ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACIÓN

Titulación:

INGENIERO TÉCNICO EN INFORMÁTICA DE GESTIÓN

Título del proyecto:

TOLERANCIA A FALLOS Y AUTOCONFIGURACIÓN EN REDES DE SENSORES

Pablo Albizu Balerdi

Tutor: José Javier Astrain Escola

Pamplona, 15 de septiembre de 2010

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

AGRADECIMIENTOS

Es tarea imposible nombrar a todas aquellas personas que me han ayudado a llegar hasta aquí y finalizar este proyecto fin de carrea, pero si quiero reconocer específicamente el valor a algunos de ellos:

Quiero dar mi más sincero agradecimiento a Don José Javier Astrain Escola, por haberme dirigido este proyecto de fin de carrera. Por todas las atenciones, por el tiempo que ha perdido conmigo, y sobre todo por su apoyo.

También me gustaría destacar la paciencia que ha tenido para conmigo Don Carlos Aristu López, sufridor constante de mis preguntas para la resolución de los distintos problemas surgidos con la utilización del hardware y el software empleado en la realización del proyecto.

Mi familia, por su apoyo y empuje incondicional. Especialmente a mis padres, si no hubiera sido por ellos no habría llegado hasta aquí y les estaré eternamente agradecido por ello, y a mi abuelo Vitoriano, por deslumbrarse con su experiencia y por su seguimiento y control día a día de cómo avanzaba el proyecto.

A mi amiga, mi compañera, Elsa Alonso Martínez, por haber vivido este proyecto como si fuese suyo y por hacerme mirar siempre hacia adelante dándome aliento y fuerza en los momentos más complicados. Te debo muchas cosas, una de ellas poder estar escribiendo ahora mismo estas líneas.

No puedo olvidarme de mis compañeros de laboratorio, especialmente de Patricia Chivite Fernández, Dani Ruíz Albéniz y Leticia Osacar Landa, por motivarme día a día y por hacer que las horas de trabajo en el laboratorio volasen.

Gracias a todos mis amigos y amigas en general, a mis compañeros de piso, a mi cuadrilla de Igúzquiza y a todos mis amigos y amigas en general, por ayudarme a ver la luz al final de este túnel constantemente y por hacerme sentir que esta tarea merecía la pena.

Gracias a todos aquellos que han puesto su granito de arena y que olvido nombrar.

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

ÍNDICE

1.- INTRODUCCIÓN 4
1.1.- Red de sensores 4
1.2.- Técnica de agregación de datos 7
1.3.- Trabajo a desarrollar 8
2.- ENTORNO DE TRABAJO 9
2.1.- Introducción 9
2.2. - HARDWARE: KIT CROSSBOW 9
2.2.1. Resumen del producto 9
2.2.2.- Diagrama de bloques del XM2110-IRIS 10
2.2.3.- Procesador y radio 11
2.2.4.- Sensores o placas sensoras 11
2.2.5.- Batería 11
3.2.6.- Memoria flash 11
2.2.7.- Estación base USB MIB520 12
2.2.- SOFTWARE 13
2.2.1 Sistema operativo 13
2.2.1.1.- TinyOS 13
2.2.2.- Lenguajes de programación para motas 15
2.2.2.1.- NESC 15
3.- ALGORITMOS DE AGREGACIÓN 17
3.1.- Introducción 17
3.1.1.- Antecedentes y trabajos relacionados 17
3.1.2.- Algoritmos propuestos 18
3.2.- Modelo del sistema 19
3.2.1 El detector de fallos Ω 20
3.3.- Nivel local (intra-regional) 20
3.3.1.- Primer algoritmo 20
3.3.2.- Segundo algoritmo 24
3.3.3.- Tercer algoritmo 26
3.4.- El consumo de energía en la elección del agregador y en la agregación de datos. 29
4.- ANÁLISIS Y DISEÑO 31
4.1. Particularidades 31
4.2. Especificación de Requisitos de Usuario 31
4.2.1.- Arquitectura 31
4.3.- Algoritmo 1 33

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

4.3.1.- Comportamiento de la aplicación 33
4.3.2.- Estructura de la trama de datos 34
4.3.3. Diagramas de secuencia 36
4.3.4.- Estructura de la aplicación 37
4.4.- Algoritmo 2 38
4.4.1.- Comportamiento de la aplicación 38
4.4.2.- Estructura de la trama de datos 39
4.4.3.- Diagramas de secuencia 40
4.4.4.- Estructura de la aplicación 41
4.5.- Algoritmo 3 42
4.5.1.- Comportamiento de la aplicación 42
4.5.2.- Estructura de la trama de datos 42
4.5.3.- Diagramas de secuencia 43
4.5.4 Estructura de la aplicación 44
5.- RESULTADOS COMPARATIVOS 45
5.1 Resultados para el algoritmo 1 45
5.2 Resultados para el algoritmo 2 51
5.3 Resultados para el algoritmo 3. 62
6.-CONCLUSIONES Y LÍNEAS FUTURAS 69
BIBLIOGRAFÍA 71

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

1.- INTRODUCCIÓN

1.1.- Red de sensores

Una red de sensores (del inglés *sensor network*) es una red de dispositivos de reducidas capacidades de cómputo y almacenamiento («nodos»), equipados con sensores, que colaboran en una tarea común. Estos dispositivos tienen por misión la obtención de datos del entorno (temperatura, sonido, vibración, presión, movimiento, contaminantes, luminosidad, etc.) para ser transmitidos hasta una estación base que los hará accesibles desde el exterior.

Un nodo de una red de sensores es una unidad autónoma que consiste físicamente en un microprocesador, un dispositivo de almacenamiento de datos, sensores, convertidores analógico-digitales, un transmisor de radio, controladores y una fuente de energía [2].

Las redes de sensores están formadas por un grupo de sensores con ciertas capacidades sensitivas y de comunicación inalámbrica los cuales permiten formar redes *ad hoc* sin infraestructura física preestablecida ni administración central.

El concepto de red de sensores es relativamente nuevo en adquisición y tratamiento de datos, y posee múltiples aplicaciones en distintos campos tales como entornos industriales, domótica, entornos militares, detección ambiental, entornos sanitarios, etc.

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

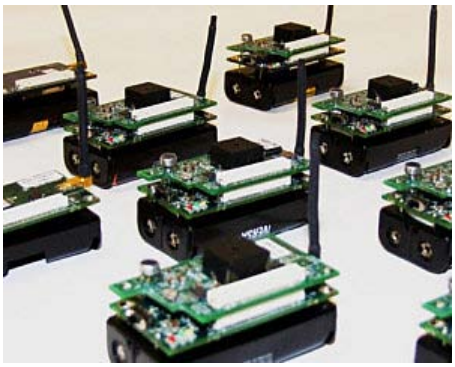


Figura 1: *Nodos de una red de sensores*

La miniaturización creciente de ordenadores dio a luz la idea de desarrollar computadoras extremadamente pequeñas y baratas que se comunican de forma inalámbrica y se organizan autónomamente. La idea de estas redes es repartir aleatoriamente estos nodos en un territorio grande, siendo estos los que observan hasta que sus recursos energéticos se agoten. La mayoría de las redes inalámbricas de sensores se basan en el estándar IEEE 802.15.4, que es una versión reducida del estándar de comunicaciones 802.11 (WiFi). En general, estas redes utilizan también la banda de 2.4 GHz, son totalmente programables e incorporan el sistema operativo TinyOS del que hablaremos más adelante [13].

Igualmente, estas redes suponen uno de los avances más investigados en los últimos años. Funcionalidades que antes eran independientes unas de otras, van a poder ser integradas gracias a este tipo de redes, con el fin de lograr máxima eficiencia en cuanto a consumo y gestión de energía.

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

Las últimas investigaciones apuntan hacia una eventual proliferación de redes de sensores inteligentes, redes que recogerán enormes cantidades de información hasta ahora no registrada que contribuirá de forma favorable al buen funcionamiento de fábricas, al cuidado de cultivos, a tareas domésticas, a la organización del trabajo y a la predicción de desastres naturales como los terremotos. En este sentido, la computación que penetra en todas las facetas de la vida diaria de los seres humanos está a punto de convertirse en realidad [3].

El diseño, implementación y operación de una red de sensores requiere la convergencia de muchas disciplinas, como procesamiento de la señal, redes y protocolos, sistemas embebidos, gestión de la información y algoritmos distribuidos. En un futuro no muy lejano, sensores muy baratos y diminutos se rociarán en carreteras, paredes o máquinas, creando una capa digital que sentirá una variedad de fenómenos físicos de interés [28].

Historia

La evolución de redes de sensores tiene su origen en iniciativas militares. Por eso no hay mucha información sobre la fuente de la idea.

