PONTIFICIA UNIVERSIDAD CATOLICA DE CHILE
ESCUELA DE INGENIERIA

SISTEMA OPERATIVO PARA REDES INALÁMBRICAS DE SENSORES

SERGIO CAMPAMÁ DERPSCH

Tesis presentada a la Dirección de Investigación y Postgrado
como parte de los requisitos para optar al grado de
Magister en Ciencias de la Ingeniería

Profesor Supervisor:
CHRISTIAN OBERLI

Santiago de Chile, Julio 2012

PONTIFICIA UNIVERSIDAD CATOLICA DE CHILE
ESCUELA DE INGENIERIA

SISTEMA OPERATIVO PARA REDES INALÁMBRICAS DE SENSORES

SERGIO CAMPAMÁ DERPSCH

Miembros del Comité:
CHRISTIAN OBERLI
MARCELO GUARINI
DIEGO DUJOVNE
IGNACIO CASAS

Tesis presentada a la Dirección de Investigación y Postgrado
como parte de los requisitos para optar al grado de
Magister en Ciencias de la Ingeniería

Santiago de Chile, Julio 2012

A mi familia y amigos

AGRADECIMIENTOS

Me gustaría agradecer primero a Christian Oberli, por darme la oportunidad de conocer el mundo de la investigación aplicada, y también a Marcelo Guarini por todo el apoyo recibido durante el proyecto.

A mi familia, en especial a mi madre por enseñarme mediante el ejemplo a no rendirme. También a mis abuelos, por toda su preocupación y sus consejos.

Finalmente, pero no por eso menos importante, a todos mis amigos que me acompañaron durante este proceso, en especial a Cami, Lurys, Melisa, Nicolás, José Luis, Santiago, Toño, Joaquín, Fernando y Francisco.

Índice

Índice de cuadros

Índice de figuras

RESUMEN

El estado del arte de la tecnología permite alta eficiencia y bajo costo en electrónica de uso masivo. Dispositivos de muy bajo costo que integran una multitud de sensores y que se comunican entre sí mediante enlaces de radio-frecuencia es lo que en la actualidad se conoce como red inalámbrica de sensores. Estos dispositivos poseen variadas limitaciones tanto en capacidad de procesamiento como en memoria y a la vez hay aplicaciones que requieren que funcionen de manera autónoma durante años, por lo que es importante que su uso sea energéticamente eficiente.

Para comprobar el correcto funcionamiento de una aplicación existen dos maneras. La primera manera consiste en comprobar el funcionamiento en dispositivos físicos, proceso que es costoso y lento. Una segunda manera es utilizar un simulador que permita analizar el comportamiento de una red antes de ser desplegada en terreno.

Existen varias soluciones que permiten desarrollar aplicaciones para redes inalámbricas de sensores bajo las restricciones anteriores, pero hasta el momento no existen soluciones que permitan realizar tanto el desarrollo de aplicaciones como simulaciones bajo un mismo lenguaje de programación. Ello obliga desarrollar los algoritmos dos veces, una vez para los dispositivos físicos y otra para la simulación. Además, las soluciones disponibles utilizan técnicas de programación complejas, lo que aumenta la curva de aprendizaje para nuevos desarrolladores.

En este trabajo se describe el diseño, desarrollo y validación de LatinOS, un sistema operativo para redes inalámbricas de sensores que a la vez está acoplado a un simulador bajo un mismo lenguaje de programación estándar.

Palabras Claves: redes inalámbricas de sensores, sistemas operativos, simulaciones

ABSTRACT

The state of the art technology enables high efficiency and low cost electronics for massive use. Very low-cost devices that integrate a multitude of sensors that communicate with each other via radio frequency links is what today is known as a wireless sensor network. These devices have various limitations in both processing power and memory, yet there are applications that require autonomous functioning for years, so it is important that its use is energy-efficient.

There are two ways to verify the correct operation of an application. The first way is to check the performance on physical devices, a process that is costly and slow. A second way is to use a simulator to analyze the behavior of a network before being deployed in the field.

There are several solutions that allow developing applications for wireless sensor networks under the above restrictions, but so far there are no solutions that allow for both the development of applications and simulations under the same programming language. This requires developing algorithms twice, once for the hardware and the second for the simulation. Furthermore, the available solutions use complex programming techniques, which increases the learning curve for new developers.

This thesis describes the design, development and validation of LatinOS, an operating system for wireless sensor networks which in turn is coupled to a simulator under one standard programming language.

Keywords: wireless sensor networks, operating systems, simulations

Capítulo 1. INTRODUCCIÓN Y DEFINICIÓN DEL PROBLEMA

Las comunicaciones inalámbricas han sido una importante área de investigación y desarrollo en el mundo en las décadas recientes. Debido a los avances tecnológicos que han permitido aumentar la eficiencia y disminuir los costos de la electrónica para uso masivo, hoy en día es posible construir pequeños dispositivos de comunicaciones de muy bajo costo y que pueden funcionar en base a baterías durante años. Un conjunto de estos dispositivos comunicándose entre sí es lo que en la actualidad se conoce como red inalámbrica de sensores.

Típicamente las redes de sensores tienen, pero no están limitadas a, cuatro modos de operación. Las redes de monitoreo son las más comunes. Un ejemplo típico de aplicación es la medición de variables meteorológicas en una región geográfica extensa, como puede ser un predio agrícola o un glaciar (Barrenetxea y cols., 2008). Las redes de respuesta ante eventos se utilizan para indicar la detección de situaciones que requieren de atención inmediata, tales como incendios forestales o la repentina baja de presión de un paciente en un hospital. Existen también redes en que los nodos, además de reportar valores de los sensores, poseen actuadores que permiten controlar el ambiente de forma remota, como por ejemplo abriendo válvulas que controlen el flujo de agua para efectos de regadío. Otro modo, menos frecuente, es el *streaming* de datos en redes que reportan, por ejemplo, el valor de la aceleración en 3 ejes durante pruebas sismológicos en un edificio. La gran variedad de aplicaciones de redes inalámbricas de sensores hacen indispensable disponer de dispositivos flexibles que puedan ser configurados fácilmente para operar en todas las condiciones.

Las redes inalámbricas de sensores se componen de pequeños nodos, cuya arquitectura se basa en un microcontrolador, un sistema de transmisión de datos por radiofrecuencia, y uno o varios sensores. A nivel comercial, existen múltiples compañías que fabrican nodos inalámbricos (*BTNodes*, s.f.; *Tiny Node*, s.f.; *MEMSIC*, s.f.; *Zolertia*, s.f.), similares al que se muestra en la Figura 1.1, donde se distinguen el área del transmisor y el microcontrolador, además de algunos sensores.

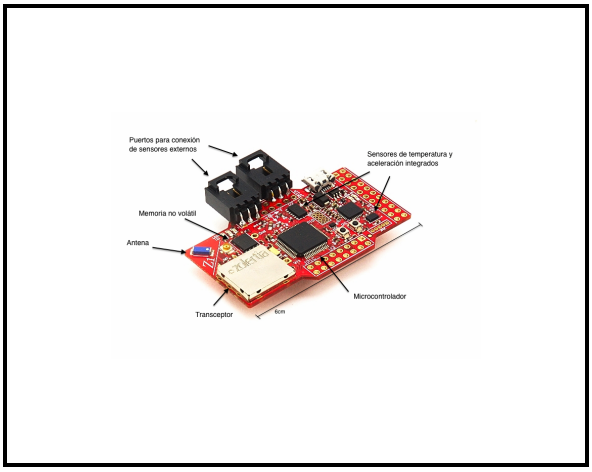


FIGURA 1.1. Nodo Zolertia Z1 (*Zolertia*, s.f.)

La gran mayoría de los nodos tradicionales comparte una arquitectura como la que se muestra en la Figura 1.2. El transceptor, compuesto por un módem de banda base y una radio, realizan la comunicación de la información digital manejada por el microcontrolador. La memoria es utilizada por el microcontrolador para almacenar información recibida de sensores propios o por vía inalámbrica desde otros nodos, además de almacenar información propia, como el número de serie del dispositivo y parámetros de configuración. El microcontrolador contiene internamente un programa, o *firmware*, que controla el funcionamiento del nodo y que es instalado utilizando el controlador USB. La batería provee alimentación de energía a los elementos recién descritos y permite almacenar energía obtenida de otras fuentes, como paneles solares.

En la mayoría de los casos, los nodos disponibles comercialmente tienen limitaciones tanto en energía como en capacidad de procesamiento, lo que impone condiciones tanto sobre las características que debe tener el *firmware* instalado en el microcontrolador para maximizar el tiempo de operación de la red de sensores, como sobre los protocolos de comunicaciones que minimizarán el número de transmisiones.

Debido a que a veces es necesario desplegar redes inalámbricas de sensores en lugares remotos, el cambio de baterías o la instalación manual, en el lugar, de nuevo *firmware* se

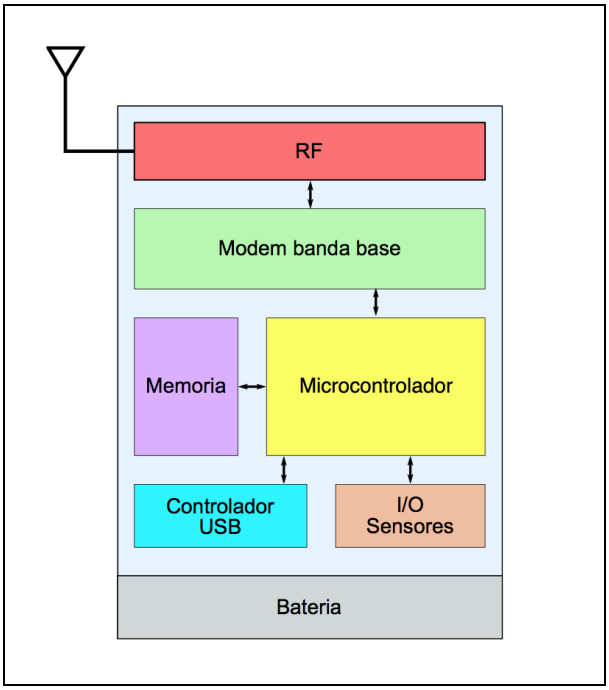


FIGURA 1.2. Arquitectura tradicional de un nodo inalámbrico

hace prácticamente imposible. Estas limitantes generan dos grandes restricciones en el diseño del *firmware*:

(a) Gestionar a los distintos componentes electrónicos del nodo procurando que la energía no se malgaste. Por ello, es vital que el diseño y desarrollo de la arquitectura del *firmware* sea capaz de proveer funcionalidades de manejo de energía. La arquitectura debe además ser lo más genérica posible para soportar a un variado número de plataformas de *hardware* sin requerir grandes modificaciones.

(b) La segunda restricción tiene que ver con la estabilidad del *firmware* instalado. Debe comprobarse de que el *firmware* a instalar en el nodo esté completamente depurado de errores que puedan perjudicar el funcionamiento normal de la red. Pruebas exhaustivas del *firmware* deben realizarse con redes similares a las que se utilizarám en la realidad. Sin embargo, producir una red de prueba que permita depurar errores para posteriormente instalar el nuevo *firmware* en todos los nodos es impracticable, debido a su alto costo y a la imposibilidad de reproducir todos los escenarios bajo los que operará la red.

