Simulación de Redes de Sensores Wireless

Javier Navarro Fernández

Bellaterra, 10 de septiembre de 2010

Universitat Autònoma de Barcelona

etse Escola Tècnica Superior d'Enginyeria

El sotasignat, Dr. Diego Javier Mostaccio Mancini Professor/a de l'Escola Tècnica Superior d'Enginyeria de la UAB,

CERTIFICA:

Que el treball a què correspon aquesta memòria ha estat realitzat sota la seva direcció per en Javier Navarro Fernández

I per tal que consti firma la present.

Signat: Dr. Diego Javier Mostaccio Mancini

Bellaterra, 9 de septembre de 2010

Índice general

Índice de figuras

Índice de cuadros

A mis padres.

Agradecimientos

En primer lugar, doy las gracias a mis padres y hermanos por ser como son y por todo el apoyo y confianza que me han dado en los momentos más difíciles.

También me gustaría agradecer de forma muy especial a todos esos amigos que han estado tan cerca, los ánimos, apoyo y ayuda que me han prestado, ya que sin vosotros me habría resultado muy difícil, sino imposible, llegar a este punto. Quiero demostrar mi máxima gratitud a todas esas personas que han estado a mi lado cuando más lo he necesitado.

A todas las personas que me han acompañado en el transcurso de la carrera por esas horas de estudio y trabajo que han ayudado a superar con su compañía y por todo lo que me han enseñado en este tiempo.

Mi más sincera gratitud al Dr. Joaquin Garcia-Alfaro por dedicarme su tiempo, explicarme su trabajo, compartir conmigo su conocimiento e inquietudes y especialmente por su amistad.

Por último, quiero dar las gracias a mi director de proyecto, Dr. Diego Javier Mostaccio por toda la ayuda, paciencia y confianza prestada.

Capítulo 1

Introducción

1.1. Redes de sensores wireless

El interés creciente en los últimos años en redes de sensores *wireless* (WSN), en parte ha sido motivado por los recientes desarrollos en el campo de los sistemas micro-electrónico-mecánicos (MEMS) y tecnología de sistemas *wireless*.

En una red de sensores, millares de nodos son desplegados una región geográfica con el objetivo de llevar a cabo un proceso de monitorización u obtención de datos. Una vez desplegados, estos nodos deben trabajar conjuntamente con el objetivo de obtener sus coordenadas. Este proceso de localización es una etapa crucial para preservar la integridad de un red de sensores[9].

Por una parte, la fase de descubrimiento de coordenadas es una etapa necesaria para aplicar un protocolo de *routing* basado en localización y dirigir los datos obtenidos hacía un usuario final o estación base. Por otra parte, esta etapa también es importante como información necesaria para la aplicación final, de manera que esta sea capaz de determinar el origen de la información obtenida.

1.1.1. Aplicaciones

Las características de una WSN pueden ser utilizadas en distintos tipos de aplicaciones. A continuación veremos algunos ejemplos reales y/o posibilidades en varios campos.

1.1.1.1. Medio ambiente

Las redes de sensores tienen varias características que las hacen atractivas para aplicaciones de monitorización y seguimiento del medio ambiente[18][21]. Ejemplos del uso de redes de sensores son la detección de incendios forestales[20][8], detección de inundaciones y monitorización exhaustiva de zonas de riesgo, en otras.

Detección de incendios forestales En estos casos se trata de una red de sensores densamente poblada, con una gran cantidad de nodos distribuidos a lo largo de toda la zona que debe monitorizarse. Estos nodos suelen alimentarse mediante baterías que recargan utilizando la electricidad de origen solar utilizando paneles fotovoltaicos, lo cual permite a la red de sensores mantenerse activa durante un largo período de tiempo.

Los autores de [20] estudiaron la viabilidad del uso de sensores inalámbricos en las redes de vigilancia de incendios forestales. En los resultados experimentales se monitorizaron dos incendios controlados en San Francisco, California. El sistema se utilizado se compone de diez nodos MICA Mota con GPS, sensores de temperatura, humedad y datos de presión barométrica. Los datos son transmitidos a una estación base donde los registros se almacenan en una base de datos y proporcionan servicios diferentes aplicaciones.

En [19], los autores estudian el comportamiento del fuego en lugar de la detección de incendios. Presentan una red de sensores portátiles con el objetivo de medir las condiciones que rodean a los incendios activos. Cuando los nodos de detectan un fuego activo, estos envían los datos a la estación central. Generalmente, las necesidades de comunicación de este tipo de aplicaciones es en tiempo real, antes de que la propagación del fuego haga que la situación se vuelva incontrolable.

Detección de inundaciones Desarrollado como estándar en los años setenta por el *National Weather Service*, ha sido utilizado por numerosas agencias y organizaciones internacionales. ALERT[2] es acrónimo de *Automated Local Evaluation in Real Time*, como método que utiliza sensores remotos

que proporcionan importantes datos de campo en tiempo real y envían la información a una estación central.

Hay un gran número de tipos y fabricantes de hardware y software ALERT, pero todos han sido diseñados para cumplir con un conjunto común de criterios de las comunicaciones. Gracias a ello, la mayoría de los equipos y programas son intercambiables, lo que ha permitido la competencia para mejorar el rendimiento y reducir costes. Los sistemas ALERT se han convertido en un estándar en la recopilación de datos en tiempo real del medio ambiente debido a su precisión, fiabilidad y bajo coste

En la red ALERT aplicada a la detección de inundaciones, cada uno de los sensores está equipado con sensores meteorológicos, precipitaciones, nivel del agua, etc. Todo esta información necesaria para evaluar la posibilidad de inundaciones o detectarlas en tiempo real.

Monitorización en zonas/climas extremos La obtención de datos en zonas de difícil acceso es un campo con gran proyección en el campo de redes de sensores. Ejemplos podrían ser proyectos como *Glasweb*[35], donde la información proporcionada por sensores situados en la superficie de los glaciares y bajo ella es importante para entender la dinámica de los glaciares, así como el calentamiento global. O el proyecto *PermaSense*[40], un sistema *wireless* de detección en entornos de alta montaña de difícil acceso donde el calentamiento y descongelación del permafrost alpino en el lecho rocoso escarpado puede afectar la estabilidad de taludes.

1.1.1.2. Aplicaciones sociales y sanitarias

Un ejemplo claro es el proyecto CodeBlue, desarrollado en la Universidad de Harvard. En este caso se han implementado distintos tipos de sensores para la monitorización de parámetros vitales: tasa de latidos del corazón, concentración de oxígeno en sangre, datos EKG de electrocardiograma, etc. Toda esta información se recoge por los sensores y se distribuye de forma inalámbrica a una PDA u ordenador portátil para su procesamiento. De este modo, cualquier señal de alerta puede detectarse a distancia en tiempo real.

1.1.1.3. Aplicaciones militares

Históricamente, uno de los principales focos de interés de las redes de sensores se basan en aplicaciones militares. Algunas de las principales características de una red de sensores hacen de esta una opción especialmente atractiva en el ámbito militar. Características citadas como tolerancia a fallos y un sistema

auto-organización en una red robusta y de bajo coste, ofrecen la ventaja de que la destrucción de algunos nodos por acciones hostiles en el campo de batalla no comprometen de forma significativa las capacidades de la red. Además, la detección distribuida tiene la ventaja de poder proporcionar información redundante y por lo tanto muy fiable sobre las amenazas, así como la capacidad de localizar las amenazas mediante la correlación y transformación de la información recibida de un conjunto de nodos de sensores distribuidos.

Entre algunos de los ejemplos del uso de redes de sensores en aplicaciones militares podrían encontrarse los siguientes:

Monitorización de equipos Armamento, munición y estado del equipamiento en general. El reducido tamaño y una arquitectura abierta y distribuida, sin necesidad de desplegar una red dedicada, facilita incluir estos sensores en vehículos, tropas, equipamiento y suministros críticos Puede llevarse a cabo una monitorización en tiempo real del estado de los suministros y equipo, generando informes automático de la situación en el campo de batalla.

El proyecto *Advanced Soldier Sensor Information System and Technology (ASSIST)*[22] es un ejemplo que demuestra las capacidades de un sistema integrado que permite a los soldados en tierra recolectar, diseminar y visualizar información clave en el campo de batalla.

Vigilancia del enemigo Una red de sensores puede desplegarse con capacidades de observación muy diferenciadas: Algunos sensores que forman parte de la red pueden ser sensibles a ciertas características ambientales (humedad, temperatura,etc), otros en cambio pueden estar orientados a monitorizar pequeñas vibraciones sísmicas o detectar ciertos productos químicos. La aplicación de este tipo de redes no es sólo militar pero claramente gozan en este campo de atractivas cualidades: Facilidad de despliegue, camuflaje, dificultad de detección y posibilidad de llevar a cabo una identificación y seguimiento de tropas,vehículos militares o maquinaría pesada, así como en la detección de armas químicas y biológicas.

El número de aplicaciones es creciente día a día e incluyen muchos otros ámbitos como aplicaciones en el sector industrial, doméstico (casas inteligentes y *greenhouse monitoring*), sector de la agricultura[27], etc. La variedad de aplicaciones genera también un gran número de nuevos conceptos nuevos como por ejemplo las llamadas redes corporales o *body sensor networks (BSN)* [1].

1.2. Propósito del proyecto

La finalidad del proyecto será modelar los algoritmos de localización propuestos en [14, 23]. Se propone el desarrollo de un simulador o conjuntos de simulaciones como herramienta que permita evaluar los algoritmos propuestos. Debe permitir contrastar los resultados y evaluar las soluciones respecto a algunas de las métricas más relevantes en el área de redes de sensores [10]: Tiempo de vida útil, uso efectivo de los recursos disponibles, número de nodos necesarios para establecer y mantener rutas efectivas y consumo de energía, por ejemplo.

1.3. Líneas de actuación

Para llevar a cabo este proyecto, es necesario definir un conjunto de pasos que deben llevarse a cabo junto con las típicas fase de análisis, diseño e implementación.

- En primer lugar, un estudio genérico de algunos sistemas similares ya existentes desde el punto de vista de su funcionamiento y arquitectura, permitirá identificar los componentes básicos del sistema real. Será necesario comprender el modelo teórico en el que se fundamenta la infraestructura propuesta.

- En segundo lugar, es imprescindible que realizar una investigación sobre las posibles metodologías existentes que permitan experimentar con un modelo del sistema físico. En la figura 1.1 se observa un esquema del proceso asociado en la calificación, validación y verificación (*Qualification, Validation and Verification, QV&V*) y aplicado al modelado y simulación (*Model and Simulation, M&S*). La transición de un sistema físico a un sistema conceptual mediante un proceso de análisis y la aplicación de los métodos computacionales necesarios para obtener un modelo computacional [33].

- Por último, una fase de verificación del modelo. La necesidad de verificación del modelo computacional se justifica por el grado añadido de incerteza al pasar a un modelo discreto en espacio y tiempo.

1.4. Estructura de la memoria

A continuación, se realizará una breve descripción de los capítulos que conforman el resto de la memoria. Para ello, se seguirán las líneas de actuación

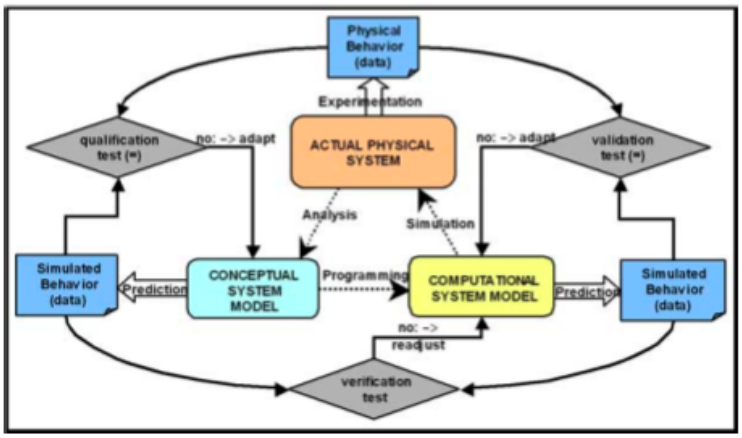


Figura 1.1: QV&V tradicional en el proceso de Modelado y Simulación

anteriormente propuestas adecuándolas a las exigencias de documentación y estructura de la memoria de un proyecto de fin de carrera.

En primer lugar se llevará a cabo una fase de documentación. El objetivo de esta primera fase es el de obtener los conocimientos necesarios para poder elaborar un estudio de viabilidad del proyecto y, posteriormente, abordar una fase de análisis del problema con ciertas garantías. El resto de los capítulos están enfocados a documentar el diseño, pruebas y conclusiones.

- **Capítulo 4.5: Estudio de viabilidad.** El objetivo final de este capítulo es concretar la viabilidad del proyecto. Esto permitirá decidir si finalmente, asumiendo unos riesgos controlados, debe iniciarse el proyecto o si debe desecharse la idea planteada evitando de esta manera un fracaso seguro.

  Se propone el análisis de un conjunto concreto de necesidades para proponer una solución que tenga en cuenta restricciones técnicas, legales y operativas. Para ello, se identifican los requisitos que se han de satisfacer y se estudia la situación actual. A partir del estado inicial, la situación actual y los requisitos planteados, se estudian las alternativas de solución, se valora su impacto y los riesgos asociados. Esta información se analiza con el fin de evaluar las distintas alternativas y seleccionar la más adecuada, definiendo y estableciendo su planificación.

- **Capítulo 3: Fundamentos teóricos.** Antes de tratar la fase de análisis es necesario plantear varias cuestiones que permitan abordar este capítulo. Es necesario un estudio más exhaustivo de los métodos y herramientas utilizadas con anterioridad en otros trabajos de simulación, tanto de forma general (herramientas de simulación), como posibles *frameworks* de

simulación especializados en redes de sensores *wireless*.

Además, se estudian varios proyectos más o menos conocidos que coinciden en algún punto, o en la mayoría de ellos, en los objetivos de este proyecto.

- **Capítulo 4: Análisis.** A partir de los conocimientos adquiridos en el capítulo anterior y los puntos descritos inicialmente en el capítulo 4.5, en este capítulo se presenta un análisis más detallado de las necesidades del sistema y se fijan con detalle los requerimientos En el último punto se describe la arquitectura básica utilizada como una aproximación a la arquitectura que se pretende desarrollar, dando paso a la fase de diseño e implementación de la simulación.

- **Capítulo 5: Diseño.** Este capítulo se ocupa de presentar el diseño de la aplicación. Esta presentación se abordará de forma gradual, basándose en las orientaciones obtenidas en el capítulo anterior, hasta llegar a obtener la estructura modular de la aplicación. Se mostrará con detalle la arquitectura que se ha dotado al sistema a desarrollado y una descripción de cada uno de los componentes que forman parte de ella.

- **Capítulo 6: Pruebas experimentación.** Se describen el conjunto de pruebas realizadas durante el desarrollo del proyecto que verifican su correcto funcionamiento.

- **Capítulo 7: Conclusiones.** Con este capítulo se concluye la memoria del proyecto. La principal motivación del capítulo es revisar el cumplimiento de los objetivos presentados al inicio de este trabajo, tratando de identificar que objetivos se han cumplido y que objetivos no se han cumplido. En este capítulo también se enumeran posibles líneas de continuación del proyecto.

Capítulo 2

Estudio de viabilidad

2.1. Presentación

Al principio de este capítulo se realizará un breve estado del arte en el apartado de modelado y simulación. Los siguientes puntos presentan la metodología de trabajo utilizada en el desarrollo del proyecto, objetivos principales y planificación.

2.2. Estado del arte

A continuación se citan algunos trabajos relacionados en de redes de sensores y algoritmos de localización. También se identifican los tipos principales de simuladores y se enumeran algunos ejemplos.

2.2.1. Trabajos relacionados

El problema relacionado con la localización en redes de sensores ha recibido en el pasado una atención considerable y recientemente varios autores han propuesto sistemas de localización especializados en redes de sensores. Los algoritmos de localización empleados varían enormemente. La elección de una u otra solución

depende de la información y *hardware* empleado por los nodos de la red al llevar a cabo el proceso de localización [43, 28, 32].

En [34] se utiliza el ángulo de llegada de la señal para estimar las distancias aplicando distintos métodos. En [28] se aplican algunos estos algoritmos utilizando *Positif* como *framework* de simulación. Otros autores proponen trabajos basados en la medida del tiempo empleado en recibir la señal (entre otros métodos) y utilizan un pequeño *testbed* de nodos para realizar la experimentación [44].

2.2.2. Tipos de simuladores

Un simulador es una herramienta que trata de imitar el comportamiento de un sistema. Es posible distinguir entre dos tipos de simuladores: Simuladores basados en tiempo contínuo y simuladores de eventos discretos. La mayoría de las herramientas de simulación disponibles son simuladores de eventos discretos.

En una simulación de tiempo continuo, el comportamiento del sistema está representado por ecuaciones diferenciales y la simulación consiste en la solución del sistema de ecuaciones. En una simulación de eventos discretos, el comportamiento del sistema se describe como una serie de eventos que deben ser tratados en un punto discreto del tiempo de simulación y que toman una cierta cantidad de tiempo real. Estos eventos son gestionados por un planificador de eventos que permiten, generalmente, simulaciones fuera de línea. No obstante, algunos simuladores integran planificadores de eventos en tiempo real que puede utilizar como entrada de datos el resultado de una aplicación o de una red real y ofrecer como salida unos datos que pueden ser enviados a una aplicación especializada o re-inyectados en la red real.

Las herramientas de simulación de eventos discretos generalmente están organizadas de la misma manera. Están compuestas de bibliotecas de modelos y un motor de simulación. Las librerías de modelos representan los modelos disponibles para describir el sistema que pretende simularse (por ejemplo, para representar elementos de red, enlaces, protocolos o aplicaciones). El motor de simulación ejecuta el escenario de simulación (topología y organización del sistema) y la conducta del escenario (descripción de lo que sucede, dónde y cuándo). Un simulador de estas características, generalmente, necesita como entrada de datos el escenario y un comportamiento como de entrada, generando como salida las trazas del resultado de la simulación.

Dependiendo del simulador utilizado, la descripción del escenario y el comportamiento del escenario puede llevarse a cabo desarrollando un *script* o un fichero de código específico o, de una manera más sencilla, utilizando una inter-

face gráfica de usuario (GUI). Otros simuladores proponen la directa generación del escenario utilizando algoritmos concretos u otra herramienta auxiliar.

2.2.3. Herramientas de simulación

Existen una gran número de simuladores orientados a la simulación de redes. A continuación se presentan algunas de las que se han considerado más relevantes para evaluar redes de sensores:

SensorSim [37], desarrollado a partir del simulador de redes NS-2 como extensión de esta herramienta, es un *framework* de simulación para redes de sensores. Proporciona modelos de canales de sensores, consumo de energía, una pila de protocolos ligeros de comunicación en micro-sensores wireless, creación de escenarios y simulación híbrida. El modelo de canales de sensores representa la interacción dinámica entre los sensores y el entorno físico. Cada uno de los sensores se integra un modelo de consumo de energía con varios estados, capaz de utilizar la energía de forma eficiente y prolongar la vida útil de los nodos . El simulador dejó de desarrollarse en el año 2001, en un estado de *pre-release* y carece de la documentación apropiada.

J-Sim (más conocido como *JavaSim*) [7], es un entorno de simulación desarrollado en Java, orientado a objetos y basado en componentes. Una de las principales ventajas *J-Sim* es su filosofía de componentes: Los módulos pueden ser añadidos y eliminados fácilmente y este aspecto suele ser interesante tanto para efectuar simulaciones de redes, como emulaciones, ya que permite incorporar uno o más dispositivos de sensores reales. *J-Sim* proporciona, además, soporte para distintos tipos de nodos, canales de sensores, modelos de comunicación *wireless*, medio físico, modelos de energía y potencia.

GloMoSim (Global Mobile Information system Simulate) [26], es un entorno de simulación escalable desarrollado en C y Parsec, capaz de paralelizar la ejecución de una simulación de eventos discretos. *GloMoSim* se compone de una colección de librerías de módulos, cada una de la cuales simula un protocolo de comunicación *wireless* específico en la pila de protocolos. *GloMoSim* es ampliamente utilizado para simular redes *Ad-hoc* y móviles de sensores *wireless*.

SENS (Sensor, Environment, and Network Simulator)[45] es un sensor de redes de alto nivel para aplicaciones de redes de sensores. Está formado básicamente por tres tipos de componentes intercambiables y extensibles: Com-

ponentes físicos, de red y de aplicación. Los componentes de aplicación se utiliza para simular la ejecución en un nodo sensor, los componentes de red están orientados a simular las funciones de envío y recepción de paquetes en el sensor y la capa física se utiliza para modelar los sensores, actuadores y modelos de potencia que interactúan con el entorno. Es posible modificar estos componentes o incluir nuevos componentes para nuevas aplicaciones, modelos de red, capacidad de sensores y entorno.

Ns-2 es una de las herramientas se simulación de redes más populares y ha sido ampliamente utilizado por la comunidad científica. Ns-2 es un simulador de eventos discretos orientado a objetos y una estructura modular permite la extensión de su funcionalidad. Las simulaciones se basan en una combinación de C++ y OTcl. En general, C++ se utiliza como lenguaje de programación para implementar protocolos y extender las librerías con nuevas funcionalidades. OTcl se utiliza para crear y controlar en entorno de simulación, incluyendo la selección de los datos obtenidos.

Ns-2 como simulador de redes de sensores, es una modificación de los módulos orientados a movilidad y redes *ad hoc* mediante un gran número de pequeñas extensiones. Ofrece soporte para algunas características peculiares en redes de sensores como hardware limitado, modelos de potencia e incluso ha sido desarrollado una extensión que permite que fenómenos del entorno de simulación actúen como *triggers* en la simulación.

OMNeT++ El principal objetivo en el desarrollo de OMNeT++ era proporcional una herramienta potente para la simulación de eventos discretos en el ámbito académico que permita modelar redes de comunicación, multiprocesadores y otros tipos de sistemas paralelos o distribuidos. OMNet++ es *open source*. Pretende cubrir el vacío entre herramientas de estilo *open source* y orientadas a la investigación (como Ns-2) y herramientas comerciales de alto coste como OPNeT (OPNET Technologies, Inc.).

OMNeT++ proporciona una estructura extensible, modular, orientada a componentes y jerárquica. Lo componentes y módulos están programados en C++ y los desarrolladores pueden ampliar la funcionalidad de las librerías del núcleo de simulación y utilidades proporcionadas para generación de números aleatorios, generación de estadísticas, topologías, etc. Se proporciona un lenguaje llamado NED (Network Description) utilizado para ensamblar componentes individuales en nuevos componentes y modelos.

Aunque pueden encontrase muchos modelos desarrollados sobre esta pla-

taforma, en especial citaremos *PAWiS* como *framework* de simulación de redes de sensores *wireless*. PAWiS se centra en el estudio detallado de los nodos que conforman una red de sensores, prestando especial atención a detalles como consumo de energía.

Existen otras numerosas opciones que no se han citado en este documento, igualmente conocidas y utilizadas como Opnet, Arvora, TOSSIM, Atemu, Sidh, EmStar.

2.3. Objetivos del proyecto

El propósito general del proyecto consiste en desarrollar un conjunto de simulaciones en el área de localización de redes distribuidas de sensores *wireless*. Pueden identificarse como objetivos principales del proyecto los siguientes puntos:

- Diseñar y desarrollar un método que facilite la generación de topologías de redes de sensores distribuidas y permita incorporarlas en las diferentes simulaciones. El objetivo principal de este punto es abstraer en lo posible el diseño de la simulación, de la arquitectura de la red. Debido a la diversidad de este tipo de redes en, por ejemplo, características de los nodos, distribución de nodos clave (*liars*, *anchors*, etc), aplicaciones y algoritmos de localización.

