transporte del recurso. El valor por defecto es puerto de destino UDP.

    c) Uri-Path: especifica un segmento de la ruta absoluta hacia el recurso. Contiene cualquier secuencia de caracteres, menos '.' o '..'.

    d) Uri-Query: especifica un argumento parametrizando el recurso. Contiene cualquier secuencia de caracteres, menos '.' o '..'.

2. Proxy-Uri y Proxy-Scheme

    a) Proxy-Uri: Es usado para hacer una solicitud a forward-proxy. Forward-proxy es consultado para el reenvío de solicitud o servir desde un cache valido, y retornar la respuesta. El forward-proxy debe reenviar los mensajes, entonces para evitar loops, el proxy debe reconocer todos

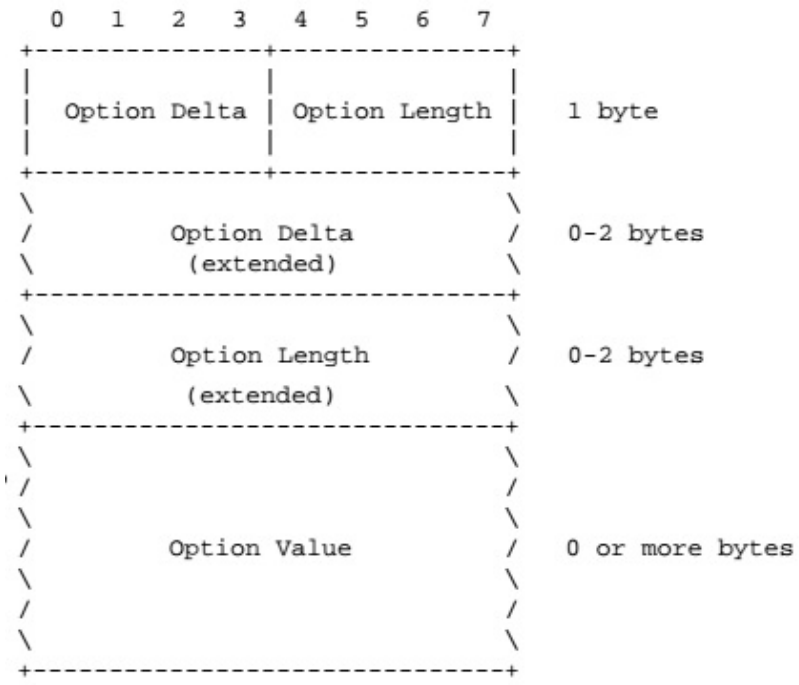


Figura C.4: Fromato del campo opciones.

Number	Name
0	(Reserved)
1	If-Match
3	Uri-Host
4	ETag
5	If-None-Match
7	Uri-Port
8	Location-Path
11	Uri-Path
12	Content-Format
14	Max-Age
15	Uri-Query
17	Accept
20	Location-Query
35	Proxy-Uri
39	Proxy-Scheme
60	Size1
128	(Reserved)
132	(Reserved)
136	(Reserved)
140	(Reserved)

Figura C.5: Tabla de opciones.

los nombres de los servidores incluyendo alias e IP. Si no es posible el forward-proxy envía un mensaje 5.05 (Proxying not suporting). Proxy-Uri tiene prioridad sobre todos los Uri-*.

b) Proxy-Scheme: Cuando aparece este caso, es porque la CoAP Uri es creada de la Uri-*.

3. Content-Format: Indica el formato del mensaje contenido en el payload del paquete.

4. Accept: Indica que Content-format es accesible para el cliente. Se indica en que formato quiere que el mensaje sea enviado, si no se envía esta opción el servidor envía en un formato cualquiera. Si el formato solicitado no es posible el servidor envía un mensaje 4.06 "Not acceptable", a menos que otro mensaje de error tenga mayor precedencia.

5. Max-Age: Es el tiempo máximo en el que una respuesta puede ser aceptada antes de ser considerada obsoleta. El valor se expresa en segundos. Si no se indica nada, el valor por defecto es de 60 segundos.

6. ETag: La entity-tag es usada como un identificador de recurso local para diferenciar entre representaciones del mismo recurso que varían con el tiempo. Es generado por el servidor entregando el recurso con su versión, un checksum, entre otras.

7. Location-path y Location-Query: Los dos juntos indican la URI relativa.

8. Conditional Request: Habilita al cliente a indicarle al servidor que envíe la respuesta solo si se cumplen ciertas condiciones especificadas por la opción. Si no se da la condición, el servidor responde con 4.12 (Precondition Failed).

   a) If-Match: Es usada para hacer una solicitud condicional. Es útil para recursos de respuesta actualizada, como por ejemplo PUT, como protección contra sobreescrituras cuando múltiples clientes están actuando en paralelo en el mismo recurso. El valor del If-Match es un ETag o un string vacío.

   b) If-None-Match: Es usada para hacer una solicitud condicional cuando no existe el recurso target. Idem If-Match.

9. Sizel: Provee el tamaño del recurso representado en la solicitud.

C.6. Observe

El observe es una solicitud GET con opción Observe en el cual se sigue un recurso indicado por la URI de la solicitud. Cuando un servidor recibe una solicitud de observe por parte de un cliente, éste agrega al cliente a una lista de subscriptores a observación de dicho recurso. Si el cliente fue agregado a la lista de observadores correctamente, responde un mensaje 2.xx (respuesta exitosa) conteniendo también

la opción Observe. Los clientes existentes en la lista serán notificados cuando hayan cambios en el recurso observado. Si el servidor no puede agregar al cliente a la lista de observadores del recurso, procesará la solicitud como un GET sin opción de observe, y dicho cliente no recibirá notificaciones de los cambios en el recurso.