La solución a este problema típicamente se aborda mediante simulaciones. Los simuladores son programas computacionales que permiten modelar ciertas características de un sistema real y recrear su comportamiento en un tiempo y costo reducido. Los simuladores además permiten generar y recrear condiciones que pueden ser difíciles de obtener en la realidad, como condiciones climatológicas específicas o redes con un número de nodos de alto costo total. Una simulación de una red completa permite comprobar que los protocolos, algoritmos, funciones, etc, que se implementarán en el *firmware* funcionan correctamente, a un costo aceptable. Los simuladores existentes no son aptos para estos propósitos. Un primer grupo utiliza el mismo código escrito para el *firmware*, permitiendo una simulación del funcionamiento del nodo consistente con la operación real, pero requieren de extensiones que simulen el ambiente de operación de los nodos en forma fidedigna (canal inalámbrico, entorno físico y variables observadas). Hasta la actualidad no se dispone de un simulador con extensiones adecuadas y precisas. Otro grupo son simuladores que representan de manera más precisa el canal de radiofrecuencia, pero están basados en un lenguaje diferente al utilizado por el *firmware*. Esto implica que toda la funcionalidad simulada debe ser re-implementada en *firmware* para su uso en terreno.

Basado en el problema descrito, la hipótesis de trabajo es que es posible generar un sistema operativo para redes inalámbricas de sensores que incluya tanto la programación de las aplicaciones como un simulador capaz de corroborar el correcto funcionamiento de estas, todo asociado a un mismo lenguaje de programación estándar y multi-plataforma.

El objetivo del presente trabajo fue diseñar y desarrollar dicho ambiente de desarrollo de *firmwares* para nodos inalámbricos acoplado a un simulador de canal que no presente las deficiencias de propuestas anteriores. De esta manera, la simulación y las pruebas sobre la plataforma real pueden realizarse con un mismo sistema. Esto permite al desarrollador programar y comprobar ideas en el simulador, con la seguridad que luego funcionarán de manera idéntica en una red real.

Esta tesis fue desarrollada en el marco del proyecto FONDEF D09I1094, cuyo fin es desarrollar una tecnología de múltiples antenas para redes inalámbricas de sensores.

Por este motivo, este trabajo busca además que el ambiente de desarrollo mencionado anteriormente facilite una futura inclusión de tecnologías de múltiples antenas.

El documento está estructurado de la siguiente forma. El Capítulo 2 presenta una revisión de los conceptos asociados a sistemas operativos y simuladores para redes de sensores. Una descripción detallada de la solución propuesta se encuentra en los Capítulos 3 y 4, donde se hace énfasis en el diseño y en la arquitectura desarrollada. En el Capítulo 5 se presentan pruebas realizadas y sus resultados. Finalmente, el Capítulo 6 resume las conclusiones y enumera las principales áreas de trabajo futuro.

Capítulo 2. MARCO TEÓRICO

En el mundo académico y en la industria existen diversas soluciones planteadas tanto para el problema de los sistemas operativos como para los simuladores. En este capítulo se realiza una descripción de conceptos necesarios para esta tesis y se describen algunas de las soluciones planteadas, para luego realizar una comparación cualitativa entre ellas.

2.1. Redes Inalámbricas de Sensores

Una red inalámbrica de sensores (RIS, o simplemente "red de sensores") es una red de comunicaciones cuyo propósito es observar un determinado sistema o fenómeno distribuido que evoluciona en el tiempo. Las redes inalámbricas de sensores están compuestas por nodos, cuya arquitectura es generalmente la mostrada en la Figura 1.2. La información obtenida por los nodos es transmitida hacia un nodo designado como acumulador. La comunicación de datos desde los nodos hacia el acumulador no es, en general, directa y se logra en varios saltos a través de otros nodos de la red, los que típicamente también son nodos de sensores (Figura 2.1). Los datos que llegan al acumulador son procesados localmente o transmitidos mediante algún canal de comunicaciones tradicional hacia un servidor remoto para almacenamiento y procesamiento. El procesamiento de los datos puede ser en tiempo real o diferido, según la aplicación.

Las aplicaciones de las redes de sensores son vastas y variadas (Sohraby y cols., 2007), pero no hay duda que aún quedan muchas aplicaciones innovadoras por descubrir. Entre los ámbitos de aplicación ya conocidos cuentan:

- Automatización industrial (líneas de producción, manufactura, minería, etc.)
- Ganadería (monitoreo de ganado, ubicación, actividad)
- Agricultura y forestal (monitoreo microclimatológico para detección de plagas, detección temprana de incendios forestales, irrigación de precisión en agricultura de secano)

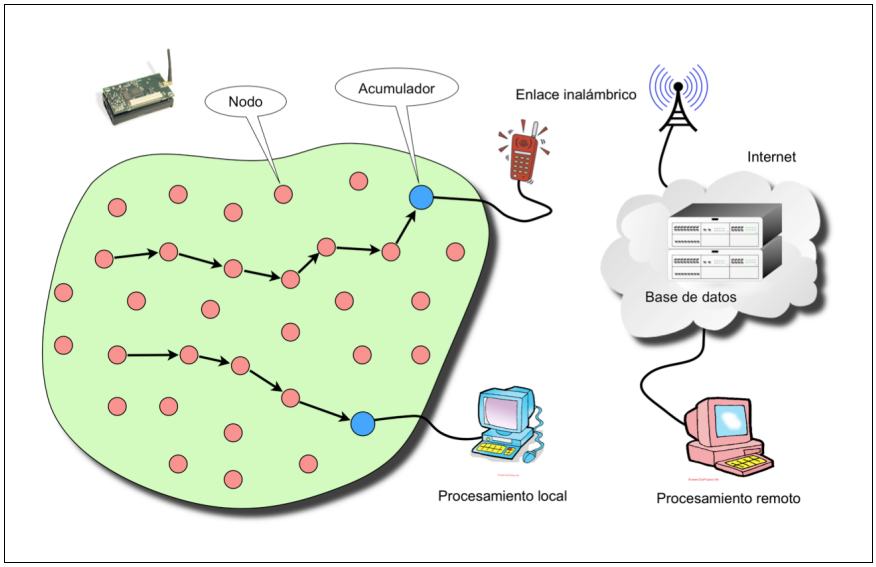


FIGURA 2.1. Red inalámbrica de sensores. El transporte de datos desde los nodos sensores hacia el acumulador ocurre mediante una ruta de múltiples saltos de un nodo a otro. Los datos pueden ser procesados localmente en un acumulador o ser desaguados hacia una base de datos y procesador remotos.

- Aplicaciones en transporte público y privado (instrumentación a bordo de automóviles, comunicaciones vehículo a vehículo y vehículo a cuneta)
- Ingeniería civil (monitoreo sísmico de estructuras, alerta temprana de fallas en puentes)
- Ingeniería ambiental (monitoreo de recursos hídricos y contaminantes)
- Medición de consumo de servicios residenciales (automatización de recolección de datos de medidores, tarificación de consumo en tiempo real basada en datos de demanda obtenidos en línea)
- Ciencia (monitoreo climatológico, medio ambiente marino)
- Alerta y monitoreo de desastres naturales (vulcanología, humedad en suelos conducente a aluviones, incendios forestales, terremotos y tsunamis)

Los parámetros fundamentales que deben ser considerados en el diseño y despliegue de una red de sensores son la cobertura y la conectividad requeridas por la aplicación (Krishnamachari, 2005). Cobertura se refiere a la densidad de nodos por unidad de área o volumen y a la frecuencia temporal con la cual cada nodo toma lecturas del fenómeno

de interés. Conectividad es un indicador de cuantas conexiones tiene cada nodo con otros nodos de la red. Este indicador habla sobre la fluidez con que la información puede circular a través de la red hasta llegar al punto de recolección y/o procesamiento. Los niveles de cobertura y conectividad alcanzados tienen relación directa con el grado de calidad de la información recolectada sobre el fenómeno bajo observación.

La vida útil de una red de sensores depende de la autonomía energética que sus nodos posean bajo los niveles deseados de cobertura y conectividad. Por ello, lograr alta eficiencia energética de los nodos es un criterio de diseño fundamental de toda red de sensores. Esta problemática ha sido el hilo conductor de innumerables contribuciones científicas y textos académicos que reportan una variedad de algoritmos, protocolos y sistemas operativos embebidos que permiten un uso eficiente de la energía para distintas y variadas aplicaciones y condiciones de operación (Kuorilehto y cols., 2007; Krishnamachari, 2005; Karl y Willig, 2005; X. Li, 2008; Y. Li y cols., 2008; Labrador y Wightman, 2009; Sohraby y cols., 2007; Verdone y cols., 2008; Zhao y Guibas, 2004).

2.2. Redes de sensores con Múltiples Antenas

No obstante la extensa literatura disponible, la búsqueda de eficiencia energética en redes de sensores mediante soluciones novedosas en la capa física ha sido escasa. En particular destaca la poca exploración que ha tenido el uso de arreglos de antenas en este ámbito. Las comunicaciones con múltiples antenas, también conocidas por "comunicaciones MIMO", constituyen un tema que ha recibido mucha atención por la comunidad científica en la última década (Goldsmith, 2005; Kühn, 2006; Tse y Viswanath, 2005). La tecnología ya ha sido incorporada a la última generación del estándar de redes inalámbricas de área local WiFi (IEEE 802.11n (*Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications — Amendment 4: Further Higher Data Rate Extension in the 2.4 GHz Band*, s.f.)) y a las próximas generaciones de telefonía móvil (4G, conocida como LTE (Astely y cols., 2009; Irmer y cols., 2009) y 5G, conocida como LTE- Advanced (Mogensen y cols., 2009)). Sin embargo, la tecnología aún no

ha encontrado su sitio en redes de sensores, lo que se atribuye principalmente a la mayor complejidad computacional necesaria para realizar comunicaciones con arreglos de antenas. Dependiendo de la tarea (por ejemplo, estimación de canal, pre-codificación o decodificación de las transmisiones), la complejidad típicamente crece con el cuadrado o con el cubo del número de antenas (elementos radiantes). Así, el esfuerzo computacional para realizar una transmisión con un arreglo con 4 elementos radiantes puede ser entre 16 y 64 veces mayor que con nodos tradicionales que tienen un sólo elemento radiante. Asumir dicho costo en sistemas WiFi y de telefonía celular, ambas tecnologías más maduras que las redes de sensores, ha sido obligatorio para mejorar su desempeño. En el caso de WiFi, la generación anterior (IEEE 802.11g (*Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications Amendment 5: Enhancements for Higher Throughput*, s.f.)) sufrió de problemas de congestión en *hot-spots* y su alcance fue menos que lo pronosticado. Así, la tecnología MIMO fue incorporada en WiFi para mejorar las tasas de datos y cobertura posibles. En el caso de la telefonía móvil, la tercera generación llegó esencialmente al límite práctico de la capacidad de tráfico que un sistema celular con enlaces de antenas simples puede tener (Huang y Valenzuela, 2005).

El pronóstico es que el crecimiento explosivo que se augura en el ámbito de las redes de sensores en los años que vienen conducirá, inevitablemente, a limitaciones tanto de congestión como de interferencia en algunos tipos de aplicaciones. Asimismo, dicho crecimiento generará interés por desarrollar nuevas aplicaciones y explotar nuevos mercados, entre los que cuentan problemas de amplia extensión geográfica. Por ahora, sin embargo, las redes de sensores son una tecnología comparativamente menos madura que WiFi y que la telefonía móvil, y la necesidad de buscar soluciones en el ámbito de las múltiples antenas para enfrentar escenarios como los anteriores aún no es evidente en la industria.