- Desarrollar un simulador para modelar los patrones de comportamiento de los distintos algoritmos aplicados. En [14] se proponen los modelos principales de adversario y algoritmos posibles que pueden emplearse.

- Diseñar un conjunto de pruebas que validen los resultados de este proyecto.

- Generalizar en lo posible la aplicación. Deber ser relativamente sencillo añadir nueva funcionalidad reutilizando el código desarrollado en la medida de lo posible. Se considera deseable la utilización de parámetros que permitan variar el comportamiento de la simulación como, por ejemplo, poder especificar el algoritmo de localización utilizado o el tipo de información intercambiada entre los nodos de la red.

- Diseño de una interface gráfica para el simulador que permita observar y depurar el comportamiento de los nodos de la red de forma visual. También resultará interesante proveer una opción en modo consola que facilite la ejecución de simulaciones en modo *batch*.

2.4. Plan de proyecto

El proyecto se estructurará siguiendo las clásicas fases definidas en el marco de los proyectos de ingeniería informática. Debido a la naturaleza del proyecto, el proyectista ha considerado necesario incluir en el plan de trabajo una metodología de investigación ligada al desarrollo natural del mismo. A continuación se presenta el método de trabajo utilizado en el proceso de estudio y desarrollo del proyecto.

2.4.1. Estimación de recursos

Los recursos necesarios para el desarrollo no son un factor decisivo para la viabilidad del proyecto ya que se encuentran localizados y disponibles. Como principales recursos de software necesarios:

- Máquina virtual (VM) como entorno estable de desarrollo, donde se instalará gran parte del software y evitará algunos problemas conocidos (como incompatibilidad entre diferentes versiones de software) y facilitará algunas acciones posteriores (difusión de la VM, generación de snapshots...)
- Simulador redes orientado a eventos o motor de simulación (Omnet++) y un framework especializado en el área de localización (PAWiS).
- Clásicas herramientas necesarias en el desarrollo de SW: IDE de desarrollo, Sistema de control de versiones (SVN) y sistema de documentación de código fuente DoXygen.

En el caso de recursos de *hardware* serán necesarios:

- Servidor de máquinas virtuales. Se ha elegido VMWare Server por ser uno de los más versátiles y utilizados, además de ofrecer un gran rendimiento.

2.4.2. Análisis de riesgos

Existen una serie de factores que deben tenerse en cuenta en todas las fase de desarrollo del proyecto ya que pueden ejercer una influencia negativa en la consecución de los objetivos finales.

En primer lugar, el área de interés está presente en una gran número de proyectos de investigación actuales. Es una área muy específica, especializada y compleja de la que el proyectista tenía un conocimiento nulo.

Como consideración final, existen un gran número de herramientas disponibles que permiten la simulación de redes de sensores centrándose en muchos

y diversos aspectos: Desde herramientas muy generalistas, hasta herramientas muy especializadas en un aspecto concreto. La elección de la herramienta apropiada a partir de la cual debe empezarse a trabajar y establecer el alcance de proyecto son factores decisivos.

2.4.3. Metodología de investigación

El proceso de investigación debe seguir un método estándar reconocido[41]. Se ha escogido un proceso secuencial de investigación representado en las fases que aparecen a continuación. Metodología de investigación secuencial:

- Revisar el campo de estudio. Intensa búsqueda bibliográfica
- Identificar correctamente los puntos clave del campo de estudio.
- Contrastar las diferentes soluciones teóricas
- Reflejar e integrar: Poner al día las ideas basadas en la pruebas realizadas y comunicar resultados

2.4.4. Metodología de desarrollo

La metodología Métrica Versión 3[11] ofrece a las Organizaciones un instrumento útil para la sistematización de las actividades que dan soporte al ciclo de vida del software dentro del marco que permite alcanzar los siguientes objetivos más importantes. Algunas de sus principales características son:

- Se define la estructura del proyecto (ciclo de vida) en procesos,
- Actividades y tareas.
- Basada en un enfoque orientado a proceso.
- Sigue el ciclo de vida principal o de cascada ("waterfall")

La nueva versión de métrica contempla el desarrollo de Sistemas de Información para las distintas tecnologías que actualmente están conviviendo y los aspectos de gestión que aseguran que un Proyecto cumple sus objetivos en términos de calidad, coste y plazos.

2.5. Planificación

Con el objetivo de organizar el trabajo, se propone dividir el ciclo de vida del proyecto en las siguientes fases:

1. **Estudio previo:** Debido a la naturaleza del proyecto, esta fase es especialmente necesaria. Durante esta fase se ha llevado a cabo todo el proceso de búsqueda bibliográfica, estudio preliminar, estado del arte y pruebas de las distintas soluciones. En esta fase también podría incluirse la parte dedicada al estudio de viabilidad.

2. **Análisis del proyecto:** Durante este período se estudia el problema que la aplicación debe solventar, se identificarán las partes implicadas, se presentan los rasgos generales de la solución adoptada y se realiza una primera aproximación que permita abordar la fase de diseño con éxito.

3. **Fase de diseño y codificación:** Fase en la que se decide como implementar y se implementa la solución del problema. En este fase se describirá arquitectura la arquitectura final del sistema y se desarrollarán los módulos de los que consta la aplicación.

4. **Pruebas:** Fase final del desarrollo que permitirá controlar la calidad del producto final y detectar los posibles errores de diseño que se hayan derivado de fases anteriores. Deberán realizarse pruebas de integración (errores de diseño), de sistema (verificación de requerimientos) y de aceptación.

2.5.1. Identificación de tareas

Una vez se han identificado los grandes bloques en los que se divide el proyecto y se ha identificado el método de trabajo que debe emplearse, se han subdividido cada uno de estos puntos en una conjunto de subtareas que permitan completar el desarrollo del proyecto, englobándolo dentro de las partes anteriormente citadas.

Puede consultarse una descripción detallada de las tareas en la tabla 2.1, donde se presenta un listado completo de las actividades e hitos del proyecto, con especial detalle en las tres primeras fases, así como una duración y fechas estimadas.

Para proporcionar una visión más completa, en la figura 2.1 se presenta la planificación temporal asignada a cada una de las tareas en forma de diagrama de Gantt.

2.6. Conclusiones

En esta primera fase del proyecto se han identificado los objetivos generales del proyecto y un breve estado del arte de varias de las herramientas disponibles en

el campo de la simulación, especialmente en redes y redes de sensores. En este punto es posible, mediante el estudio del conjunto de pruebas y comparativas, optar entre una de la alternativas disponibles. Finalmente, se ha establecido una metodología de trabajo y una planificación que permita continuar con el resto de fases de análisis, diseño e implementación.

Project stamp	
Project name	**PFC - Simulador Redes de Sensores Wireless**
Planned: Start-End	12/01/2009 - 09/02/2010 (136 days / 2 hours work)
Expected: Start-End	12/01/2009 - 09/02/2010 (136 days / 2 hours work done)
Objective	This is an template for a typical development project. It is prepared with some detailed phases where you can easily adopt your needed activities. Delete all phases, you don't need.

Merlin2

Describing project- and activity information

MS	Title	Group	Start	End	Work
◆	Kick-Off	PFC - Simulador Redes de Sensores Wireless	12/01/2009	12/01/2009	
	Estudio previo	PFC - Simulador Redes de Sensores Wireless	12/01/2009	01/23/2010	25 days
	Búsqueda bibliográfica inicial	Estudio previo	12/01/2009	01/15/2010	10 days
	Taxonomía de WSNs	Estudio previo	12/01/2009	12/23/2009	5 days
	Algoritmos de localización y routing	Estudio previo	12/16/2009	01/07/2010	5 days
	Estudio previo y pruebas de entornos de simulación	Estudio previo	01/01/2010	01/23/2010	5 days
	Documentation	PFC - Simulador Redes de Sensores Wireless	01/27/2010	09/02/2010	47 days
	Propuesta PFC	Documentation	01/27/2010	02/19/2010	10 days
◆	Estudio previo completo	PFC - Simulador Redes de Sensores Wireless	02/07/2010	02/07/2010	
	Estudio de Viabilidad	PFC - Simulador Redes de Sensores Wireless	02/07/2010	03/06/2010	10 days
	Estado del arte	Estudio de Viabilidad	02/07/2010	02/18/2010	5 days
	Análisis/tratamiento de datos	Estudio de Viabilidad	02/21/2010	02/27/2010	3 days
	Definición de objetivos del sistema	Estudio de Viabilidad	02/24/2010	02/26/2010	1 days
	Estudio de alternativas de solución	Estudio de Viabilidad	02/28/2010	03/04/2010	1 days
	Documentación código	Documentation	03/01/2010	08/31/2010	2 days
◆	Evaluación viabilidad del proyecto	PFC - Simulador Redes de Sensores Wireless	03/07/2010	03/07/2010	
	Análisis	PFC - Simulador Redes de Sensores Wireless	03/11/2010	03/25/2010	7 days
	Definición del sistema	Análisis	03/11/2010	03/19/2010	4 days
	Establecimiento de requisitos	Análisis	03/21/2010	03/25/2010	2 days
	Interfaces de usuario	Análisis	03/21/2010	03/23/2010	1 days
◆	Aprobación Análisis	PFC - Simulador Redes de Sensores Wireless	03/24/2010	03/24/2010	
	Diseño	PFC - Simulador Redes de Sensores Wireless	03/31/2010	04/23/2010	10 days
	Fases de diseño	Diseño	03/31/2010	04/23/2010	10 days
◆	Diseño completo	PFC - Simulador Redes de Sensores Wireless	05/01/2010	05/01/2010	
	Codificación	PFC - Simulador Redes de Sensores Wireless	05/02/2010	06/17/2010	20 days
	Tareas de codificación	Codificación	05/02/2010	06/17/2010	20 days
	Memoria Proyecto Final de Carrera	Documentation	06/01/2010	08/24/2010	30 days
	Pruebas de verificación	PFC - Simulador Redes de Sensores Wireless	06/16/2010	07/31/2010	17 days
	Descripción técnica del plan de pruebas	Pruebas de verificación	06/16/2010	06/26/2010	5 days
◆	Codificación completa	PFC - Simulador Redes de Sensores Wireless	06/16/2010	06/16/2010	
	Realización de pruebas	Pruebas de verificación	07/01/2010	07/17/2010	7 days
	Tratamiento de datos obtenidos	Pruebas de verificación	07/16/2010	07/27/2010	5 days
◆	Finalización de pruebas	PFC - Simulador Redes de Sensores Wireless	08/01/2010	08/01/2010	
	Presentación	Documentation	08/22/2010	09/02/2010	5 days
◆	Documentación completa	PFC - Simulador Redes de Sensores Wireless	09/01/2010	09/01/2010	

Cuadro 2.1: Tabla de actividades

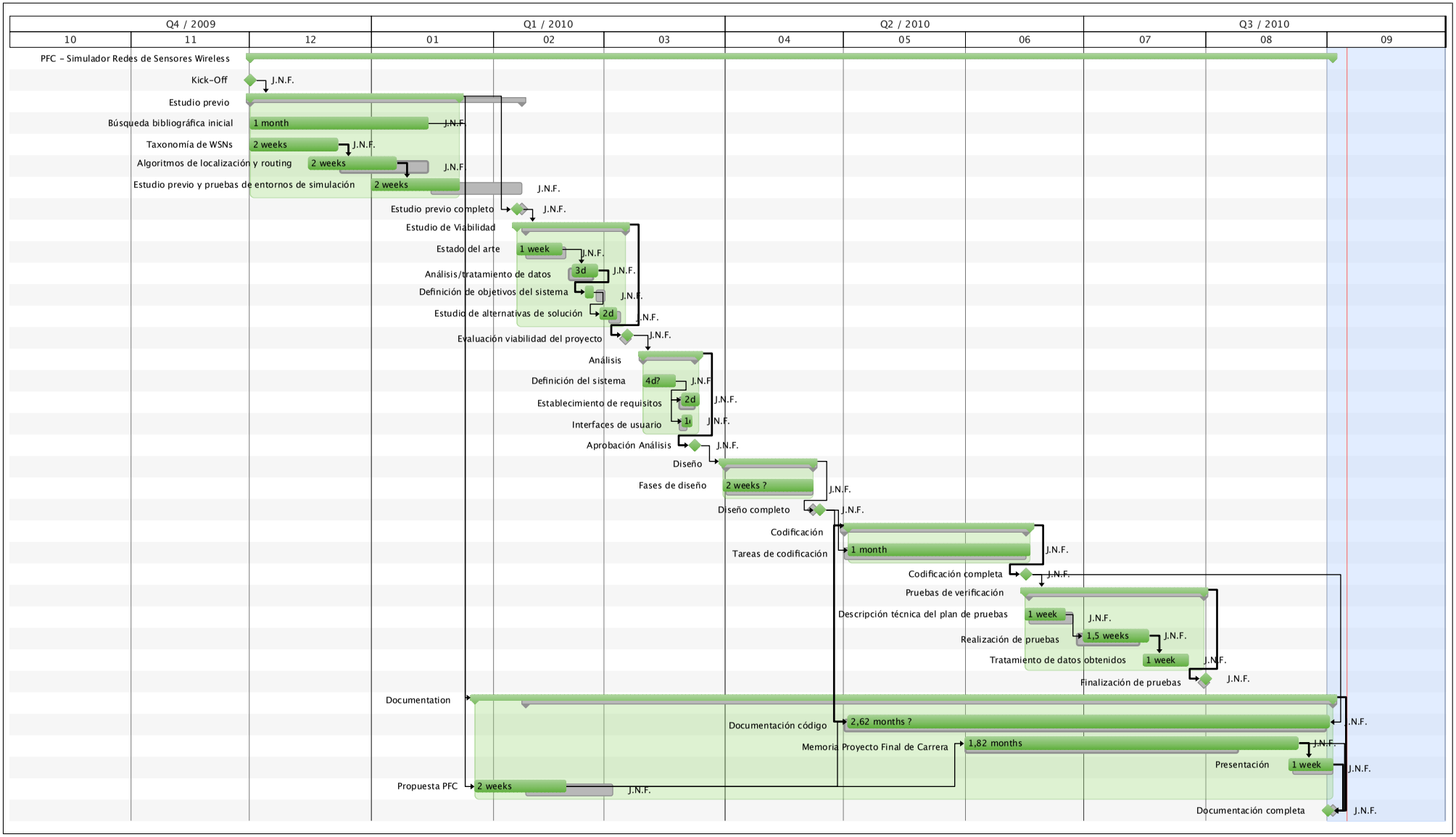


Figura 2.1: Diagrama de Gantt del proyecto

Capítulo 3

Fundamentos teóricos

3.1. Presentación

En este capítulo se tratan algunos conceptos clave para poder llevar a cabo una posterior etapa de evaluación, análisis y diseño. Por un lado, se define con detalle que es una red de sensores, sus principales aplicaciones, los elementos que la forman y como interactúan entre ellos. Se plantean ciertas restricciones y características diferenciadoras en este tipo de redes. Además se introducen algunos conceptos básicos para entender ciertas limitaciones y posteriores decisiones de diseño en el simulador.

3.2. Redes de sensores

Una red de sensores está formada por un número determinado de sensores. Estos pequeños nodos sensores son elementos de bajo coste y consumo, con unas capacidades de monitorización, procesamiento de datos y comunicación muy específicas y limitadas. Poseen la capacidad de auto-organizarse y comunicarse con otros nodos cercanos. En general, aunque no siempre es así, cada uno de los nodos juega dos papeles fundamentales en la red: Por una parte debe ocuparse

de retransmitir datos recibidos de otros vecinos y, además, utilizar su limitada capacidad de proceso para tratar sus propios datos y determinar los datos que debe transmitir.

Una red de sensores se basa en la idea de construir un entorno donde un gran número de pequeños elementos (sensores) colaboren para llevar a cabo una tarea, que generalmente consiste en tareas de monitorización de uno u otro tipo, y suelen están formadas por un gran número de sensores desplegados en el fenómeno o cerca de fenómeno que desea observarse.

Intuitivamente se pueden distinguir dos formas básicas de llevar a cabo este proceso: Por un lado puede llevarse a cabo un estudio de ingeniería cuidadoso en el momento de desplegar la red de sensores, de forma que la posición de los nodos está establecida en base a las condiciones más favorables que ofrece el terreno o, por el contrario, una gran cantidad de nodos pueden desplegarse alrededor del área de interés, estableciendo así un perímetro en el que la red puede ser efectiva.

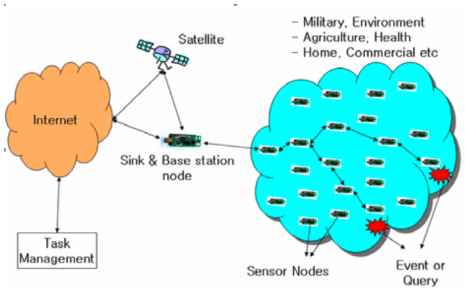


Figura 3.1: Despliegue de una red de sensores

En este último caso, la posición de los nodos no tiene porque estar preestablecida o ser conocida, lo que permite a los nodos ser desplegados de forma aleatoria en zonas de difícil acceso. Esto también significa que una red de sensores debe ser capaz de auto-organizarse, lo cual implica en última instancia que un nodo debe ser capaz de aplicar esta característica para comunicarse con sus nodos vecinos, delegando en ellos la responsabilidad de reenviar la información de forma que esta sea capaz de alcanzar su destino. En la arquitectura básica de una red de sensores (Fig.3.2) se pueden distinguir tres elementos principales:

Sensor field formado por la zona cubierta por el conjunto de sensores de la red

Task Manager donde pueden llevarse a cabo tareas de tratamiento/almacenamiento de datos o gestión activa de la red.

Sink o *gateway* como intermediario, que se encarga de enlazar la red de sensores con el resto.

Las redes de sensores representan una significativas diferencias ante las redes más tradicionales:

- El número de sensores en una red de sensores puede ser en varios órdenes de magnitud mayor que los nodos en una red de sensores *ad hoc*.
- Estos nodos son, en general, densamente desplegados en un área de interés. Esto también genera nuevos problemas como, por ejemplo, disminuir el radio de alcance para evitar colisiones.
- Limitaciones en potencia, energía, capacidad de cálculo y memoria. Límites en los recursos que limitan y perfilan los protocolos y algoritmos utilizados.
- Estos sensores están sujetos a fallos. No únicamente los posibles fallos provocados por el *hardware* en sí, también pueden ser provocados por elementos del entorno: Corrosión, altas temperaturas, etc.
- La topología de una red de sensores puede variar por motivos externos: Objetos móviles, obstáculos, etc.

3.2.1. Factores de diseño en redes de sensores

El diseño de este tipo de redes está influenciado por muchos factores, incluyendo aspectos como tolerancia a fallos, escalabilidad, costes de producción, entorno operativo, topología de la red de sensores, restricciones de *hardware*, medio de transmisión y consumo de energía.

Estos factores son importantes ya que sirven de guía para diseñar los distintos protocolos o algoritmos utilizados, adaptándose a una situación particular y son utilizados en muchos casos para comparar distintos esquemas propuestos.

3.2.1.1. Tolerancia a fallos

En una red de sensores desplegada en el terrero, algunos de los sensores pueden llegar a fallar por varios motivos: Fallo provocado por algún defecto o daño físico, falta o fallo de energía, interferencia del entorno, etc.

La tolerancia a fallos en una red de sensores es la capacidad de mantener la funcionalidad de la red sin interrupción, aunque fallen algunos de los nodos desplegados. La fiabilidad o tolerancia a fallos de un sensor es modelado en [17]

utilizando un distribución de *Poisson* para la calcular la probabilidad de tener un fallo en un intervalo de tiempo $(0, t)$.

$$R_k(t) = e^{-\lambda_k t} \tag{3.1}$$

Donde λ_k es la tasa de fallo de un sensor k en un período de tiempo t.

3.2.1.2. Escalabilidad

El número de sensores desplegados para estudiar un fenómeno puede ser del orden de cientos o miles y, dependiendo del tipo de aplicación, características del entorno u otros factores, el número puede alcanzar valores extremos incluso superiores. Los esquemas propuestos deben ser capaces de trabajar con este alto número de nodos y ser capaces de utilizar la alta densidad presente en las redes de sensores.

La densidad

$$\mu(R) = \frac{(N \cdot \pi R^2)}{A} \tag{3.2}$$

Donde N es el número de sensores dispersos en una región A y R es el radio de transmisión. Básicamente, $\mu(R)$ proporciona el número de nodos dentro del radio de transmisión de cada nodo en la región A.

3.2.1.3. Costes de producción

Dado que las redes de sensores están formadas por un elevado número de sensores, el coste de cada uno de los elementos que forman esta red (sensores) suele ser importante para justificar el coste total del sistema. Si el coste de una red de este tipo es más alto que desplegar una red de sensores tradicional, en muchos casos no justificaría su uso. Como resultado, el coste de un sensor debe mantenerse lo más bajo posible y esto tiene un impacto directo en varios aspectos como autonomía y *hardware*.

3.2.1.4. Restricciones de *hardware*

El mismo tipo de *hardware* utilizado impone ciertos límites, como pueden ser: Capacidad de procesamiento de datos y almacenamiento limitada, tamaño, energía disponible, arquitectura, modularidad, capacidad de comunicación, etc. Para conocer con más detalle el tipo de restricciones es necesario conocer con más profundidad el *hardware* empleado.

Detalles de la arquitectura se describen más adelante. Algunos ejemplos de *hardware* actualmente disponible en el anexo B.

3.2.1.5. Topología de la red de sensores

El despliegue de un alto número de nodos que pueden estar a pocos metros unos de otros y hace necesario un manejo cuidadoso en el mantenimiento de la topología. Pueden identificarse tres frases relacionadas en las cuestiones de mantenimiento y cambio en una red de sensores:

- **Fase de despliegue y preparación previa:** Los sensores pueden ser desplegados de múltiples formas en la zona de interés: De forma masiva, uno a uno. Pueden ser lanzados desde un aeroplano, artillería o depositados uno a uno por un ser humano, robot o cualquier otro medio.
- **Fase posterior al despliegue:** Posteriormente al despliegue, hay muchos factores que pueden provocar cambios en la topología: Interferencias, ruido, obstáculos móviles, terreno, etc. Una red de sensores puede estar sometida a frecuentes cambios después de la fase anterior.
- **Nuevas fases de despliegue:** Podría llegar a se necesario una nueva fase de despliegue por varios motivos como reemplazar los sensores que han fallado en la fase anterior por diversos motivos o por necesidades de cambio tareas dinámicas.

3.2.1.6. Entorno

Redes de este tipo generalmente se encontrarán desplegadas en áreas remotas y desatendidas, donde pueden estar sometidas a gran número de condiciones adversas: Contaminación biológica o química, campo de batalla, en un edificio, una fábrica, sometidos a una gran presión atmosférica o temperaturas extremas, entornos con una gran cantidad de ruido electromagnético, etc. Esta gran diversidad de escenarios y condiciones extremas donde pueden estar obligadas a trabajar, tiene una gran influencia que debe considerarse en aspectos como, por ejemplo, comunicación entre nodos y tasa de fallos.