Como predecesor de las redes de sensores modernos se considera *Sound Surveillance System (SOSUS)*, una red de boyas sumergidas instaladas en los Estados Unidos durante la Guerra Fría para detectar submarinos usando sensores de sonido.

La investigación en redes de sensores cerca de 1980 comenzó con el proyecto *Distributed Sensor Networks (DSN)* de la agencia militar de investigación avanzada de Estados Unidos *Defense Advanced Research Projects Agency (DARPA).*

Es probable que hoy mismo haya proyectos militares que sigan haciendo investigaciones en esa área.

Áreas de aplicación

Haciendo caso omiso de las aplicaciones militares, el ámbito de aplicación de las redes de sensores es muy amplio y diverso, en particular cabe destacar las siguientes áreas de aplicación:

- Eficiencia energética: Se utilizan redes de sensores para controlar el uso eficaz de la electricidad, como en el caso de Japón.

- Entornos de alta seguridad: Existen lugares que requieren altos niveles de seguridad para evitar posibles ataques, tales como centrales nucleares, aeropuertos, edificios del gobierno de paso restringido. Aquí gracias a una red de sensores se pueden detectar situaciones que con una simple cámara sería imposible.

- Sensores ambientales: El control ambiental de vastas áreas de bosque o de océano, sería imposible sin las redes de sensores. Las redes de sensores controlan múltiples variables, como la temperatura, la humedad, el fuego o la actividad sísmica. También ayudan a los expertos a diagnosticar o prevenir un problema o urgencia y además minimizan el impacto ambiental de la presencia humana.

- Sensores industriales: Dentro de fábricas existen complejos sistemas de control de calidad, el tamaño de estos sensores les permite estar allí donde se requiera.

- Automoción: Las redes de sensores son el complemento ideal a las cámaras de tráfico, ya que pueden informar de la situación del tráfico en ángulos muertos que no cubren las cámaras; también pueden informar a conductores de la situación, en caso de atasco

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

o accidente, con lo que estos tienen una mayor capacidad de reacción para tomar rutas alternativas.

- Medicina: Es otro campo bastante prometedor. Con la reducción de tamaño que están sufriendo los nodos sensores, la calidad de vida de pacientes que tengan que tener controlada sus constantes vitales (pulsaciones, presión, nivel de azúcar en sangre, etc.), podrá mejorar sustancialmente.

- Domótica: Su tamaño, economía y velocidad de despliegue, las hacen una tecnología ideal para automatizar tareas cotidianas en el hogar a un precio asequible.

Es imaginable que los nodos no sólo puedan observar sino también *reaccionar* para activar funciones de otros sistemas.

Características de una red de sensores

Las redes de sensores tienen una serie de características propias y otras adaptadas de las redes Ad-Hoc:

- Topología Dinámica: En una red de sensores, la topología puede ser cambiante, debido a que los nodos pueden dejar de funcionar en cualquier momento, y éstos tienen que ser capaces de adaptarse para poder comunicar nuevos datos adquiridos.

- Variabilidad del canal: El canal radio es un canal muy variable en el que existen una serie de fenómenos como pueden ser la atenuación, desvanecimientos rápidos, desvanecimientos lentos e interferencias que puede producir errores en los datos.

- No se utiliza infraestructura de red: Una red de sensores no tiene necesidad alguna de infraestructura para poder operar, ya que sus nodos pueden actuar como emisores, receptores o routers. Sin embargo, hay que destacar en el concepto de red de sensores la figura del nodo recolector (también denominados sink node o estación base), que es el nodo que recolecta la información generada normalmente en tiempo discreto. Esta información generalmente es adquirida por un ordenador conectado a este nodo y es este ordenador el que tiene la posibilidad de transmitir los datos por tecnologías inalámbricas o cableadas según sea el caso.

- Tolerancia a fallos: Un dispositivo sensor dentro de una red de sensores tiene que ser capaz de seguir funcionando incluso cuando se produce algún fallo en su sistema.

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

- Consumo energético: Es uno de los factores más sensibles debido a que tienen que conjugar autonomía con capacidad de proceso, ya que actualmente cuentan con una unidad de energía limitada. Un nodo sensor tiene que contar con un procesador de consumo ultra bajo así como de un transceptor radio con la misma característica. A esto hay que agregar un software que también conjugue esta característica haciendo el consumo aún más restrictivo.

- Limitaciones hardware: Para que el consumo se lo más bajo posible, se hace indispensable que el hardware sea lo más sencillo posible, así como su transceptor

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

radio, esto nos deja una capacidad de proceso limitada. Por ejemplo, la radio utilizada tiene que emplear un protocolo de bajo consumo de energía.

- Costes de producción: Dada que la naturaleza de una red de sensores tiene que ser en número muy elevada, para poder obtener datos con fiabilidad, los nodos sensores una vez definida su aplicación, son económicos de producir si son fabricados en grandes cantidades.

1.2.- Técnica de agregación de datos

Las redes inalámbricas de sensores están constituidas por una gran cantidad de nodos, y por lo tanto es impracticable utilizar algoritmos de encaminamiento centrados en direcciones de red (*address –centric*), en los cuales es necesario identificar cada nodo unívocamente [3]. Por este motivo, los algoritmos que se utilizan habitualmente en redes de sensores inalámbricas son centrados en datos (*data-centric*). En los protocolos centrados en datos, generalmente se busca aplicar algún tipo de procesamiento en los nodos de forma de reducir el tiempo de transmisión [5].

En este sentido, se han desarrollado técnicas de agregación de datos, con el fin de lograr el mensaje más compacto posible que represente al conjunto de mensajes individuales. El mensaje agregado se transmite a través de la red en lugar del conjunto de mensajes procesados [5].

Al incorporar agregación, los algoritmos prolongan la vida útil de la red, reduciendo la cantidad de mensajes a difundir. La reducción del consumo de energía depende en gran medida de la aplicación a desarrollar, pues son sus características las que determinan la función de agregación a utilizar. Algunas aplicaciones admiten agregados sencillos como la obtención del valor máximo, mínimo o promedio. El estudio de otros fenómenos en cambio, requiere de la transmisión de todos los valores obtenidos o gran parte de ellos. Estos últimos casos son los más desafiantes a la hora de diseñar alguna técnica de agregación que permita cumplir con los requerimientos de la aplicación [6].

Si bien el ahorro de energía es un requerimiento fundamental para el diseño de una técnica de agregación, se evalúan también otros aspectos los algoritmos que la utilizan son evaluados también en otros aspectos [7]:

- Vigencia de los datos: diferencia entre el momento de obtención del dato y el de llegada a la estación base.
- Proporción de lecturas recibidas: Cantidad de lecturas recibidas por la estación base respecto del número total de lecturas realizadas.
- El peso que se le dé a cada requerimiento dependerá de la aplicación particular.

El problema a abordar será el que determine la magnitud de error aceptable, el período admisible de espera para la llegada de los datos (latencia) o el porcentaje de mensajes que pueden perderse, sin afectar gravemente a la eficiencia esperada para la aplicación.

A lo largo del desarrollo de este proyecto fin de carrera vamos a aplicar esta técnica de agregación de datos, mejorando como bien hemos dicho la eficacia de los modelos de enrutamiento para la comunicación en redes sensores. Se trata de una técnica de optimización que divide la red en zonas. Los nodos se agruparán en regiones con un líder, responsable de enviar la información desde esa zona a los líderes de otras zonas o a la estación base. Los datos viajan desde los sensores “inferiores” de una zona a los sensores “superiores” o líderes de la misma lo que nos permite cubrir mayores distancias y que los datos se envíen a mayor velocidad. Por esto este modelo es especialmente adecuado para grandes superficies con un número elevado de sensores.

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

Clasificación de las técnicas de agregación

Este trabajo se enfoca a aplicaciones donde es habitual realizar una adquisición periódica de datos, con el fin de mantener un estado de situación actualizado del fenómeno a medir. Se habla de "rondas de recolección o medida", donde los nodos obtienen los valores de las variables de interés y los transmiten hacia la estación base. Estas "rondas" se repiten en períodos cuya duración queda determinada por la naturaleza del fenómeno bajo estudio. En general, podemos clasificar las técnicas de agregación periódicas como [4] [7]:

- Periódica simple: En este tipo de algoritmos, cada nodo espera un período fijo, agrega los datos recibidos en un único paquete de datos y transmite. Si algún nodo se retrasa en el envío de datos o falla, su información no formará parte del agregado enviado hacia la estación base. Un ejemplo muy conocido es el algoritmo de difusión dirigida [8].
- Periódica por salto: En estos algoritmos, cada nodo sabe quiénes de entre sus vecinos le enviarán información (nodos hijos). Entonces, los nodos agregan los datos recibidos y los envían cuando han colectado la información de todos sus hijos. En caso de que algún hijo no informe, se espera un período predeterminado, y si no se obtiene respuesta, se difunde el agregado logrado hasta ese momento. El algoritmo que proponemos en este trabajo pertenece a esta categoría.
- Periódica por salto ajustado: Estos algoritmos utilizan el mismo principio que los de agregación periódica por salto, pero incorporan un esquema de retardos para la difusión de los mensajes, basado en la posición de cada nodo en el árbol de distribución. El algoritmo propuesto por Solís [7] cae dentro de esta categoría.