C.6.1. Notificaciones

Las notificaciones enviadas al cliente subscrito a la observación de un recurso contarán con el token especificado en la solicitud GET enviada por el cliente, una opción Observe con diferentes números de secuencia para cada mensaje, y un Payload en el mismo formato que la respuesta inicial.

Las notificaciones pueden ser del tipo confirmable, o del tipo no confirmable. En caso de ser confirmables, el cliente debe responder con un mensaje de reconocimiento al recibir una notificación por parte del servidor. El servidor al recibir la confirmación entiende que el cliente sigue interesado en observar el recurso.

C.6.2. Cancelación de subscripción

Si un cliente recibe una notificación confirmable pero el token es desconocido no debe enviar una confirmación, en su lugar debe rechazar el mensaje respondiendo con un Reset. Si el servidor recibe un mensaje de Reset o una nueva solicitud GET a un recurso ya observado, pero sin opción de Observe, se eliminará de la lista de observadores del recurso a dicho cliente.

En caso de que el servidor no reciba un mensaje de confirmación como respuesta a un mensaje confirmable, éste asumirá que el cliente ya no está interesado en observar el recurso y puede eliminarlo de su lista de observadores.

En la Figura C.6 se puede ver el desglose de un paquete CoAP correspondiente a una solicitud de observación. Esta imagen surge de una simulación capturada en Wireshark, donde se realizó una solicitud de observación al recuso test/push del ejemplo er-example-server, el cual se puede encontrar en la carpeta examples/er-rest-example de Contiki.

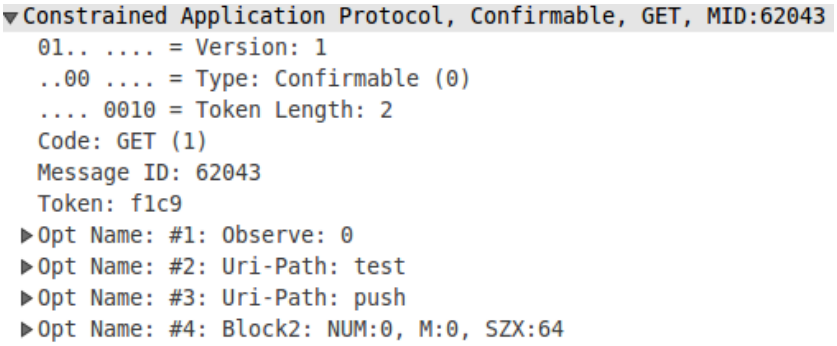
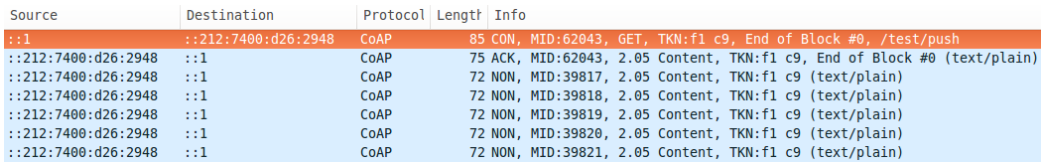


Figura C.6: Mensaje de solicitud de observe en Wireshark.

En la Figura C.7 se puede ver la solicitud de observación de la Figura C.6, seguido de los mensajes de notificación enviados por el servidor. En este caso, como se puede ver en la imagen los mensajes son del tipo no confirmable. También se puede observar que los mensajes tienen el mismo token, debido a que corresponden a la misma solicitud.



Source	Destination	Protocol	Length	Info
::1	::212:7400:d26:2948	CoAP	85	CON, MID:62043, GET, TKN:f1 c9, End of Block #0, /test/push
::212:7400:d26:2948	::1	CoAP	75	ACK, MID:62043, 2.05 Content, TKN:f1 c9, End of Block #0 (text/plain)
::212:7400:d26:2948	::1	CoAP	72	NON, MID:39817, 2.05 Content, TKN:f1 c9 (text/plain)
::212:7400:d26:2948	::1	CoAP	72	NON, MID:39818, 2.05 Content, TKN:f1 c9 (text/plain)
::212:7400:d26:2948	::1	CoAP	72	NON, MID:39819, 2.05 Content, TKN:f1 c9 (text/plain)
::212:7400:d26:2948	::1	CoAP	72	NON, MID:39820, 2.05 Content, TKN:f1 c9 (text/plain)
::212:7400:d26:2948	::1	CoAP	72	NON, MID:39821, 2.05 Content, TKN:f1 c9 (text/plain)

Figura C.7: Mensajes de observe en Wireshark.

C.7. Copper

Copper (Cu) es un navegador para Firefox que permite controlar los mensajes REST de protocolo COAP y brinda una interfaz de usuario para interactuar con los dispositivos de una red. Cu se convirtió en una herramienta popular para el grupo de trabajo IETF CoRE y fueron los primeros en adoptar COAP en la industria y el mundo académico [6].

Para utilizar Copper primero se debe descargar del siguiente link:
https://addons.mozilla.org/En-us/firefox/addon/copper-270430/

Luego de haberlo instalado, al abrir FireFox y escribir en la URL coap://[aaaa::212:4b00:89ab:2]:5683, se puede comenzar a interactuar con un dispositivo de la red, en donde aaaa::212:4b00:89ab:2 es la dirección IP del dispositivo. Como se puede observar en la figura C.8, Copper cuenta con botones para poder enviar solicitudes GET, POST, PUT, DELETE y OBSERVE. También permite

enviar mensajes PING para verificar la comunicación con el dispositivo.