3.2.1.7. Medio de transmisión

En una red de sensores que utiliza una comunicación *wireless* pueden establecerse links entre nodos mediante radio, sistemas ópticos o infrarrojos pero en el caso de quererse utilizar de forma global, este medio debe estar disponible de

forma internacional. Es muy común la utilización de bandas ISM (*Industrial, Scientific and Medical*).

En el caso de las restricciones de *hardware* aplicables a redes de sensores de bajo consumo y bajo coste, existe una gran número de componentes basados en la banda ISM de 433 y 915 MHz o circuitos RF operando en el rango de 2.4 GHz, así como es posible el uso combinado para llevar a cabo ciertas tares de localización, como en el proceso de estimación de distancias. El uso de una banda u otra tiene ciertas implicaciones en algunos fenómenos como atenuación, absorción y relación señal-ruido, entre otras. Se concreta con más detalle a este respecto en el anexo A.

3.2.1.8. Consumo de energía

Un nodo sensor *wireless* está generalmente equipado con una fuente limitada de energía. En muchos escenarios, la utilización de una fuente de energía alternativa es imposible, limitando la fuente de energía del nodo al uso de baterías o acumuladores. La vida útil del sensor queda limitada entonces al tiempo de vida útil de la batería.

Debido a la naturaleza de este tipo de redes, cada uno de sus elementos juega dos papeles principales dentro de ella: Generador (datos locales) y enrutador (datos externos). El fallo de alguno de ellos implica un cambio en la topología y obliga a la reorganización de la red, modificando las tablas de *routing* de los nodos, por ejemplo.

La conservación y gestión de energía adquiere una importancia adicional; motiva el empleo y desarrollo de nuevos protocolos y algoritmos. Las principales tareas de un nodo sensor son: Detectar eventos, realizar un mínimo tratamiento de datos local y finalmente transmisión y retransmisión de datos. El consumo de energía se distribuye entonces en tres dominios principales: Detección, comunicación y procesamiento de datos.

3.3. Protocolo de pila en WSN

La capa de protocolos utilizada en los nodos sensores (figura 3.3) permite algunas de las siguientes tareas: Gestión de la energía al utilizar diferentes protocolos de localización o enrutamiento, integración de los datos con los protocolos de red, gestión eficiente del medio inalámbrico y facilita la cooperación entre diferentes nodos sensores.

Módulo de gestión de energía Gestiona como debe utilizar el nodo la energía disponible; por ejemplo el nodo puede apagar su receptor después de re-

cibir un mensaje de sus vecinos, para evitar recibir mensajes duplicados, cambiar a un estado de baja energía, retransmitir a sus vecinos que no pueden participar en el enrutamiento reservando la energía restante para tareas de detección.

Módulo de gestión de tareas Balancea y planifica el conjunto de tareas que se llevan a cabo en el nodo, ya sea internas o externas, como tareas de monitorización llevadas a cabo en una región determinada.

Módulo de gestión de movilidad Detecta y registra el movimiento de los sensores o variación del entorno. Puede ser utilizada para mantener o como apoyo a un algoritmo de enrutamiento determinado, como mantener el estado de un conjunto de vecinos.

Capa física Utilizada en la selección de la frecuencia utilizada, generación de frecuencia portadora, detección de señales, modulación, cifrado, etc.

Capa de enlace de datos Responsable de la multiplexación de flujos de datos, detección de *frames* de datos, control de acceso al medio y control de errores, etc. Se utilizan varios tipos de protocolos MAC enfocados a una gestión eficiente de la energía como S-MAC, T-MAC, EARS, B-MAC.

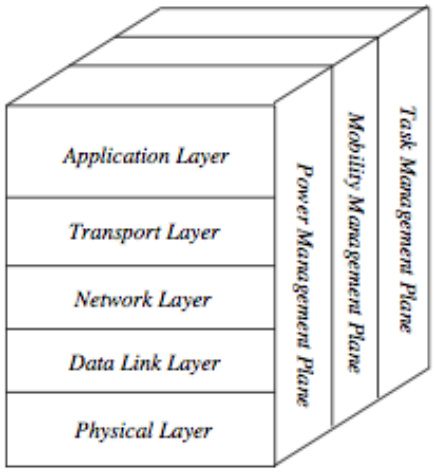


Figura 3.2: **Protocol Stack** en WSN

3.4. Arquitectura de un Nodo

El esquema básico de un nodo sensor consta básicamente de cuatro componentes clave: Unidad de detección (o sensores), unidad de proceso, transceptor y módulo de alimentación. Este esquema básico podría ampliarse dependiendo de la necesidad de las aplicaciones o la especialización del hardware, incluyendo módulos como un generador de energía, un sistema de movilidad o un sistema de localización o posicionamiento.

3.4.0.9. Componentes de un nodo sensor *wireless*

Los cuatro subsistemas básicos que componen el *hardware* de un nodo pueden apreciarse con detalle en la figura 3.3:

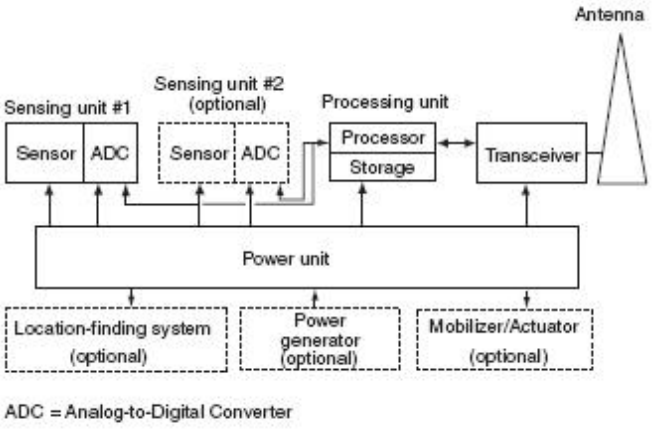


Figura 3.3: Diagrama de *hardware*

1. **Fuente de energía**. Un suministro de energía apropiado debe ser capaz de alimentar al nodo durante horas, meses o años, dependiendo de la aplicación. Como ejemplo de plataforma desatendida, suelen incorporar una alimentación mediante un sistema de alimentación autónomo (generalmente baterías), combinado con alguna fuente de recarga (células solares, p.e.) o energía auxiliar. Estas estrategias de recarga suelen dar buen resultado en operaciones donde no es posible llevar a cabo un reemplazo de la batería. Para optimizar el uso de energía, esta unidad puede soportar funciones avanzadas de gestión de energía como *dynamic voltage scaling.*

2. **Microcontrolador**. Proporciona la lógica computacional y de almacenamiento. Utilizado en tareas de procesamiento y manipulación de datos,

almacenamiento transitorio, cifrado, corrección de errores (FEC), modulación y transmisión digital. Los requisitos de cómputo y almacenamiento en una WSN dependen de la aplicación y pueden ir desde la utilización de un microcontrolador de 8 bits a un microprocesador de 64 bits. Los requerimientos de almacenamiento pueden igualmente oscilar entre 0,01 hasta 100 gigabytes (GB).

Un microcontrolador proporciona la integración necesaria para llevar a cabo las tareas de procesamiento y coordinación sin necesidad de hardware adicional. Además de la memoria proporcionada por el microcontrolador no es extraño encontrar modelos que incluyan memoria externa adicional, por ejemplo en forma de memoria *flash*.

3. **Unidad/es de detección o sensores**. Son el medio de comunicación entre el entorno, la red y el sensor.

   Un sensor es un dispositivo que mide alguna cantidad física y la convierte en una señal que se procesado por el microcontrolador. Entre la amplia gama de tipos de sensores existentes figuran los sensores de aceleración, sísmicos, humedad, iluminación, presión, sonido, térmicos, acústicos, visuales, infrarrojos y magnéticos. Los sensores pueden ser pasivos (sin sensor manipulación activa del medio ambiente) o activa (mediante la manipulación activa o sondeo del medio medio con un radar, p.e.), direccional u omnidireccional.

   La detección de una magnitud física como las descritas, generalmente son resultado la producción de una señal analógica continua, por esta razón, una unidad de detección se compone típicamente de una número de sensores y un conversor analógico digital (DAC) que digitaliza la señal.

4. **Transceptor**. Esta unidad permite la transmisión y recepción de datos a otros dispositivos, conectando al nodo en la red de sensores. Un nodo sensor *wireless* típico se comunica utilizado un sistema de RF (radio frecuencia) y algún tipo de tecnología PAN (Personal Area Network)

Un sensor típico está compuesto de cinco subsistemas principales de *software* (Fig. 3.4) :

1. **Sistema operativo (*middleware*)**. Este es el microcódigo utilizado por el todo el *software* de alto nivel en el nodo, proporcionando apoyo a diversas funciones básicas. Como cualquier otro sistema operativo, el objetivo será proteger y abstraer al *software* de la funcionalidad/complejidad del microprocesador a nivel puro de *hardware*.

Es conveniente disponer de sistemas operativos de código abierto diseñado específicamente para este tipo de aplicaciones; estos sistemas operativos suelen utilizar una arquitectura que permite la implementación rápida y reduciendo al mínimo el tamaño del código. TinyOS es un ejemplo de un sistema operativo de uso general.

2. ***Drivers***. Los sensores pueden estar diseñados utilizando un sistema modular o tipo *plug-in*, dependiendo de su sofisticación. Estos módulos de *software* específicos gestionan las funciones básicas del transceptor del sensor, gestión de módulos, la configuración adecuada y los ajustes utilizados en el sensor. Estos *drivers* proporcionan la abstracción necesaria de funcionalidad a nivel de máquina y dan soporte al sistema operativo utilizado.

3. **Procesadores de Comunicación**. Este código gestiona las funciones de comunicación, incluyendo el envío, almacenamiento en búfer de paquetes y transmisión, mantenimiento de topología, control de acceso medio (p.e. mecanismos de contención), cifrado, por citar algunos de ellos.

4. ***Drivers* de comunicación (codificación y capa física)**. Estos módulos de *software* de encargan de la gestión de aspectos menores de la capa del canal de transmisión de radio, incluyendo aspectos de sincronización, gestión de la frecuencia del reloj (*clocking*), codificación de la señal, recuperación de *bits*, niveles de señal y modulación.

5. **Mini-aplicaciones de procesamiento de datos**. Para llevar a cabo muchas de las tareas es necesario llevar a cabo numerosas operaciones numéricas, de procesamiento de datos, almacenamiento y manipulación de

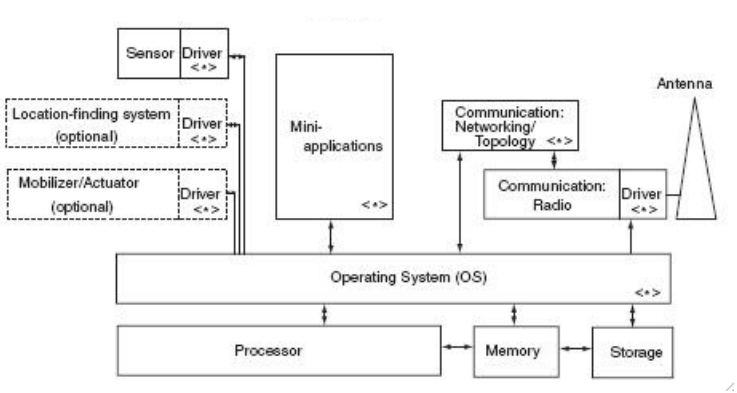


Figura 3.4: Diagrama de *software*

señales y otras operaciones básicas que sirven de apoyo a otros módulos a nivel de nodo.

3.5. Conclusiones

A lo largo del capítulo se han presentado varios aspectos clave que deben servir de base en las siguientes etapas de diseño y análisis: Grandes diferencias entre las aplicaciones de este tipo de redes, despliegue, topologías, detalle de algunos condicionantes y restricciones en el diseño, arquitectura y módulos principales; todos estos elementos dan una visión a varios niveles de profundidad, de los elementos que componen este tipo de redes y la lógica de funcionamiento que existe entre ellos externa e internamente.

Capítulo 4

Análisis

4.1. Presentación

A lo largo del capítulo se analiza los distintos métodos de localización y se presentan los algoritmos utilizados. Una vez cubierto el análisis de los aspectos claves, se propone una solución que permita alcanzar los objetivos principales. En el último punto se describe la arquitectura básica del *framework* de simulación empleado.

4.2. Localización en redes de sensores

En un proceso de localización distribuido, el conjunto de sensores desplegados trabajan de forma cooperativa para obtener los datos que luego permitirán confeccionar un mapa de red. Existen varios requisitos que influyen en el diseño de sistemas de localización en redes de sensores como son: Escalabilidad, eficiencia energética y precisión [38].

Generalmente, estos sistemas requieren el despliegue de un conjunto de nodos que por medio de algún método conocen su posición. La posición podría ser obtenida por cualquier nodo en la red empleando un método de posicionamiento

(como un sistema GPS) aunque este método resulta demasiado caro (coste, energía), sobretodo en redes con una alta densidad de nodos.

Una estrategia consiste un reducir el número de nodos que equipan este tipo de *hardware* estratégicamente, e implementar un protocolo que permita obtener la posición al resto de nodos de la red. En el proceso de localización llevado a cabo por un nodo sin sistema de posicionamiento descrito en [14, 23] pueden identificarse dos grandes fases:

Fase 1: Estimación de distancias. En esta primera fase se determinan las distancias entre los nodos que no conocen su posición y el resto. El objetivo es conseguir que el nodo que llevará a cabo el proceso de localización conozca la distancia a la que se encuentran el resto de vecinos. Dependiendo del tipo de método empleado en la siguiente fase podría ser necesario disponer, además de la distancia, de otro tipo de información (ángulo o posición).

Fase 2: Obtención de coordenadas, donde se aplica algunos de los algoritmos de localización (multilateración, triangulación, trilateración, etc.) y se lleva a cabo un proceso de corrección y refinamiento que permite obtener la posición de un nodo.

A continuación se presenta una descripción del proceso general centrando el detalle en cada una de las fases.

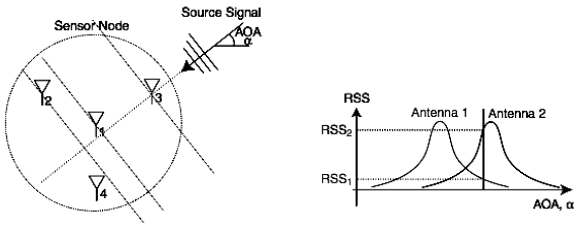
4.3. Estimación de distancias

Actualmente existen varios métodos que hacen posible calcular la distancia entre dos elementos de una red de sensores, de forma activa o pasiva, evaluando las distintas propiedades de la señal recibida/emitida entre ellos. Estos incluyen, aunque no son los únicos, los siguientes: Time Difference of Arrival (TDOA), Time of Arrival (ToA), Time of Flight (ToF), Received Signal Strength (RSS), Angle of Arrival (AoA).

4.3.1. Angle of Arrival (AoA)

Determinar el ángulo de llegada de la señal no proporciona información de la distancia entre nodos por sí mismo, pero obtener información sobre la dirección de un nodo vecino puede aportar importante información complementaria a los métodos ToA4.3.2 o RSS 4.3.3 ya que permite, bajo ciertas circunstancias, romper la ambigüedad al utilizar ciertos algoritmos de localización.

Existe dos formas sencillas utilizadas para llevar a cabo la medición AoA por un sensor. La más común consiste en utilizar un conjunto de sensores RF



(a) ToA/RSS (b) Ratio RSS_1/RSS_2

Figura 4.1: Métodos de estimación *Angle of Arrival (AoA)*

dispuestos de forma que la diferencia en el RSS o ToA de cada uno de ellos puede ser utilizado para determinar el origen (Fig.4.1a). Otra forma de llevar a cabo el proceso es la utilización de dos o más antenas direccionales apuntando a diferentes direcciones, de forma que la superposición de las ondas (Fig.4.1b) puede ser utilizada para determinar el AoA a partir del ratio RSS_1/RSS_2 de señales RSS individuales.

4.3.2. Time of Arrival (ToA)

TOA es el tiempo medido en el que una señal (RF, acústica o de algún otro tipo) tarde en llegar al receptor. El tiempo de medición TOA se compone del tiempo de transmisión más el tiempo necesario utilizado por la señal para propagarse por el medio.

El tiempo utilizado ($t_{i,j}$) entre la transmisión de la señal por parte de un sensor i y la recepción de la señal por un sensor j, será igual a la distancia $d_{i,j}$ entre el transmisor y el receptor, divida por la velocidad de propagación de la señal en el medio (v_p).

El aspecto más importante en el éxito en este tipo de técnicas es la habilidad por parte del receptor en estimar con precisión el tiempo de llegada de la señal. Esta estimación normalmente se ve obstaculizada por efectos como el ruido aditivo (*additive noise*) y *multipath*.

4.3.2.1. Ultra Wide Band (UWB)

El término *Ultra Wide Band (UWB)* se usa para hacer referencia a cualquier tecnología de radio que usa un ancho de banda mayor de 500 MHz o del 25 % de la frecuencia central, de acuerdo con la *FCC (Federal Communications Commission)*. En comunicaciones UWB (Fig.4.2) se emplean pulsos de corta duración, *Pulse Repetition Interval (PRI)*, para estimar el la distancia en el receptor mediante el *Time-of-Arrival (ToA)*.

Las ventajas que ofrece UWB son su bajo consumo, bajo coste aunque la principal ventaja de la modulación *UWB* reside en el hecho de que conduce a una resolución extremadamente fina por lo que resulta ideal en aplicaciones de radiolocalización de alta precisión. Además, el empleo de una frecuencia portadora muy baja permite una modulación directa y una buena penetración en gran número de materiales[13].

Algunas de las implementaciones en estimación de distancias basadas en UWB han demostrado unos errores RMS[1] (Probabilidad: 63/68 %)[12] comprendidos entre 0,12m. y 1,5m. En [15] describe con detalle el proceso de localización basado en UWB.

4.3.2.2. Time Difference of Arrival (TDOA)

En redes de sensores que utilizan relojes sincronizados puede determinarse el tiempo de retardo sustrayendo el tiempo de transmisión conocido al tiempo resultado de la medición TOA. La precisión en los algoritmos de sincronización utilizados y las diferentes velocidades de propagación, implica que este método resulte adecuado para ser utilizados en señales acústicas (menor velocidad de propagación) pero no para señales RF.

En redes de sensores asíncronas, una práctica común es utilizar una medición TOA de dos vías. Mediante este método un sensor transmite una señal al segundo sensor que inmediatamente responde con su propia señal, de forma que es posible calcular el tiempo de propagación sin necesidad de mantener ambos sensores sincronizados. Se nombra como TDOA a la diferencia entre los tiempos de llegada de la misma señal en dos sensores y una de las principales ventajas es que no depende de la sincronización o desplazamiento del reloj interno de nodo

[1] Root Mean Square (RMS).

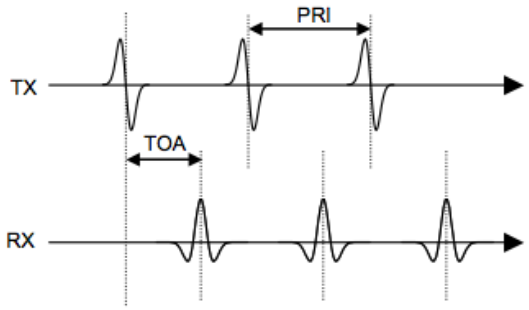


Figura 4.2: Utilización de pulsos en sistemas UWB.

transmisor.

4.3.3. Received signal Strength (RSS)

RSS se define como el voltaje medido por el receptor mediante un circuito que indica la potencia de señal recibida o *received strength indicator (RSSI)*. Utilizando la ecuación de Friis 4.1 [29], por ejemplo, es posible determinar la distancia del emisor.

$$S_r = S_s \left(\frac{\lambda}{4\pi d} \right)^2 \tag{4.1}$$

Donde S_s es la potencia de transmisión del emisor, S_r la potencia de recepción o *receive signal strength (RSS)* de la señal en el receptor, λ representa la longitud de onda de la señal y d la distancia entre emisor y receptor.

Llevar a cabo este tipo de medidas es relativamente barato y sencillo de implementar en el *hardware*. Además, si en una red de sensores *wireless* cada uno de los sensores se comunica con sus vecinos mediante señales de radiofrecuencia (RF), es posible llevar a cabo una medición RSS sin coste adicional de energía o ancho de banda.

Este método está lejos de ser perfecto al utilizarse en el proceso de estimación de distancias entre nodos vecinos, ya que el error en el proceso es demasiado alto para un gran número de aplicaciones. Las principales fuentes de errores pueden encontrarse en los efectos descritos en el anexo A y aunque existen técnicas que permiten minimizar los efectos de fading o shadowing, además de procesos de calibración y sincronización entre emisor-receptor, diversas irregularidades en el terreno (muros, vegetación,etc.) pueden provocar una atenuación de la señal imposible de predecir *a priori*.

Tal y como presentan en [38], la diferencia entre la potencia recibida en la medición por parte del receptor y su media es proporcional a su radio con un determinado factor multiplicativo constante. La conclusión es que los errores utilizando este método (RSS) son multiplicativos, en comparación a los errores aditivos que presenta el método TOA, lo cual implica que este método puede llegar a ser interesante en redes de sensores de muy alta densidad, con una distancia media entre vecinos muy escasa.

En numerosos trabajos aparece este último método *(AoA)* utilizado como apoyo a cualquiera de los anteriormente citados. La elección de utilizar uno u otro método depende de varios factores, aunque en la práctica es común encontrar combinaciones de los métodos presentados.

Según lo propuesto en [14][23] y debido a las características del *framework* de simulación, el método de estimación de distancias utilizado en la simulación será RSS. Este método encaja perfectamente con las características del modelo de entorno (atenuación, SNR, etc.) implementado en el simulador.

4.4. Obtención de coordenadas

El proceso de obtención de coordenadas puede llevarse a cabo empleando varios métodos. La elección de uno u otro suele depender del tipo de información obtenido de la red. Entre algunos de los que pueden aplicarse utilizando las técnicas detalladas en la sección anterior encontraremos los siguientes:

Multilateración es el proceso de posicionar un objeto calculando el *time difference of arrival (TDOA)* de una señal emitida desde este objeto a tres receptores (como mínimo) o posicionar a un receptor utilizando el TDOA de tres o más emisores sincronizados.

Triangulación como el proceso de determinar la posición de un punto midiendo los ángulos (y no distancias) de este respecto a unos puntos conocidos. Este punto puede obtenerse como el tercer punto de un triángulo del cual se conocen dos ángulos y uno de los lados.

Trilateración como método para determinar la intersección de tres esferas de radio y centros conocidos. Utilizando la geometría de las esferas (3D) o triángulos (2D), este método implica la obtención de una posición relativa o absoluta una vez determinada la distancia a la que se encuentran los centros de la esferas o circunferencias.

Asumiendo que el proceso de posicionamiento se basa en trilateración, puede considerarse un punto $A = (a_x, a_y)$, tal que $(a_x, a_y) = \mathcal{F}(B_1, B_2, B_3)$ para cualquier de los tres puntos B_1, B_2, B_3 y donde la función $\mathcal{F}$ retorna el punto obtenido de la intersección de tres círculos con centro B_1, B_2, B_3 y radio $d(A, B_1), d(A, B_2)$, y $d(A, B_3)$, respectivamente (Figura 4.3). $\mathcal{F}(B_1, B_2, B_3)$ es un punto único cuando los puntos A, B_1, B_2, B_3 están situados en posiciones generales (no colineales).