El problema fundamental que subyace al uso de múltiples antenas es determinar cómo deben ajustarse ganancias de magnitud y fase en cada antena para lograr un desempeño óptimo. La optimalidad puede estar orientada a explotar la llamada ganancia de multiplexación o bien la ganancia de diversidad (Tse y Viswanath, 2005). La ganancia de multiplexación es aquella que aprovecha cuando el canal MIMO es utilizado para lograr aumentos

en la tasa de datos. En este caso, todos los elementos radiantes del transmisor emiten señales que contienen información distinta. Las señales se combinan en el aire (fenómeno de superposición) y el receptor, con sus propias múltiples antenas, obtiene suficientes observaciones independientes de la señal combinada en el aire tal que pueda resolver cada una de las transmisiones originales mediante un sistema de ecuaciones. Se obtiene así una ganancia efectiva en la tasa de datos del enlace transmisor-receptor.

La ganancia de diversidad resulta de explotar el extremo opuesto: todos los elementos radiantes del transmisor emiten señales que contienen la misma información (Yacoub, 1993; Jakes, 1994; Kühn, 2006). Las señales, eso sí, son distintas, y deben ser preparadas cuidadosamente considerando las propiedades del canal hacia el receptor. Con ello, las señales transmitidas se combinan constructivamente en el aire generando una ganancia efectiva al llegar al receptor. Las múltiples antenas del receptor operan en forma similar al sumar constructivamente las varias señales recibidas de una misma información. Existen diferentes métodos para implementar la ganancia de diversidad. Estos varían en el desempeño logrado (tasa de error) y complejidad de implementación. La solución óptima es conocida por el acrónimo MRC (*Maximum Ratio Combining*) y exige conocimiento completo del canal de transmisión, pero logra una ganancia de diversidad igual al número de antenas. "Selección pura", por su parte, utiliza siempre la señal de la antena con mayor razón señal a ruido. Por ello, el método no necesita estimaciones del canal, pero logra un factor de mejora menor que MRC y con rendimientos decrecientes por cada antena adicional.

2.3. Microcontroladores

Como fue mencionado en el Capítulo 1, uno de los componentes más importantes de un nodo inalámbrico es el microcontrolador. Un microcontrolador es la implementación de una Unidad de Procesamiento Central (CPU) en un circuito integrado. Contiene una máquina de estados que controla la ejecución de instrucciones de programa, registros para almacenar variables de control y de datos temporales y una Unidad de Lógica Aritmética (ALU) para realizar cálculos numéricos. Un microcontrolador es un circuito integrado

fabricado en una porción de silicio con dimensiones menores a 7 mm^2, tamaño cercano a la cabeza de un plumón de pizarra (Wittie, s.f.).

Los microcontroladores están compuestos internamente por una memoria *flash*, una memoria RAM, una unidad de control, periféricos, entre otros (Wittie, s.f.). La memoria *flash* es utilizada para almacenar el *firmware*, mientras que la RAM es utilizada para el almacenamiento de datos temporales que cambian durante la ejecución del *firmware*. Tamaños típicos de memoria *flash* en microcontroladores rondan entre los 92 kB y los 116 kB, mientras que para la memoria RAM, estos valores se encuentran entre 1 kB y 8 kB (*MSP430F2618 Datasheet*, s.f.). En comparación con los computadores de escritorio, en los que la memoria de programa puede llegar a ser de hasta 500 GB y la memoria RAM de 16 GB, los microcontroladores son muy limitados en cuanto a la cantidad de programas y datos que puede almacenar.

Los microcontroladores pueden responder rápidamente a eventos externos mediante rutinas de interrupción. Al momento de generarse una interrupción, típicamente desde algún periférico, la ejecución normal del *firmware* en el microcontrolador se detiene y se ejecuta una función definida especialmente para atender a la interrupción. Definiremos esta función como *interruption callback*. Cuando finaliza la ejecución de la *interruption callback*, se continua ejecutando el *firmware* desde donde fue interrumpido. Durante la ejecución de la *interruption callback* no se puede volver a interrumpir al microcontrolador, por lo que una interrupción se denomina *evento bloqueante*, a diferencia de la ejecución normal del *firmware*, denominada *tarea no bloqueante*.

Existen dos maneras de utilizar la memoria RAM. Al momento de compilar el *firmware*, el compilador sabe cuanta memoria necesita reservar en la RAM del microcontrolador. Esta asignación de memoria se denomina estática, ya que la utilización de la RAM no cambia durante la ejecución del *firmware*. Existe además, una manera de requerir memoria adicional de la RAM utilizando una función de C llamada *malloc()*, que solicita al microcontrolador el número deseado de bytes adicionales a los ya asignados al momento

de compilar. Esta asignación de memoria adicional se denomina memoria dinámica, ya que la utilización de la RAM varía durante la ejecución del *firmware*.

El lenguaje de programación más utilizado en el desarrollo de aplicaciones para microcontroladores y para computadores comunes es C (Kernighan y Ritchie, 1988). Para generar un *firmware* se escribe código C, y se compila utilizando un compilador apropiado para el microcontrolador en el que se instalará. Para el caso de los computadores de escritorio, el compilador más utilizado para C es *gcc* (Gough y Stallman, 2004), mientras que para microcontroladores se deben utilizar adaptaciones de *gcc*, dependiendo de la marca y modelo del microcontrolador. Para el caso de los microcontroladores de la familia MSP430 de Texas Instruments, usado en diversos nodos comerciales, se puede utilizar *mspgcc* (*mspgcc - GCC toolchain for MSP430*, s.f.).

2.4. Sistemas Operativos

Un sistema operativo (Abraham Silberschatz, 1998) es un programa que actúa como intermediario entre el *hardware* y una aplicación. Por ejemplo la aplicación Microsoft Word puede leer y escribir archivos en el disco duro utilizando el sistema operativo Microsoft Windows. El propósito principal de un sistema operativo es abstraer el uso del *hardware*, permitiendo desarrollar aplicaciones independientes al *hardware* utilizado. El segundo propósito es utilizar el *hardware* de manera eficiente, utilizando los recursos de *hardware* disponibles de manera de evitar el mal uso del procesador, y como consecuencia, de la energía disponible. Como se mencionó anteriormente, un microcontrolador está limitado en términos de memoria, por lo que en general es posible programar sólo una aplicación junto al sistema operativo. De esta manera, el *firmware* contiene simultáneamente tanto al sistema operativo como a la aplicación.

Uno de los sistemas operativos más utilizados para redes inalámbricas de sensores es TinyOS (Hill y cols., 2000; Levis y cols., 2004). Es un sistema basado en eventos bloqueantes y tareas no bloqueantes. Debido a que fue uno de los primeros sistemas operativos diseñados para redes de sensores, es de uso común. TinyOS está escrito en nesC,

lenguaje basado en C y creado por investigadores de Universidad de California, Berkeley (Gay y cols., 2003). TinyOS es compilado estáticamente, por lo que las aplicaciones no son modificables una vez instaladas. Además, TinyOS no permite memoria dinámica, es decir, el tamaño definitivo de la memoria RAM queda definido al momento de compilación.

Contiki (Dunkels y cols., 2004) es un sistema operativo para microcontroladores creado en el Swedish Institute of Computer Science (SICS). Las aplicaciones en Contiki siguen un modelo basado en eventos bloqueantes y en *proto-threads* no bloqueantes, en los que se emula un comportamiento de *multi-threading* con un sistema de semáforos entre los varios procesos. Además, Contiki opera con compilación dinámica lo que permite reemplazar la aplicación dinámicamente. No obstante, al igual que TinyOS, Contiki tampoco posee soporte para memoria dinámica, por lo que los tamaños de memorias *flash* y RAM quedan definidos al momento de compilar. Contiki incluye protocolos de comunicaciones compatibles con TCP/IP (Stevens y cols., 2004).

El sistema operativo Mantis OS (Bhatti y cols., 2005) también sigue un diseño clásico de *multi-threading* para soportar procesos concurrentes, pero requiere reservar memoria adicional dentro de la aplicación para almacenar los distintos contextos de los *threads*. Posee un sistema de reprogramación dinámica e incluye un sistema de control remoto que sirve para obtener información del nodo en tiempo real. Nano-RK (Eswaran y cols., 2005) es un sistema operativo con una arquitectura estática de compilación, por lo que la organización de su memoria queda definida al momento de compilar y no cambia al momento de ser ejecutado. Su diseño sigue un paradigma de reserva de recursos, en el que cada tarea le indica al sistema operativo los recursos que necesita y la cantidad de ciclos que requiere para terminar. Tanto Mantis OS como Nano-RK carecen de un simulador asociado, por lo que solo es posible probar su funcionamiento en nodos reales.

En el cuadro 2.1 se presenta una comparación cualitativa entre los sistemas operativos mencionados. En él se puede observar que para aplicaciones que requieren conectividad a

CUADRO 2.1. Comparación de sistemas operativos.

Sistema Operativo	Basado en tareas	Soporte para *threads*	Simulador asociado	Reprogramación inalámbrica	Compatibilidad TCP/IP
TinyOS	X		TOSSIM		No
Contiki		X	Cooja	X	Sí
Mantis		X	N/A	X	No
Nano-RK		X	N/A	X	No

Internet es buena opción utilizar Contiki debido a que incluye una implementación TCP/IP en su distribución.

2.5. Simuladores

Un simulador es un programa que imita la operación de un proceso real en un ambiente controlado. Los simuladores requieren el desarrollo de modelos de los procesos reales que permitan a estos ser representados por comportamientos y/o características claves de su funcionamiento. Una simulación corresponde a la operación de los modelos en el tiempo en un ambiente controlado por parámetros configurables.

Las simulaciones son utilizadas en variados contextos, como la simulación de tecnologías para optimizar su rendimiento y el entrenamiento de manera segura para operadores de maquinarias riesgosas. En el contexto de las redes inalámbricas de sensores, los simuladores son utilizados con el fin de comprobar el correcto funcionamiento de los algoritmos implementados en el *firmware*. Además, permiten generar y recrear condiciones climatológicas difíciles de encontrar en situaciones reales.

TinyOS incorpora como parte de su distribución una extensión llamada TOSSIM (Levis y cols., 2003) que permite simular aplicaciones completas a nivel de protocolos de comunicaciones. Existe también una extensión a este, llamado PowerTOSSIM (Shnayder y cols., 2004), que simula además los consumos energéticos de los algoritmos de la aplicación. Si bien TOSSIM permite realizar simulaciones de las aplicaciones de TinyOS,

está desarrollado en lenguaje *Python*, por lo que su uso junto a TinyOS requiere interoperabilidad de código escrito en dos lenguajes distintos. Finalmente, TOSSIM no soporta la simulación de tecnologías de múltiples antenas.

Contiki incluye en su distribución un simulador llamado Cooja (Österlind, 2006), basado en Java. Este simulador utiliza probabilidades de error entre enlaces para simular canales de variadas estadísticas, y está diseñado en forma modular, lo que permite incorporar nuevos modelos de canal en las simulaciones. Si bien Cooja permite realizar simulaciones de las aplicaciones de Contiki, está desarrollado en un lenguaje distinto que las aplicaciones y que el sistema operativo. Al igual que TOSSIM, Cooja no soporta simulaciones de tecnologías de múltiples antenas.

Para redes de sensores también existen simuladores *stand-alone*. Uno de ellos es el simulador Worldsens (Fraboulet y cols., 2007), compuesto por un simulador de nodos inalámbricos llamado WSim, y WSNet, un simulador del canales inalámbricos y entorno. Ambos se conectan mediante *sockets* TCP/IP. WSim permite emular el comportamiento exacto a nivel de ciclos de la operación de un microcontrolador MSP430 lo que posibilita utilizar para la simulación el mismo *firmware* que se utiliza para los nodos. WSim y WSNet se utilizan cuando se requieren simulaciones precisas para la evaluación de algoritmos en el microcontrolador. Al igual que con TOSSIM y Cooja, WSNet tampoco soporta simulaciones con tecnologías de múltiples antenas.