1.3.- Trabajo a desarrollar

El objetivo primordial de este proyecto se centra en el estudio de los mecanismos para garantizar la autoconfiguración de redes inalámbricas de sensores y su tolerancia y recuperación frente a fallos.

Nuestro trabajo en este proyecto va a consistir en la implementación de tres algoritmos jerárquicos de elección del agregador y agregación de datos en grandes WSN basados en detector de fallos Omega propuesto por Chandra en [14]. Los algoritmos en cuestión los estudiaremos en el capítulo 3 y una vez los hayamos implementado, realizaremos un análisis comparativo de los resultados obtenidos por cada uno de ellos.

Finalmente, observaremos cuál de ellos es el que mejores resultados proporciona en diferentes escenarios de trabajo y comprobaremos si nos proporcionan sus dos objetivos fundamentales: tolerancia a fallos en la red y la minimización del consumo de energía por parte de los sensores para aumentar su tiempo de vida.

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

2.- ENTORNO DE TRABAJO

2.1.- Introducción

En este capítulo se va a realizar una descripción de los componentes, tanto hardware como software, que han sido necesarios para llevar a cabo el presente proyecto.

En cuanto al software, se ha utilizado el sistema operativo TinyOS, especialmente diseñado para las características intrínsecas de los nodos sensores y sus aplicaciones de red potenciales, y en él programaremos aplicaciones para los nodos sensores en el lenguaje de programación nesC. En el laboratorio se ha trabajado con un equipo dotado con Windows XP y un kit de desarrollo comercial de Crossbow. Dado que TinyOS requiere para su correcto funcionamiento del uso de diversos comandos y funcionalidades de Linux, ha sido necesaria la instalación de la herramienta Cygwin, encargada de emular un entorno Linux en Windows.

2.2. - Hardware: kit Crossbow

2.2.1. Resumen del producto

Para la realización de este proyecto hemos elegido unos sensores de la compañía Crossbow, una de las más extendidas y conocidas en cuanto al desarrollo de kits de hardware para la investigación en redes inalámbricas de sensores.

Concretamente, las motas elegidas se denominan IRIS (ver figura 2). Las motas IRIS pertenecen a la compañía Crossbow Technology. El IRIS o XM2110 (2400 MHz a 2483,5 MHz) utiliza el Atmel RF230, compatible con el estándar IEEE 802.15.4, con un transceptor de radiofrecuencia integrado listo para comunicaciones ZigBee y con un micro-controlador Atmega128. Estas mejoras brindan hasta tres veces más de radio alcance y el doble de memoria de programa en comparación con la generación anterior de motas MICA. Al igual que la familia de motas MICA, cuentan con un conector de expansión de 51 pines, y utilizan la misma memoria flash de serie; todo el software de aplicación y las placas de sensores son compatibles con el XM2110.

Las características del producto incluyen:

- Un radio de alcance tres veces superior y el doble de memoria de programa en comparación con la generación anterior de motas MICA.
- Pruebas de línea de visión al aire libre han alcanzado rangos de más de 500 metros entre nodos sin amplificación.
- Transceptor RF compatible con el estándar IEEE 802.15.4.
- Entre 2.4 y 2.48 GHz, globalmente compatible con la banda ISM.
- Secuencia directa de radio espectro extendido que es resistente a las interferencias de la radiofrecuencia y ofrece seguridad en los datos.
- Alta velocidad de transmisión de datos: 250 kbps.
- Compatible con la plataforma de redes inalámbricas de sensores Mote Works para redes de mallas ad-hoc fiables.
- Plug & Play con las placas sensoras, las placas de adquisición de datos, las estaciones bases y el software de Crossbow.

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

- Conector de expansión para la medición de luz, temperatura, humedad relativa, presión, aceleración / sísmica, acústica, magnetismo y para conectar otras placas sensores de la misma compañía.

MoteWorks permite el desarrollo de aplicaciones de sensores de encargo y es una herramienta específicamente optimizada para el bajo consumo de energía en redes que funcionan con baterías. MoteWorks se basa en el sistema operativo de código abierto de TinyOS y proporciona fiabilidad, redes de malla ad-hoc, capacidades de programación remota, herramientas de desarrollo de Crossbow, servidor middleware para la integración de redes empresariales y cliente de interfaz para el análisis y la configuración.

Figura 2: *Foto de una mota XM2110-IRIS con una antena estándar.*

2.2.2.- Diagrama de bloques del XM2110-IRIS

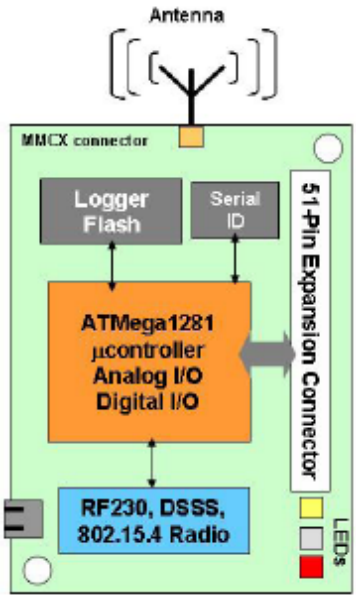


Figura 3: *Diagrama de bloques de un IRIS.*

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

2.2.3.- Procesador y radio

El XM2110CA se basa en el procesador de Atmel ATmega1281. El ATmega1281 es un microprocesador de bajo consumo, el cual ejecuta MoteWorks desde su memoria flash interna.

El procesador XM2110 puede ser configurado para ejecutar la aplicación del sensor / procesamiento y las comunicaciones de red / radio de forma simultánea. El conector de expansión de 51 pines soporta entradas analógicas, entradas y salidas digitales, e interfaces I2C, SPI y UART. Estas interfaces permiten conectar al IRIS una amplia variedad de periféricos externos.

2.2.4.- Sensores o placas sensoras

Son las placas que tienen los mecanismos de medición (termómetros, medidores de presión, detectores de luz, etc.) que se conectan a la mote a través del conector de 51 pines (hembra en el sensor, macho en la mote).

Crossbow ofrece una gran variedad de placas sensoras y de adquisición de datos para las motas IRIS. Todas estas placas se conectan a la mota vía el conector de expansión estándar de 51 pines. Otras placas sensoras y de adquisición de datos más complejas también son compatibles.

Las placas sensoras utilizadas en la realización de este proyecto han sido las MDA100. La figura 4 muestra una placa sensora MDA100.

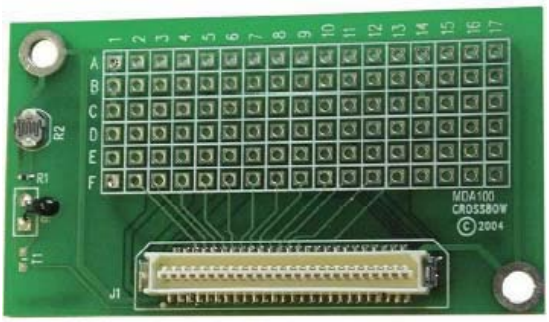


Figura 4: *Placa sensora MDA100*

2.2.5.- Batería

Todas las motas IRIS están diseñadas para la alimentación por batería. Esta batería consiste en dos pilas AA; sin embargo, se puede utilizar cualquier combinación de pilas (células AAA, C, D, etc.) siempre y cuando la salida esté entre 2,7 VDC y 3,6 VDC.

Un interruptor mecánico a un lado del dispositivo proporciona la capacidad de apagar y encender la batería.

3.2.6.- Memoria flash

Las motas están dotadas de una memoria flash de 4Mbits (Atmel AT45DB041) para almacenar datos, mediciones y otro tipo de información definida por el usuario. Está conectada a uno de los USART del ATmega128L/1281. Este chip es compatible con TinyOS que usa este chip como un mini sistema de archivos. El dispositivo de memoria flash es compatible con más de 100.000 lecturas de medición. Este chip también se utiliza para los servicios de programación remota disponibles en TinyOS.

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

2.2.7.- Estación base USB MIB520

La estación base permite la agregación de los datos de los sensores de la red en un PC o en cualquier otra plataforma. Cualquier mota IRIS puede funcionar como estación base cuando está conectada a una interfaz estándar de un PC o a una estación base. Las estación base elegida en nuestro caso es la MIB520. Se alimenta mediante el bus USB del ordenador y proporciona una interfaz USB tanto para la programación como para la comunicación de los datos. La placa programadora MIB520 utiliza un conector de expansión de 51 pines como vía de comunicación con los motes. La placa nos permite cargar código en los motes y hace la función de puerta de enlace entre los motes y el PC. La figura 5 muestra el aspecto de la placa MIB520.