Si estos puntos son sensores, $\mathcal{F}$ es calculada por el sensor A cuando recibe las coordenadas $B_1 = (b_{1x}, b_{xy}), B_2 = (b_{2x}, b_{2y}), B_3 = (b_{3x}, b_{3y})$ y es capaz de determinar las distancias $d(A, B_1), d(A, B_2), d(A, B_3)$ utilizando algunas de las técnicas descritas (p.e. RSS). Finalmente, la posición $A = (a_x, a_y)$ es obtenida como única solución al siguiente sistema de ecuaciones:

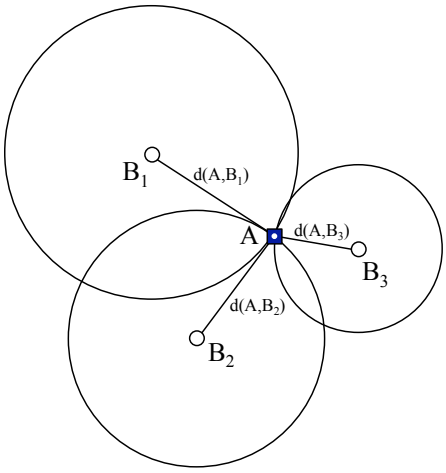


Figura 4.3: El Sensor A intenta determinar su posición procesando las señales de radiolocalización recibidas de los nodos vecinos B_1, B_2, y B_3 que se encuentran dentro de su radio de alcance (un *hop*). [23].

$$(b_{1x} - a_x)^2 + (b_{1y} - a_y)^2 = d(A, B_1)^2 \quad (4.2)$$

$$(b_{2x} - a_x)^2 + (b_{2y} - a_y)^2 = d(A, B_2)^2 \quad (4.3)$$

$$(b_{3x} - a_x)^2 + (b_{3y} - a_y)^2 = d(A, B_3)^2. \quad (4.4)$$

En los siguientes apartados se plantea como el sensor A podría llegar a recibir información errónea del entorno, proporcionada por algunos nodos maliciosos de la red. Estos nodos mentirán acerca de la distancia y posición. El sensor A podrá detectar la presencia de estos nodos, informar acerca de ellos y aun así podrá llegar a determinar su posición.

4.4.1. Restricciones aplicadas

En [23] se define como nodo mentiroso (*liar*) a cualquier nodo que proporcione al resto de nodos de la red algún tipo de información errónea, ya sea distancia o coordenadas. Esa información errónea puede ser enviada de forma intencionada con el objetivo de intentar que el resto de nodos no sea capaz de encontrar su posición. También puede ser un efecto no deseado producido por obstáculos u otras circunstancias físicas de un medio no-ideal.

Se utiliza un espacio de dos dimensiones y distancias euclidianas sin errores en la estimación. En estas condiciones, dadas dos posiciones (x, y),(x', y'), un

nodo puede determinar sin son o no son iguales, rechazando una de las dos. Además se aplican las siguientes restricciones:

- La comunicación es bidireccional.
- Idéntica potencia de emisión en todos los nodos en la red, excepto en el caso de nodos *liar*.
- Suficiente densidad de nodos en la red.
- La identidad de un nodo en la red es única (*NodeId*).
- Los nodos se encuentran situados en posiciones generales, por ejemplo, no colineales.

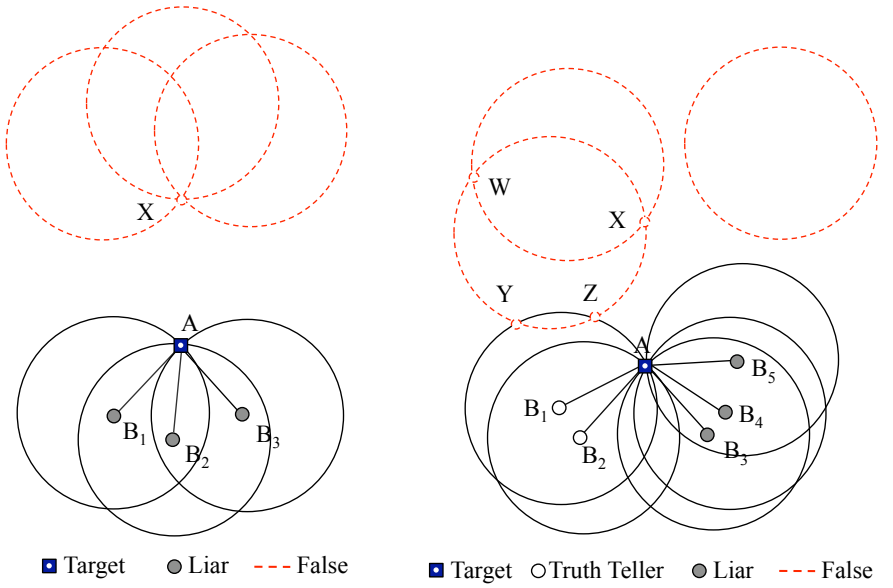
4.4.2. Modelos de adversario

Las capacidades de un adversario o conjuntos de adversarios en la red quedan definidas como:

- Capacidad de analizar las comunicaciones entre un objetivo A y uno o varios nodos *Anchor* (*truth teller*). De esta manera se puede construir una posición errónea consistente para uno o varios nodos.
- Capacidad de un adversario de proporcionar una distancia errónea...
- ...o mentir acerca de su posición.
- Construcción de un canal de colaboración donde dos o más adversarios intercambian la información necesaria para proporcionar la información errónea al nodo víctima.

La descripción detallada de los modelos de adversarios y algoritmos de localización empleados pueden encontrarse en [14][23]. De los posibles modelos de adversario presentados en el artículo, los modelos 3 y 4 son los más sencillos de implementar: El modelo 4 (Fig 4.4b) consistirá simplemente en un nodo que, posiblemente de forma no intencionada, envían una información errónea en la posición, distancia o ambas. La aplicación del modelo 3 queda representado en la Figura 4.4a e incluye la capacidad de los nodos de colaborar en el proceso de proporcionar las posición al resto de nodos de la red.

Un proceso de inicialización erróneo del sistema de posicionamiento de los nodos determinados como *liar* será suficiente para simular un Modelo de adversario 4. Los demás modelos se proponen como posible línea futura ya que



(a) Three liars tamper their positions and distances. They collude and force A to conclude that it is located at X.

(b) Three liars tamper their positions and distances. They do not collude.

Figura 4.4: Ejemplos de los Modelos de Adversario 3 y 4.

necesitarían implementar un medio de colaboración entre nodos *liar*. El desarrollo de la lógica necesaria para simular los modelos 1 y 2 necesita de una complejidad añadida ya que, previamente a realizar el ataque, sería necesario realizar tareas de monitorización del tráfico generado por los nodos de la red.

4.4.3. Algoritmos aplicados

En la simulación se modela el proceso de localización empleando los algoritmos 1, 2 y 3 descritos por el autor. Estos permiten resolver el problema de determinar la posición de un nodo ante los modelos de adversario nombrados anteriormente, siempre que se respeten los límites y restricciones descritos con detalle en el trabado. En futuras implementaciones podría definirme un modelo que permita definir nodos de confianza (*trusted nodes*) en la red y permitirá llevar a cabo pruebas con los algoritmos 4 y 5, como se especifica en el artículo.

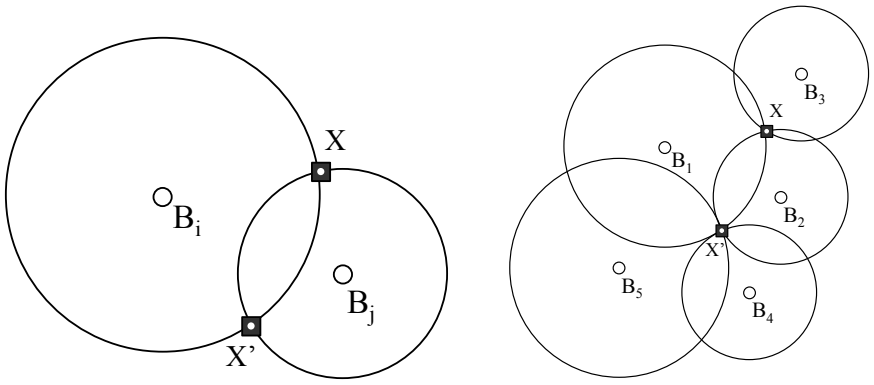
Los siguientes puntos reproducen los algoritmos originales junto con las características más remarcables.

4.4.3.1. Algoritmo 1: *Majority-ThreeNeighborSignals*

En el primer caso es necesario obtener la posición correcta de tres nodos de manera que sea posible resolver la posición A aplicando un proceso de trilateración para cada una de las tripletas de nodos vecinos.

Algorithm 1 Majority-ThreeNeighborSignals

1: Sensor A requests the location of its neighbors.
2: Every sensor in $N_1(A)$ sends its location to A.
3: For each triple t of neighbors $B_i, B_j, B_k \in N_1(A)$, A computes (x_t, y_t).
// (x_t, y_t) is the point of intersection of the three circles
// centered at B_i, B_j, B_k and with radii $d(A, B_i)$,
// $d(A, B_j)$, and $d(A, B_k)$.
4: A accepts the majority as its location, and reports the nodes lying about the resulting position.
// if there is no consensus, then A aborts the process,
// and declares that it fails compute its location.



(a) Sensor A aplicando el algoritmo. Se obtienen dos posibles posiciones.

(b) Resolución de la ambigüedad en aplicación del algoritmo

Figura 4.5: Ejemplos de la aplicación del Algoritmo 2.

4.4.3.2. Algoritmo 2: *Majority-TwoNeighborSignals*

Este algoritmo describe el proceso en que un sensor A utiliza parejas de dos nodos vecinos para encontrar su posición. La posición correcta de este sensor será uno de los los dos puntos de intersección $\{X, X'\}$ de dos círculos centrados en los puntos B_i, B_j con distancias $d(A, B_i)$, $d(A, B_j)$, respectivamente (Figura 4.5a). En la Figura 4.5b se determina la posición del nodo en un entorno con

Algorithm 2 Majority-TwoNeighborSignals

1: Sensor A requests the location of its neighbors.
2: Every sensor neighbor of A sends its location to A.
3: For each pair p of neighbors $B_i, B_j \in N_1(A)$, A computes $(x_p, y_p), (x'_p, y'_p)$.
// The locations computed are the two points of
// intersection of the two circles centered at B_i, B_j
// with radii $d(A, B_i)$ and $d(A, B_j)$, respectively.
4: A calculates the frequencies of occurrence of each position and accepts the position that has majority. It reports the nodes lying about the resulting position.
// If there is no consensus, then A aborts the process, and
// declares that it fails to compute its location.

un número de nodos vecinos de $n = 5$.

4.4.3.3. Algoritmo 3: *MostFrequent-TwoNeighborSignals*

En este algoritmo se utiliza un método alternativo que permite obtener a un nodo no posicionado A encontrar su posición correcta respetando las restricciones entre el número de nodos y número de nodos *liar*. De forma similar al caso anterior, calculará los puntos de intersección de todas las parejas. En este caso se elegirá como posición el punto con frecuencia de aparición más alta.

Algorithm 3 MostFrequent-TwoNeighborSignals

1: Sensor A requests the location of its neighbors.
2: Every sensor neighbor of A sends its location to A.
3: For each pair p of neighbors $B_i, B_j \in N_1(A)$, A computes $(x_p, y_p), (x'_p, y'_p)$.
// The locations computed are the two points of
// intersection of the two circles centered at B_i, B_j
// with radii $d(A, B_i)$ and $d(A, B_j)$, respectively.
4: A calculates the frequencies of occurrence of each position, accepts as correct the most frequently occurring value, and reports the nodes lying about it.
// If there is no any position whose frequency of
// occurrence is, at least, twice the frequency of
// occurrence of the second most frequent position,
// then A aborts the process, and declares failure to
// compute its location.

4.5. Valoración de alternativas

En la capítulo se han presentado algunos conceptos y herramientas concretas en el área de simulación de redes y redes de sensores. En una primera valoración de las alternativas posibles de solución al problema que plantea el proyecto pueden distinguirse las siguientes:

- Programar un motor de simulación desde cero, completamente adaptado al problema teniendo en cuenta todas las necesidades del proyecto.
- Utilizar un motor de simulación de redes de carácter general y adaptarlo al propósito del proyecto.
- Utilizar y adaptar alguno de los simuladores especializados en el área de la simulación de redes distribuidas de sensores o emulador de algunos de los modelos *hardware* que se adecuen al problema.

La elección de una u otra solución es un factor de riesgo en el desarrollo de proyecto y existen ciertos factores de decisiónque deben evaluarse cuidadosamente antes de optar por alguna de las opciones propuestas:

- Nivel de detalle. La herramienta utilizada, indiscutiblemente debe tener la potencia necesaria para expresar el modelo que debe simularse.
- Modelos disponibles. El número de protocolos y modelos disponibles en la herramienta de simulación, la capacidad de extender y generar nuevos módulos son cuestiones de relevancia si debe adaptarse una herramienta a un modelo concreto.
- Definición de la topología de red. Como se define la topología de la red y las posibilidades que ofrece la herramienta para hacerlo.
- Modelo de programación utilizado por el entorno de simulación. En este tipo de herramientas se utilizan básicamente dos modelos: Funciones de parseo de mensajes o hilos de ejecución. Cada una ofrece sus ventajas y desventajas en cuestiones críticas como, por ejemplo, escalabilidad.
- Existen otros aspectos igualmente importantes como rendimiento, disponibilidad de código fuente, documentación, herramientas de depuración, etc.

4.5.1. Solución adoptada

Una vez analizado con detalle el modelo físico, teórico y presentado el modelo conceptual, debe elegirse la solución que emplee los métodos computacionales adecuados que permitan obtener un modelo computacional.

Teniendo en cuenta lo expuesto en todas las secciones anteriores, se considera como la elección más apropiada el uso de *OmNet++* como motor de simulación orientado a eventos y *PAWiS* como *framework* de simulación especializado en el área de redes de sensores *wireless*.

4.5.2. Especificación de requerimientos

En este apartado se citan las características que se deben tenerse en cuenta en el desarrollo del proyecto: Necesidades específicas del simulador, la funcionalidad y diversas cuestiones útiles en el diseño de la aplicación. A continuación se efectúa una descripción de los requerimientos identificados.

1. Una red de sensores estará formada esencialmente por dos tipos de nodos, tal y como se cita en [14]:
   a) Un nodo equipado con un módulo GPS similar al modelo cuyas características pueden apreciarse en el anexo B.
   b) Nodos sin *hardware* de posicionamiento, dedicados a efectuar tareas de monitorización y que previamente llevarán a cabo una fase de localización como parte de su proceso de *start up*.
2. Ambos tipos de nodos en el futuro efectuarán tareas de enrutamiento y retransmisión de paquetes. Debe tenerse en cuenta en el diseño.
3. En la capa de red se implementarán las diferentes tareas necesarias de localización.
4. En la capa de localización un nodo podrá llegar a tener la siguiente información:
   a) Para llevar a cabo el proceso de lateración es necesario: Un lista de nodos *anchor* detectados, junto su posición y distancia estimada.
   b) Nodos vecinos locales, es decir, el conjunto de nodos dentro del alcance que son posibles de descubrir monitorizando el medio de forma pasiva.
   c) Opcionalmente, una lista de nodos vecinos detectados por los nodos colindantes (nodos vecinos descubiertos por los vecinos).

5. En la capa de localización puede especificarse el algoritmo de lateración que se utilizará en la fase de cálculo de la posición. En un principio es conveniente implementar un algoritmo sencillo de trilateración pero debe considerarse la posibilidad de incluir más métodos en un futuro. Estos algoritmos utilizarán la información disponible en la capa de localización.

6. Se dispondrá de dos tipos de visualización: Una interface gráfica que permita y una vista en modo texto para llevar acabo las simulaciones.

4.6. PAWiS Simulation Framework

El *framework* de simulación PAWiS (*Power Aware Wireless Sensors*) facilita el diseño y la simulación de modelos de redes de sensores *wireless*. Los objetivos principales de este *framework* están orientados a cubrir la simulación en la comunicación internodo e intranodo. También provee una librería de módulos (`ModLib`) que proporciona varias implementaciones de algunos módulos, como punto de partida al desarrollo de una simulación completa.

4.6.1. Estructura

La estructura básica del simulador puede observarse en la Figura 4.6. El *framework* está basado en el sistema de simulación de eventos discretos OMNeT++ y desarrollado utilizando el lenguaje de programación C++. Generalmente, los modelos del programador deberían interactuar con el *framework* de simulación (utilizando C++).

En un simulador de eventos discretos básicamente cada módulo puede generar un nuevo *evento* que es almacenado en una lista llamada *Future Event List (FEL)*. El kernel de simulación tratará continuamente los eventos más cercanos en el tiempo que se encuentren en la FEL, utilizando un llamado "tiempo de simulación" gestionado por el propio núcleo de simulación.

Para implementar los módulos del nodo, el *framework* de simulación PAWiS utiliza el modelo orientado a objetos mediante clases C++ y llamadas a interfaces funcionales (llamadas a procedimientos remotos, p.e.) para las interfaces entre módulos. Estas interfaces funcionales son utilizadas para implementar las comunicaciones en las capas de red, CLAMP y otros bloques (capa de seguridad, gestión del nodo, etc). Cada llamada es gestionada por el motor de simulación de eventos discretos de OmNet utilizando una lista de eventos futuros y un sistema de entrega de mensajes entre varios módulos.

Las interfaces funcionales de una capa (MAC, aplicación...) o bloque parti-

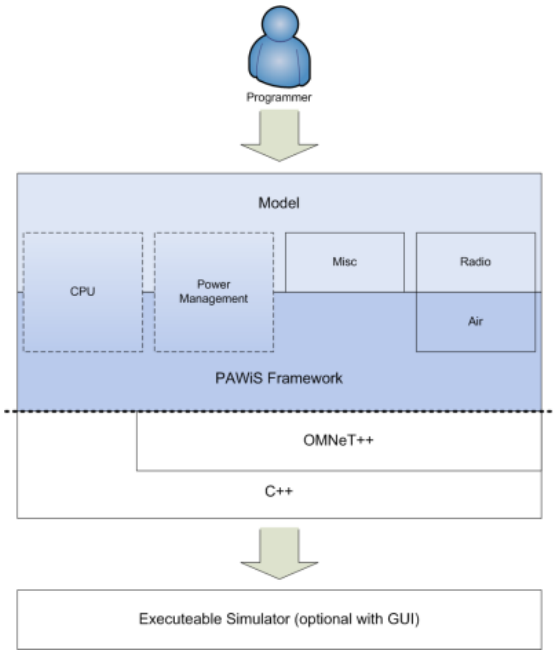


Figura 4.6: Estructura del *framework* de simulación PAWiS

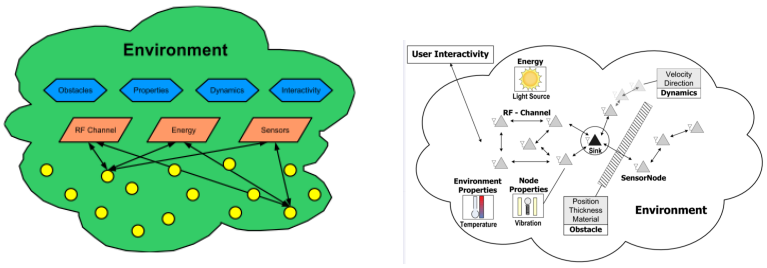
cular (CLAMP, EMM...) son invocadas por otros módulos utilizando el método `invoke(module name, ...)`. Por ejemplo, suponiendo que la capa de aplicación necesita enviar un paquete hacia la capa de red, puede hacerlo utilizando el método `invoke("network","send",&paramIn,&paramOut)`. Para consultar más detalles del funcionamiento interno de PAWiS puede consultarse [16].

4.6.2. Entorno

El entorno en el simulador (fig. 4.7) se define como un escenario tridimensional que contiene todos los nodos y obstáculos. Es responsable de proporcionar el entorno a cada uno de los nodos y proporcionar un medio físico cuantitativo (temperatura, humedad, etc).

La transmisión de radio entre nodos es implementada mediante un objeto *Air* que forma parte de este entorno. La transmisión se realiza en paquetes completos y considera efectos y características del medio como:

- Atenuación de la señal debido al medio de transmisión (ver anexo A) y atenuación adicional (obstáculos).
- Distintos de canales de frecuencia y bandas de frecuencia.
- *Code Division Multiple Access (CDMA)*



(a) (b)

Figura 4.7: El entorno en el simulador con propiedades, objetos y nodos

- Errores de transmisión. Un nodo es notificado cuando recibe una transmisión del medio y mediante la potencia de señal recibida, el SNR y BER son calculados para ese paquete de datos.

4.6.3. Tipos de módulos

La principal función de la librería de módulos `ModLib` es ofrecer un conjunto de módulos acordes al los tipos definidos de módulos (fig. 4.8). Esto permitirá al usuario intercambiar fácilmente diferentes implementaciones sin modificar el comportamiento o código de otros módulos. Los tipos principales de módulos están divididos en grupos.

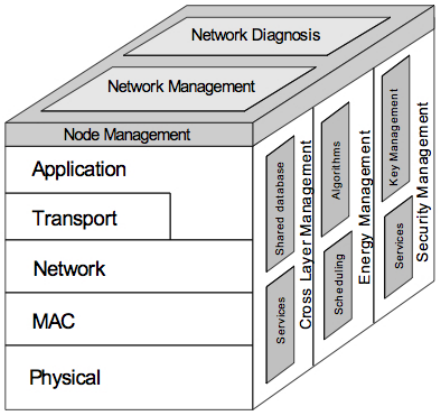


Figura 4.8: Arquitectura de protocolos propuesta por PAWiS

4.6.3.1. Pila de protocolos

Los módulos que forman la pila de protocolos (fig.4.6) son:

- Capa de aplicación: Esta capa generalmente implementa tareas como monitorización, creación y recepción de datos.
- Transporte: Esta capa es opcional y utilizada para asegurar una comunicación punto a punto.
- La capa de red se ocupa del enrutamiento de datos: Mantenimiento de las tablas y búsqueda de rutas, *forwarding* de paquetes.
- En un canal RF, como en cualquier medio compartido en el que puedan darse problemas de colisión, un módulo MAC lleva a cabo todas las tareas necesarias de coordinación, contención y retransmisión de los nodos al acceder al medio.
- Esta última capa provee las tareas básicas de comunicación entre los nodos y el entorno (fig. 4.7).

4.6.3.2. Bloques de gestión

Un conjunto de bloques independientes de las capas de la pila de protocolos controlan la lógica de gestión:

- CLAMP (Cross Layer Management Plane) gestiona las variables globales de los demás módulos.
- La capa de gestión de energía (*Energy Management Plane*) lleva a cabo tareas como: Estimación de energía restante y aspectos de gestión de energía de otros módulos.
- El bloque de gestión del nodo (*Node Management*) es responsable de llevar a cabo tareas comunes como la inicialización y puesta en marcha del nodo.

4.6.3.3. Módulos *hardware*

Además del conjunto de módulos *software* citados, también se dispone de la capacidad de implementar módulos *hardware*. El módulo de CPU que simula el consumo de energía y *timing* de un microcontrolador real. Otro ejemplos son el *timer*, *DAC*, etc.

Para cada uno de los módulos incluido en la librería de módulos, se incluye un datasheet detallado que documenta su función, los parámetros utilizados en las interfaces inicialización, interrupciones, etc.