NS-2 (*The Network Simulator - NS-2*, s.f.) es un simulador genérico para redes, ya sean inalámbricas o por cable. No incorpora características particulares de las redes de sensores, por lo que no es ideal para las necesidades abordadas por esta tesis.

OMNeT++ (*OMNeT++ Network Simulation Framework*, s.f.) es otro simulador de redes de uso general basado en un diseño modular de componentes que permite especializar la simulación para redes arbitrarias. OMNeT++ está orientado a redes tradicionales como ethernet y Wi-Fi.

Tanto NS-2 como Omnet++ pueden simular tráfico de TCP/IP y enlaces ad-hoc, lo que los hace ideal para simular tráfico en redes de computadores. Sin embargo, para utilizarlos

CUADRO 2.2. Comparación de simuladores.

Simulador	Lenguaje	Dedicado para RIS	Simulación de Canal	Múltiples antenas
TOSSIM	Python	Sí	Probabilística	No
Cooja	Java	Sí	Probabilística	No
Omnet++	Propio	No	Probabilística	No
NS-2	Propio	No	Probabilística	No

es necesario implementar las aplicaciones de la red de sensores en el lenguaje específico del simulador en cuestión (*The Network Simulator - NS-2*, s.f.; *OMNeT++ Network Simulation Framework*, s.f.). Esto implica varias restricciones que hacen a estos simuladores inapropiados para este proyecto.

- Implica una duplicidad de código, uno para la plataforma física en que operará la aplicación y protocolos y otro para el simulador propiamente tal. Esto conlleva riesgos como los errores de transcripción, que si bien son solucionables, son una pérdida de tiempo innecesaria. Además, implica reescribir el código del simulador con cada nueva iteración del código del *firmware*
- Al ser simuladores genéricos, estos no captan propiedades específicas del *hardware* en donde se utilizará el *firmware*.
- La simulación de algoritmos, si bien puede detectar errores de funcionamiento, no permite la simulación de un sistema operativo embebido, por lo que no se capturan errores posibles fuera de los algoritmos.

En el cuadro 2.2 se presenta un resumen acerca de los simuladores descritos. En él puede observarse que no existen simuladores que permitan simultáneamente extensión hacia múltiples antenas y la simulación en un mismo lenguaje de programación.

Capítulo 3. EL SISTEMA OPERATIVO: LATINOS

En este capítulo se describe LatinOS, el sistema operativo desarrollado en la presente tesis para redes inalámbricas de sensores. Este capítulo consta de 4 secciones. En la Sección 3.1 se describe LatinOS y las capacidades que posee. La Sección 3.2 detalla el diseño y la arquitectura modular de LatinOS. En la Sección 3.4 se explican las interfaces entre los distintos módulos que componen LatinOS y finalmente en la Sección 3.3 se presenta la estructura de carpetas diseñada para el código fuente de LatinOS.

3.1. Descripción de LatinOS

LatinOS es un sistema operativo para microcontroladores de nodos de redes inalámbricas de sensores. LatinOS está basado en tareas que son procesos de ejecución no prioritaria y a menudo lenta (procesamiento de la información obtenida desde los sensores, re-transmisión de un paquete recibido, etc.), y en eventos o interrupciones de alta prioridad y de relativa baja duración. En LatinOS, las tareas son no-bloqueantes y los eventos son bloqueantes.

LatinOS tiene un diseño modular, de forma que cualquiera de sus partes, a excepción del mismo sistema operativo, puede ser reemplazada de manera simple por otro módulo similar, permitiendo adecuar la funcionalidad de un nodo sin tener que intervenir el *firmware* completo.

En adelante, el término *aplicación* será utilizado para indicar una configuración específica de LatinOS asociada a un tipo de red en particular. El término *plataforma* se asociará a un modelo específico de nodo inalámbrico, por ejemplo, el el ilustrado en la Figura 1.1. El término *arquitectura* se utilizará para referirse a una familia de procesadores específica, tales como x86, ARM, MSP430, AVR, entre otros.

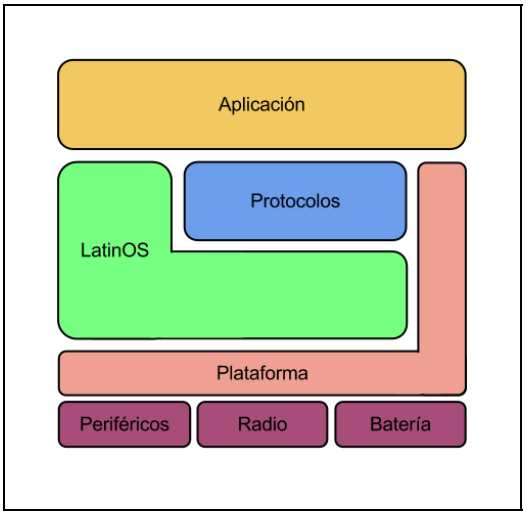


FIGURA 3.1. Estructura de LatinOS

3.2. Diseño Modular

LatinOS está compuesto por 4 módulos (Figura 3.1). La capa inferior corresponde al módulo Plataforma que provee *drivers* necesarios para que el sistema operativo pueda interactuar con periféricos como sensores, memorias *flash*, etc., específicos del *hardware* de cada modelo de nodo. El módulo Plataforma posee las funcionalidades de detección de eventos y generación de interrupciones. Por ejemplo, si en un nodo un sensor de temperatura supera un límite previamente configurado en el sensor, este debe indicarle esta condición a la aplicación para que ésta actúe en consecuencia. Asimismo, si la radio indica haber recibido un paquete, éste debe ser enviado al módulo Protocolo para su procesamiento y administración. Ello se logra mediante interrupciones al microcontrolador. En LatinOS, la estrategia es usar eventos breves que agregan una tarea a una cola de tareas que el sistema operativo luego ejecuta durante el tiempo ocioso entre interrupciones. Este aspecto se describe en detalle en la Sección 3.3.

El sistema operativo (bloque LatinOS), en la segunda capa, provee librerías generalmente requeridas por la mayoría de las aplicaciones en una red inalámbrica de sensores. En este módulo se encuentran los códigos correspondientes a las tareas y los *timers*.

El módulo de Protocolos corresponde a la tercera capa. Este módulo tiene una interfaz directa al módulo de aplicación. El módulo de protocolo de comunicaciones se considera parte del sistema operativo, pero debido a que es un conjunto de funciones de comunicaciones auto-contenidas y variable según el tipo de red que se desea implementar, puede ser considerado un módulo de por sí, que actúa como intermediario entre la aplicación y la red. Este módulo incluye los códigos necesarios para lograr una transmisión correcta hasta el acumulador, como el control de topología, protocolos de enrutamiento y transporte y protocolos de direccionamiento.

El código fuente tanto del sistema operativo como de los protocolos son genéricos, es decir, son independientes tanto de la plataforma sobre la que se está ejecutando como de las aplicaciones. Por lo tanto, no deben hacer referencia directamente a funciones implementadas tanto en el módulo Plataforma como en el módulo Aplicación. El módulo de Protocolo de comunicaciones corresponde a las capas 2 a 6 del modelo OSI (Day, 1995). Las capas 1 y 7 se encuentran en la radio y en la aplicación respectivamente.

Finalmente, la capa superior corresponde al Módulo Aplicación, donde se encuentra el código que se encarga de obtener datos de los sensores y administrarlos de acuerdo a la operación deseada de la red (transmisión, compresión, almacenamiento, etc.). También es responsabilidad de la aplicación configurar los *drivers* de la plataforma y la inicialización del protocolo de comunicaciones.

3.3. Funcionalidades de LatinOS

En esta sección se describen los cuatro componentes sobre los que se basa LatinOS:

- eventos
- tareas
- *timers*
- *Hardware Abstraction Layer (HAL)*

El componente tareas contiene el código que permite administrar la utilización de los recursos del microcontrolador. Se compone de una lista de tareas en la cual se ingresan a la cola los procesos requeridos por los distintos módulos del *firmware*. El componente de los eventos, a su vez, contiene la funcionalidad para responder a las interrupciones generadas hacia el microcontrolador. Este componente esta muy integrado con los *drivers* de la plataforma, ya que cada componente generador de eventos funciona de manera distinta. El componente de *timers* contiene código que permite ingresar procesos a la lista de tareas en tiempos futuros predeterminados. La *HAL* corresponde a la definición de la interfaz entre LatinOS y la plataforma. Este componente busca estandarizar los nombres de las funciones implementadas en el módulo de las plataformas con el objetivo de mantener intacto al resto de los módulos en el momento en que se reemplaza una plataforma por otra.

3.3.1. Eventos

Como se indicó anteriormente, LatinOS es un sistema operativo basado en eventos. Los eventos son interrupciones al microcontrolador y que éste debe ser capaz de atender inmediatamente. Los eventos que ocurren en una red de sensores pueden provenir de estímulos externos, tales como un sensor de temperatura que superó cierto umbral o un acelerómetro que detectó un golpe. Los eventos también pueden tener origen en estímulos internos, tales como un *timer* que indica que ha transcurrido un tiempo predeterminado tras el cual debe ejecutarse una acción.

La arquitectura electrónica de un microcontrolador no permite responder simultáneamente a dos interrupciones, por lo que hay que esperar que la primera interrupción sea ejecutada antes de procesar la segunda. Los procesos gatillados por interrupciones deben ejecutarse en el menor tiempo posible para mantener al microcontrolador en un estado de alta disponibilidad para responder a otros eventos apenas estos se gatillen.

En el caso de que los eventos requieran procesos cuyo tiempo esperado de ejecución supere el tiempo aceptable, LatinOS ofrece un sistema de tareas, el que se describe a continuación.

3.3.2. Tareas

Las tareas en LatinOS son procesos no bloqueantes ingresados por distintos módulos a una cola de tareas en respuesta a un estímulo o evento. Las tareas pueden ser, por ejemplo, procesar el contenido de un paquete recibido o ejecutar un proceso ingresado a la cola de tareas por un *timer*.

Las tareas son asíncronas ya que no se ejecutarán en el instante en que se produce la interrupción, si no que cuando el microcontrolador tenga disponibilidad para ellas.

LatinOS tiene un mecanismo de priorización de tareas. La prioridad de una tarea se especifica al momento de su creación. De esta forma, por ejemplo, tareas críticas pueden tener mayor prioridad de ejecución que un *timer* que realiza mediciones periódicas.

3.3.3. Timers

Los *timers* son mecanismos del sistema operativo que interrumpen al microcontrolador cuando se cumple el instante en el que se debe ejecutar una tarea. Los *timers* son parte del sistema operativo ya que son requeridos frecuentemente por aplicaciones de redes de sensores. Son utilizados tanto por los protocolos de red para generar *timeouts*, como por las aplicaciones para regular el ciclo de trabajo del microcontrolador.

El módulo de *timers* de LatinOS posee internamente una lista que contiene el instante y tarea de *timers* ya registrados. Esta lista puede contener un máximo de 255 *timers*. Una vez que es tiempo de ejecutar un *timer*, la tarea asociada al *timer* es agregada a la lista de tareas de LatinOS.