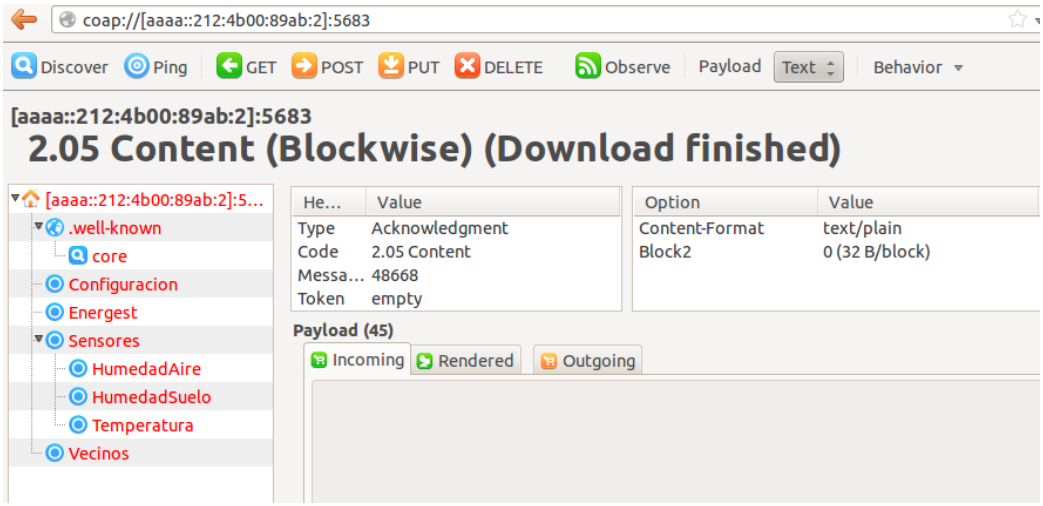


Figura C.8: Ventana de Copper.

Apéndice D

IPv6

D.1. Introducción

Internet Control Message Protocol Version 6 (ICMPv6) son mensajes de información y de error de la red IPv6. Estos mensajes pueden aparecer como una extensión de IPv6, y si se incluye en el paquete, se indica en el campo Next Header del encabezado IPv6. Protocolos como Neighbour discovery, autoconfiguración y RPL dependen de ICMPv6. La información presentada en este anexo fue extraída del documento "An Overview of Enabling Technologies for THE INTERNET OF THINGS" [7].

D.2. Encabezado IPv6

En la Figura D.1 se visualiza el encabezado de IPv6, el cual consta de los campos:

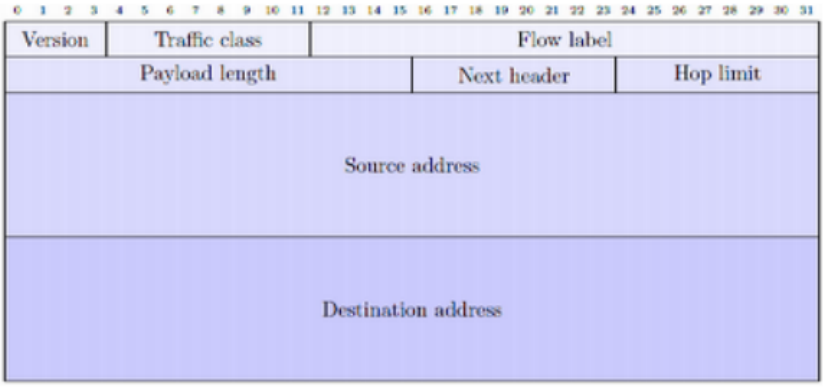


Figura D.1: Encabezado IPV6

1. Version: versión de IP (6 en este caso) – 4 bits.
2. Traffic Class: define diferentes prioridades para los paquetes – 8 bits.
3. Payload Length: cantidad de bytes de la carga – 16 bits.
4. Next Header: indica qué tipo de encabezado le sigue al de IP – 8 bits.
5. Hop Limit: cantidad de saltos límite donde el paquete sigue siendo válido – 8 bits.
6. Source Address: IP origen.
7. Destination Address: IP destino.

IPv6 permite extensiones de encabezado para evitar enviar información innecesaria. En el campo Next Header se indica qué tipo de encabezado le sigue al de IP, esto puede ser, un encabezado de extensión, un encabezado de un protocolo de capa superior, o el encabezado de un paquete IP encapsulado.

ICMPv6 se indica con el valor 58 en el encabezado IPv6, y siempre empieza con tres campos:

1. Type – 8 bits
   a) Mensajes de error (0 – 127)
   b) Mensajes de información (128 – 255)
2. Code: agrega opciones adicionales al mensaje – 8 bits
3. Checksum: comprobación de errores

La tabla que se ve en la Figura D.2 muestra el resto del contenido para algunos ejemplos de los mensajes más usados.