4.6.4. Gestión del nodo

- **Inicialización:** La primera parte de código que es ejecutada en un nodo está definido que sea la capa de gestión del nodo. Este bloque llamará a todas las funciones de inicialización de otros módulos del nodo.
- **Planificación (*Scheduling*):** Implementa la ejecución y planificación de tareas periódicas llevadas a cabo en cualquiera de los módulos del nodo como, por ejemplo, medición, monitorización y/o estados de "escucha".
- ***Timer*:** El bloque de gestión del nodo contiene un *driver* que implementa un *timer* "abstracto" para permitir que los módulos sean independientes al microcontrolador utilizado. La implementación de esta abstracción ofrece funciones comunes como *wait* de duración constante o condiciones de *stop*, entre ellas.
- **Interrupciones:** Un sistema de interrupciones (vectores y rutinas de servicio, etc) debe ser adaptarse en la combinación de microcontrolador y los módulos utilizados, manteniendo la independencia de la plataforma.
- **Bucle principal (*Main Loop*):** Después de las rutinas de inicialización, las operaciones principales del nodo son puesta en marcha. Esta parte cubre (e implementa) todas las tareas no controladas por las rutinas de servicio del sistema de interrupciones.

Muchas de estas tareas son llevadas a cabo implícitamente por *firmware* real. Únicamente es necesario implementarlos para la simulación y con motivo de mantener una independencia entre los distintos módulos y un determinado tipo de microcontrolador.

4.6.5. Cross Layer Management Plane (CLAMP)

La idea principal de CLAMP es proporcionar un conjuntos de parámetros de red a las diferentes capas para adaptarse dinámicamente a los requerimientos de la aplicación. CLAMP proporciona a cada una de las capas de la pila de protocolos un conjunto de interfaces que les permiten operar con ella: `publish`, `update`, `query` y `subscribe`. Esta funcionalidad ayuda a salvar la limitación impuesta por una arquitectura de capas en el funcionamiento y gestión interno de nodo, ya que las interfaces están disponibles únicamente entre capas adyacentes de la pila de protocolos.

Inicialmente la base de datos de CLAMP está vacía y no conoce nada acerca de ningún parámetro. Cada una de las capas puede publicar cualquiera de los

parámetros propios que desee compartir con otros módulos (`publish`) o puede suscribirse a cualquiera de los parámetros publicados (`suscribe`). CLAMP es capaz de notificar cambios realizados en estos valores a las capas suscritas, utilizando la interface funcional `onChange`, disponible en cada capa. En el caso de que no sea necesario mantener una "suscripción", cualquier capa es capaz de consultar un parámetro particular mediante la interface `query`.

4.6.6. Interfaces

Las interfaces disponibles para llevar a cabo la comunicación entre módulos se agrupan en tres grupos tipos: Obligatorias, opcionales y definidas por el usuario.

Existen dos interfaces comunes a todas las capas en la pila de protocolos utilizadas para enviar y recibir datos:

- Envío de datos (`Send()`): Generalmente se origina desde la capa de aplicación y reenvía el paquete de datos desde esta a la capa física.
- Recepción de datos (`Receive()`): Disponible desde la capa de red hasta la capa de aplicación puede ser invocada de forma asíncrona *bottom-up*. La comunicación entre la capa MAC y física utiliza una interface dedicada que es invocada por la capa de MAC, de acuerdo al protocolo implementado.

Las interfaces comunes en todos los bloques de gestión incluyen las ofrecidas por CLAMP (ver sección 4.6.5) y la función de inicialización (`init()`) de la capa de gestión del nodo.

4.6.7. Lenguaje NED

En Omnet, la topología de un modelo se describe utilizando el lenguaje NED. Este lenguaje facilita la descripción modular de la red. Esta descripción consiste en la especificación de un conjunto de componentes: Canales, módulos simples y/o módulos compuestos. Estas descripciones (módulos, redes, etc) pueden ser reutilizados en otras descripciones. Para una referencia más detallada del lenguaje ejemplos, puede consultarse [46, 25].

4.6.8. Lenguaje LUA

Lua es un lenguaje de programación extensible diseñado para una programación procedimental general con utilidades para la descripción de datos. También ofrece un buen soporte para la programación orientada a objetos, programación funcional y programación orientada a datos. Se pretende que Lua sea usado

como un lenguaje de script potente y ligero para cualquier programa que lo necesite [3].

Siendo un lenguaje de extensión, LUA no tiene noción de programa principal (*main*); sólo funciona embebido en un cliente anfitrión, denominado programa contenedor o simplemente anfitrión (*host*). Éste puede invocar funciones para ejecutar un trozo de código LUA, puede escribir y leer variables de LUA y puede registrar funciones C para que sean llamadas por el código LUA.

4.7. Conclusiones

A lo largo del capítulo y una vez presentados los aspectos teóricos generales, se presentan tres puntos clave para el desarrollo del proyecto: En la primera sección se presenta un análisis detallado de las características y procesos que ocurren en las distintas fases del proceso de localización. En la siguiente sección se fijan con detalle los objetivos, requerimientos y funcionalidades necesarias respecto al proceso anteriormente especificado. En el último punto se ofrece una aproximación a la arquitectura sobre la que se desarrollará el simulador.

Capítulo 5

Diseño

5.1. Presentación

A lo largo del capítulo se detalla el diseño de utilizado en la implementación de la simulación. Se inicia con la descripción de los módulos principales que forman la red (configuración y nodos). Más tarde una descripción de los submódulos que forman la capa de protocolos y gestión de los nodos y, finalmente, una descripción de los métodos utilizados para generar las topologías de red.

5.2. Red de sensores

La estructura de una red de sensores en la simulación (Fig.5.1) se compone de un módulo de configuración, el conjunto de nodos de la red y nodo de *sink*.

Empleando esta filosofía, además de llevar a cabo el diseño e implementación propio de cada uno de los módulos, es necesaria una etapa de especificación de como deben interconectarse cada uno de estos módulos utilizando los componentes propios proporcionados por OmNet y PAWiS. Por un lado, la estructura modular más básica la proporciona OmNet (módulos, submódulos, canales, etc) y por encima de esta estructura, PAWiS aporta los elementos necesarios para

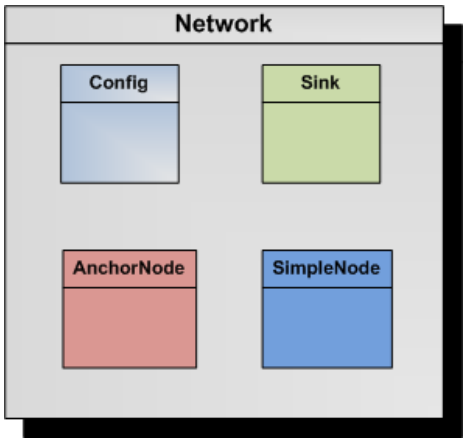


Figura 5.1: Estructura modular de la red utilizada en la simulación

crear la estructura básica de una red de sensores (Fig. 3.2).

5.2.1. Configuración general

Dentro de la red general, el módulo de configuración define parámetros generales utilizados por otros submódulos (nodos de red u otros elementos) que dependen del diseño de los nodos y la implementación específica de la capa de protocolos y módulos. Estos parámetros pueden ser accedidos y modificados dinámicamente en tiempo de simulación desde PAWiS (API de la librería) o simplemente llevar a cabo tareas de inicialización, parametrizando partes generales de la simulación.

Los valores de estos parámetros pueden definirse de varias maneras: Especificarse directamente al definir el módulo mediante NED o LUA (cuadro 5.3), en el archivo de inicio de OmNet `omnetpp.ini` (cuadro 5.1) o manualmente al ejecutar la simulación.

```
[General]
.
.
[Run 1]
myNetwork.config.InitScript = "./omnet/config.lua"
myNetwork.config.LocationEqualRange = 0.0025
myNetwork.config.ErrorRangeDistance = 0.005
```

Cuadro 5.1: Ejemplo de configuración en `omnetpp.ini`

5.2.2. Tipos de nodos

En la simulación está definida la estructura de varios tipos de nodos basados en un modelo que aporta la funcionalidad común básica. Se definen dos tipos de nodos principales que implementan el algoritmo de localización y un tercer modelo completamente opcional como teórico nodo de enlace (fig. 5.3a) con una red exterior.

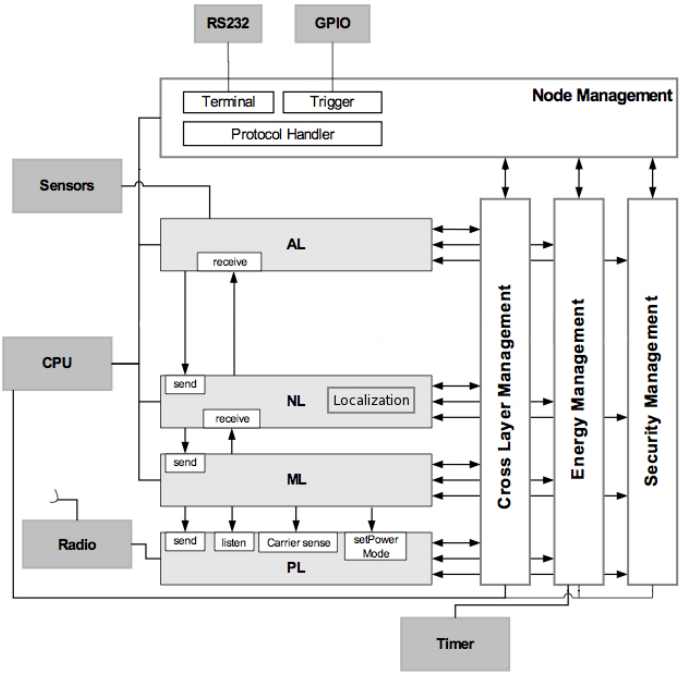
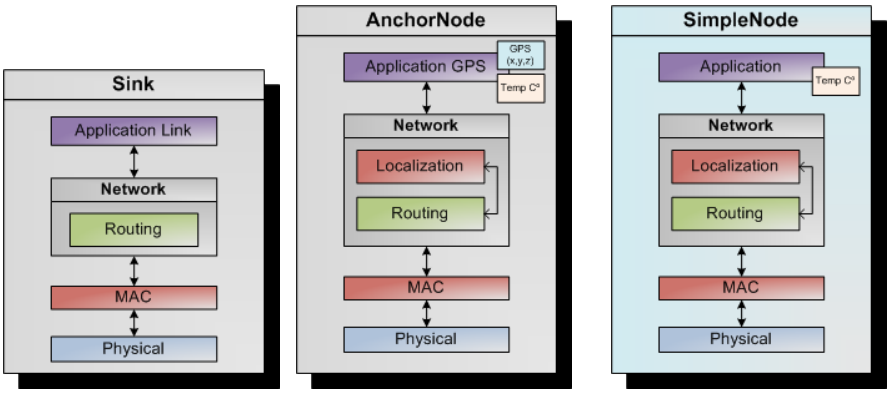


Figura 5.2: Interconexión de módulos e interfaces

Un nodo básico se compone de un conjunto de módulos interconectados tal y como se presenta esquemáticamente en la figura 5.2. La interconexión efectiva de estos módulos y enlaces se específica mediante NED. Estos enlaces forman una arquitectura básica de protocolos con la funcionalidad general necesaria para cubrir aspectos de acceso al medio, comunicación, alimentación, etc.

Ambos nodos comparten el diseño de interconexión y módulos en las capas física, MAC, enrutamiento, localización, aplicación y bloques CLAMP, gestión de energía, gestión del nodo y seguridad. Los parámetros generales de un nodo incluyen un identificador único (utilizado posteriormente como *id* en la red) y tres campos que determinan la posición del nodo en el terreno (PosX, PosY, PosZ).

Los elementos finales son nodos especializados (*hardware/software*) para llevar a cabo tareas específicas y esencialmente se definirá su comportamiento



(a) Nodo *Sink* (b) Nodo *Anchor* (c) Nodo *Simple*

Figura 5.3: Estructura modular de los nodos en la red de sensores

especificando mediante un conjunto de parámetros en las capa de aplicación y localización. Se definirán dos tipos de nodos:

Anchor node Este tipo de nodos se diseminan por la red como "radiofaros" en el proceso de localización: Conocerán su posición e informarán al resto de nodos. Para determinar este tipo de comportamiento, un nodo *Anchor* dispondrá esencialmente de dos parámetros: Parámetro `GPS=true` determinará que está equipado con un sistema de posicionamiento global en la capa de aplicación. `LiarProbability` como valor del intervalo $[0, 1]$ y determina la probabilidad de este nodo de funcionar como nodo tipo *liar*.

Simple node El número mayor de nodos que componen la red son de este tipo debido a su menor coste, consume energético, etc. Este tipo de nodos desconocerán su posición en una fase de *startup* previa necesitarán llevar a cabo una serie de tareas de interacción con el entorno (medio y nodos vecinos) para intentar determinar su posición con la mayor precisión posible. El algoritmo de localización e información empleada por este nodo se especifica mediante parámetros en el módulo de localización. En el módulo de aplicación se define el parámetro `GPS=false`.

En ambos casos, los detalles de la implementación de las capas de *software* y funcionamiento interno de los módulos se citan en el apartado 5.3. En el caso de implementar comportamientos más complejos en los nodos *"liars"* convendría generar un nuevo tipo de nodo de forma que el diseño y comportamiento de la capa de aplicación/red facilite el modelado de una red paralela en un entorno de cooperación.

5.2.2.1. LUA *scripting* en PAWiS

El *framework* de simulación PAWiS soporta LUA como sistema de *scripting* embebido, proporcionando acceso y control mediante este lenguaje a varias de las funcionalidades claves de PAWiS y Omnet. Uno de los usos más interesantes de LUA es su utilización en la fase de inicialización y configuración de la red, especificación de módulos (nodos) y valor de sus parámetros, además de permitir la creación de nuevas funciones que pueden utilizarse para, por ejemplo, dotar a los nodos de propiedades como movimiento.

Unos de los ejemplos utilizados durante el desarrollo de la simulación se expone a continuación. En el cuadro 5.2, el módulo de configuración *Config* ofrece un parámetro *InitScript* que contendrá la ubicación del fichero de *script* de Lua que se ejecutará en el inicio de la simulación. En el *script* de configuración de inicio *"config.lua"* (cuadro 5.3) se especifica la función `init()` que se ejecutará al inicio del *script*. En este ejemplo se generan los nodos (módulos) *Anchor* y *Simple* en posiciones aleatorias y se modifican algunas de sus características (parámetros).

```
simple Config
  parameters:
    InitScript: string,
endsimple

module MyNetwork
  submodules:
    config: Config;
      parameters:
         InitScript = "./omnet/config.lua";
endmodule
network myNetwork : MyNetwork
endnetwork
```

Cuadro 5.2: Declaración de un *script* de configuración LUA en el módulo de configuración

Pueden encontrarse ejemplos de la utilización de LUA en PAWiS en [16], así como ejemplos de especificación de interfaces funcionales de PAWiS mediante LUA y acceso a otro tipo de funcionalidades como movimiento de nodos, agrupación, etc.

```
function init()
   -- Semilla Random
   math.randomseed(12345);
   -- Ecuación de Friis de propagación en dB. despreciando modelo
   .
   .
   local AntennaArea=Range*pawis.PositionFactor;
   local SinkAntennaArea=Range*pawis.PositionFactor;
   local Radius=pawis.PositionFactor;
    -- Create nodes Simple Nodes
    pawis.NodeClass = "SimpleNode";
    for i=1,90 do
        x=0.5+GridX*math.random(); y=0.5+GridY*math.random();
        local aNode = pawis.createDefaultNode("Node"..id, id,  x, y, 0);
        pawis.setDisplayString(aNode, "b="..Radius..",
                       "..Radius..",oval;o=red,,0;r="..AntennaArea..",,black,1");
        id = id+1;
    end
     -- Create anchor nodes
     pawis.NodeClass = "AnchorNode";
     for i=1,10 do
         x=0.5+GridX*math.random(); y=0.5+GridY*math.random();
         local aNode = pawis.createDefaultNode("Node"..id, id,  x, y, 0);
         pawis.setDisplayString(aNode, "b="..Radius..",
                        "..Radius..",oval;o=black,,0;r="..AntennaArea..",,black,1");
         id = id+1;
     end
end
```

Cuadro 5.3: *Script* de inicio LUA

5.3. Diseño de la capa de protocolos

La capa de protocolos, representada por la estructura modular de los nodos parte del modelo teórico descrito en 3.3. Utiliza el modelo básico de módulos proporcionado por PAWiS, excepto por el hecho que se ha eliminado la capa de transporte. La capa de transporte únicamente suele ser necesaria cuando un sistema debe comunicarse con alguna otra red de comunicación (p.e. internet u otra d de sensores), en este caso la comunicación se llevará a cabo nodo a nodo, sin que haya implícita una noción de entrega punto a punto [18].

A continuación de presentan la características más relevantes del diseño de cada una de las capas utilizadas en la simulación.

5.3.1. Capas física y MAC

Ambas capas utilizadas son proporcionadas directamente por la librería de módulos `ModLib` aunque gracias a la estructura modular del *framework*, podría utilizarse cualquier otro módulo proporcionado por la librería o implementar uno nuevo.

La capa MAC utiliza la implementación más sencilla de un protocolo de control de acceso al medio. En envío de un paquete se realizada utilizando un tiempo de espera aleatorio. En el caso de que el canal se encuentre ocupado, se programa el reenvío un número determinado de veces.

También se han realizado pruebas utilizando el protocolo CSMA-MPS [31] orientado a redes de bajo consumo. Las principales características de este protocolo son: Un tiempo de *listen* del nodo reducido y la posibilidad de planificación en el envío de paquetes. Las características detalladas de ambas implementaciones (interfaces, parámetros, etc.) pueden consultarse en el *datasheet* de los módulos.

En la capa física utilizada implementa la abstracción de interfaces de *hardware* y comunicación del transceptor ChipCon CC2400 2.4GHz, cuyas características principales se detallan en el anexo B.

5.3.2. Capa de *routing*

La capa de red realmente se encuentra formada por dos módulos: Módulo de *routing* y módulo de localización.

El módulo de *routing* implementa el protocolo *Energy Aware Distance Vector (EADV)*, diseñado específicamente para su aplicación en redes de sensores formadas por un nodo de *sink* y un número arbitrario de nodos sensores (ver Fig. 3.2) [42].

En una primera etapa de inicialización, el nodo de *sink* genera y envía mediante *broadcast* un mensaje del tipo *Initial Broadcast Vector (IBV)*. Todos los nodos vecinos a su alcance recibirán este vector de inicialización y lo añadirán a su tabla de rutas. El nodo controlará el reenvío de este mensaje mediante una serie de condiciones que controlaran aspectos como *multipath* y *routing loops*. Una vez se finaliza la etapa de *broadcast* inicial, cada nodo dispondrá de una tabla de rutas que permitirá el envío y reenvío de mensajes hacia el nodo de *sink*.

Para obtener más detalles de su implementación (interfaces, parámetros,etc) puede consultarse en [42, 16, 4].

5.3.3. Capa de localización

Esta capa se encargará de implementar toda la lógica del nodo descrita en el análisis de la fase de localización: Envío y recepción de datos, protocolo y algoritmos de localización, detección de movilidad, etc. Generalmente la capa de localización trabaja conjuntamente con la capa de enrutamiento, muchas veces compartiendo información (posición, nodos vecinos identificados en la red) de forma que la implementación final puede desembocar en el uso de una única capa (capa de red).

Teniendo en cuenta que en [14] [23] no se especifica ningún algoritmo concreto de *routing*, a estas alturas del proyecto no se ha considerado necesario completar una implementación conjunta de ambas capas. La capa de localización proporciona información a la capa de aplicación mediante los parámetros del bloque CLAMP y el protocolo de *routing* utilizado no necesita información de esta capa para llevar a cabo sus funciones.

Estados

Se ha determinado un conjunto de estados (Cuadro 5.4) asociado a los dos tipos de nodos (*Anchor* y *Simple*). El estado de un nodo determinará su comportamiento a lo largo de la simulación. Este comportamiento corresponde a la lógica del proceso de localización ya descrito:

- Un nodo *Anchor* debe obtener una posición antes de poder retransmitir su posición al resto de nodos.
- Un nodo *Simple* debe intentar obtener la suficiente información del medio para determinar su posición para, posteriormente, colaborar en que los demás nodos obtengan la suya.

Las transiciones posibles entre los diferentes estados pueden observarse en la Figura 5.4.

Estructuras de datos

En esta capa se almacena la información necesaria para llevar aplicar los algoritmos de localización. La cantidad y uso de la información dependerá de los parámetros de simulación empleados (uso de vecinos, p.e.) y esto derivará en el

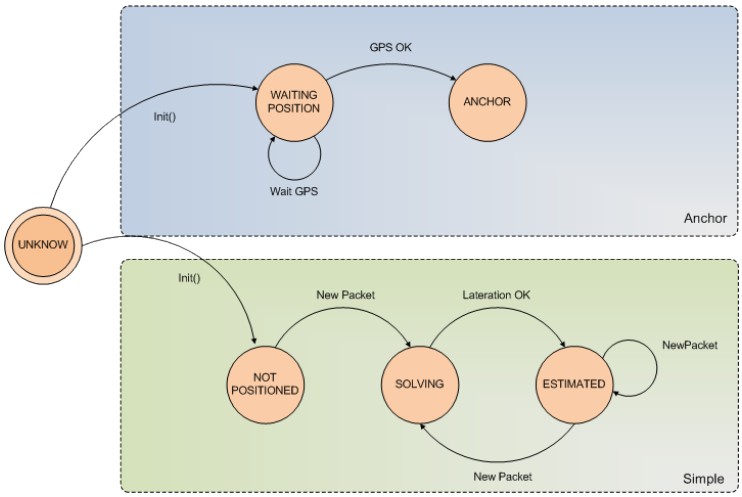


Figura 5.4: Diagrama de estados de la capa de localización

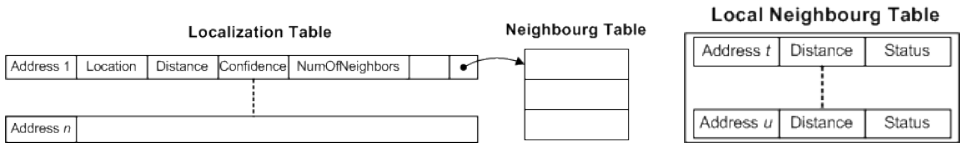
Nodo	Estado	Descripción
All	`UNKNOWN`	Estado inicial previo a la inicialización de la capa de localización
Anchor	`WAITING_POSITION`	Se encuentra en espera de obtener una localización desde el *HW* del nodo de posicionamiento.
	`ANCHOR`	Posicionado y disponible a responder los mensajes de petición de coordenadas del resto de nodos.
Simple	`UNPOSITIONED`	El nodo no ha determinado la posición ni recibido información al respecto del medio. Tablas de enrutamiento vacías.
	`UNSOLVED`	Se ha recibido información pero no la suficiente como para determinar una posición mediante el algoritmo de lateración.
	`ESTIMATED`	Utilizando la información obtenida el nodo ha conseguido determinar una posición

Cuadro 5.4: Descripción de estados del módulo de localización

uso de más o menos recursos por parte de esta capa, sobre todo en cuestión de energía y tiempo de CPU.

La finalidad de esta capa es obtener el suficiente conocimiento del entorno como para poder aplicar el algoritmo seleccionado. Toda esta información se mantendrá en una tabla de localización (Fig. 5.5a) que contendrá información como: Identificador del nodo en la red (*id*), distancia al nodo, su posición, su estado y un valor denominado *confidence*.