Para su funcionamiento, los *timers* de un sistema operativo se enlazan a *timers* del *hardware* propiamente tal del microcontrolador, que pueden variar tanto es su funcionamiento como cantidad. Por ello, los timers son complicados de implementar. Ante esta restricción, el módulo de *timers* se diseñó de tal forma que los *timers* de LatinOS utilizan solo un *timer* del microcontrolador. Esto es logrado configurando el *timer* de *hardware* para crear un evento correspondiente al primer *timer* de LatinOS. Una vez generado el evento correspondiente al *timer* configurado, se configura el siguiente *timer* de LatinOS.

3.3.4. Hardware Abstraction Layer (HAL)

Para que LatinOS tenga el potencial de funcionar en cualquier plataforma fue necesario diseñar una interfaz genérica entre LatinOS y el *hardware* de la plataforma. De esta forma se evita hacer modificaciones mayores tanto a LatinOS como a las aplicaciones cada vez que se desee soportar una nueva plataforma. Así que por cada nueva plataforma sólo se debe agregar una nueva carpeta con el código de abstracción de hardware específico.

Para diseñar la *HAL*, se estudió cuales eran los módulos comunes que toda plataforma debe tener para ser considerado un nodo inalámbrico. Todos los nodos tienen al menos un microcontrolador, una radio y una memoria externa. Debido a que no existe un consenso sobre qué tipos de sensores tendrá el nodo, no es posible incluir estas interfaces en la *HAL*.

La *HAL* debe definir una serie de funciones genéricas en torno a la radio; para el manejo del microcontrolador, tanto para configurar el modo de bajo consumo como los *timers* internos; y la memoria, para leer y escribir en ella.

3.4. Descripción de Interfaces

Las interfaces más importantes de LatinOS se observan en la Figura 3.2, en la que se ilustran las funciones mínimas que debe implementar cada módulo. Como se aprecia, todas las funciones poseen nombres genéricos, lo que permite modificar o reemplazar cualquiera de estos módulos sin tener que intervenir en la programación de los otros.

La interfaz entre la aplicación y el protocolo de comunicaciones está diseñada para configurar al protocolo. Para transmitir datos, la aplicación utiliza la función *protocol_transmit()*. Para recibir datos lo hace mediante la función *app_receive()*. A su vez, el protocolo intercambia datos y configuraciones mediante la interfaz *os_transmit()* con el sistema operativo.

La interfaz entre la aplicación y LatinOS provee funciones como el registro de *timers*, el ingreso de procesos a la lista de tareas, entre otros. Además, esta interfaz incluye las

funciones *app_boot()* y *app_shutdown()* para señalizarle a la aplicación los eventos de inicialización y terminación respectivamente.

La interfaz entre la plataforma y LatinOS es la denominada *Hardware Abstraction Layer*. Esta interfaz incluye funciones de nombre y prototipo genéricos como *modem_transmit()* y *memory_write()*, pero que deben ser implementadas en forma específica según las características de cada periférico en cada plataforma.

Finalmente, la interfaz entre la aplicación y la plataforma es necesaria para que la primera pueda interactuar con los sensores. Esta interfaz no está definida genéricamente debido a que depende de las características del sistema que se desea observar con la red inalámbrica de sensores.

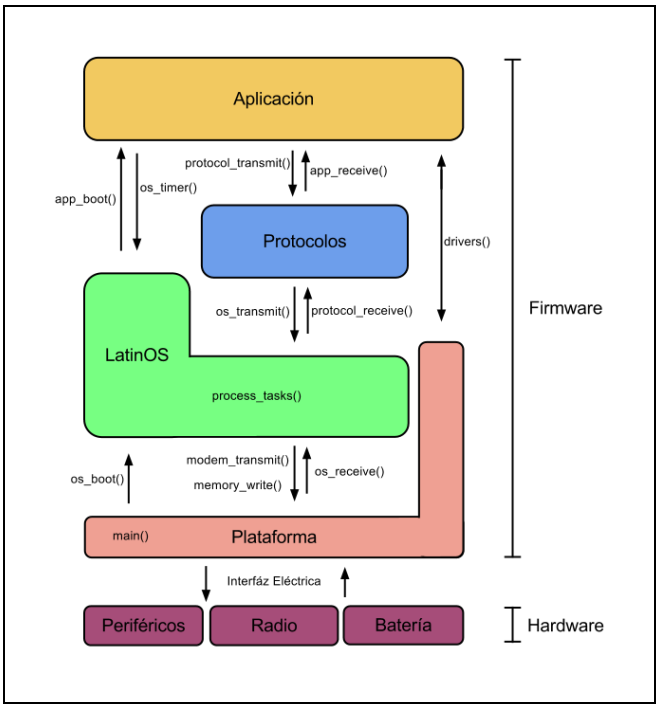


FIGURA 3.2. Definición de las interfaces entre módulos de LatinOS

3.5. Estructura de Carpetas

En el diseño de un sistema operativo es deseable que tanto la estructura del código como el código mismo sean simples de entender y así permitir a futuros desarrolladores

realizar actualizaciones y nuevas aplicaciones sin dificultad. Basado en TinyOS y Contiki (Hill y cols., 2000; Levis y cols., 2004; Dunkels y cols., 2004), se diseñó una organización del código en seis subcarpetas, en la que cada uno de los módulos de la Figura 3.2 corresponde a una subcarpeta dentro del proyecto, como se observa en la Figura 3.3. Esta estructura está diseñada para acomodar una amplia variedad de casos posibles en una aplicación de redes inalámbricas de sensores. El objetivo es que ninguna modificación requiera agregar una nueva subcarpeta.

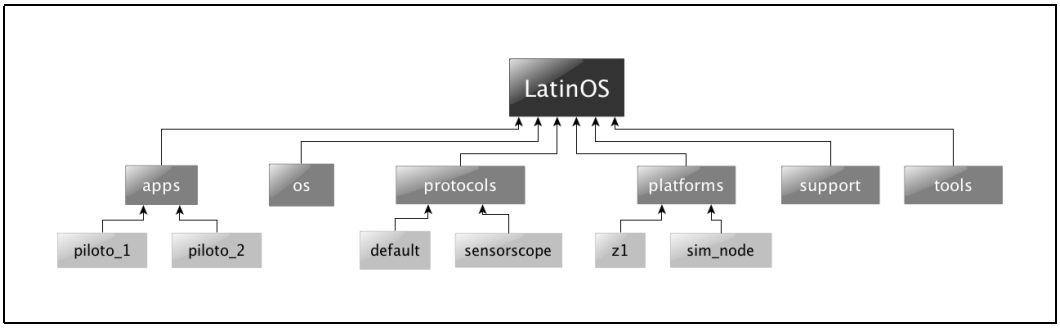


FIGURA 3.3. Estructura de carpetas de LatinOS

En la carpeta *apps* se encuentran las aplicaciones disponibles. Cada subcarpeta dentro de *apps* define una aplicación en particular. Luego, en la carpeta *os* se encuentra todo el código relevante al sistema operativo, tales como los *timers*, la lista de tareas, y la declaración de la HAL, explicados en la Sección 3.3. En la carpeta *protocols* residen los protocolos de red disponibles. Cada subcarpeta dentro de *protocols* define un protocolo de red, compuesto por técnicas de control de topología, enrutamiento, control de transporte, etc., según sea el caso.

La carpeta *platforms* contiene subcarpetas con los códigos fuente específico para cada plataforma. Dentro de cada plataforma existe una subcarpeta *drivers* en la que se definen los códigos que permiten a LatinOS interactuar con los componentes integrados en un nodo de la plataforma correspondiente. Luego, la carpeta *support* contiene código que no corresponde a ninguna de las carpetas anteriores. Finalmente, la carpeta *tools* contiene herramientas de diagnóstico para LatinOS.

3.5.1. Aplicaciones

Las aplicaciones se estructuran en subcarpetas dentro de la carpeta *apps*. Cada una de estas subcarpetas contiene un archivo de código y un archivo llamado *Rakefile*. El código de la aplicación debe contener dos funciones, *app_boot* que se encarga de la configuración inicial, y *app_receive*, función que especifica cómo procesar los paquetes recibidos a nivel de aplicación.

El archivo *Rakefile* contiene instrucciones de compilación y configura tanto el protocolo de comunicaciones a utilizar como las plataformas para las cuales es posible compilar la aplicación. Es necesario especificar las plataformas posibles a compilar, ya que si una aplicación utiliza un elemento específico de una plataforma que no está presente en otras, es posible que la aplicación genere errores y no compile.

3.5.2. LatinOS

La carpeta *os* contiene el código fuente del sistema operativo, incluyendo la programación de los *timers*, el control de la cola de tareas, y la HAL. Cada uno de estos componentes está asociado a un archivo propio (*timer.c*, *task.c*, etc.), a excepción de la HAL que se encuentra distribuida en múltiples archivos, separados por funcionalidades comunes. Cada uno de los archivos correspondientes a la *HAL* contiene las declaraciones y prototipos de las funciones incluidas en la interfaz entre LatinOS y la plataforma. Por ejemplo, la declaración de las funciones relativas a la utilización de la memoria *flash* (*flash_enable(), flash_write(), flash_read()*, etc.) se encuentran en un mismo archivo llamado *flash.h*. De igual manera, la declaración de las funciones correspondientes al modem se almacenan en el archivo *modem.h*.

3.5.3. Plataformas

Al igual que las aplicaciones, cada plataforma queda definida mediante una subcarpeta en la carpeta *platforms*. En esta carpeta deben existir por lo menos tres elementos. El primero es el código que contiene la inicialización del programa en C del *firmware*, que generalmente queda definida en dentro de la función *main()*.

El segundo elemento corresponde a un archivo *Rakefile.platform_name*, que incluye la lista de archivos que son requeridos por la plataforma *platform_name* al momento de compilar, tales como librerías específicas para la arquitectura del microcontrolador.

El tercer elemento que debe estar presente es una subcarpeta llamada *drivers* que contiene los códigos específicos para interactuar con los distintos componentes que integran la plataforma como los sensores, la radio y la memoria. Como no es posible predecir los componentes que tendrá cada plataforma, el programador es responsable de definir una interfaz ordenada para comunicarse con la aplicación.

Como parte de este trabajo de tesis, se han implementado 2 plataformas. La plataforma *z1* corresponde a la mostrada en la Figura 1.1, que posee un microcontrolador de la familia MSP430. También se ha desarrollado la plataforma *node_sim*, que es una plataforma virtual que permite simular redes de sensores. Este aspecto se describe en detalle en el Capítulo 4.

3.5.4. Protocolos

Dentro de la carpeta *protocols* se encuentran los distintos protocolos de comunicaciones implementados.

Al igual que las plataformas, cada protocolo incluye un archivo *Rakefile.protocol_name* que contiene la lista de archivos necesarios para compilar el protocolo *protocol_name*. Las funciones mínimas que debe implementar un protocolo son *protocol_transmit()* y *protocol_receive()*, que componen la interfaz básica entre los módulos Aplicación y LatinOS de la Figura 3.2.

Como parte del proyecto Fondef en el que se encuentra inserta esta tesis, se han implementado los protocolos *sensorscope* y *default*. El protocolo Sensorscope se basa en el trabajo descrito en (Barrenetxea y cols., 2008), mientras que el protocolo *default* no realiza ningún procesamiento en los paquetes, y se los entrega directamente a la aplicación.

3.5.5. Soporte y Herramientas

Las carpetas *support* y *tools* contienen códigos de soporte para LatinOS. En la carpeta *support* se encuentra código que es común para más de alguna plataforma. Por ejemplo, dentro de *support*, la carpeta *cpu/msp430* contiene código que es común a todas las plataformas que posean un microcontrolador de la familia MSP430. A su vez, también se encuentra la carpeta *simulation*, que incluye código utilizado tanto por la plataforma *node_sim* como por el simulador (Capítulo 4).