Type	Name
1	Destination Unreachable
128	Echo Request (ping)
129	Echo Reply (ping)
130	Multicast Listener Query
131	Multicast Listener Report
132	Multicast Listener Done
133	Router Solicitation
134	Router Advertisement
135	Neighbour Solicitation
136	Neighbour Advertisement
137	Redirect Message
155	RPL Control Message

Figura D.2: Opciones del contenido de la extencion

D.3. Neighbor Discovery

Este protocolo describe mensajes para la resolución del direccionamiento, detección de destinos inalcanzables, autoconfiguración y detección de direcciones duplicadas en la red. Se diferencian los siguientes tipos de mensajes:

1. Router Solicitation (RS)

   Los nodos envían mensajes RS consultando por un Router Advertisment (RA). El destino de estos mensajes es normalmente una dirección multicast de todos los routers locales. El origen puede ser una dirección sin especificar o la dirección real de quien lo envía, si es que tiene. Si quien envió el mensaje tiene dirección IPv6, también enviará su dirección "link-layer".

2. Router Advertisment (RA)

   Los routers envían periódicamente mensajes RA a los nodos locales, o como respuesta a mensajes RS. Estos mensajes contienen flags que indican si se está usando DHCPv6, y puede tener información adicional, como información sobre DNS.

3. Neighbor Solicitation (NS)

   Estos mensajes se usan para la resolución de direccionamiento, detección de destinos inalcanzables (NUD) y detección de direcciones duplicadas en la red (DAD). Para resolver el direccionamiento o para DAD, el mensaje se envía como multicast, y si se quiere NUD, se envía.

4. Neighbor Advertisment (NA)

   Los mensajes NA se envían en respuesta a mensajes NS, o para informar a los demás nodos de su presencia. En el mensaje se incluyen flags que indican si quien envía el mensaje es router, o si tiene una nueva dirección "link-layer". Así como en los mensajes NS, NA tiene un campo de dirección de destino. Para mensajes que fueron solicitados, este campo contiene la dirección de quien hizo la solicitud, en el caso que no haya sido solicitado, tiene como destino la dirección de aquellos nodos que hayan cambiado su dirección de link-layer.

5. Redirect Message

   Si un router detecta que hay un camino mejor para los mensajes que está enviando un emisor, le envía esta información a través de un redirect message, que incluye el destino y el mejor primer salto para alcanzarlo.

D.4. Multicast listener discovery

Este protocolo permite a los routers consultar a los nodos a qué grupo multicast pertenecen.

Los nodos pueden registrar o borrar su pertenencia a un grupo multicast al router con mensajes report/done.

Apéndice E

Cooja

En algunas ocasiones cuando se trabaja con RSI es imprescindible contar con una herramienta de simulación para desarrollar y emular la red. Aunque sea preferible ejecutar los programas en nodos reales para probarlos esto no siempre es posible, por ejemplo, cuando se quiere evaluar una red de cientos de nodos. Montar una RSI de esta característica es inviable en un laboratorio.

Cooja es el simulador de Contiki que permite simular pequeñas y grandes RSI. Es totalmente compatible con Contiki y permite que el mismo código compilado sea ejecutado y simulado sin necesidad de ser modificado. Cooja permite incluso simular redes formadas por algunos nodos reales y otros simulados.

Si bien Cooja no soporta al MCU CC2538 fue utilizada durante el presente proyecto y resulta una herramienta muy útil para desarrollar aplicaciones en Contiki. En este anexo de describen los pasos básicos para utilizar esta herrmaienta de simulación. Cooja está incluida en el árbol de Contiki desde la versión 2.0 y se encuentra en tools/cooja y platform/cooja.

Para simular una aplicación en Cooja siga los siguientes pasos:

1. Abra una nueva terminal en Ubuntu, diríjase a la ruta donde se encuentra el código de Cooja (por defecto: tools/cooja) y ejecute el siguiente comando:

   ant run

2. Espere a que aparezca la ventana del simulador. Cuando Cooja se inicia por primera vez debe compilarse, lo que puede demorar.

3. Cree una nueva simulación presionando en **File** → **New Simulation**. Debería aparecer una ventana como la de la Figura E.1. Ingrese un nombre para su simulación y luego presione el botón *Create*.

4. Para agregar nodos a la simulación pulse en **File** → **Motes** → **Add motes** → **Create new** → **Mote type** → **Sky mote** (por ejemplo) como se muestra en la Figura E.2.