Este parámetro es aplicado por algunos autores [28] como estimador de la fiabilidad de la información obtenida. En este caso se utiliza con una significado



(a) Tabla de localización (b) Tabla de nodos vecinos

Figura 5.5: Estruturas de datos del módulo de localización

similar obteniendo dos posibles valores: 1 en el caso de que se trate de una entrada normal y 0 si se ha determinado que la información obtenida desde este nodo se ha determinado como no válida (error o *Liar*). En este último caso esta información no será utilizada al aplicar el algoritmo de localización.

Opcionalmente también se mantiene una tabla con el conjunto de nodos vecinos (Fig. 5.5b) detectados en la red (*id* y distancia) con la posibilidad de incluir una posición y una lista de vecinos al nodo asociada.

Tipos de mensajes

`SimplePacket:` Hereda de la clase `BaseMessage`. Los campo comunes a todos los mensajes de la capa de localización se incluyen la definición de este mensaje.

`BCASTPacket:` Este mensaje se genera en los nodos *Simple* en estado `UNPOSITIONED` para alertar a un conjunto determinado de nodos vecinos, *Anchor* y nodos *Simple* posicionados (estado `ESTIMATED`) que necesitan información para encontrar su posición. En este mensaje se añade información del estado del nodo.

`DataPacket:` Dependiendo de los parámetros especificados en la simulación también pueden incluirse una lista con el conjunto de nodos vecinos y nodos *Liar* detectados por ese nodo.

Parámetros

`LocalNeighbor` Habilitar el uso de *vecinos* en el módulo. Si se habilita este parámetros se almacenará la información de los nodos detectados en la red (mensajes *broadcast* y datos). Está información podrá utilizarse posteriormente por los algoritmos de lateración implementados.

`OtherNeighbor` Al habilitar esta opción, en los mensajes de datos del protocolo también se incluirá la información conocida respecto a los vecinos locales a

un nodo. Esta información podrá almacenarse en la tablas de localización y ser aplicada en otros algoritmos de lateración.

`InformLiars` Al igual que en el caso anterior, se informará al resto de nodos vecinos respecto a los nodos "Liar" detectados localmente.

`LaterationMethod:` Este parámetro especifica el algoritmo de lateración aplicado por el nodo. Los distintos parámetros corresponden a la implementación llevada a cabo en la clase `Lateration`. El constructor de esta clase inicializa el estado interno de manera que se habilita es uso de un algoritmo u otro.

Un nodo simplemente invocará a un método denominado `doLateration()` que, si es posible, devolverá una posición de acuerdo a la información que se encuentra en ese momento en la capa de localización (tablas de localización, etc.).

Esta capa también hará referencia a los parámetros especificados en la configuración general de la red: `ErrorRangeDistance` y `LocationEqualRange`. Estos parámetros pueden obligar a que se genere un error al estimar la distancia de un nodo y determinar cuando una posición en la red es equivalente a otra, respectivamente.

Interfaces

`Init:` Inicialización del módulo de localización desde la capa de gestión del nodo. Actualización del estado de localización y estado previo.

`OnChange:` Notifica cuando alguno de los siguientes parámetros es modificados: `Location{X,Y,Z}`, `noOfNeighbors`.

`Send:` Realiza un *forward* del paquete directamente hacia las capas inferiores.

`Receive:` Recepción de mensajes desde la capa inferior. Al recibir un mensaje se analiza el paquete recibido para determinar de que tipo de los definidos se trata: `Broadcast`, `data` o `ACK`. Cada uno de estos tipo de datos tiene una lógica asociada.

`handleBCASTPacket:` Se prepara la información necesaria para generar un paquete de datos como respuesta a un paquete de este tipo. El conjunto de datos dependerá de los parámetros de simulación (`informLiars`, `LocalNeighbors`, etc). Una vez generado este paquete se envía a las capas inferiores con dirección al remitente del mensaje.

`handleDataPacket:` Se lleva a cabo el tratamiento de los datos recibidos y actualización de las tablas de localización. Opcionalmente la respuesta del protocolo podría ser un "acknoledge" para evitar posibles retransmisiones. En este punto el nodo también debe determinar la distancia del emisor mediante algunos de los métodos propuestos. Actualmente puede actualizarse la distancia al emisor de forma perfecta, utilizando un distribución estadística (ZipF) que simule un cierto tipo de error (ver. 4.3) al realizar la medición o utilizar el parámetro relativo a la potencia de señal recibida, publicado en la capa CLAMP por la capa física.

`handleAckPacket:` Se lleva a cabo un *acknowledge* silencioso.

`Lateration:` Lleva a cabo la operación programada (gestión del nodo) de lateración aplicando el algoritmo seleccionado en la configuración de la simulación de entre los descritos en la sección 4.4.

`Bcast:` Envía un mensaje de `broadcast` al medio como petición de información de localización al resto de nodos vecinos. Programada en el bloque de gestión del nodo.

5.3.4. Capa de aplicación

En el supuesto caso de que se llevasen a cabo todas las fases de inicialización del nodo y la red, y no únicamente la fase de *startup* previa de localización, el objetivo final de este sensor será enviar un dato de medición del medio (temperatura) junto con una posición y una marca de tiempo (opcional). Esta tarea debe llevarse a cabo una vez que se hayan completado las fases de localización y se disponga de la información de enrutamiento necesaria. Para determinar el estado de localización del nodo, esta capa de aplicación se encuentra suscrita a parámetro `LocationStatus` publicado en la capa CLAMP.

El módulo que contiene la capa de aplicación llevará a cabo dos funcionalidades básicas controladas por el gestor de tareas programadas de la capa de gestión del nodo. La primera de ellas será una típica tarea de monitorización del medio simulando la existencia de un sensor de temperatura. La siguiente dependerá del tipo de nodo: En el caso de un tipo *Simple* únicamente inicializará el estado del nodo como `UNSOLVED` y se suscribirá a los valores de posicionamiento publicados en la capa CLAMP, delegando las tareas de posicionamiento al módulo de localización.

Con la finalidad de simular la lógica de funcionamiento de un módulo de posicionamiento (GPS) en un nodo tipo *Anchor*, en el hilo principal de ejecución

de esta capa se invoca a un conjunto de funciones que implementan esta lógica de forma sencilla. En este estado de desarrollo del proyecto no se implementará un módulo *hardware* de posicionamiento (GPS) completo, aunque podría utilizarse como modelo alguno de los componentes que aparecen en el anexo B.

Estructuras de datos y mensajes

Mensajes: La estructura de los mensajes utilizados en esta capa contendrá un campo de temperatura y posición.

Datos: En esta capa se mantendrá la información relativa a localización actual (subscripción), dirección de red, temperatura, estado del nodo (localización) y cierta información relacionada con la inicialización del protocolo de enrutamiento [42].

Parámetros e interfaces

Cualquier módulo de PAWiS hereda de una clase `PawisModule` . Las interfaces funcionales son declaradas e inicializadas mediante funciones de inicialización (`onStartup()`, `onInit()`) generales definidas como virtuales por la clase padre. PAWiS utilizará estas funciones para inicializar cualquiera de los módulos.

El conjunto de interfaces funcionales implementadas en esta capa son:

`Init`**:** Inicialización y puesta en marcha del módulo desde la capa de gestión del nodo. En el caso que el nodo esté declarado como *Liar* (después de la evaluación del parámetro `LiarProbability`), la posición que obtendrá el nodo será errónea y por lo tanto proporcionará información falsa a sus vecinos.

`Send` y `Receive`: Envío y recepción de datos de las capas inferiores.

`Measure`**:** Lleva a cabo un "medición" del medio aunque realmente genera una temperatura aleatoria. En el caso de que la variación respecto a la medición anterior sea importante y se den las condiciones adecuadas de inicialización del nodo, se realiza un envío de datos a la red.

`globalPosition`**:** Obtiene la posición del nodo en un tiempo t. Este código se ejecutará si el nodo se encuentra en el estado `WAITING_POSITION` y generalmente únicamente será invocada si el parámetro `isLocated` está declarado como `false`. En este caso también evaluará el parámetro `LiarProbability`.

El resto de funcionalidad ofrece apoyo a las interfaces principales y permanecen privadas a la clase.

5.4. Diseño de los bloques de gestión

A continuación de presentan la características más relevantes del diseño de los módulos de gestión utilizados en la simulación.

5.4.1. *Node Management (NM)*

La implementación del bloque de gestión del nodo, aunque sencilla, es totalmente funcional utilizando los módulos `TimerSimple` y `CpuSimple`. Proporciona la siguiente funcionalidad: Inicialización, mapeo de interrupciones, ejecución de tareas periódicas y *timer*.

Además de este conjunto de tareas internas, también se ocupa de planificar y ejecutar las siguientes tareas periódicas invocando las interfaces funcionales de los módulos (Fig. 5.6): `globalPosition()` (capa de aplicación), `bcast()` y `lateration()` (capa de localización).

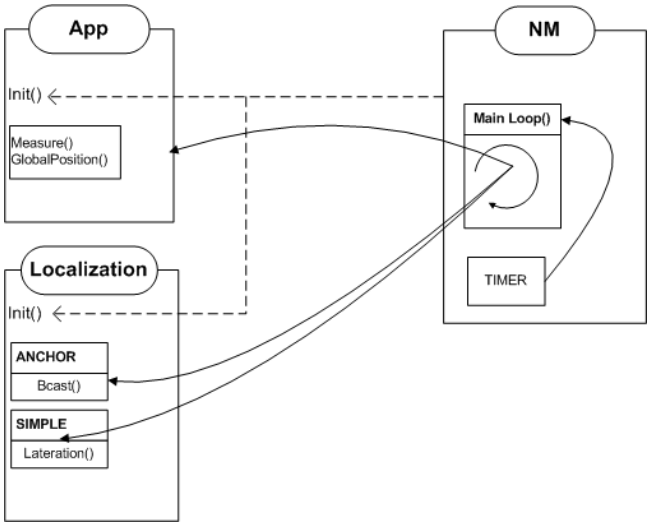


Figura 5.6: Esquema de la especificación NM en el diseño

5.4.2. CLAMP

Los objetivos y funcionalidades principales de este bloque ya han sido especificados en 4.6.5. En esta simulación se han añadido un conjunto de parámetros esenciales a los generales a los ya proporcionados por los módulos principales. Las relaciones principales (`publish`, `subscribe`) de estos parámetros con los módulos de la capa de protocolos puede apreciarse en la Figura 5.7 y una descripción de su uso en la simulación en la Tabla 5.5.

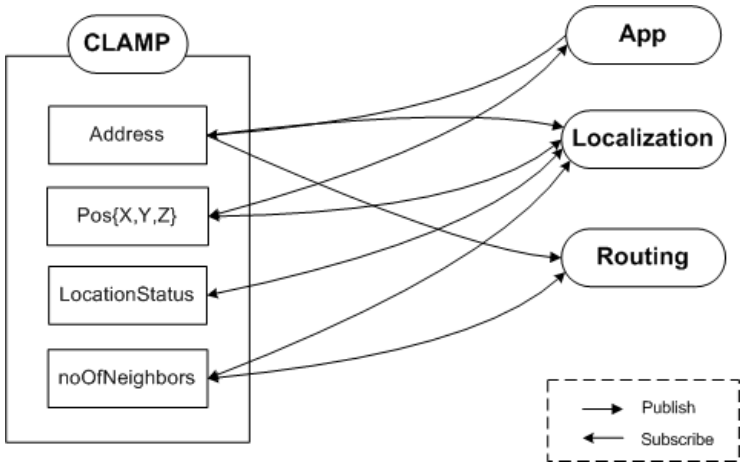


Figura 5.7: Esquema de la especificación CLAMP en el diseño

Parámetro	Descripción
`Address`	Dirección del nodo publicada por la capa de *routing*
`LocationStatus`	Estado del módulo de localización en el nodo
`LocationX,Y,Z`	Posición obtenida por el módulo de posicionamiento o la capa de localización
`noOfNeighbors`	Número de vecinos conocido. Esta información puede obtenerse del módulo de localización y *routing*

Cuadro 5.5: *Profile* de parámetros CLAMP

5.5. Topologías de red

Unos de los objetivos del proyecto incluye facilitar un o varios métodos para especificar una topología de red de sensores, de forma que se facilite la simulación *batch* de un gran conjunto de modelos distintos al llevar a cabo los experimentos con el simulador.

El uso de PAWiS (y por lo tanto OmNet) como *framework* de simulación permite utilizar esencialmente dos métodos:

- Utilizando lenguaje NED proporcionado por OmNet, es posible proporcionar una descripción de la topología de una red de sensores al simulador. Una red es tratada como un módulo que contiene submódulos, en este caso los nodos que forman la red de sensores. En general, el conjunto de topologías de red suele ser estático; un fichero de especificación NED que contiene la definición y declaración de un módulo de red del tipo `GeneratedNetwork` (red de sensores). Este a su vez contiene dos módulos del tipo `AnchorNode` y `SimpleNode` (nodos de la red de sensores) además de

los parámetros de configuración de este módulo, especificados en `Config`.

Aunque la especificación suele ser estática, NED proporcional elementos básicos (aunque no proporcione todas las características de LUA) para generar topologías y submódulos de forma iterativa o parametrizada. Además de lo anterior, en OmNet, NED soporta una función de `include` que permite la generación de la topología de red en el momento de iniciar la simulación.

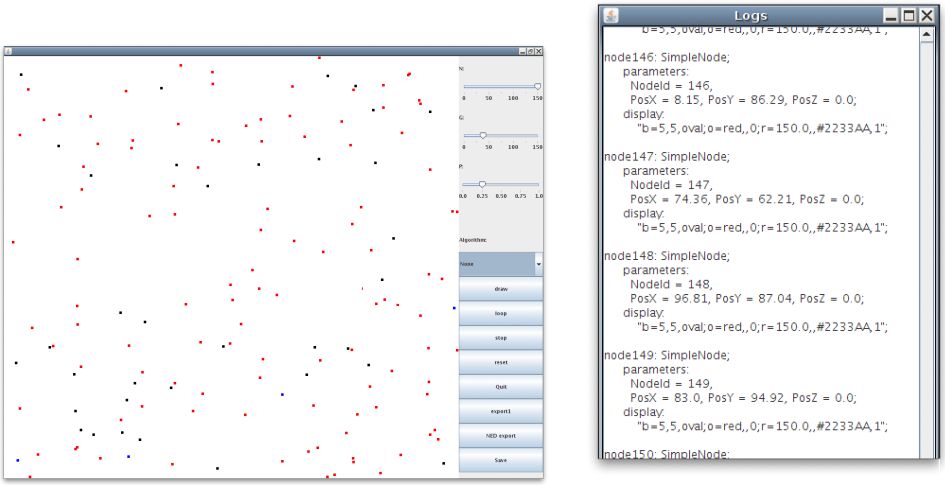
- En combinación con NED, PAWiS también permite generar topologías utilizando *scripting* de LUA. Mediante este método es posible acceder a todas las características proporcionadas por el *framework* y, tal y como se cita en el apartado 5.2.2.1, LUA extiende la funcionalidad de OmNet ofreciendo soporte a ciertas funciones estadísticas, generación de números aleatorios, especificación de interfaces a la librería y módulos proporcionado por PAWiS, entre ellas. En el archivo de configuración `config.lua` de ejemplo (cuadro 5.3) se muestra como se accede a algunos de los parámetros de configuración del módulo de red `GenNetwork` y se generan de forma una topología de nodos `AnchorNode` y `SimpleNode` situados en una posición $(x, y, 0)$ aleatoria.

Debido a la flexibilidad tanto del simulador (OmNet) como del *framework* de simulación, es posible combinar varios métodos en un mismo tipo de simulación; por ejemplo, sería posible generar una configuración básica de red especificada con NED que podría contener un nodo de *sink* y un conjunto de nodos *Anchor*. Más tarde, mediante funciones de inicialización implementadas con LUA podrían generarse el resto de nodos (*Simple*) de forma dinámica siguiendo alguna distribución estadística (o de forma aleatoria si es conveniente), modificar parámetros de configuración de la red e implementar funciones de con un patrón de movimiento diferente en distintos grupos de nodos.

5.5.1. Generación de topologías de red

Durante el desarrollo del simulador y para validar el correcto funcionamiento del mismo a varios niveles (protocolo de comunicación, algoritmos de localización, congestión, simulación en redes con una alta densidad de nodos, p.e.), se han utilizado varios métodos anteriormente citados:

- Para el test en fases previas de desarrollo se han diseñado un conjunto de redes sencillas y genéricas. Estas pruebas han permitido llevar a cabo una prueba comparativa con algunos de los simuladores presentados en la sección 2.2.3 sin entrar en excesivos detalles.



(a) Topología de una red de sensores (b) Topología en formato NED

Figura 5.8: Aplicación utilizada en la generación de topologías de red.

- Durante la fase de desarrollo se han descrito, por un lado, un conjunto reducido de redes estáticas de reducido tamaño y densidad con un resultado fácilmente previsible y por otro, un conjunto de redes dinámicas generadas de forma aleatoria para evaluar cuestiones de rendimiento y funcionamiento generar. Este conjunto ha servido de *testbed* durante todo esta fase.

Tal y como se especifica en los objetivos planteados en las secciones anteriores, se prevee utilizar un tercer método que facilite el diseño y ejecución de experimentos utilizando la librería de consola de OmNet y permita utilizar un *corpus* de redes de sensores donde realizar pruebas comparativas utilizando varios parámetros de simulación.

5.5.2. Aplicación en la simulación

En el apartado 3.2 se han presentado algunas de las cuestiones relativas a las características de una red de sensores, en especial a su diversidad. Con el fin de adecuarse a esta variabilidad en la topología donde se pueden encontrar diseños más complejos que simples distribuciones aleatorias, se ha considerado necesario plantearse métodos alternativos, optando por un sistema como el citado a continuación por su versatilidad en el momento de diseñar futuros experimentos y nuevas simulaciones.

Para generar el conjuntos de topologías de red utilizado en el conjunto final

de simulaciones, se ha modificado la aplicación (figura 5.8) utilizada en fases previas de [14], introduciendo la capacidad de exportar las redes generadas por la aplicación en formato NED (fig. 5.8b). Este conjunto de redes podrá ser importado a la simulación en combinación de algunos de los métodos anteriormente expuestos y lanzar la ejecución de todas ellas utilizando *shell scripts*.

La aplicación utilizada está desarrollada en Java, proporcionando una sencilla interface principal (fig. 5.8a) donde pueden especificarse varios parámetros como número de nodos que forman la red de sensores, nodos equipados con un sistema de posicionamiento global asistido (p.e. GPS), probabilidad de que ese nodo tenga un funcionamiento descrito como *"liar"*, generación automática de topologías, exportar en fichero de texto `ned` y formato NED la topología de red generada, etc. Aunque utilizando esta aplicación se cubren las necesidades de este proyecto, sería relativamente sencillo proporcionar cualquier otro método o utilizar otro tipo de aplicaciones para llevar a cabo este proceso.

5.6. Conclusiones

A lo largo del capítulo se han presentado los aspectos claves del diseño e implementación de la simulación utilizando la arquitectura propuesta por PAWiS.

En las primeras secciones se describe el proceso de generación de módulos y se detalla la estructura propuesta. Posteriormente se describe como se ha aplicado la funcionalidad del sistema de *scripting* soportado por el *framework* y como puede utilizarse en simulaciones que requieran operaciones más complejas de los nodos como, por ejemplo, movilidad.

Por último se presentan características más importantes del diseño realizado de la capa de protocolos y gestión, los métodos empleados y la solución utilizada en el apartado de generación de topologías de red que permite llevar a cabo el conjunto de pruebas.

Capítulo 6

Pruebas

6.1. Presentación

En este capítulo se describen brevemente el conjunto de pruebas a las que ha sido sometido la simulación implementada.

6.2. Conjunto de pruebas

Ha sido necesario depurar la simulación durante todos los puntos del desarrollo del proyecto para asegurar que el conjunto de resultados obtenido es el deseado.

Durante el proceso se han utilizado un conjunto de redes con una distribución de nodos y resultados conocidos de manera que, utilizando la interface gráfica de la simulación (Fig. 6.3), y los resultados finales (Fig. 6.2) y de depuración obtenidos de la simulación (Fig. 6.1), ha facilitado el estudio de resultados del simulador para su posterior verificación.

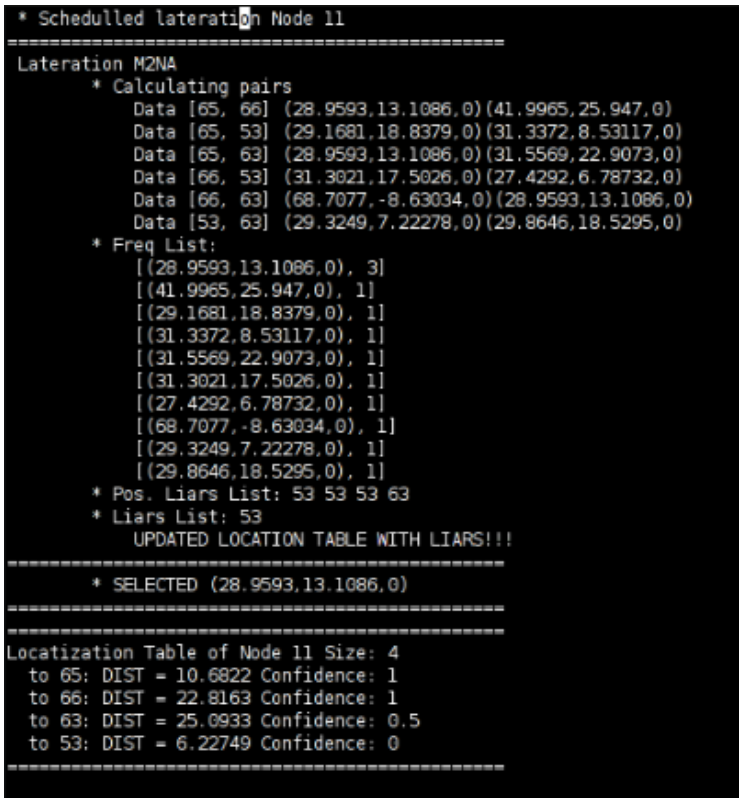


Figura 6.1: Detección de nodos *liar*

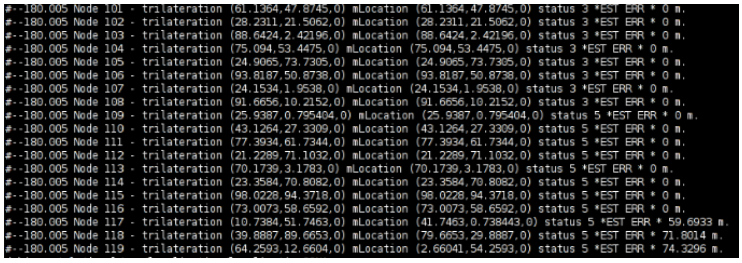


Figura 6.2: Salida estándar una vez finalizado el proceso de localización

6.3. Análisis de resultados

Una vez se han concluido las pruebas de depuración, pueden empezarse a realizar el conjunto de experimentos utilizando distintos conjuntos de redes de sensores y parámetros de simulación. A continuación se presentan algunos ejemplos del tipo de información que puede obtenerse.