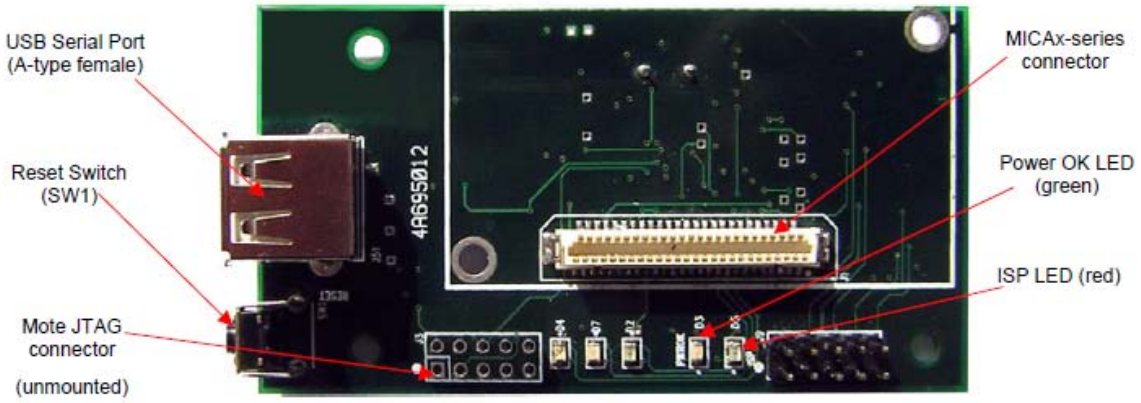


Figura 5: *Fotografía de la parte de arriba de una MIB520CA.*

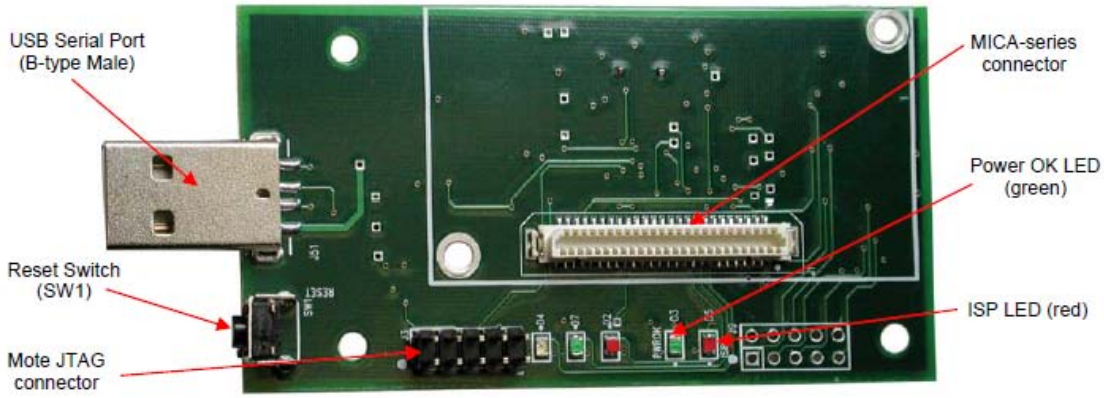


Figura 6: *Fotografía de la parte de abajo de una MIB520CA.*

Las especificaciones de esta interfaz, a través del puerto serie, muestran una velocidad de transferencia de 57.6 kbaudios para las comunicaciones con los motes y el PC, y una velocidad de programación de 115.2 kbaudios.

El programador incorpora tres LEDs: amarillo, verde y rojo, que nos indica si la placa está conectada a la corriente eléctrica (verde) y si se están cargando los datos en el mote acoplado desde el PC (rojo).

También incorpora un interruptor para deshabilitar la transmisión serie con los motes, y un botón para reiniciar la placa programadora y el mote acoplado. La placa se alimenta de 5 voltios y unos 50mA de corriente. El mote acoplado no necesita alimentación ya que se lo cede la misma MIB520, que se encuentra conectada al PC.

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

2.2.- SOFTWARE

2.2.1 Sistema operativo

Para la programación de motas contamos con diversos sistemas operativos de los cuales aquí enumeramos los más conocidos:

- Bertha (pushpin computing platform) Una plataforma de software diseñada e implementada para modelar, testear y desplegar una red de sensores distribuida de muchos nodos idénticos.
- Nut/OS: Es un pequeño sistema operativo para aplicaciones en tiempo real, que trabaja con CPUs de 8 bits.
- Contiki: Es un Sistema Operativo de libre distribución para usar en un limitado tipo de computadoras, desde los 8 bits a sistemas embebidos en microcontroladores, incluidas motas de redes inalámbricas.
- CORMOS: A Communication Oriented Runtime System for Sensor Networks, específico para redes de sensores inalámbricos como su nombre indica.
- eCos: (embedded Configurable operating system) es un sistema operativo gratuito, en tiempo real, diseñado para aplicaciones y sistemas embebidos que sólo necesitan un proceso. Se pueden configurar muchas opciones y puede ser personalizado para cumplir cualquier requisito, ofreciendo la mejor ejecución en tiempo real y minimizando las necesidades de hardware.
- EYESOS: se define como un entorno para escritorio basado en Web, permite monitorizar y acceder a un sistema remoto mediante un sencillo buscador.
- MagnetOS: es un sistema operativo distribuido para redes de sensores o adhoc, cuyo objetivo es ejecutar aplicaciones de red que requieran bajo consumo de energía, adaptativas y fáciles de implementar.
- MANTIS (MultimodAl NeTworks In-situ Sensors)
- TinyOS: Sistema Operativo desarrollado por la universidad de Berkeley, se hablará de él en el próximo apartado.
- t-Kernel: es un sistema operativo que acepta las aplicaciones como imágenes de ejecutables en instrucciones básicas. Por ello, no importará si está escrito en C++ o lenguaje ensamblador.
- LiteOS: Sistema operativo desarrollado en principio para calculadoras, pero que ha sido también utilizado para redes de sensores.

2.2.1.1.- TinyOS

TinyOS es un sistema operativo de código libre basado en eventos y destinado para utilizarse en sensores.

Después de introducirse en el mundo de las redes de sensores con el primer capítulo de este trabajo, es fácil darse cuenta de la utilidad y necesidad de un sistema operativo adaptado a sensores y basado en eventos.

La actividad de un nodo sensor está basada en los impulsos externos que recibe: mensajes radio de otros nodos sensores, detección de sonido, detección de una temperatura más alta de lo habitual y que debe generar un mensaje de alerta que se propague por la red, etc.

TinyOS utiliza un modelo de programación basado en el concepto de "*wiring*" (enlazar o cablear) componentes software para producir un programa final. Este modelo de programación

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

además, pone requisitos sobre cómo deben ser escritos los programas. TinyOS, siguiendo los patrones de las redes de sensores, tiene en cuenta que puede haber muy pocos recursos disponibles (por ejemplo, 512 bytes de RAM) y que esto requiere una utilización de los recursos muy eficiente. Otro requerimiento tiene que ver con el concepto *wiring*, los programas deben ser capaces de mapear una sola llamada a función (input wire) para que sean llamadas múltiples funciones (output wires).

Además, TinyOS utiliza frecuentemente macros del preprocesador de C para permitir modos de compilación alternativos (como pueden ser simuladores).

La página en Internet donde se puede descargar el sistema operativo de distribución libre TinyOS es www.tinyos.net, además en esa página se encuentra la comunidad de desarrolladores.

Modelo de componentes

En TinyOS la distinción entre el sistema operativo en sí y las aplicaciones es meramente semántica; ambas cosas se compilan de forma conjunta y comparten un mismo espacio de memoria. A medida que los componentes de aplicación son probados, se vuelven más estables y crecen para ser componentes que utilicen otras aplicaciones, pueden acabar convirtiéndose en componentes del sistema.

Los componentes de TinyOS pueden dibujarse formando una pirámide invertida: encima de todo se encuentran los componentes del nivel de aplicación, mientras que la capa más baja la ocupan los componentes que se asientan directamente sobre el hardware.

TinyOS presenta tres abstracciones de computación: los eventos, los comandos y las tareas.

Los comandos representan “llamadas hacia abajo” en la pirámide de componentes; es decir, un componente llama a comandos que pertenecen a componentes que se encuentran por debajo de él en la pirámide.

Los eventos representan “llamadas hacia arriba” en la pirámide de componentes; es decir, un componente advierte que ha sucedido un evento a componentes que se encuentran por encima de él en la pirámide.

La forma más precisa de llamar a los eventos sería mediante capturadores de eventos, pues como ya indicábamos anteriormente, un evento propiamente dicho es un suceso externo que tiene repercusión en la actividad de un nodo sensor.