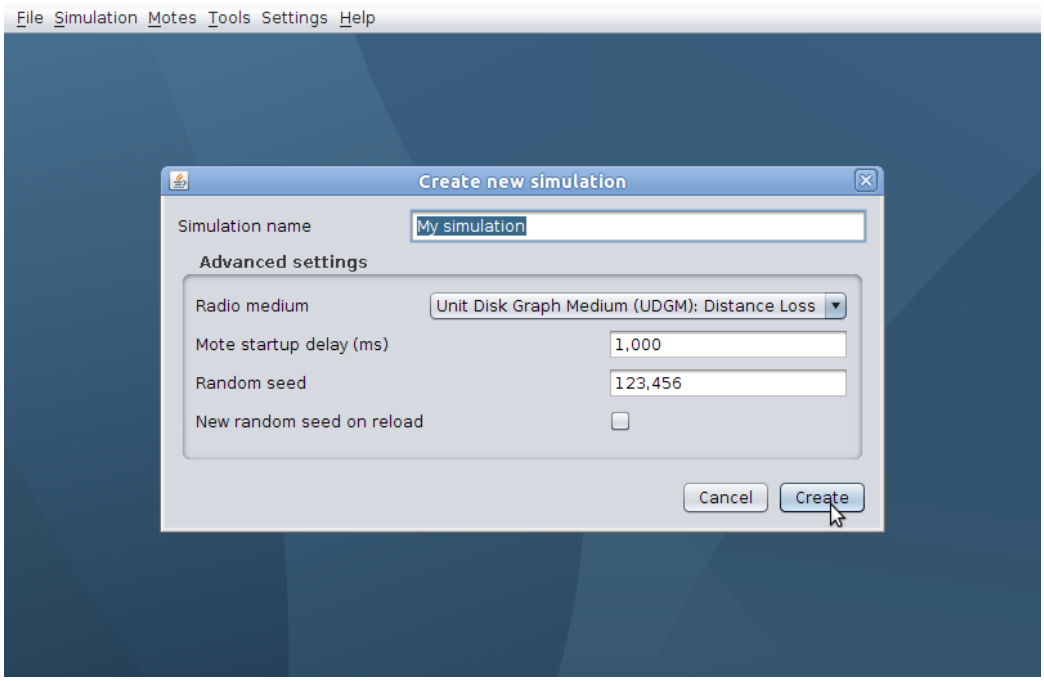


Figura E.1: Crear una nueva simulación en Cooja.

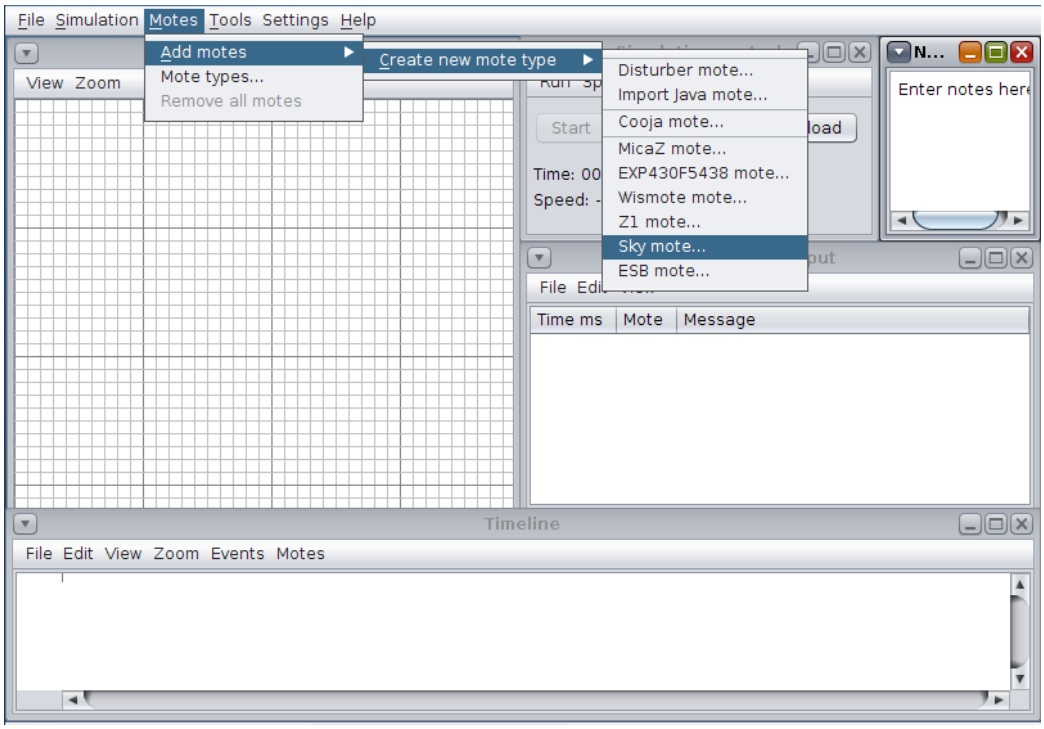


Figura E.2: Agregar motes en Cooja.

5. Pulse en *Browse* para seleccionar el programa que desea cargar en los nodos (archivo.c) y luego en *Compile* para que el programa sea compilado. Nótese que de esta forma el código se compilará cada vez que se recargue la simulación, permitiendo hacer cambios sobre el código y probarlo sin necesidad de volver a crear los nodos.

6. Si el código compila correctamente el programa permite crear los nodos presionando en *Create* y luego en *Add motes*. Los nodos se visualizados en la grilla, los cuales pueden ser ubicados como se prefieran.

Una vez creada la simulación se comienza a ejecutar el código cargado en los nodos de la red presionando el botón *Start* en la ventana *Simulation control* como se muestra en la Figura E.3.