- En primer lugar, se encuentran disponibles las herramientas que ofrece OMNet como soporte. Se encuentran opciones muy utilizadas en el registro

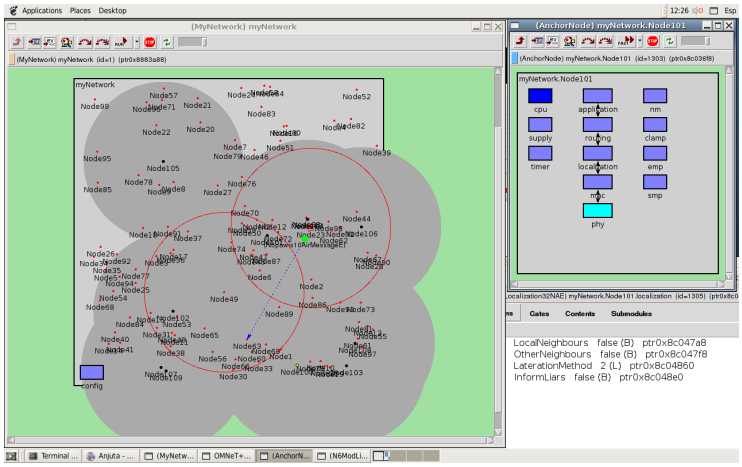


Figura 6.3: Simulación de una red de sensores

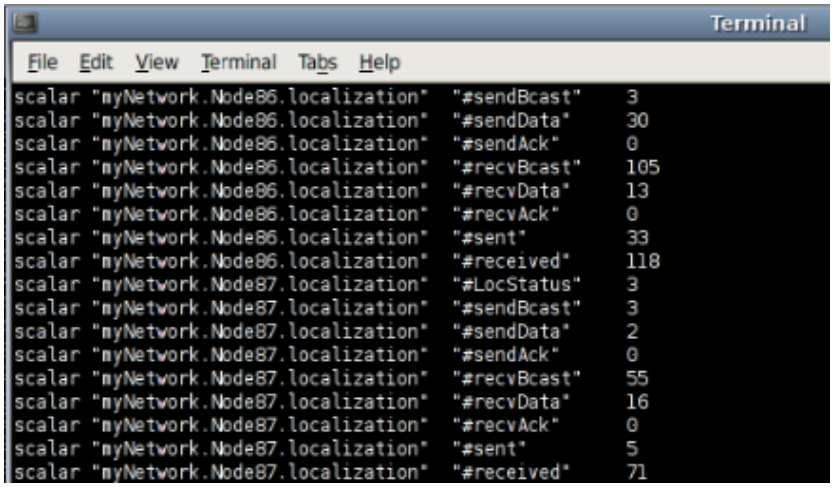


Figura 6.4: Registro de resultados: Escalares

de resultados de la simulación como son el registros de escalares y eventos. Estos registros quedan guardados en ficheros específicos de salida. En la capa de localización se han implementado varios registros asociados al tráfico de red. En la figura 6.4 puede observarse un ejemplo asociado al número de mensajes de los nodos en esta capa. OMNet ofrece un conjunto de herramientas propio para tratar este tipo de resultados y existe un gran número de herramientas de terceros que dan soporte ha este tipo de resultados generando gráficas y otras opciones.

- En segundo lugar se encuentran disponibles las utilidades propias de PA-WiS en el apartado de generación y tratamiento de eventos asociados al

```
I 0.1693623 0.024 12 Node51::phy
V 0.1693623 transition_done_in_Rx Node51::phy
I 0.18565398 9.2e-06 12 Node18::timer
I 0.18565398 2.12e-05 12 Node18::supply
V 0.18581098 listenmode:On Node18::phy
V 0.18581098 transition_from_Off_to_Rx Node18::phy
I 0.18575398 2.62e-05 12 Node18::supply
I 0.18575398 5e-06 12 Node18::phy
I 0.18581098 0.0120287 12 Node18::supply
I 0.18581098 0.0120075 12 Node18::phy
I 0.18582198 0.012040498 12 Node18::supply
I 0.18582198 1.1798313e-05 12 Node18::cpu
I 0.18694098 0.0012329983 12 Node18::supply
I 0.18694098 0.0012 12 Node18::phy
V 0.18694098 transition_from_Idle_to_Rx Node18::phy
I 0.18694098 0.012632998 12 Node18::supply
I 0.18694098 0.0126 12 Node18::phy
I 0.18704098 0.0062329983 12 Node18::supply
I 0.18704098 0.0062 12 Node18::phy
V 0.18704098 transition_from_FsOn_to_Rx Node18::phy
I 0.18704098 0.015132998 12 Node18::supply
I 0.18704098 0.0151 12 Node18::phy
I 0.18708098 0.024032998 12 Node18::supply
I 0.18708098 0.024 12 Node18::phy
V 0.18708098 transition_done_in_Rx Node18::phy
I 0.21037931 9.2e-06 12 Node10::timer
I 0.21037931 2.12e-05 12 Node10::supply
V 0.21053631 listenmode:On Node10::phy
V 0.21053631 transition_from_Off_to_Rx Node10::phy
I 0.21047931 2.62e-05 12 Node10::supply
I 0.21047931 5e-06 12 Node10::phy
I 0.21053631 0.0120287 12 Node10::supply
I 0.21053631 0.0120075 12 Node10::phy
I 0.21054731 0.012040498 12 Node10::supply
I 0.21054731 1.1798313e-05 12 Node10::cpu
```

Figura 6.5: Log de eventos de energía

consumo de energía de los módulos (Fig. 6.5). PAWiS proporciona una herramienta específica para tratar los resultados de este apartado, aunque también pueden utilizarse otro tipo de herramientas. En la figura 6.6 se ha generado la gráfica del consumo de energía total, sin tener en cuenta el sistema de posicionamiento, de cada uno de los nodos durante el proceso de localización.

- Por último, también es posible obtener los resultados obtenidos de la salida estándar del simulador. Al finalizar su ejecución, cada uno de los nodos proporciona información relativa a la posición obtenida y estimada mediante el algoritmo de localización, porcentaje de error relativo, estado del nodo (*LocalizationStatus*), número de mensajes enviados y recibidos, etc.

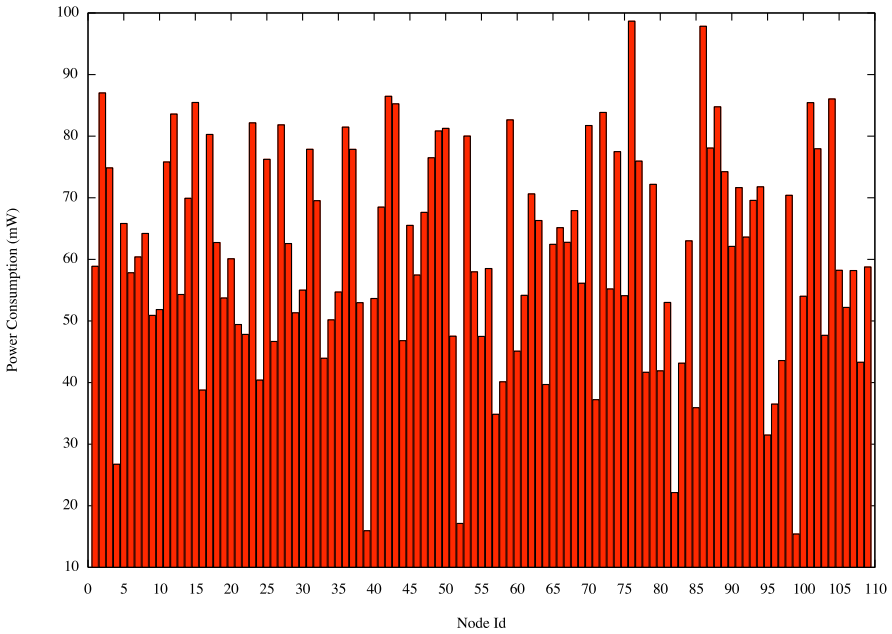


Figura 6.6: Consumo de energía en los nodos

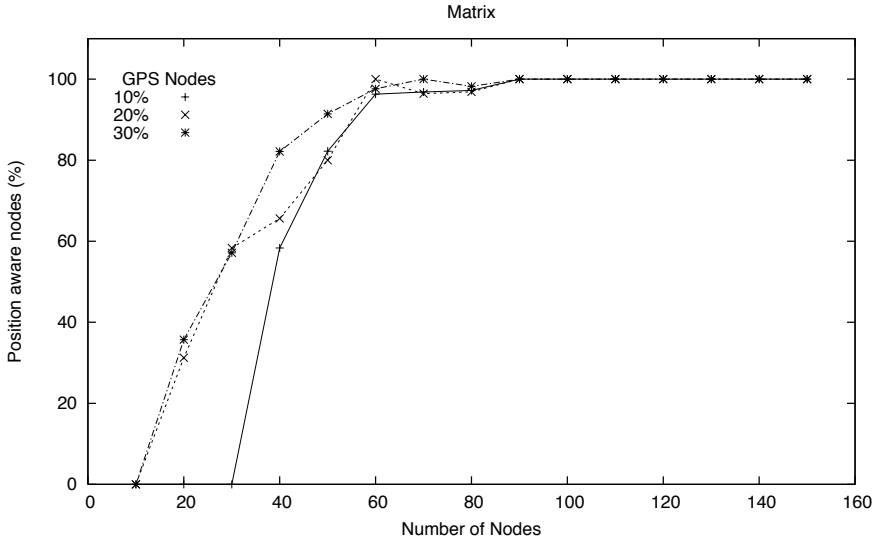


Figura 6.7: Porcentaje de nodos posicionados

Mediante este tipo de información pueden graficarse resultados relativos al porcentaje de nodos posicionados (6.7) o número de mensajes enviados durante la simulación (6.8).

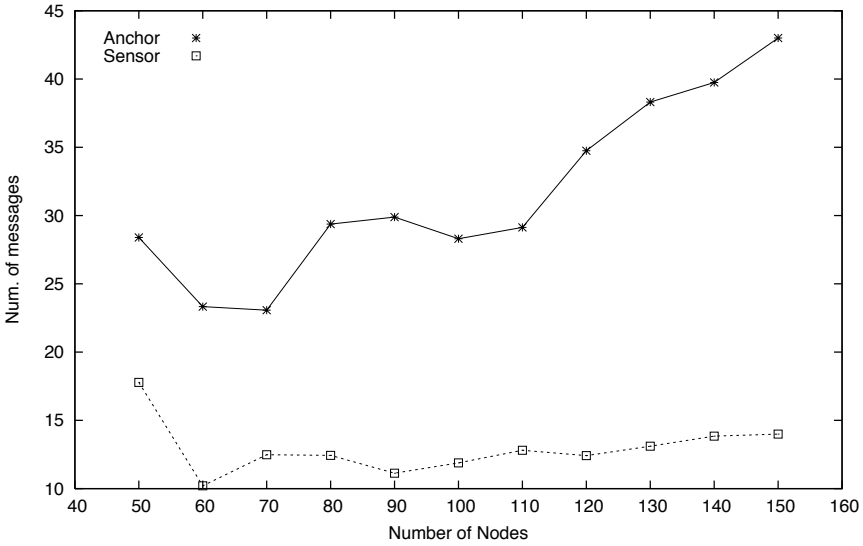


Figura 6.8: Mensajes capa de localización

6.4. Conclusiones

En este capítulo se han descrito el conjunto de pruebas realizados durante el desarrollo del proyecto con el objetivo de verificar los resultados de la simulación. También se han descrito de forma general las distintas opciones disponibles para tratar los resultados obtenidos en la simulación. En última instancia, el paso que realmente verificará los resultados será diseñar un conjunto de experimentos amplio, utilizando un "corpus" de topologías de red extenso, y contrastar los resultados con detalle entre el modelo teóorico y los resultados de la simulación.

Capítulo 7

Conclusiones

7.1. Introducción

Para finalizar con el trabajo desarrollado en el proyecto, en este último capítulo se recogen las conclusiones finales que se pueden desprender del estudio e implementación realizados. Se ha decidido estructurarlo en dos partes, en una primera se realiza un resumen global del proyecto y se revisan los objetivos que se propusieron al inicio de éste, exponiendo aquellos que han sido cumplidos y cuáles no. En una segunda parte, se consideran las futuras líneas de continuación por donde se podría encauzar el trabajo actual.

7.2. Resumen y revisión de objetivos

Tras la implementación y pruebas a los que se ha sometido el desarrollo, es posible retomar los objetivos que se plantearon inicialmente y determinar si se han alcanzado. En primer lugar, se han cumplido los objetivos relativos al diseño y la implementación necesarios para llevar a término este proyecto. De esta manera, se han estudiado las diferentes métodos, técnicas y herramientas empleados en el ámbito de la simulación, motores de simulación orientados a eventos especializados en redes *ad-hoc* y *wireless*.

En este proceso, además de las herramientas, se ha estudiado con profundidad el marco teórico necesario siguiendo un análisis *top-down* desde el nivel de red hasta la arquitectura lógica y *hardware* de los elementos la forman. Una vez determinado el marco teórico, detalle de funcionamiento de cada una de sus partes y concretado las necesidades específicas, se han presentado las bases de la arquitectura y diseño utilizados.

Se han conseguido otros objetivos planteados al inicio del proyecto y considerados como primordiales: Un simulador que implementa los algoritmos propuestos en [14] mediante una arquitectura modular versátil y parametrizable que ofrece varias interfaces (gráfica y consola). Se han presentado varios métodos de generación de topologías de red y otras funcionalidades, como el apartado de movilidad.

7.3. Líneas futuras

A lo largo de este proyecto se ha comprobado la dificultad que entraña un trabajo de estas características y aunque en líneas generales se han conseguido los objetivos más importantes, el desarrollo expuesto es únicamente la base al desarrollo de un modelo más complejo.

Las posibles líneas de continuación que se presentan una vez llegados a este punto pasan obviamente por utilizar la herramienta adecuadamente, aunque se han dejado abiertos ciertos puntos de mejora en aspectos como incluir mejoras en la librería de módulos (módulos *hardware*), monitorización interno de simulador y optimización del *framework*, mejorar la funcionalidad de la aplicación utilizada para generar las topologías de red, etc. Este trabajo puede llegar a representar un gran número de horas, lo que está fuera del alcance de este proyecto. A pesar de ello, es posible enumerar un conjunto de aspectos que pueden ser interesantes de afrontar en un futuro inmediato:

- Trabajar en la implementación del resto de algoritmos propuestos en el artículo. Introducir nuevos parámetros en la simulación como, por ejemplo, determinar que la información proporcionada por cierto tipo de nodos es válida, simulando que se dispone de *trusted nodes* desplegados en la red. Esta característica permitiría realizar experimentos utilizando el conjunto de algoritmos presentado en el artículo y que requieren de este nodos, antes de desarrollar un modelo de confianza en la red.

- Investigar la forma de llevar a cabo tareas de validación de la información o llevar a cabo la implementación de un modelo de confianza que permita

validar como *trusted nodes* a ciertos elementos de la red.

- Actualmente, la creación de un nodo *liar* se lleva a cabo mediante un parámetro de simulación al generar un nodo *anchor* en la red. Resultaría conveniente introducir un nuevo tipo de nodo en el simulador que proporcione modelos más complejos en sus capacidades como adversario: Posibilidad de modificar convenientemente la potencia de emisión al comunicarse con otros nodos, capacidad de llevar a cabo tareas de monitorización y *sniffing* del tráfico de red y en base a esta información, dotar a los nodos de una cierta habilidad en el momento de informar sobre una posición. Estos capacidades facilitarán el desarrollo del punto siguiente.

- Modificar la arquitectura de los nodos destinados a ser adversarios simulando realmente el despliegue de una red paralela de nodos dedicados a llevar a cabo un ataque. Incluir modificaciones en los módulos de la capa de física y MAC para proporcionar una red que opere sobre otro canal y/o banda, estableciendo un canal de comunicación privado entre estos nodos. Dotar a estos nodos de un módulo de aplicación que permita llevar a cabo tareas de colaboración complejas con otros nodos de la red.

- Por el momento el *framework* de desarrollo no está basado en la última versión disponible de OMNeT++. Sería conveniente revisar la posibilidad de integrar la librería con las nuevas funcionalidades proporcionadas en la nueva versión, especialmente en relación al análisis de datos en los modelos simulados.

Apéndice A

Antenas

A.1. Introducción

Para entender la dificultades que aparecen en el proceso de comunicación entre dos elementos *wireless*, previamente es necesario definir y explicar algunos términos comunes, así como presentar algunos del los efectos que pueden rodean a la propagación de ondas de radio en medios no ideales. De la misma forma y relacionado con los conceptos anteriores, existen diferentes tipos de antenas, cada una de ellas con unas características y aplicaciones específicas.

A.2. Fundamentos básicos

A continuación se presentan brevemente algunos de los conceptos básicos relacionados con conceptos que aparecen en [4, 16]. Es posible encontrar información más detallada sobre este tema en [13][24][30].

A.2.1. Ganancia

Al contrario de lo que puede esperarse del comportamiento ideal de una antena isotrópica A.5.1, en la práctica cualquier antena irradiará más energía en unas direcciones que otras y dado que no puede generar una energía, el total de energía irradiada es idéntica a la que podría irradiar una antena isotrópica pero de forma irregular. Este efecto implica que en algunas direcciones se irradie más energía que en otras.

La ganancia de una antena en una dirección determinada es la cantidad de energía radiada en esa dirección, comparada con la energía que sería capaz de radiar en esa dirección una antena ideal (isotrópica) con la misma cantidad de

potencia.

La ganancia puede calcularse como

$$G_{dBi} = 10 \cdot \log_{10}\left(\eta \frac{4\pi}{\lambda^2} A\right) \tag{A.1}$$

en referencia a un radiador isotrópico, η es la eficiencia de la antena.

A.2.2. Ley del cuadrado inverso

La intensidad de señal se calcula en intensidad de campo eléctrico por metro cuadrado. Como la señal sale de forma radial desde la antena hacia todas las direcciones, así si enviamos un vatio de potencia ésta se reparte en la esfera alrededor de la antena, a medida que la señal se va alejando de la antena la esfera crece y la misma potencia se reparte en esta esfera mayor, así la intensidad de la señal por metro cuadrado ha bajado.

La intensidad de la señal disminuye a razón del cuadrado de la distancia ($\frac{1}{r^2}$ donde r es la distancia a la antena). Por ejemplo, entre dos antenas donde la potencia recibida decrece en proporción al cuadrado de la distancia entre ellas, si se dobla la distancia r, únicamente $\frac{1}{4}$ de la potencia es recibida.

A.2.3. Relación señal-ruido

Siempre habrá lugares de nuestra organización donde haya más obstáculos e interferencias. Cuando se analizan señales afectadas por ruido, es útil definir una medida del mismo que sea relativa a la propia señal. Se define la relación señal-ruido, *SNR (Signal to Noise Ratio)*, como el cociente entre el valor de la señal y el ruido. De ese modo, una relación $SNR = 100$ indica que la señal es 100 veces más relevante que el ruido que la afecta.

A.2.4. Eficiencia

Se define como la relación entre la potencia radiada y la potencia total entregada a la antena, a una frecuencia dada. Se puede considerar que la resistencia total de la antena está formada por dos resistencias en serie: R y r. Una antena será tanto más eficiente cuanto mayor sea la relación $\frac{R}{r}$. En general podemos esperar eficiencias entre un 50 y 80 % [39]. El coeficiente de eficiencia de una antena

$$\eta = \left[\frac{R}{(R+r)}\right] \cdot 100 \tag{A.2}$$

donde η se expresa en porcentaje (%).

A.2.5. Atenuación

La atenuación o *pérdidas* entre dos antenas, un transmisor con ganancia G_T (dB) y un receptor con ganancia G_R (dB) puede estimarse utilizando convenientemente la ecuación de Friis para el cálculo de pérdida en la transmisión [24].

$$At(dB) = 10 \cdot \log \left(\frac{4\pi d}{\lambda} \right)^2 - G_T(dBi) - G_R(dBi) \tag{A.3}$$

donde la distancia d es la distancia entre ambas antenas, emisor y receptor.

A.2.6. Patrones de Radiación

El patrón de radiación de una antena, como el ejemplo de la Fig.A.2 y Fig.A.1a, se puede representar como una gráfica tridimensional de la energía radiada vista desde fuera de esta. Los patrones de radiación usualmente se representan de dos formas, el patrón de elevación y el patrón de azimuth. El patrón de elevación es una gráfica de la energía radiada por la antena vista de perfil. El patrón de azimuth es una gráfica de la energía radiada vista directamente desde arriba. Al combinar ambas gráficas se tiene una representación tridimensional de como es realmente radiada la energía desde la antena.

A.3. Propagación de las ondas de radio

Para instalar una red inalámbrica y, en particular, ubicar los puntos de acceso a fin de obtener el máximo alcance posible, se deben conocer algunos datos con respecto a la propagación de las ondas de radio.

Las ondas de radio (abreviado RF por *Radio Frequency*) se propagan en línea recta en varias direcciones al mismo tiempo. En vacío, las ondas de radio se propagan a 3,108 m/s. En cualquier otro medio, la situación deja de ser ideal y la señal está sujeta a efectos como: Atenuación, reflexión, refracción, difracción y *fading*.

A.3.1. Absorción

Cuando una onda de radio se topa con un obstáculo, parte de su energía se absorbe y se convierte en otro tipo de energía, mientras que otra parte se atenúa y sigue propagándose. Es posible que otra parte se refleje.

La atenuación se da cuando la energía de una señal se reduce en el momento de la transmisión. La atenuación se mide en belios (B) y equivale al logaritmo de

base 10 de la intensidad de salida de la transmisión, dividida por la intensidad de entrada. Por lo general, se suelen usar los decibelios (dB) como unidad de medida. Cada decibelio es un décimo de belio. Siendo un belio 10 decibelios, la fórmula sería:

$$R(dB) = (10) \cdot \log \frac{P2}{P1} \tag{A.4}$$

Cuando R es positivo, se denomina amplificación, y cuando es negativo se denomina atenuación. En los casos de transmisiones inalámbricas, la atenuación es más común. La atenuación aumenta cuando sube la frecuencia o se aumenta la distancia. Asimismo, cuando la señal choca con un obstáculo, el valor de atenuación depende considerablemente del tipo de material del obstáculo. Los obstáculos metálicos tienden a reflejar una señal, en tanto que el agua la absorbe.

A.3.2. Reflexión

Cuando una onda de radio choca con un obstáculo, parte o la totalidad de la onda se refleja y se observa una pérdida de la intensidad. La reflexión es tal que el ángulo de incidencia equivale al ángulo de reflexión.

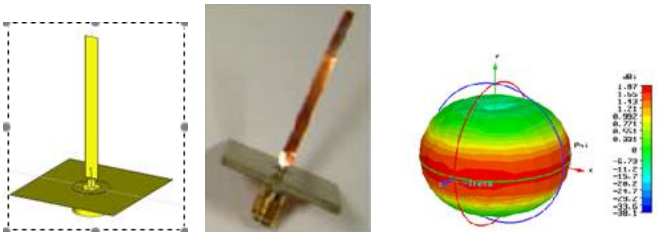
Por definición, una onda de radio es susceptible de propagarse en varias direcciones. Después de reflejarse varias veces, una señal de origen puede llegar a una estación o punto de acceso después de tomar muchas rutas diferentes (llamadas multirutas o *muti-path*).

La diferencia temporal en la propagación (llamada retraso de propagación o *fading*) entre dos señales que toman diferentes rutas puede interferir en la recepción, ya que los flujos de datos que se reciben se superponen entre sí.