Es importante diferenciar que los capturadores de eventos pueden llamar a comandos, pero un comando nunca puede señalizar un evento.

Las tareas son un mecanismo para computación asíncrona de larga duración. Una tarea se ejecuta síncronamente respecto a otras tareas, todas tienen la misma prioridad y se ejecutan en orden. Sin embargo, una tarea puede verse obligada a abandonar el procesador si llega un evento de mayor prioridad. Por lo tanto, si realizamos tareas con requisitos de tiempo real, éstas deben ser cortas para evitar que puedan ser interrumpidas por sucesivos eventos.

También es crítico que el código invocado por un capturador de eventos sea lo más corto posible. Esto se debe a que normalmente se ejecutan como resultado de una interrupción, por tanto es posible que no puedan ser interrumpidos por otras interrupciones posteriores hasta su finalización, manteniendo detenida la ejecución del código principal y manteniendo en espera interrupciones que podrían ser críticas para la aplicación.

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

2.2.2.- Lenguajes de programación para motas

La programación de sensores es relativamente compleja, entre otras dificultades está la limitada capacidad de cálculo y la cantidad de recursos. Y así como en los sistemas informáticos tradicionales se encuentran entornos de programación prácticos y eficientes para generar y depurar código, incluso en tiempo de ejecución, en estos microcontroladores todavía no hay herramientas comparables.

Podemos encontrar lenguajes como:

- nesC: lenguaje que utilizamos para nuestras motas, y que está directamente relacionado con TinyOS.
- Protothreads: específicamente diseñado para la programación concurrente, provee hilos de dos bytes como base de funcionamiento.
- SNACK: facilita el diseño de componentes para redes de sensores inalámbricas, sobre todo cuando la información o cálculo a manejar es muy voluminoso, complicado con nesC, este lenguaje hace su programación más eficiente. Luego es un buen sustituto de nesC para crear librerías de alto nivel a combinar con las aplicaciones más eficientes.
- C@t: iniciales que hincan computación en un punto del espacio en el tiempo (Computation at a point in space (@) Time)
- DCL: Lenguaje de composición distribuido (Distributed Compositional Language)
- GalsC: diseñado para ser usado en TinyGALS, es un lenguaje programado mediante el modelo orientado a tarea, fácil de depurar, permite concurrencia y es compatible con los módulos nesC de TinyOS • SQTL (Sensor Query and Tasking Language): como su nombre indica es una interesante herramienta para realizar consultas sobre redes de motas.

De todos los lenguajes de programación citados anteriormente, en este proyecto se utiliza nesC, puesto que es el lenguaje de programación que utiliza TinyOS y el cual se describirá brevemente a continuación.

2.2.2.1.- NESC

nesC (Network Embedded Systems C) es un dialecto del lenguaje de programación C optimizado para las limitaciones de memoria de las redes de sensores. Existen varias herramientas que completan y facilitan su uso, escritas en su mayoría en Java y en Bash. Otras herramientas y librerías asociadas están escritas principalmente en C.

nesC es un lenguaje orientado a componentes y está especialmente diseñado para programar aplicaciones sobre redes de sensores, en particular en el sistema operativo TinyOS [16].

Originalmente TinyOS estaba programado en C (hasta su versión 0.6), sin embargo, en la transición entre la versión 0.6 y la 1.0 se reimplementó todo el sistema operativo en nesC.

Las ventajas que proporcionaba nesC eran: aparición de las interfaces (de las que se hablará posteriormente), detección de errores de "*wiring*", generación automática de documentos (similar a javadoc), y mayor facilidad para la optimización del código significativo del sistema operativo respecto al anterior modelo de TinyOS. Su mayor éxito fue conseguir que el código de TinyOS fuese más limpio, más sencillo de entender y más sencillo de escribir.

TinyOS define un número importante de conceptos que tienen su correspondencia en el lenguaje de programación nesC.

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

En primer lugar, las aplicaciones nesC están construidas por componentes con interfaces bidireccionales y bien definidas. En segundo lugar, nesC define un modelo concurrente, basado en tareas y capturadores de eventos y capaz de detectar "*data races*" en tiempo de compilación.

Componentes

Una aplicación nesC consiste en uno o más componentes enlazados entre ellos para formar un ejecutable; un componente proporciona y utiliza interfaces, estas interfaces representan el único punto de acceso a un componente y son bidireccionales. Una interfaz declara un conjunto de funciones llamadas comandos que el proveedor de la interfaz debe implementar, y otro conjunto de funciones llamadas eventos que, en este caso, deben ser implementadas por el usuario de la interfaz.

Un componente puede utilizar o proporcionar múltiples interfaces y múltiples instancias de una misma interfaz (por ejemplo, un componente que necesita dos temporizadores deberá instanciar dos veces a la interfaz que proporcione el temporizador).

Existen dos tipos de componentes en nesC: módulos y configuraciones. Los módulos proporcionan el código de la aplicación, implementando una o varias interfaces. Las configuraciones se utilizan para ensamblar otros componentes conjuntamente, conectando las interfaces utilizadas por los componentes con las interfaces proporcionadas por otros. A esto se le llama "*wiring*", un concepto que de forma abstracta se introdujo en el apartado sobre TinyOS.

Cada aplicación nesC requiere de una configuración de alto nivel que enlace entre sí los componentes utilizados. NesC utiliza la misma extensión de archivo (.nc), para todos sus archivos, es decir, tanto para interfaces como para configuraciones y módulos. Sin embargo, se proponen unas convenciones de nombre para diferenciar unos componentes de otros.

Modelo concurrente

TinyOS ejecuta únicamente un programa consistente en los componentes de sistema y los componentes propios necesarios para que funcione la aplicación.

Existen dos hilos de ejecución: las tareas y los capturadores de eventos. Si recordamos, los capturadores de eventos lanzaban la ejecución de un código vinculado al evento ocurrido y/o llamaban a comandos que se encargaban así mismo de tareas específicas vinculadas al evento.

El lenguaje nesC debe tener conocimiento de qué comandos y eventos serán ejecutados como parte de un capturador de eventos, para ello en su definición se les debe declarar con la palabra clave *async*.

Debido a que las tareas y los capturadores de eventos pueden ser detenidos para permitir la ejecución de código asíncrono de mayor prioridad, los programas en nesC son susceptibles a padecer *data races*. Las condiciones de aparición de *data races* pueden evitarse mediante el acceso a datos compartidos exclusivamente en el interior de las tareas o bien encapsulando todos los accesos dentro de secciones *atomic*.

Las secciones *atomic* aseguran la ejecución de todo el código que encapsulan, si llega una interrupción, ésta esperará hasta que la ejecución abandone la sección *atomic* para ejecutarse. El compilador de nesC advierte de *data races* potenciales en tiempo de compilación para que el programador pueda rediseñar su código.

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

3.- ALGORITMOS DE AGREGACIÓN

La contribución de este proyecto fin de carrera consiste en la implementación y estudio de tres algoritmos para la elección del agregador y para la agregación de datos en una red inalámbrica de sensores donde los sensores pueden fallar y recuperarse. La red está dividida en varias regiones. Los algoritmos garantizan la elección de un agregador común de datos dentro de cada región, el cual se encargará de la recogida de los datos locales y de la elección de un único sensor súper-agregador, encargado a su vez de la recogida de datos globales, entre todos los agregadores. Ambas elecciones se logran implementando el detector de fallos Omega [14], que proporciona un servicio de elección de líder auto-configurable y tolerante a fallos. Cada algoritmo está basado en una hipótesis de conectividad diferente. El primer algoritmo asume que todo par de sensores en una región pueden comunicarse directamente. El segundo algoritmo solamente requiere algunos sensor(es) correctos para comunicarse directamente con el resto de los sensores. Finalmente, el tercer algoritmo solamente requiere la existencia de un camino *multi-hop* bidireccional desde algunos sensor(es) correctos hasta el resto de los sensores. Nosotros además introducimos un umbral de agotamiento de la batería para mejorar la calidad de servicio (QoS) de la red inalámbrica de sensores.

3.1.- Introducción

3.1.1.- Antecedentes y trabajos relacionados

Las redes inalámbricas de sensores (WSN) proporcionan la recogida fiable de datos en las aplicaciones reduciendo al máximo la intervención humana (auto-configuración y auto-mantenimiento). Las aplicaciones pueden actuar de manera autónoma sobre una red de sensores configurando los sensores remotamente y recuperando la información recogida por los sensores periódicamente o bajo demanda. Las redes inalámbricas de sensores pueden estar expuestas a varias fuentes de problemas como la medición, la comunicación, los accidentes y/o los errores de alimentación. La redundancia de los dispositivos permite obtener redundancia en la información, y por lo tanto garantiza un cierto nivel de tolerancia a fallos. Nosotros consideramos una red de sensores inalámbricos con un cierto grado de agrupamiento donde cada grupo se centra en áreas específicas y recoge información siguiendo un esquema de detección distribuida.