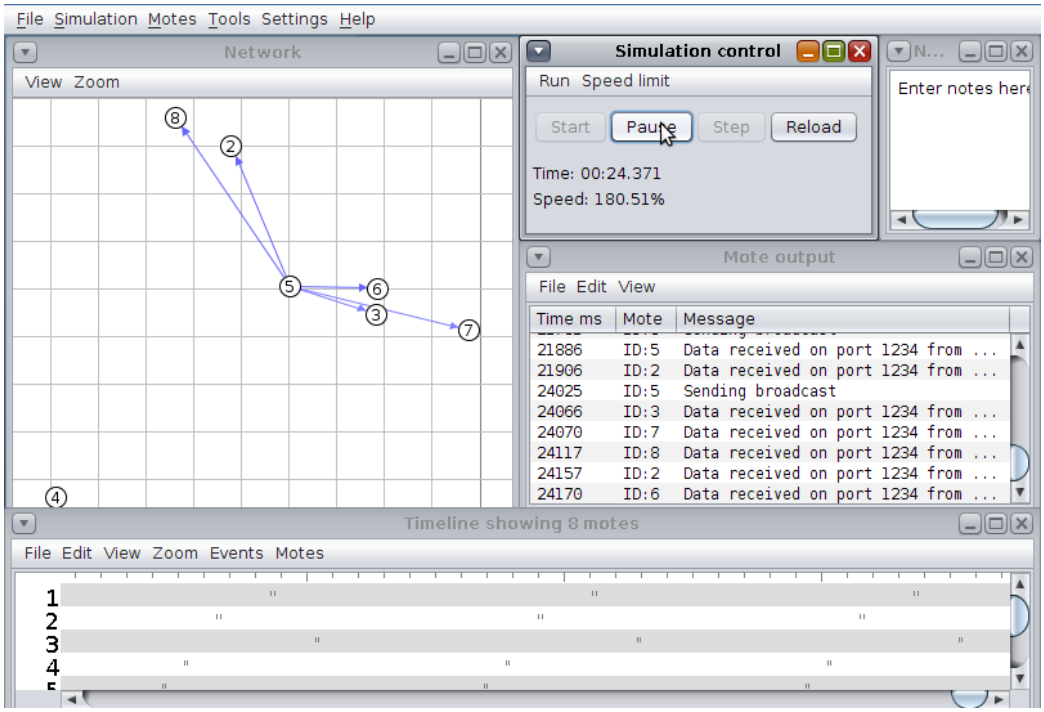


Figura E.3: Simulación en Cooja.

Referencias

[1] Antenas multidireccionales. http://www.ehowenespanol.com/antenas-multidireccionales-mas-potentes-uhf-vhf-info_260872/.

[2] Constrained application protocol. http://en.wikipedia.org/wiki/Constrained_Application_Protocol.

[3] Contiki os. http://www.contiki-os.org/.

[4] Contiki rdc y coap. http://m.blog.csdn.net/blog/tietao/8457460.

[5] Cooja. http://anrg.usc.edu/contiki/index.php/Cooja_Simulator.

[6] The copper (cu) coap user-agent for chrome (b/l). http://www.vs.inf.ethz.ch/edu/abstract.html?file=theses/mkovatsc-cu4chrome.

[7] The copper (cu) coap user-agent for chrome (b/l). https://eva.fing.edu.uy/pluginfile.php/80445/mod_resource/content/2/IoT.pdf.

[8] Redes con pérdidas. https://docs.google.com/document/d/1QHYWQT5Cej_KNl_PNWA_h7_vao7IKdSj-2Lwzih9PMU/pub#h.5hrlvbq9658w.

[9] Stack de comunicación. https://docs.google.com/presentation/d/1c4FtNfCOOGT_ULAiB1LcBXiOV1L7qjkBHukQfqP50jo/pub?start=false&loop=false&delayms=3000&slide=id.g371f83428_086.

[10] Zona de fresnel. http://es.wikipedia.org/wiki/Zona_de_Fresnel.

[11] Embedded Adventures. Mod-1018. http://www.embeddedadventures.com/datasheets/MOD-1018_hw_v1_doc_v1.pdf.

[12] Atmel. Sniffer. http://www.atmel.com/tools/RZUSBSTICK.aspx?tab=overview.

[13] T. Berners-Lee. Uniform resource identifiers (uri): Generic syntax. https://tools.ietf.org/html/rfc2396.

[14] Sergio R. Caprile. Equisbí: Desarrollo de aplicaciones con comunicación remota basadas en zigbee y 802.15.4. https://books.google.com.uy/books?hl=es&lr=&id=xTXv5-AhOhMC&oi=fnd&pg=PA1&dq=canales+802.15.4&ots=RiEZcpHDqD&sig=EXAfYgpRzAcYP6kn9p_oawhk5-8&redir_esc=y#v=onepage&q=canales&f=false.

[15] Fredrik Osterlind y Thiemo Voigt Carlo Alberto Boano, Kay Romer. Demo abstract: Realistic simulation of radio interference in cooja. http://soda.swedish-ict.se/4109/1/boano11realistic-demo.pdf.

[16] Eduardo Coquet. Bus i2c. http://www.comunidadelectronicos.com/articulos/i2c.htm.

[17] Facultad de Ingeniería de la Universidad de la República. Redes de sensores inalámbricos. https://docs.google.com/document/d/1WJw36Yn6gT0ZcdEhq-ZAy3jw67cFhhIHUWSjL_-Nbec/pub.

[18] Decagon. 10hs. http://manuals.decagon.com/Manuals/13508_10HS_Web.pdf.

[19] Decagon. Ec-05. http://www.decagon.com/products/soils/volumetric-water-content-sensors/ec-5-soil-moisture-small-area-of-influence/.

[20] Dr. Peter Friess Dr. Ovidiu Vermesan. Internet of things:converging technologies for smart environments and integrated ecosystems. http://www.internet-of-things-research.eu/pdf/Converging_Technologies_for_Smart_Environments_and_Integrated_Ecosystems_IERC_Book_Open_Access_2013.pdf.

[21] Adam Dunkels. Adam dunkels. http://dunkels.com/adam/.

[22] OKW Enclosures. G 155. http://www.okw.com/en/category/fe162660-c2e5-11e2-8e2c-0050568225d7/products?vs=af775970-c2e5-11e2-8e2c-0050568225d7%24%24bb2cbfc1-d9bb-11e2-a369-0050568225d7.

[23] OKW Enclosures. G 190. http://www.okw.com/en/Shell-Type-Cases/A9413341.htm.

[24] ENERGIZER. Energizer l91. http://data.energizer.com/PDFs/l91.pdf.