En la carpeta *tools* se encuentran herramientas de soporte para el desarrollo de aplicaciones en LatinOS. En esta carpeta se encuentra el código del simulador desarrollado en conjunto con LatinOS, descrito en el Capítulo 4.

Capítulo 4. SIMULADOR

Disponer de un simulador que permita ensayar la operación de un *firmware* en un ambiente controlado y reproducible es una característica muy atractiva para disminuir los tiempos de programación de los algoritmos en los nodos, pues evita el trabajoso ciclo de ensayo en nodos reales. En el marco del proyecto LatinOS y de la presente tesis, se ha desarrollado un simulador capaz de reproducir de manera exacta una red desplegada, utilizando el mismo lenguaje de programación y código fuente utilizado en el *firmware*, que será descrito en este capítulo.

La Sección 4.1 presenta la estructura del simulador y los mecanismos de comunicación entre los nodos virtuales y el controlador. La Sección 4.2 describe la configuración y funcionamiento de la simulación.

4.1. Estructura del Simulador

El simulador está estructurado como un controlador que interactúa con un conjunto de nodos virtuales. El controlador es una aplicación que se responsabiliza de llevar a cabo la simulación, gestionando el flujo de eventos que ocurren durante la simulación. Los nodos virtuales corresponden a una aplicación y representan el *firmware* a utilizar en la red de sensores.

El nodo virtual almacena exactamente los mismos códigos utilizados en nodos reales, compilados para una plataforma distinta llamada *node_sim*, como se explicó en la Sección 3.2. La plataforma *node_sim* contiene *drivers* que emulan los componentes básicos de un nodo, como la radio y la memoria *flash*. El *driver* de la radio es el encargado de realizar la comunicación con el controlador, mientras que la memoria *flash* emula el comportamiento de una memoria *flash* real mediante un archivo de texto independiente para cada nodo virtual. En términos prácticos, los nodos virtuales son procesos independientes que ejecutan el *firmware* compilado para el computador utilizado en la simulación (típicamente de arquitectura x86). Debido a que los nodos virtuales corresponden a una instancia de

LatinOS, su arquitectura interna no será descrita en este capítulo ya que fue desarrollada en el Capítulo 3.

El controlador está compuesto por cuatro módulos que se comunican entre sí para llevar a cabo la simulación. Sus funciones se describen a continuación:

Módulo de eventos: Al igual que LatinOS, el simulador está basado en eventos. El módulo de eventos define las representaciones virtuales de los eventos descritos en la Sección 3.3.1. En el simulador, las estructuras de datos que representan los eventos reales contienen el tiempo en el que ocurrirá el evento y un identificador que permite distinguir a qué nodo corresponde cada evento. Los eventos son almacenados en una agenda interna, ordenados por tiempo de ocurrencia.

Los eventos se clasifican en dos tipos: eventos genéricos y eventos de radio. La mayoría de los eventos que ocurren en el nodo real pueden ser representados en un mismo tipo de evento genérico debido a que se diferencian en la clase de evento que representan pero no en su estructura. Los eventos genéricos implementados en el simulador corresponden a los de poder (nodo se encendió, nodo se apagó), de *timers* y de sensores (temperatura, aceleración, entre otros). Es posible agregar tantas clases de eventos genéricos al módulo de eventos como sea necesario.

El segundo tipo de evento corresponde a los eventos de radio, que indican la llegada de un paquete a un nodo e incluyen, además del tiempo de recepción, el paquete enviado. Los eventos de radio contienen dos arreglos de *floats*, uno con los símbolos en fase y otro con los símbolos en cuadratura, representando una comunicación en banda base. Esta estructura está diseñada para ser extensible a simulaciones de múltiples antenas, en las que por cada antena se debe agregar a dicha estructura un par de arreglos adicionales.

Módulo de Nodos: El módulo de nodos se encarga de administrar los nodos virtuales utilizados en la simulación. Los procesos que representan los nodos virtuales son instanciados por este módulo.

En el módulo de nodos se encuentra toda la información acerca de los nodos virtuales, como las coordenadas de los nodos en un espacio virtual y las ganancias de antena de cada nodo, necesarias para el correcto funcionamiento de la simulación. En este módulo también se encuentran los mecanismos mediante los cuales el controlador se comunica con los procesos que ejecutan el *firmware*.

Módulo de Canal: El módulo de canal permite simular los canales inalámbricos que afectan la transmisión de los paquetes de radio. Cuando un nodo transmite un paquete, dicho paquete es procesado por este módulo, que agrega tantos eventos de radio como nodos en la red, sin re-transmitirlo al nodo originario. El tiempo de recepción y la potencia recibida de cada paquete depende de la distancia entre el nodo emisor y el receptor. Esta información es consultada por el módulo de canal al módulo de nodos.

La pérdida de potencia por distancia se calcula utilizando la ecuación de presupuesto de enlace (Rappaport, 2001; Goldsmith, 2005) que se presenta en la expresión 4.1, donde P_r corresponde a la potencia recibida y P_t a la potencia transmitida. G_t y G_r corresponden a las ganancias de las antenas transmisora y receptora respectivamente. El parámetro α corresponde al coeficiente de pérdida por distancia dependiente del ambiente en donde se realiza la transmisión. Un valor $\alpha = 2$ representa el caso ideal en el cual no existen obstrucciones y $\alpha = 3{,}5$ representa un alto nivel de dificultad para propagarse (Goldsmith, 2005). d_0 es una distancia de referencia para el campo lejano de la antena, λ corresponde a la longitud de onda de la señal y d es la distancia entre la antena transmisora y la receptora.

$$P_r = \frac{P_t\, G_t\, G_r\, \lambda^2}{(4\,\pi\, d_0)^2} \times \left(\frac{d_0}{d}\right)^{\alpha} \tag{4.1}$$

De acuerdo a la expresión (4.1), la potencia recibida es constante cuando la distancia entre los nodos no cambia, supuesto que no corresponde apropiadamente al comportamiento en la realidad, ya que existen distintos tipos de obstrucciones y

condiciones de canal. Existen diversos modelos matemáticos que permiten representar los distintos efectos que pueden ocurrir en un canal inalámbrico. Para esta tesis fueron implementados dos de estos modelos (Goldsmith, 2005; Rappaport, 2001):

1. El desvanecimiento de gran escala, que corresponde a una pérdida de potencia de la señal transmitida causada por la "sombra" de edificios y montañas.
2. El modelo Rayleigh, que permite modelar canales con múltiples trayectorias.

Módulo de Control: El módulo de control contiene el código que implementa el algoritmo de la simulación. Es responsable de inicializar, configurar, ejecutar y finalizar la simulación. Este módulo gestiona los eventos contenidos en la agenda del módulo de eventos y los nodos a través del módulo de nodos.

Las comunicaciones entre el controlador y los nodos virtuales se realizan mediante *sockets* TCP/IP (Stevens y cols., 2004). Estos *sockets* garantizan integridad de los datos intercambiados entre el controlador y los nodos virtuales, asegurando el correcto funcionamiento de la simulación.

Utilizando los *sockets* TCP/IP, se desarrolló una interfaz mediante la cual se comunica el controlador con los nodos virtuales. Esta interfaz se compone de los mensajes resumidos en la Tabla 4.1.

4.2. Funcionamiento de la Simulación

El controlador posee la capacidad de interpretar un archivo que configura la simulación. Los parámetros a configurar son, por ejemplo, el *firmware* que se cargará en el nodo virtual, el número de nodos de la red a simular y las posiciones de cada uno de los nodos en el espacio virtual de la simulación, entre otros. Una vez iniciada la simulación, el controlador instancia el número de nodos virtuales especificado en el archivo de configuración.

CUADRO 4.1. Descripción de los mensajes entre controlador y el nodo virtual.

Nombre	Emisor	Detalle
event_add	Nodo	Permite agregar un nuevo evento a la lista de eventos del controlador
event_remove	Nodo	Permite eliminar un evento de la lista de eventos del controlador
event_ack	Controlador	Indica que el evento se agregó/eliminó correctamente
event_execute	Controlador	Indica al nodo virtual que debe procesar un evento
event_complete	Nodo	Indica al controlador que terminó de ejecutar el evento
node_identify	Nodo	Se identifica al controlador, utilizada en la fase inicial
radio_transmit	Controlador, Nodo	Transmite un paquete

El archivo de configuración permite además configurar parámetros específicos para cada nodo, por ejemplo la ganancia de antena, así como condiciones del canal inalámbrico. También es posible especificar en el archivo de configuración el tiempo máximo de simulación y eventos predeterminados para ser agregados a la lista de eventos, tales como el evento de apretar un botón en un nodo.

En el Archivo 4.1 se muestra un ejemplo de archivo de configuración. En línea (1) se puede apreciar que la simulación se realizará con dos nodos virtuales, con identificadores 1 y 2 que se encontrarán en un espacio virtual en las posiciones $(0\text{ cm}, 1000\text{ cm}, 0\text{ cm})$ y $(0\text{ cm}, 2000\text{ cm}, 0\text{ cm})$ respectivamente, como se detalla en las líneas (5) y (6). El *firmware* para los nodos virtuales se encuentra en el archivo *bin/node_sim.bin* (línea 3). Finalmente se agregó un evento del tipo *BUTTON* en el segundo 10 para el nodo 1, como se muestra en la línea (8). La línea (10) configura un tiempo máximo de simulación de 100 minutos.

ARCHIVO 4.1. Ejemplo de archivo de configuración de la simulación

```
1 n_nodes 2
2
```

```
 3 default_node_path bin/testnode_sim.bin
 4
 5 node_pos 1 0 1000 0
 6 node_pos 2 0 2000 0
 7
 8 add_event 1 10 BUTTON_EVENT
 9
10 max_time 100 m
```

El flujo de la simulación ocurre como se ilustra en la Figura 4.1. En la fase inicial se inicia el controlador. Durante esta fase el controlador interpreta el archivo de configuración, instanciando el número indicado de nodos virtuales. Al instanciarse los nodos virtuales, éstos se conectan mediante *sockets* TCP/IP al controlador, el que recibe las conexiones y almacena el identificador del *socket* en el módulo de nodos. Luego se configuran los parámetros de cada nodo virtual y del canal inalámbrico. Finalmente se agrega un evento por cada nodo virtual a la agenda del módulo de eventos. Este evento inicial es del tipo genérico indicando que el nodo se debe iniciar.

Una vez finalizada la fase inicial se procede a iniciar la simulación. El controlador recibe el primer evento a simular desde la agenda, que corresponderá a uno de los eventos de inicialización para uno de los nodos virtuales. Una vez recibido el evento, el controlador primero incrementa el tiempo de simulación a el instante de ejecución del evento y luego lo envía al nodo para ser ejecutado utilizando el mensaje de interfaz *event_execute*. El nodo virtual, dependiendo del *firmware* compilado, utilizará los mensajes *event_add* o *radio_transmit* para agregar eventos en la agenda del módulo de eventos. Terminada la ejecución del evento el nodo virtual finaliza enviando un mensaje *event_complete*. Al recibir este último mensaje, el controlador elimina el evento recién ejecutado de la agenda y procede a enviar el siguiente evento. El controlador continúa en el ciclo de recibir, procesar y enviar eventos desde la agenda hasta que se cumpla alguna de las siguientes condiciones:

1. La agenda del módulo de eventos se encuentra vacía. Esto indica que no quedan eventos por simular.
2. Que se alcance el tiempo de simulación previamente definido en el archivo de configuración.