Esta interferencia se incrementa a medida que aumenta la velocidad de transmisión, ya que los intervalos de recepción de los flujos de datos se hacen cada vez más cortos. Por lo tanto, la multiruta limita la velocidad de transmisión en redes inalámbricas.

A.3.3. Las propiedades de los medios

El debilitamiento de la señal se debe en gran parte a las propiedades del medio que atraviesa la onda. Los materiales tienen grados de atenuación muy diversos; el aire por ejemplo tiene un grado de atenuación bajo, el agua posee una grado medio y materiales como los metales, poseen lo grados más altos.



(a) Modelo (b) Prototipo (c) Patrón de radiación

Figura A.1: Antena planar monopolo de poliamida

A.4. Clasificación de antenas

Una antena es un dispositivo hecho para transmitir (radiar) y recibir ondas de radio (electromagnéticas). Existen varias características importantes de una antena que deben de ser consideradas al momento de elegir una específica para su aplicación[6]:

- Patrón de radiación
- Ganancia
- Directividad
- Polarización

Las antenas utilizadas en la realización de los sistemas de comunicación para redes inalámbricas necesitan para ofrecer un alto grado de rendimiento en toda la gama de entornos posibles. También tienen que estar diseñados para las especificaciones comunes al resto del sistema, es decir, las pérdidas, el tamaño pequeño y alto rendimiento. En otras palabras, existe una necesidad de antenas para ser utilizadas en sistemas de tamaño reducido que mantengan un nivel de prestaciones[36].

La figura A.1 muestra una ejemplo del desarrollo de una antena planar monopolo de poliamida.

Sin embargo, este tipo de antenas, debido a su reducido tamaño, se acercan a los límites fundamentales de desempeño. Esto puede ser debido a reducciones en el área de captura de la antena, el vínculo indisoluble que existe entre la frecuencia de resonancia y longitud de onda, límites de funcionamiento debido a una fuente de alimentación más ajustada, etc [5].

A continuación se definen los dos tipos básicos de antenas: Omnidireccionales y direccionales. No se presenta el caso de antenas parabólicas ya que no es relevante en el caso de redes de sensores *wireless*, debido a su escaso tamaño.

A.5. Antenas omnidireccionales

Como su propio nimbore indica, una antena omnidireccional es aquella que es capaz de radiar energía prácticamente en todas direcciones.

A.5.1. Antena Isotrópica

Una hipotética antena isotrópica se define como una antena puntual y actúa como una fuente que irradia de forma idéntica en todas las direcciones en una esfera perfecta. Físicamente esta antena no existe. El hecho que la cantidad de energía radiada sea exactamente la misma en todas las direcciones es importante ya que esto nos indica que esta antena podrá enviar o recibir señal con las mismas condiciones esté en la posición que esté.

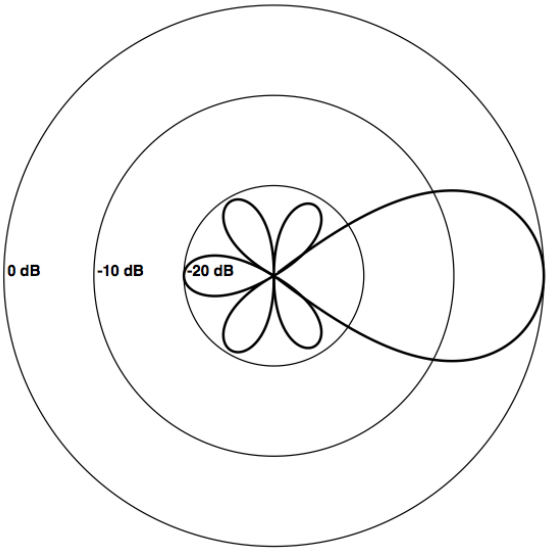


Figura A.2: Diagrama de radiación: Antena direccional

A.6. Antenas direccionales

Las antenas direccionales son aquellas que han sido concebidas y construidas para favorecer que la mayor parte de la energía sea radiada en una dirección en concreto. Puede darse el caso en que se desee emitir en varias direcciones, pero siempre se trata de un número de direcciones determinado donde se encontrarán el lóbulo principal y los secundarios (Fig.A.2).

Como ejemplo de antena direccional es acertado plantear el caso de una antena tipo Yagi. Este tipo de antena consiste en una antena de dipolo a la

cual se le añaden unos elementos llamados "parásitos"para hacerla direccional. Estos elementos pueden ser directores o reflectores.Los elementos directores se colocan delante de la antena y refuerzan la señal en el sentido de emisión.Los elementos reflectores se colocan detrás y bloquean la captación de señales en la dirección opuesta al emisor.

Apéndice B

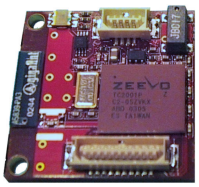
Hardware survey

B.1. Modelos de sensores *wireless*

Intel Imote/Imote 2

El modelo IMote está basado en un sistema *bluetooth* Zeevo TC2001 y un ARM7 integrado. Las principales características de este nodo son:

- Zeevo module
- ARM core, SRAM and Flash memory
- Bluetooth



Nodo	Imote 1.0
CPU	ARM 7TDMI 12-48 MHZ
Memoria	64KB SRAM, 512 KB Flash
Sensores I/O	USB, UART, GPIO,i2C,SPI
Radio	Bluetooth with the range of 30 m
Otros	Sistema operativo TinyOS

Cuadro B.1: Caractersticas Intel IMote 1.0

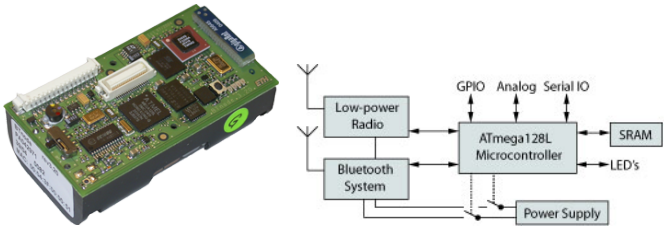
El modelo Imote2 (IPR2400) (Fig. B.2) es una plataforma avanzada de nodo

sensor wireless, construido con un procesador de bajo consumo PXA271 XScale, integra un sistema de radio 802.15.4 (CC2420) con una antena integrada de 2.4GHz. Este nodo es una plataforma modular que puede expandir su funcionalidad mediante ranuras de extensión, lo que permite customizar el sistema para aplicaciones específicas. Mediante los conectores de la tarjeta de expansión del sensor, se pueden utilizar interfaces analógicas o digitales. El sistema puede ser alimentado mediante una batería o a través de la interfaz integrada USB.

- Intel PXA 271 Processor
- Low active power @ 13 MHz
- Ultra low voltage at low speeds (0.85V up to 104 MHz)
- Enhanced Low power modes
- Deep sleep (0.1mW)
- Enhanced I/O options (sensor boards, alternate radios)
- I2C, SPI, UART, CIF, USB, SDIO
- All I/O pins can be configured as GPIOs
- Scalable performance (DVS : power/performance tradeoff)
- Based on a Zeevo TC2001 Bluetooth system-on-chip with integrated ARM7.

Nodo	Imote 2
CPU	Intel PXA 271 Processor
Memoria	Internal 256K SRAM, Stacked 32MB FLASH/SDRAM option, to reduce form factor size
Sensores I/O	Enhanced I/O options (sensor boards, alternate radios
Radio	Wireless MMX and ARM5VTE DSP
Otros	Sistema operativo TinyOS

Cuadro B.2: Caractersticas Intel IMote 2



(a) BTNode (b) Diagrama arquitectura sistema

Figura B.1: Sensor BTNode rev.3 - Imagen y arquitectura

BTnode

El nodo BTnode Rev.3 es dispositivo de radio dual compatible con la anterior revisión 2. Este dispositivo puede operar utilizando dos radios simultáneamente o desconectarlas de forma independiente cuando no están en uso. Es compatible con los sistemas operativos BTnut y TinyOS. Algunas de las nuevas características que proporciona este nodo son un nuevo sistema Bluetooth Zeevo ZV4002 que soporta hasta 4 piconets independientes y a 7 esclavos con un sistema de radio de baja potencia Chipcon CC1000. El sistema B.1b está construido alrededor de un microcontrolador Atmel ATmega128l alimentado con dos células AA.

Nodo	BTNode
CPU	Atmel ATmega128L(AVR RISC 8 MHz @ 8 MIPS)
Memoria	64+180 Kbyte SRAM, 128 Kbyte Flash ROM, 4 Kbyte EEPROM
Sensores I/O	UART, SPI, I2C, GPIO, ADC, Clock, Timer, LEDs Standard Molex 1.25mm Wire-to-Board and Hirose DF17 Board-to-Board connectors
Radio	Chipcon CC1000 operating in ISM Band 433-915 MHz)
Otros	Sistema operativo BTnut System Software o TinyOS

Cuadro B.3: Caractersticas BTNode rev.3

TMote Sky

También conocido como Telos B. Es un nodo de consumo ultra-bajo compatible con la especificación EEE 802.15.4, basedo en un TI MSP430 y un sistema de radio Chipcon CC2420.

Nodo	TMote Sky
CPU	TI MSP430F1611 microcontroller at up to 8 MHz
Memoria	10k SRAM, 48k Flash + 1024k serial storage
Sensores I/O	16-pin expansion port
	Programming and interface via USB
Radio	250kbps 2.4 GHz Chipcon CC2420
	IEEE 802.15.4 Wireless Transceiver
Otros	Contiki, TinyOS, SOS and MantisOS Support
	Ultra-low current consumption
	Fast wakeup from sleep (¡6usec)
	Serial ID chip
	Dimensiones: 32x80 mm

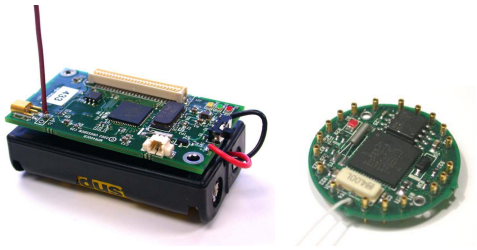
Cuadro B.4: Características TMote Sky

Principales características:

- 10k SRAM, 48k Flash, 1024k serial storage.
- Transceptor Wireless Chipcon CC2420 250kbps 2.4 GHz
- Compatible con la especificación IEEE 802.15.4.
- Sensores integrados en placa de humedad, temperatura y luz.

Mica2/Mica2Dot/MicaZ

La empresa Crossbow ofrece tres familias de nodos Mote en combinaciones de radio/procesador: MICAz (MPR2400), MICA2 (MPR400) y MICA2DOT (MPR500). La radio del nodo MICAz utiliza la banda global de 2.4GHz ISM y soporta las especificaciones IEEE802.15.4 y ZigBee. La família MICA2 y MICA2DOT utiliza las configuraciones disponibles en 315,433,868/900MHz.



(a) Mica2 (b) Mica2Dot

Figura B.2: Sensores Mica2 y Mica2Dot

Nodo	Mica2
CPU	Atmel ATmega128L (Mica2)
	Atmel ATmega128 (MicaZ)
Memoria	4K RAM 128K Flash
Sensores I/O	Large expansion connector
Radio	315, 433 or 868/916Mhz Multi-Channel (38 Kbaud).(Mica2)
	802.15.4/ZigBee compliant RF transceiver (MicaZ)
Otros	Sistema operativoTinyOS o Crossbow

Cuadro B.5: Caractersticas Mica2/MicaZ

Gateway Sensor Nodes

Nodo	Gateway Node
Microcontroller	Stargate
Tranceiver	IntelPXA255
Interface (USB/Serial/Wifi/Ethernet	802.11 Serial connection to WSN
Program Memory	64 MB SDRAM
External Memory	32 MB Flash
Otros	802.15.4/ZigBee compliant, FlatMesh FMG-S 16 MHz
	660 sensor readings

Cuadro B.6: Características nodo gateway

B.2. Módulos

iSense Weather Sensor Module

El módulo sensor iSense Weather proporciona información de gran precisión de la temperaturam humedad relativa y presión barométrica. Es un módulo muy adecuado en aplicaciones de larga duración ya que tiene un consumo $< 1\mu A$ en reposo. Este módulo forma parte de la plataforma de hardware iSense para redes de sensores inalámbricos. Posee dos conectores de 34 pines que pueden ser fácilmente conectados a otros módulos como el módulo principal iSense.

iSense GPS Module

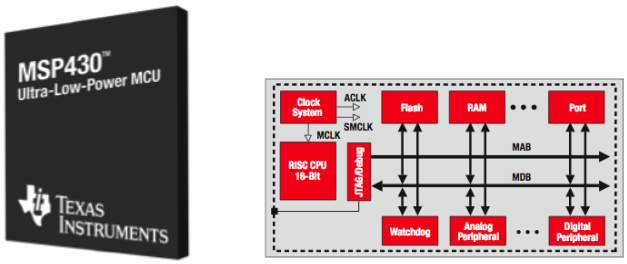
El módulo iSense GPS suministra información de posición a los nodos sensore iSense en todas las implementaciones al aire libre. Los nodos equipados con un módulo GPS puede servir como *anchors* en los protocolos de localización y también puede proporcionar actualizaciones continuas de localización en nodos sensores para móviles.

Este módulo se basa en el chipset SiRF Star 3. Debido a sus 200.000 correladores y 20 canales, es conocido por su precisión y gran sensibilidad. Combina un bajo consumo de energía con un período de fijación extremadamente rápido. Este módulo dispone de una conexión de alto rendimiento, una batería integrada para reservar los datos del sistema y facilitar una adquisición rápida de los datos proporcionados por el satélite, y un led para la indicación de GPS.

Transceptor CS2400

El CC2400 2.4 GHz RF es un transceptor single-chip diseñado para aplicaciones wireless de bajo coste, bajo consumo y altamente integrada que permite la comunicación inalámbrica robusta en la banda ISM de 2,4 a 2,4835 GHz. El transceptor de RF está integrado con un módem de banda base que admite velocidades de datos de hasta 1 Mbps, compatible con tasas de transferencia de 10 kbps, 250 kbps y 1 Mbps, sin necesidad de ninguna modificación en el hardware.

El CC2400 ofrece soporte de hardware extenso para manejo de paquetes, buffering de datos, transmisiones en modo *burst*, codificación de datos y detección de errores, reduciendo la carga de trabajo en el microcontrolador principal. Los principales parámetros de funcionamiento de CC2400 se puede programar a través de un bus SPI.



(a) MCU MSP430 (b) Arquitectura MSP430

Figura B.3: Hardware de sensores: Micro-Controller Unit (MCU)

Microprocesador MSP430

La plataforma MSP430 (Fig.B.3) es un microprocesador de consumo ultra bajo de 16 bit RISC. Sus aplicaciones incluyen una amplia gama de baja potencia y aplicaciones portátiles. El microcontrolador MSP430 ofrece un diseño robusto y soporte, incluyendo documentos técnicos, capacitación, herramientas y software.

Texas Instruments CC2400	
Frequency(Min/Max)(MHz)	2400/2483
Standby Current(uA)	1.5
Operating Voltage(Min/Max)(V)	1.6/2
Current Consumption (RX/TX)(mA)	24/19
Programmable Output Power Ranging From(dBm)	-25 to 0
RSSI Ouput	Digital
Frequency Resolution(MHz)	1
Data Buffering	32 bytes FIFO
RF Chip Interface	Differential
Device Type	Transceiver
Frequency Range	2.4GHz
TX Power(dBm)	0
RX Current (Lowest)(mA)	24
Sensitivity (Best)(dBm)	-101
Data Rate(Max)(kbps)	1000

Cuadro B.7: Low-Cost Single-Chip 2.4 GHz ISM Band Transceiver

Bibliografía

[1] Body sensor networks [online]. Available from: http://ubimon.doc.ic.ac.uk/bsn [cited Enero 2010].

[2] Flood warning technologies [online]. Available from: http://www.alertsystems.org/ [cited Enero 2010].

[3] Manual de referencia de lua 5.1 [online]. Available from: http://www.lua.org/manual/ [cited Marzo 2010].

[4] Pawis project homepage [online]. Available from: http://www.ict.tuwien.ac.at/pawis [cited Febrero 2010].

[5] Tyndall national institute [online]. Available from: http://www.tyndall.ie [cited Febrero 2010].

[6] Wni - wireless solutions [online]. Available from: http://www.wni.com.mx [cited Febrero 2010].

[7] J-sim tutorial [online]. Diciembre 2008. Available from: http://www.j-sim.org/ [cited 20 Diciembre 2008].

[8] Y. Her B. Son and J. Kim. A design and implementation of forest-fires surveillance system based on wireless sensor networks for south korea mountains. *International Journal of Computer Science and Network Security (IJCSNS), 6(9):124-130*, 2006.

[9] Archana Bharathidasan, Vijay An, and Sai Ponduru. Sensor networks: An overview. Master's thesis, Department of Computer Science University of California.

[10] David Curren. A survey of simulation in sensor networks. 2004.

[11] Consejo Superior de Administración Electrónica (CASE). Documentación metrica v3. Available from: http://www.csi.map.es/csi/metrica3 [cited Diciembre 2008].

[12] F. Diggelen. Gnss accuracy: Lies, damn lies, and statistics. *GPS World, vol. 18, no. 1, pp. 26-32*, January 2007.

[13] A.R. Figueiras. *Una panorámica de las telecomunicaciones.* Prentice Hall, 2002.

[14] J. Garcia, M. Barbeau, and E. Kranakis. Secure localization of nodes in wireless sensor networks with limited number of truth tellers. In *7th Annual Communication Networks and Services Research (CNSR) Conference, IEEE Communications Society, Moncton, New Brunswick, Canada*, May 2009.

[15] Sinan Gezici, Zhi Tian, Georgios B. Biannakis, Hisashi Kobayashi, Andreas F. Molisch, H. Vincent Poor, Zafer Sahinoglu, Sinan Gezici, Zhi Tian, Georgios B. Giannakis, Hisashi Kobayashi, Andreas F. Molisch, H. Vincent Poor, and Zafer Sahinoglu. Localization via ultra-wideband radios, 2005.

[16] Johann Glaser and Daniel Weber. Pawis simulation framework overview. 2008.

[17] G. Hoblos, M. Staroswiecki, and A. Aitouche. Optimal design of fault tolerant sensor networks. pages 467–472, August 2002.

[18] Y. Sankarasubramaniam I. Akyildiz, S. Weilian and E. Cayirci. A survey on sensor networks. *IEEE Communications Magazine, 40(8):102-114, August 2002.*, 1997.

[19] In Proc. of International Conference on Mobile systems, Applications and Services (MobiSys'06), pages 28-41, Uppsala, Sweden. *FireWxNet: A multi-tiered portable wireless system for monitoring weather conditions in wildland fire environments*, June 2006.

[20] In Proc. of SPIE Symposium on Smart Structures and Materials, pages 477-484, San Diego, CA. *Wireless sensors for wild-fire monitoring*, March 2005.

[21] In Proc. of the First International Workshop on Wireless Sensor Networks and Applications (WSNA'02), pages 88-97, Atlanta, Georgia. *Wireless sensor networks for habitat monitoring*, September 2002.

[22] Information Processing Techniques Office (IPTO). Advanced soldier sensor information system and technology (assist) [online]. Available from: `http://www.darpa.mil/ipto/programs/assist/assist_obj.asp` [cited Diciembre 2009].

[23] M. Barbeau J. Garcia and E. Kranakis. Positioning of wireless sensor nodes in the presence of liars. May 2010.

[24] H. Jasik. *Antenna Engineering Handbook*. McGraw-Hill, 1961.

[25] A. Köpke, M. Swigulski, K. Wessel, D. Willkomm, P.T.K. Haneveld, T. Parker, O.W. Visser, H.S. Lichte, and S. Valentin. Simulating Wireless and Mobile Networks in OMNeT++ The MiXiM Vision. In *OMNeT++ Workshop*, 2008.

[26] UCLA Parallel Computing Laboratory. Glomosim user manual, global mobile information systems simulation library. Available from: `http://pcl.cs.ucla.edu/projects/glomosim/` [cited 12 Diciembre 2008].

[27] K.G. Langendoen, A. Baggio, and O.W. Visser. Murphy loves potatoes: Experiences from a pilot sensor network deployment in precision agriculture. In *14th Int. Workshop on Parallel and Distributed Real-Time Systems (WPDRTS)*, apr 2006. Available from: `http://www.st.ewi.tudelft.nl/~koen/papers/WPDRTS06.pdf`.

[28] Koen Langendoen and Niels Reijers. Distributed localization in wireless sensor networks: A quantitative comparison, 2003.

[29] C. H. Liu and D. J. Fang. *Propagation in antenna handbook: Theory, applications, and design.* Van Nostrand Reinhold, 1988.

[30] Y. T. Lo and S.W. Lee. *Antenna Handbook: Theory, applications, and design*. Van Nostrand Reinhold, 1988.

[31] Stefan Mahlknecht and Michael Bck. Csma-mps: A minimum preamble sampling mac protocol for low power wireless sensor networks.

[32] Guoqiang Mao, Bar Fidan, and Brian D. O. Anderson. Wireless sensor network localization techniques. *Comput. Netw.*, 51(10):2529–2553, 2007.

[33] John Michopoulos and Sam Lambrakos. On the fundamental tautology of validating data-driven models and simulations. In *Proceedings ICCS?05*.

[34] Dragos Niculescu and Badri Nath. Ad hoc positioning system (aps). In *IN GLOBECOM*, pages 2926–2931, 2001.

[35] University of Southampton. Glacsweb. glacier monitoring [online]. Available from: `http://envisense.org/glacsweb/index.html` [cited Diciembre 2009].

[36] D. Laffey J. Barton S.C. O'Mathuna O'Flynn., J Buckley. Simulation, design, development and test of antennas for wireless sensor network systems. *Microelectronics International, Volume 24, Number 2, 2007, pp. 3-6*, 2007.

[37] S. Park, A. Savvides, and M. B. Srivastava. Networked & embedded systems laboratory (nesl), university of california at los angeles (ucla). Available from: `http://nesl.ee.ucla.edu/projects/sensorsim/` [cited Diciembre 2008].

[38] N. Patwari, J. Ash, Alfred O. Hero Iii, A. O. Hero, R. M. Moses, N. S. Correal, Motorola Labs, S. Kyperountas, and Neiyer S. Correal. Locating the nodes: Cooperative localization in wireless sensor networks, 2005.

[39] Proceedings of Microwave Update 88, ARRL, pages 57-59. *Design Considerations for Amateur Microwave Antennas*, 1988.

[40] PermaSense project. Permasense [online]. Available from: `http://www.permasense.ch/` [cited Enero 2010].

[41] Pressman R. *Ingeniería del Software: Un enfoque práctico.* McGraw-Hill, 2001.

[42] S. A. Madani S. Mahlknecht and M. Roetzer. Energy aware distance vector routing scheme for data centric low power wireless sensor networks.

[43] Prasan Kumar Sahoo, I-Shyan Hwang, and Shi-Yao Lin. A distributed localization scheme for wireless sensor networks. In *Mobility '08: Proceedings of the International Conference on Mobile Technology, Applications, and Systems*, pages 1–7, New York, NY, USA, 2008. ACM. `doi:http://doi.acm.org/10.1145/1506270.1506366`.

[44] Andreas Savvides, Chih chieh Han, and Mani B. Strivastava. Dynamic fine-grained localization in ad-hoc networks of sensors, 2001.

[45] Simulation Symposium, 2004. Proceedings 37th Annual. *SENS: a sensor, environment and network simulator*, 2004.

[46] András Varga. Omnet++. discrete event simulations system. user manual. 2006.