[25] FPTA-INIA. Fpta-inia gervasio. http://iie.fing.edu.uy/investigacion/grupos/microele/Projects_es.htm.

[26] GitHub. Github. https://github.com/contiki-os/contiki/wiki.

[27] Gabriel Firme Ignacio de Mula, Germán Ferrari. Contikiwsn. http://iie.fing.edu.uy/publicaciones/2011/DFF11/.

[28] Texas Instrument. Cc2538 per-test. http://www.ti.com/tool/cc2538-sw.

[29] Texas Instrument. Flash programmer 2. http://www.ti.com/tool/Flash-Programmer.

[30] Texas Instrument. A powerful system-on-chip for 2.4-ghz ieee 802.15.4, 6lowpan and zigbee applications. http://pdf1.alldatasheet.com/datasheet-pdf/view/511197/TAOS/CC2538.html.

[31] Texsas Instrument. Cc2420. http://www.ti.com/product/cc2420.

[32] Texsas Instrument. Msp 430 f1611. http://www.ti.com/product/msp430f1611.

[33] Texas Instruments. cc2538. http://www.ti.com/lit/ds/symlink/cc2538.pdf.

[34] Texas Instruments. Cc2538dk. http://www.ti.com/lit/ug/swru321a/swru321a.pdf.

[35] Texas Instruments. cc2592. http://www.ti.com/lit/ds/symlink/cc2592.pdf.

[36] Texas Instruments. Emb-z2538pa. http://www.embit.eu/wp-content/datasheets/EMB-Z2538PA-datasheet_rev1.2.pdf.

[37] Texas Instruments. Guia de usuario cc2538 rom. http://www.ti.com/lit/ug/swru333a/swru333a.pdf.

[38] Texas Instruments. Tmp275. http://www.ti.com/lit/ds/symlink/tmp275.pdf.

[39] Texas Instruments. Tmp75c. http://www.ti.com/product/tmp75c.

[40] Texas Instruments. Tps62740. http://www.ti.com/product/tps62740.

[41] Texas Instruments. Using a dc-dc converter to reduce power (current) consumption in cc430 systems. http://www.ti.com/lit/an/slaa500/slaa500.pdf.

[42] Texas Instruments. Xds100v3 usb cjtag/jtag emulator. http://processors.wiki.ti.com/index.php/XDS100#What_is_the_XDS100.3F.

[43] Mote iv. Tmote sk. http://www.eecs.harvard.edu/~konrad/projects/shimmer/references/tmote-sky-datasheet.pdf.

[44] Ed. J. Hui. 6lowpan. https://tools.ietf.org/html/draft-ietf-6lowpan-hc-15.

[45] Cisco Systems JP Vasseur, Cisco Fellow. Rpl: The ip routing protocol designed for low power and lossy networks. http://www.ipso-alliance.org/wp-content/media/rpl.pdf.

[46] P. Levis. The trickle algorithm. https://tools.ietf.org/html/rfc6206.

[47] Torex Semiconductor Ltd. Xcl209. http://www.torex.co.jp/english/products/dcdc_converters/data/XCL208_209.pdf.

[48] Nicolás Wainstein Mauricio González, Javier Schandy. Plagavisión. http://iie.fing.edu.uy/publicaciones/2014/GSW14/.

[49] MetaGeek. Zigbee and wi-fi coexistence. https://support.metageek.com/hc/en-us/articles/203845040-ZigBee-and-Wi-Fi-Coexistence.

[50] G. Montenegro. Transmission of ipv6 packets over ieee 802.15.4 networks. https://tools.ietf.org/html/rfc4944#page-11.

[51] Optoelectronics. Smd led. http://optoelectronics.liteon.com/upload/download/DS22-2000-223/S_110_LTST-C191KRKT.pdf.

[52] J. Villaverde F. Silveira G. Fierro A. Otero C. Saravia N. Barlocco P. Vergara D. Garín P. Mazzara, L. Steinfeld. Despliegue y depuración de redes de sensores inalámbricos para aplicaciones al agro. http://iie.fing.edu.uy/investigacion/grupos/microele/papers/rpic11.pdf.

[53] Panasonic. Alkaline-zinc/manganese dioxide. http://na.industrial.panasonic.com/sites/default/pidsa/files/1.5vseries_datasheets_merged.pdf.

[54] Jesús Serna Sanchis. Redes de sensores inalámbricas. http://www.uv.es/~montanan/ampliacion/trabajos/Redes%20de%20Sensores.pdf.

[55] Fairchild Semiconductor. Fpf2004. https://www.fairchildsemi.com/datasheets/FP/FPF2004.pdf.

[56] SENSIRION. Sht11. http://www.sensirion.com/fileadmin/user_upload/customers/sensirion/Dokumente/Humidity/Sensirion_Humidity_SHT1x_Datasheet_V5.pdf.

[57] Sensirion. Sht25. http://www.sensirion.com/fileadmin/user_upload/customers/sensirion/Dokumente/Humidity/Sensirion_Humidity_SHT25_Datasheet_V3.pdf.

[58] Sensirion. Sht75. http://www.sensirion.com/fileadmin/user_upload/customers/sensirion/Dokumente/Humidity/Sensirion_Humidity_SHT7x_Datasheet_V5.pdf.

[59] Z. Shelby. Constrained application protocol (coap). https://tools.ietf.org/html/draft-ietf-core-coap-18.

[60] Adomas Svirskas. Tcp/ip reference model. http://www.mif.vu.lt/~adam/courses/npij/scsu-mcs426-fall-1999-3.pdf.