En la fase de finalización de la simulación, el controlador envía a cada nodo virtual un último evento que corresponde al fin de la simulación. Recibido este evento, los nodos virtuales terminan su ejecución cerrando el *socket* TCP/IP. Una vez que esto ocurre en todos los nodos virtuales, el controlador termina su ejecución, y da por finalizada la simulación.

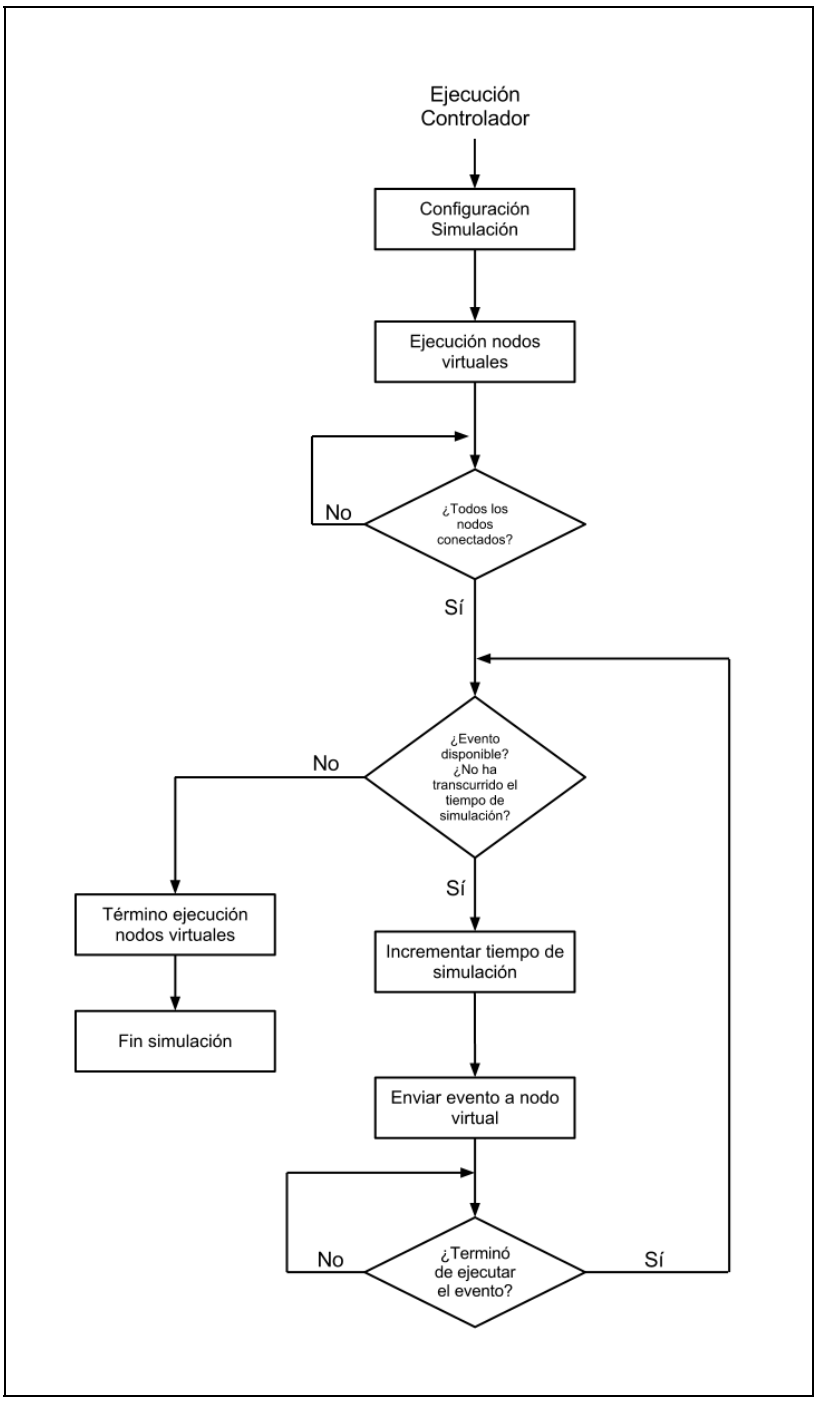


FIGURA 4.1. Algoritmo de flujo de la simulación.

Capítulo 5. SIMULACIÓN Y RESULTADOS

Para que el simulador tenga utilidad para el desarrollador de *firmwares* para redes inalámbricas de sensores, es necesario verificar que genera resultados realistas. Concretamente, los resultados obtenidos mediante las simulaciones deben coincidir con los resultados obtenidos de un experimento equivalente en terreno con nodos reales.

Con el fin de demostrar que se cumple la hipótesis propuesta en el Capítulo 1 de que es posible generar un ambiente de desarrollo de aplicaciones para redes inalámbricas de sensores en un mismo lenguaje de programación, se implementó el experimento de margen de cobertura (Sección 5.1) propuesto por (Barros, 2012). La Sección 5.2 describe los resultados obtenidos.

5.1. Descripción del Experimento Simulado

El experimento consiste en obtener curvas que representen las distancias de cobertura de nodos inalámbricos. Las curvas deseadas indican el porcentaje promedio de paquetes recibidos sin errores en función de la distancia entre nodos para distintas condiciones de propagación. La configuración del experimento consiste en disponer 10 nodos en línea recta, como se observa en la Figura 5.1. Por turnos, cada uno de los nodos transmite 500 paquetes de datos de tamaño predeterminado, los que son recibidos por todos los demás nodos. Debido a la distancia entre nodos y a las estadísticas del canal inalámbrico (Goldsmith, 2005), no todos los paquetes serán recibidos. Nodos más alejados detectan, en promedio, menos paquetes. Cada nodo almacena en su memoria local la cantidad de paquetes recibidos de cada nodo transmisor.

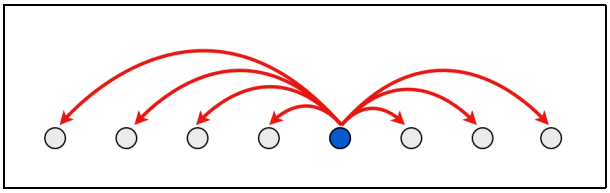


FIGURA 5.1. Distribución de los nodos en la simulación. Se ilustra el caso cuando el quinto nodo se encuentra transmitiendo.

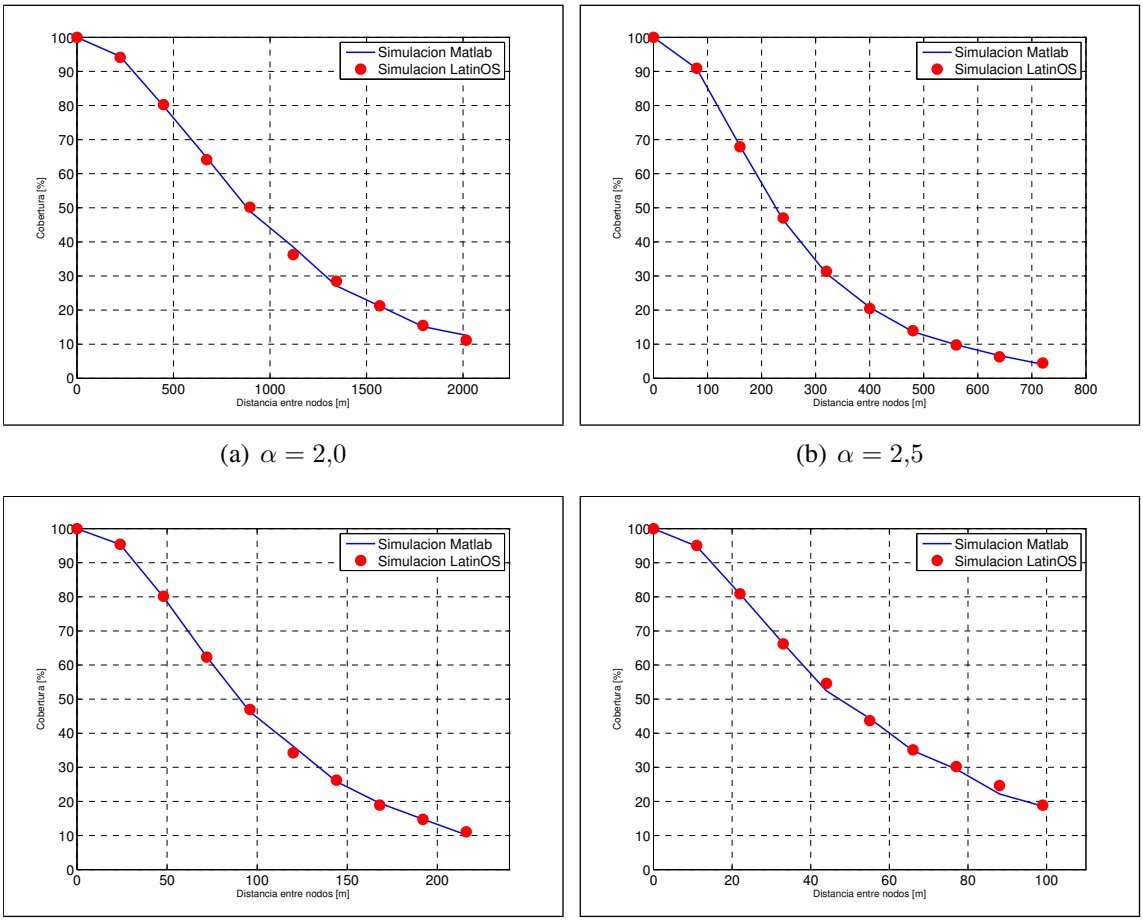
Los efectos de canal considerados en la simulación corresponden a los descritos en la Sección 4.1: pérdida por distancia, desvanecimiento de sombra y desvanecimiento local Rayleigh (Goldsmith, 2005).

Los parámetros de simulación relevantes son la frecuencia portadora (f_c), tamaño de la modulación digital (M) y ancho de banda (W) de la transmisión. En las simulaciones se utilizó $f_c = 2{,}4$ GHz, $M = 4$ (QPSK) y $W = 125$ kHz, las que corresponden a los utilizados por la radio CC2420 (*CC2420 Datasheet*, s.f.) de uso común en redes inalámbricas de sensores.

5.2. Resultados

Se realizaron cuatro simulaciones representando los casos en que α vale 2, $2{,}5$, $3{,}0$ y $3{,}5$. La Figura 5.2 presenta las curvas de cobertura para los casos recién nombrados. Cada gráfico incluye dos simulaciones: los marcadores corresponden a los valores obtenidos en la simulación con LatinOS, mientras que la línea contínua corresponde a una simulación simple realizada en Matlab con el mismo escenario de propagación.

La coincidencia entre Matlab y la simulación de LatinOS es importante ya que verifica que LatinOS, pese a tener una complejidad sustantivamente mayor que la simulación Matlab, entrega resultados que coinciden con lo esperado. El error de los resultados de LatinOS con respecto a la curva de Matlab es un $2{,}45\,\%$ en el peor de los casos.



(a) $\alpha = 2{,}0$ (b) $\alpha = 2{,}5$

(c) $\alpha = 3{,}0$ (d) $\alpha = 3{,}5$

FIGURA 5.2. Resultados de las simulaciones: Porcentaje de cobertura vs. distancia para varios valores del exponente de pérdida por distancia.