Las aplicaciones sobre redes de sensores inalámbricas deben asegurar una recogida y agregación fiable de los datos, mientras satisfacen el menor coste y las limitaciones de funcionamiento de bajo consumo de dichas aplicaciones [1]. El intento de minimizar el consumo de energía pasa por minimizar la cantidad de datos transmitidos mediante el uso de la agregación de datos. Algunos trabajos se centran en la implementación de protocolos de agregación eficientes en el tiempo como ocurre en [11]. En [17] se consideran tres esquemas diferentes de agregación de datos : *in-network, grid-based e hybrid data*. El objeto de estos esquemas es incrementar el rendimiento de proceso, reducir la congestión y ahorrar energía. Otros trabajos manejan la agregación de datos con la ayuda de un algoritmo en consenso como se hace en [18]. Los datos de los sensores pueden ser recogidos y agregados por ciertos nodos de la red de sensores llamados líderes o agregadores [19]. Sin embargo, esta obra no se ocupa de la selección de los agregadores, sino que simplemente asume que existen. La selección de los nodos agregadores se analiza en [10], donde se presenta un protocolo jerárquico de selección del agregador de energía eficiente. El protocolo es probabilístico y no considera el fallo de los sensores. Cualquier esfuerzo encaminado a prolongar la vida útil de la red requiere tanto al sensor en sí mismo, como la estrategia de colaboración que coordina los nodos en la tarea de detección, la mayor eficiencia energética posible.

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

Con respecto al sensor de comunicación en sí mismo, algunos mecanismos de prevención de colisiones que se pueden considerar son TDMA, CDMA y CSMA. Los principales inconvenientes de TDMA y las técnicas de CDMA están relacionados con los requisitos de sincronización de un sensor y de una autoridad central encargada de asignar las franjas horarias, y con el complejo hardware de modulación (que es difícil de implementar debido a la reducción del tamaño y las prestaciones de los sensores) respectivamente. El uso de protocolos CSMA tales como IEEE 802.11 (WiFi), permite reducir al mínimo el consumo de energía debido a las colisiones en las WSN sin necesidad de capacidades especiales y complejidad en los sensores.

La selección de un agregador puede ser considerada un problema de elección de líder, que ha sido ampliamente estudiada en la literatura, por ejemplo, en [12]. Para nuestro conocimiento, el primer algoritmo de elección de líder con un miembro desconocido para distribuir sistemas propensos a fallar ha sido propuesto en [19]. Podemos encontrar también varios trabajos que se centraron en la jerarquización y la agrupación de las redes inalámbricas ad hoc, por ejemplo, [20] y [21], que podrían adaptarse al problema de la elección de líder. Estos trabajos están orientados a un modelo de fallo-recuperación. En[22], donde se presenta un sistema de comunicación eficiente probabilístico (que podría ser usado para la elección del líder) se consideran los frecuentes cambios de conectividad de la red, resultando posiblemente en particiones de red. Del mismo modo, el fallo y la recuperación del sensor también se consideran.

Las redes móviles ad hoc han introducido un nuevo aspecto en el problema de elección de líder: la movilidad. En [23] se presentan protocolos de elección de líder de un único salto para redes móviles. En [24] se proponen dos algoritmos multi-hop de elección de líder, con base [25], donde cualquier componente cuya topología es estática durante un tiempo suficientemente largo tiene exactamente un líder. Por último, los algoritmos propuestos en [26] y [27] mejoran lo anteriores basándose en la difusión de los cálculos y de la mayoría de los procesos correctos, respectivamente.

3.1.2.- Algoritmos propuestos

Recordemos que este proyecto tiene como objetivo primordial el estudio de los mecanismos para garantizar la autoconfiguración de redes inalámbricas de sensores y su tolerancia y recuperación frente a fallos.

Nuestro trabajo en este proyecto va a consistir en implementar tres algoritmos jerárquicos de elección del agregador y agregación de datos en grandes WSN basados en detector de fallos Omega propuesto por Chandra en [14]. Debido a los cambios en el conjunto de los sensores alcanzables, el sensor agregador de cada región puede cambiar en el tiempo. Los algoritmos garantizan que todos los sensores de una región están de acuerdo en un agregador común y que todos los agregadores están de acuerdo en un super-agregador común. Cada algoritmo está basado en una hipótesis de conectividad diferente:

- El primer algoritmo asume que todo par de sensores en una región puede comunicarse directamente.
- El segundo algoritmo relaja esa hipótesis, solamente requiere algunos sensores correctos para comunicarse directamente con el resto de los sensores.
- Finalmente, el tercer algoritmo va un paso más lejos y no requiere que haya ningún sensor conectado directamente con el resto, pero solamente requiere la existencia de un camino multi-hop bidireccional desde algunos sensores correctos hasta el resto de los sensores.

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

Mediante la implementación de estos tres algoritmos pretendemos alcanzar ese doble objetivo que comentamos al inicio de este capítulo. Además, observaremos cuál es el algoritmo que mejores resultados proporciona.

Una vez hayamos implementado los tres algoritmos propuestos y que estudiaremos a continuación, realizaremos un análisis comparativo de los resultados obtenidos por cada uno de ellos. En primer lugar, comprobaremos si nos proporcionan sus dos objetivos fundamentales: tolerancia a fallos en la red y la minimización del consumo de energía por parte de los sensores para aumentar su tiempo de vida.

3.2.- Modelo del sistema

Vamos a considerar una red inalámbrica de sensores donde nosotros intentamos recoger los datos de los sensores minimizando el consumo de energía. Esta red está dividida en diferentes regiones donde cada sensor conoce a priori su región de operación y se ocupa de transmitir los datos que ha percibido a un sensor agregador, el cual está encargado de recoger los datos de todos los sensores de una región específica. Cada sensor tiene un identificador de su región de operación el cual es difundido en sus mensajes. Este identificador permite a los sensores rechazar mensajes de entrada procedentes de sensores pertenecientes a otras regiones. La región de operación de un sensor puede ser cambiada bajo demanda, así como la propia delimitación de la región. La comunicación de los sensores cumple el protocolo 802.11 (WiFi).

En este proyecto vamos a utilizar el detector de fallos Omega para elegir un agregador común dentro de cada una de las regiones de acuerdo a la fiabilidad y a la batería disponible de los sensores. Un sensor es candidato a ser elegido como líder o agregador si es un sensor fiable sin errores durante cada período de operación. Omega asegura que entre todos los sensores sólo uno será elegido como agregador. Cómo ya hemos dicho, este agregador será el encargado de recolectar los datos en su área de cobertura.

Más formalmente, nosotros vamos a considerar un sistema *S* compuesto por un conjunto finito de sensores que se comunican transmitiendo mensajes a través de la red inalámbrica. Se considera un modelo de fallo crash&recovery, es decir, los sensores pueden recuperarse después de fallar. Nosotros vamos a considerar como un caso particular de fallo el agotamiento de la batería del sensor, siendo su recuperación asociada el hipotético recambio de la batería. La hibernación de un sensor no se considerad un fallo, ya que es una tarea planificada. De acuerdo a lo anterior, y durante la vida útil de su batería, un sensor puede ser:

1. *Eventually up*. Es un sensor que, después de fallar y recuperarse un determinado número de veces, se mantiene funcionando sin fallar hasta que su batería se agota. Los sensores que nunca fallan pertenecen a esta clase.
2. *Eventually down*: Es un sensor que, después de fallar y recuperarse un número finito de veces, no se vuelve a recuperar hasta que se le agote la batería. Los sensores que nunca comienza su ejecución pertenecen a esta clase.
3. *Unestable*. Es un sensor que está continuamente fallando y recuperándose hasta que se le agota la batería.

Por definición, los sensores *eventually up* son correctos, y los sensores *eventually down* y *unestable* son incorrectos. Asumiremos que el número de sensores correctos en el sistema es al menos uno. Como veremos, los sensores correctos serán candidatos a convertirse en líderes, es decir, en agregadores [1].

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

3.2.1 El detector de fallos Ω

En [14], Chandra, Hadzilacos y Toueg definen el detector de fallos Ω. La salida de un módulo detector de fallos Ω en un proceso *p* es un proceso simple *q*, que *p* actualmente considera que es correcto (nosotros decimos que *p confía en q).* El detector de fallos Ω satisface la siguiente propiedad:

Propiedad 1: A partir de un determinado instante de tiempo todos los procesos correctos siempre confían en el mismo proceso correcto.

La salida del módulo detector de fallos Ω en un proceso *p* puede cambiar a lo largo del tiempo, es decir, *p* puede confiar en diferentes procesos en diferentes tiempos. Además, dado un tiempo *t* al azar, dos procesos *p* y *q* pueden confiar en procesos diferentes.