[61] K. Hartke C. Bormann Z. Shelby, Sensinode. Draft coap. https://tools.ietf.org/html/draft-ietf-core-coap-18.

Glosario

A

ACK Reconocimiento de llegada de paquete (del ingles: Acknowledgment).

ADC Conversor de analógico a digital (del inglés: Analog to Digital Converter).

B

Border Router Dispositivo que proporciona conectividad encaminando paquetes de datos de una red a otra.

Broadcast Mensaje dirigido a todos los nodos de la red.

BSL Referido a la carga en el arranque (Del inglés: Bootstrap Loader.

C

CCA Sensado del canal previo a enviar datos, si el canal está libre envía, (del inglés: Clear Channel Assessment).

CCR Magnitud en Hz, que indica la cantidad de chequeos del canal de comunicación por segundo (del ingles: Channel Check Rate).

CoAP Protocolo de aplicación comprimido (del ingles: Constrained Application Protocol).

CRC Chequeo de datos mediante comprobación de redundancia cíclica que permite detectar errores de comunicación.

D

DAO Paquetes que propagan información hacia arriba y completan las tablas de ruteo de los nodos padres (del ingles: Destination Advertisement Object).

DC/DC Dispositivo que convierte una tensión continua en otra (del inglés: Direct current / Direct current).

DIO Paquete que contienen información de la configuración del DODAG y sirven para crear, mantener y descubrir el DODAG (del ingles: DODAG Information Object).

DIS Paquetes que se envía para solicitar mensajes DIO a los nodos vecinos (del ingles: DODAG Information Solicitation Message).

DODAG Árbol de nodos RPL (del ingles: Destination Oriented Directed Acyclic Graph).

E

ETX Número de transmisiones esperada de un paquete, necesaria para recibir un paquete sin errores. Se usa como métrica utilizada para elegir el camino desde el nodo que envía el paquete al nodo raíz (del inglés: Expected Transmission Count).

F

Flash Memoria de lectura/escritura no volátil.

G

Gateway O puerta de enlace, es un dispositivo que permite interconectar redes con protocolos y arquitecturas diferentes a todos los niveles de comunicación. Su propósito es traducir la información del protocolo utilizado en una red al protocolo usado en la red de destino.

GPIO Puerto de uso general del (del inglés: General Port Input Ouput).

H

HTTP Protocolo de transferencia de hipertexto (del ingles: Hypertext Transfer Protocol).

I

I2C Protocolo de comunicación serie de dos hilos (del inglés: Inter Integrated Circuit).

IoT Internet de las cosas (del inglés: Internet of Things).

J

JTAG Protocolo serie utilizado para programación y depuración (del inglés: Joint Test Action Group).

K

Kernel El kernel (núcleo) es un programa de computadoras que maneja las E/S (entrada/salida) solicitadas por software, y las traduce dentro de la instrucción de proceso de datos.

L

LED Diodo emisor de luz (Del inglés: light-emitting diode.

LLN Red de bajo consumo y pérdidas (del inglés:Low-power and Lossy Network).

M

MCU Microcontrolador.

MRHOF OF que calcula el rank mínimo (del inglés: Minimum Rank with Hysteresis Objective Function).

MRM Es un modelo se simulación de Cooja, que permite una simulación realista de radio interferencia.

N

NAT Traducción de direcciones de red (del inglés: Network Address Translation).

O

OF Función objetivo en la cual se define la métrica del enlace y cómo ésta es utilizada para determinar el rank de los diferentes nodos, (del inglés: Objective Function).

overflow Desbordamiento aritmético que se genera cuando el código almacenado en un registro supera su valor máximo.

P

PA/LNA Amplificador de radio (del inglés: Power Amplifier / Low Noise Amplifier).

PAN Red utilizada para transmición de datos entre dispositivos, (del inglés: Personal Area Networks).

PCB Circuito impreso (del inglés: Printed Circuit Board).

PER Porcentaje de paquetes perdidos frente al total de paquetes enviados (del ingles: Packet Error Rate).

PER-test Este software consiste en un test genérico de la taza de paquetes con error. Esta aplicación permite variar la potencia de transmisión (22dbm a 7dbm).

PING Unidad diagnóstica que comprueba el estado de la comunicación del host local con uno o varios equipos remotos de una red por medio del envío de paquetes ICMP de solicitud y de respuesta.

R

RAM Memoria de acceso aleatorio (del inglés: Ramdom Acces Memory).

Rank Parámetro de RPL quee sirve para evaluar qué tan bueno son los enlaces y determinar cuál es el mejor camino al root.

RDC Ciclo de trabajo de la radio (del ingles: Radio Duty Cycle).

Root El nodo que se encuentra en la raíz del árbol de comunicación.

RSI Red de sensores inalámbricos.

S

SoC Chip que integra un microcontrolador con una radio (del inglés: Sistem on a Chip).

T

TCP Protocolo de control de transmisión (del ingles: Transmission Control Protocol).

U

UART Protocolo serie asíncrono (del inglés: Universal Asynchronous Receiver-Transmitter).

UDP Protocolo basado en datagramas(del ingles: User Datagram Protocol).

URI Identificador de recursos uniforme, el cual es una cadena de caracteres que identifica los recursos de una red de forma unívoca (del ingles: Uniform Resource Identifier).

Índice de tablas

Índice de figuras

Esta es la última página.
Compilado el domingo 16 agosto, 2015.
http://iie.fing.edu.uy/