Capítulo 6. CONCLUSIONES Y TRABAJO FUTURO

6.1. Revisión de los Resultados y Comentarios Generales

En este trabajo se presenta una nueva plataforma de desarrollo de aplicaciones para redes inalámbricas de sensores, la que incluye un sistema operativo y un simulador. Las principales contribuciones de esta tesis son:

1. La utilización de un mismo lenguaje de programación tanto en el sistema operativo como en el simulador, y gracias a ello una propuesta de *software* que permite utilizar exactamente los mismos códigos tanto para programar los nodos como para realizar simulaciones. Ello evita la doble programación de los algoritmos y acelerar tiempos de desarrollo.
2. Las técnicas y lenguaje de programación utilizados son de uso común. Ello permite una rápida curva de aprendizaje para el desarrollador de aplicaciones para redes inalámbricas de sensores.
3. Tanto el sistema operativo LatinOS y el simulador están preparados para una incluir soporte para tecnologías de múltiples antenas, lo que permite una integración con dicha tecnología de manera sencilla en el futuro.

Se desarrollaron experimentos que comprueban el correcto funcionamiento de la plataforma de desarrollo y la validez del concepto propuesto.

6.2. Temas de Investigación Futura

Existen varios ámbitos para depurar y extender el trabajo aquí presentado. Primero, se pueden investigar mecanismos para actualizar el *firmware* de los nodos utilizando la misma red de sensores. Esta capacidad, conocida como *over the air programming (OTAP)* (Hagedorn y cols., 2008), permitiría evitar que por cada actualización de *firmware* se deba acceder físicamente a cada nodo, uno por uno, lo que a menudo no es viable.

Segundo, es posible incluir en LatinOS la capacidad de *multi-threading*. Esto significa que el microcontrolador podría realizar más de una tarea a la vez, o dejar en pausa tareas que se encuentran a la espera de nueva información, para procesar tareas que se encuentren listas para ser ejecutadas. Esto podría ayudar a mejorar el rendimiento energético del nodo. Cabe señalar que debido a las diferencias entre las arquitecturas de un microcontrolador y un computador de escritorio, se debe tener cuidado con la implementación de esta funcionalidad en el simulador.

Respecto al simulador, si bien la implementación actual utiliza *sockets* TCP/IP, la funcionalidad para distribuir la simulación entre varios computadores no se encuentra completamente desarrollada. Extendiendo dicha funcionalidad se podría implementar un sistema de simulación distribuido. Esto permitiría utilizar recursos de más de un computador para realizar la simulación, por lo que no sería necesario un computador muy potente para realizar simulaciones con un número elevado de nodos.

La sintaxis utilizada en los archivos de configuración de las simulaciones podría ser expandida para incluir más opciones para la simulación de distintos tipos de aplicaciones o nodos.

Finalmente, es posible optimizar el funcionamiento interno del simulador. En términos específicos de las comunicaciones inter-procesos, actualmente resuelto con *sockets* TCP/IP, podría evaluarse la utilización de un mecanismo de transporte basado en publicación - subscripción como Redis (*Redis*, s.f.).

Referencias

Abraham Silberschatz, P. B. G. (1998). *Operating System Concepts*. Addison-Wesley.

Astely, D., Dahlman, E., Furuskar, A., Jading, Y., Lindstrom, M., y Parkvall, S. (2009). LTE: the evolution of mobile broadband - [LTE part II: 3GPP release 8]. *IEEE Communications Magazine*, *47*(4), 44–51.

Barrenetxea, G., Ingelrest, F., Schaefer, G., Vetterli, M., Couach, O., y Parlange, M. (2008). SensorScope: Out-of-the-Box Environmental Monitoring. En *Information processing in sensor networks, 2008. ipsn '08. international conference on* (p. 332 -343).

Barros, S. d. (2012). *Diseño de Protocolos para Redes Inalámbricas de Sensores sobre LatinOS: Metodología y Ejemplos. Tesis de Magíster.*

Bhatti, S., Carlson, J., Dai, H., Deng, J., Rose, J., Sheth, A., y cols. (2005). MANTIS OS: an embedded multithreaded operating system for wireless micro sensor platforms. *Mob. Netw. Appl.*, *10*, 563–579.

BTNodes. (s.f.). http://www.btnode.ethz.ch/Documentation/BTnodeRev3HardwareReference. (Access December 7, 2011)

CC2420 Datasheet. (s.f.). http://www.ti.com/lit/gpn/cc2420. (Access April 13, 2012)

Day, J. (1995). The (un)revised OSI reference model. *SIGCOMM Comput. Commun. Rev.*, *25*, 39–55.

Dunkels, A., Gronvall, B., y Voigt, T. (2004). Contiki - A Lightweight and Flexible Operating System for Tiny Networked Sensors. En *Proceedings of the 29th annual ieee international conference on local computer networks* (pp. 455–462). Washington, DC, USA: IEEE Computer Society.

Eswaran, A., Rowe, A., y Rajkumar, R. (2005). Nano-RK: An Energy-Aware Resource-Centric RTOS for Sensor Networks. En *Proceedings of the 26th ieee international real-time systems symposium* (pp. 256–265). Washington, DC, USA: IEEE Computer Society.

Fraboulet, A., Chelius, G., y Fleury, E. (2007). Worldsens: development and prototyping tools for application specific wireless sensors networks. En *Proceedings of the 6th international conference on information processing in sensor networks* (pp. 176–185). New York, NY, USA: ACM.

Gay, D., Levis, P., Behren, R. von, Welsh, M., Brewer, E., y Culler, D. (2003). The nesC language: A holistic approach to networked embedded systems. En *Proceedings of the acm sigplan 2003 conference on programming language design and implementation* (pp. 1–11). New York, NY, USA: ACM.

Goldsmith, A. (2005). *Wireless communications*. Cambridge University Press.

Gough, B., y Stallman, R. (2004). *An introduction to GCC: for the GNU compilers gcc and g++*. Network Theory.

Hagedorn, A., Starobinski, D., y Trachtenberg, A. (2008). Rateless Deluge: Over-the-Air Programming of Wireless Sensor Networks Using Random Linear Codes. En *Proceedings of the 7th international conference on information processing in sensor networks* (pp. 457–466). Washington, DC, USA: IEEE Computer Society.

Hill, J., Szewczyk, R., Woo, A., Hollar, S., Culler, D., y Pister, K. (2000). System architecture directions for networked sensors. *ACM SIGARCH Computer Architecture News*, *28*(5), 93–104.

Huang, H., y Valenzuela, R. A. (2005). Fundamental Simulated Performance of Downlink Fixed Wireless Cellular Networks with Multiple Antennas. En (Vol. 1, pp. 161–165). Berlin.

Irmer, R., Mayer, H. p., Weber, A., Braun, V., Schmidt, M., Ohm, M., y cols. (2009). Multisite field trial for LTE and advanced concepts. *IEEE Communications Magazine*, *47*(2), 92–98.

Jakes, W. C. (1994). *Microwave Mobile Communications* (2nd ed.). Wiley-IEEE Press.

Karl, H., y Willig, A. (2005). *Protocols and Architectures for Wireless Sensor Networks*. John Wiley & Sons.

Kernighan, B., y Ritchie, D. (1988). *The C Programming Language*. Prentice Hall.

Krishnamachari, B. (2005). *Networking Wireless Sensors*. Cambridge University Press.

Kühn, V. (2006). *Wireless Communications over MIMO Channels: Applications to CDMA and Multiple Antenna Systems*. John Wiley & Sons.

Kuorilehto, M., Kohvakka, M., Suhonen, J., Hämäläinen, P., Hännikäinen, M., y Hämäläinen, T. D. (2007). *Ultra low energy wireless sensor networks in practice: Theory, realization and deployment*. John Wiley & Sons.

Labrador, M. A., y Wightman, P. M. (2009). *Topology control in wireless sensor networks: with a companion simulation tool for teaching and research*. Springer Science.

Levis, P., Lee, N., Welsh, M., y Culler, D. (2003). TOSSIM: accurate and scalable simulation of entire TinyOS applications. En *Proceedings of the 1st international conference on embedded networked sensor systems* (pp. 126–137). New York, NY, USA: ACM.

Levis, P., Madden, S., Gay, D., Polastre, J., Szewczyk, R., Woo, A., y cols. (2004). The emergence of networking abstractions and techniques in TinyOS. En *Proceedings of the 1st conference on symposium on networked systems design and implementation - volume 1* (pp. 1–1). Berkeley, CA, USA: USENIX Association.

Li, X. (2008). *Wireless Ad Hoc and Sensor Networks: Theory and Applications*. Cambridge University Press.

Li, Y., Thai, M. T., y Wu, W. (Eds.). (2008). *Wireless Sensor Networks and Applications*. Springer.

MEMSIC. (s.f.). http://www.memsic.com/products/wireless-sensor-networks/wireless-modules.html. (Access December 7, 2011)

Mogensen, P. E., Koivisto, T., Pedersen, K. I., Kovacs, I. Z., Raaf, B., Pajukoski, K., y cols. (2009). LTE-Advanced: The path towards gigabit/s in wireless mobile communications. En *Wireless communication, vehicular technology, information theory and aerospace & electronic systems technology, 2009. wireless VITAE 2009. 1st international conference on* (pp. 147–151). Aalborg.

MSP430F2618 Datasheet. (s.f.). 12500 TI Boulevard, Dallas, Texas 75243, USA.

mspgcc - GCC toolchain for MSP430. (s.f.). http://mspgcc.sourceforge.net/. (Access January 26, 2012)

The Network Simulator - NS-2. (s.f.). http://http://isi.edu/nsnam/ns/. (Accessed October 13, 2011)

OMNeT++ Network Simulation Framework. (s.f.). http://www.omnetpp.org/. (Accessed October 13, 2011)

Österlind, F. (2006). *A Sensor Network Simulator for the Contiki OS Abstract A Sensor Network Simulator for the Contiki OS*. Tesis Doctoral no publicada, Uppsala Universitet.

Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications — Amendment 4: Further Higher Data Rate Extension in the 2.4 GHz Band (n.º 802.11g-2003). (s.f.).

Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications Amendment 5: Enhancements for Higher Throughput (n.º 802.11n-2009). (s.f.).

Rappaport, T. (2001). *Wireless Communications: Principles and Practice* (2nd ed.). Upper Saddle River, NJ, USA: Prentice Hall PTR.

Redis. (s.f.). http://redis.io. (Access April 16, 2012)

Shnayder, V., Hempstead, M., Chen, B.-r., Allen, G. W., y Welsh, M. (2004). Simulating the power consumption of large-scale sensor network applications. En *Proceedings of the 2nd international conference on embedded networked sensor systems* (pp. 188–200). New York, NY, USA: ACM.

Sohraby, K., Minoli, D., y Znati, T. (2007). *Wireless Sensor Networks: Technolgy, Protocols, and Applications*. Wiley-Interscience.

Stevens, W., Fenner, B., y Rudoff, A. (2004). *UNIX Network Programming: The sockets networking API*. Addison-Wesley.

Tiny Node. (s.f.). http://www.tinynode.com/. (Access December 7, 2011)

Tse, D., y Viswanath, P. (2005). *Fundamentals of Wireless Communication*. Cambridge University Press.

Verdone, R., Dardari, D., Mazzini, G., y Conti, A. (2008). *Wireless Sensor and Actuator Networks: Technologies, Analysis and Design*. Academic Press.

Wittie, L. D. (s.f.). Microprocessors and microcomputers. En *Encyclopedia of computer science* (pp. 1161–1169). Chichester, UK: John Wiley and Sons Ltd.

Yacoub, M. D. (1993). *Foundations of Mobile Radio Engineering*. CRC Press.

Zhao, F., y Guibas, L. J. (2004). *Wireless sensor networks: an information processing approach*. Morgan Kaufmann.

Zolertia. (s.f.). http://zolertia.com/products/z1. (Access December 7, 2011)