Se debe observar que esta definición de Ω considera que al final el líder común lo es para siempre. En nuestro modelo de sistema, debido a la duración limitada de la batería de los sensores, esta definición no es adecuada para nuestro propósito. Redefiniremos Ω en la propiedad 2 considerando que finalmente el líder común lo es hasta que se le agote la batería.

Propiedad 2: Hay un tiempo tras el cual todos los sensores correctos siempre confían en el mismo sensor correcto agregador hasta que se le agota la batería [1].

3.3.- Nivel local (intra-regional)

En esta sección, vamos a presentar de nuevo los tres algoritmos que hemos citado anteriormente. Estos tres algoritmos de elección del agregador y agregación de datos para el nivel intra-regional son los siguientes:

- El primer algoritmo asume que todo par de sensores en una región puede comunicarse directamente.
- El segundo algoritmo asume que algunos sensor(es) correctos pueden comunicarse directamente con el resto de los sensores de la región.
- El tercer algoritmo asume la existencia de un camino multi-hop bidireccional desde algunos sensores correctos hasta el resto de los sensores de la región.

3.3.1.- Primer algoritmo

Nosotros presentamos aquí el primer algoritmo de elección del agregador y agregación de datos para el nivel local, en el cual asumimos que los sensores se despiertan periódicamente para proveer sus datos recogidos e hibernan el resto del tiempo. La hibernación de un sensor no se considera un fallo, ya que es una tarea planificada. Esta suposición implica un cambio programado de encendido/apagado de los sensores con la ayuda de un reloj, es decir, cada $\Delta_{ACTIVATION}$ unidades de tiempo. Nosotros asumimos que los períodos de hibernación son más largos que los de actividad (ver figura 7). Asumimos un máximo desplazamiento de reloj (*clock skew*) ε entre cualquier par de sensores.

El algoritmo requiere que los identificadores de los sensores estén totalmente ordenados, pero no tienen por qué ser consecutivos. Además, los sensores no necesitan conocer por adelantado los identificadores del resto de los sensores. Asumimos que todos los sensores de una región pueden comunicarse directamente, existiendo un límite desconocido δ en el retraso de los mensajes. Igualmente, consideramos que la ejecución de cada línea de código del algoritmo requiere al menos σ unidades de tiempo.

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

En la figura 8 se puede ver el algoritmo en detalle. El algoritmo usa tanto memoria estable como volátil. El sensor elegido como agregador por un sensor *p*, es almacenado en la variable $leader_p$. La variable *incarnation* (número de encarnación), inicializada a 0, es incrementada durante la inicialización y cada vez que el sensor se recupera de un desplome. Las variables $incarnation_p$, $leader_p$, $incarnation_{leader}$ y $Timeout_p$ son almacenadas persistentemente mientras que la variable $scheduled\_wakeup_p$ permanece en memoria volátil. Además de esto, cada sensor tiene un control de tiempo local (*timer*) usado para detectar el desplome del sensor agregador. La variable $scheduled\_wakeup_p$ mantiene su valor durante los períodos de hibernación, mientras que si el sensor falla o se queda sin batería, la variable perderá su valor ($scheduled\_wakeup_p$). La constante $\Delta_{TIMEOUT}$ determina el crecimiento del *timeout* para alcanzar el acuerdo. Cuanto mayor sea este valor, el acuerdo sobre el agregador común será más rápido. Sin embargo, un valor excesivo de $\Delta_{TIMEOUT}$ puede inducir a los sensores a los sensores al gasto de sus baterías, y podría incluso retrasar la detección del fallo del agregador. La variable $Timeout_p$ es un crédito, una especia de margen que deja cada sensor a su agregador para recibir la confirmación de que efectivamente es agregador. Si este margen se sobrepasa, significa que algo no va bien y por lo tanto el sensor que está esperando confirmación se ofrecerá a ser agregador. La constante $\Delta_{DATA\_ACQUISITION}$ representa el máximo tiempo gastado por el agregador durante la recogida de los datos proporcionados por los sensores.

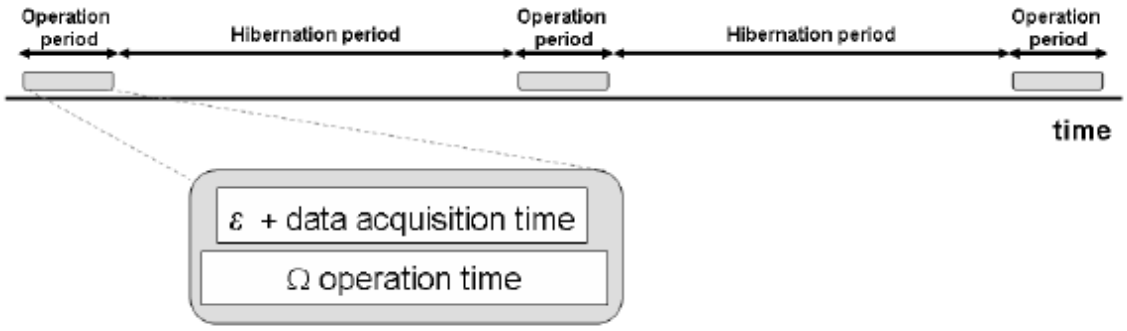


Figura 7: *Funcionamiento del sistema en la línea de tiempo.*

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

Every sensor p executes the following:

procedure *GoToHibernation*()

(1) write ($incarnation_p$, $leader_p$, $incarnation_{leader}$, $Timeout_p$) in stable storage

(2) $scheduled\_wakeup_p \leftarrow$ TRUE

(3) *hibernate*()

end procedure

Initialisation:

(4) read ($incarnation_p$) from stable storage

(5) if [$scheduled\_wakeup_p = FALSE$] then

(6) $incarnation_p \leftarrow incarnation_p + 1$

(7) write $incarnation_p$ in stable storage

(8) end if

(9) $scheduled\_wakeup_p \leftarrow FALSE$

(10) read ($leader_p$, $incarnation_{leader}$, $Timeout_p$) from stable storage

(11) if [$leader_p = p$] then

(12) start tasks 1 and 2

(13) else

(14) reset $timer_p$ to $Timeout_p + 2\epsilon$

(15) start tasks 2 and 3

(16) end if

Task 1

(17) wait ϵ time units

(18) broadcast (I-AM-ALIVE, p, $incarnation_p$)

(19) receive data from sensors during $\Delta_{DATA\_ACQUISITION}$ time

(20) *GoToHibernation*()

Figura 8: *Elección de agregador y agregación de datos de nivel intra-regional (Algoritmo 1)*

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

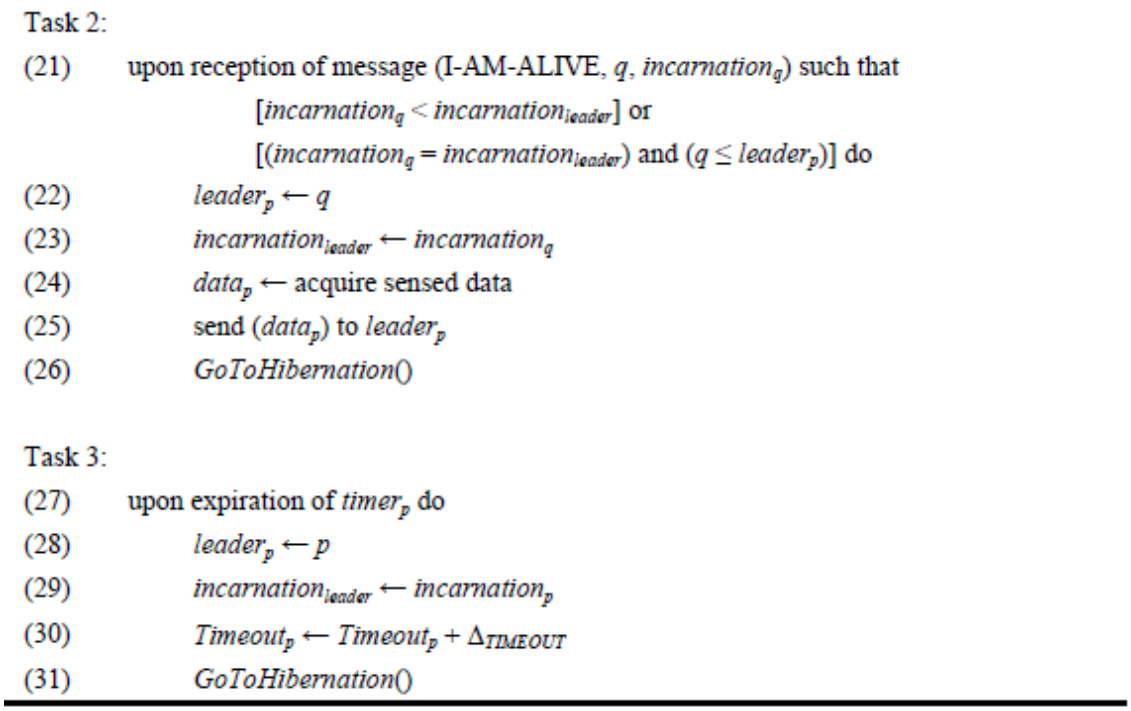


Figura 8: *Elección de agregador y agregación de datos a nivel intra-regional (Algoritmo 1) (continua)*

El algoritmo (ver figura 8) comienza con la *Inicialización* donde todos los valores de las variables son correctamente recuperados. Después de esto, si el sensor se considera a sí mismo el actual agregador ($leader_p=p$), el algoritmo empieza las tareas 1 y 2. Por otro lado, si el sensor no se considera a sí mismo agregador, el algoritmo resetea el *timer* local y comienza las tareas 2 y 3. Con la tarea 1 se anuncia a los demás sensores quien es el agregador y se recogen los datos que cada sensor proporciona. La tarea 2, que realizan todos los sensores, es utilizada para enviar los datos al agregador, y para cambiar el agregador si fuera requerido (si ha aumentado su número de encarnación). Finalmente, en la tarea 3 el sensor se propone a sí mismo como agregador cuando no ha recibido notificación por parte del agregador mientras no se ha agotado el *timer*, y además incrementa el timeout en $\Delta_{TIMEOUT}$ unidades de tiempo. Todas las tareas terminan llamando al procedimiento *GoToHibernation()*, que hace que comience el período de hibernación. Cada sensor permanecerá en este estado hasta la siguiente vez que le toque despertarse.

Vamos a denotar por c_{min} al sensor correcto de *S* con el identificador más bajo entre los que tienen el mínimo número de encarnaciones $incarnation_{min}$. Con este algotirmo, hay un tiempo tras el cual todo sensor p correcto tiene como líder a c_{min} hasta el final de sus baterías. Finalmente, solo el sensor c_{min} emite un nuevo mensaje (I-AM-ALIVE, c_{min}, $incarnation_{min}$) en cada período de operación, que reciben el resto de los sensores correctos.

En [1] se explica cómo siguiendo este algoritmo, hay un tiempo tras el cual cada sensor correcto tiene como líder a c_{MIN}, siendo c_{MIN} el sensor correcto con el identificador más pequeño con respecto a aquellos sensores que tienen un número de encarnación mínimo, igual al almacenado en $incarnation_{leader}$. Luego queda establecido el criterio para elección del líder: será aquel que de entre todos los que tiene un número de encarnación mínimo tiene el identificador más bajo.

El coste del algoritmo, medido como el número de mensajes enviados en estabilidad durante el período de adquisición de los datos, es lineal respecto al número de sensores de la región, desde que el sensor agregador emite un mensaje por la tarea 1 y el resto de sensores envían un mensaje al agregador por la tarea 2.

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

La demostración de corrección de este algoritmo puede consultarse en [1].

3.3.2.- Segundo algoritmo

Se presenta aquí el segundo algoritmo de elección del agregador y agregación de datos para el nivel local. Contrariamente al algoritmo de la sección anterior, se asume, que en general, no todo par de sensores de una región puede comunicarse directamente. Sin embargo, existe un subconjunto de sensores en la región que sí son capaces de alcanzar directamente al resto de los sensores de la región y además son también capaces de recibir mensajes emitidos por todos los sensores de la región (ver figura 9). Como ya hemos dicho, nosotros asumimos que existe un límite desconocido δ en el retraso de los mensajes y que la ejecución de cada línea de código del algoritmo requiere al menos σ unidades de tiempo. En este algoritmo, los sensores deben conocer por adelantado los identificadores del resto de los sensores.

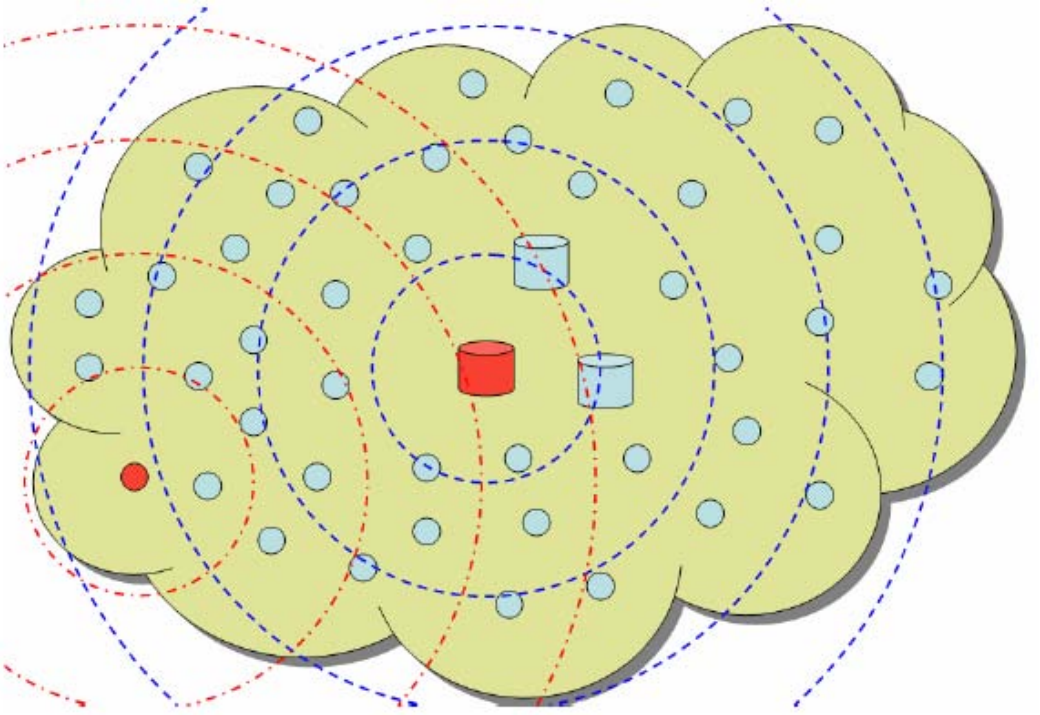


Figura 9: Distribución de los sensores dentro de una red
Nota: los cilindros representan los sensores bien comunicados, es decir, los candidatos a ser agregadores.

La figura 10 presenta el algoritmo en detalle. Con este algoritmo todos los sensores $p \in$ *correctos* tienen permanentemente $leader_p = l$, siendo l el sensor menos sospechoso entre los que pueden comunicarse directamente con el resto de los sensores de la región.

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

Every sensor p executes the following:

procedure *updateLeader*()
(1) $leader_p \leftarrow l$ such that $counter_p[l] = min\{counter_p\}$
end procedure

Initialisation:
(2) increment $incarnation_p$ by 1 in stable storage
(3) read ($incarnation_p$) from stable storage
(4) $\forall q \neq p : Timeout_p[q] \leftarrow \eta + incarnation_p$
(5) $\forall q \neq$ p : reset $timer_p(q)$ to $Timeout_p[q]$
(6) $\forall q \neq$ p : $counter_p[q] \leftarrow 0$
(7) $counter_p[p] \leftarrow incarnation_p$
(8) $leader_p \leftarrow p$
(9) start tasks 1, 2 and 3

Task 1:
(10) loop forever
(11) $data_p \leftarrow$ acquire sensed data
(12) broadcast (I-AM-ALIVE, p, $counter_p$, $data_p$)
(13) wait(η)

Task 2:
(14) upon reception of message (I-AM-ALIVE, q, $counter_q$, $data_q$) do
(15) reset $timer_p(q)$ to $Timeout_p[q]$
(16) $\forall r : counter_p[r] \leftarrow max\{(counter_p[r], counter_q[r])\}$
(17) *updateLeader*()
(18) if [$leader_p = p$] then
(19) collect $data_q$
(20) end if

Task 3:
(21) upon expiration of $timer_p(q)$ do
(22) $counter_p[q] \leftarrow counter_p[q] + 1$
(23) $Timeout_p[q] \leftarrow Timeout_p[q] + 1$
(24) reset $timer_p(q)$ to $Timeout_p[q]$
(25) *updateLeader*()

Figura 10: *Elección de agregador y agregación de datos a nivel intra-regiona (Algoritmo 2)*

El algoritmo trabaja como sigue. Todo sensor p tiene una variable $counter_p[q]$ para cada sensor, la cual representa la estimación de p sobre el número de veces que q ha sido sospechoso. El sensor p elige como su líder al sensor l con el valor de $counter_p[l]$ más pequeño. Con el fin de adquirir los datos del sensor y mantener la variable $counter_p$ hasta la fecha, todo sensor p emite cada η unidades de tiempo un mensaje (I-AM-ALIVE, p, $counter_p$, $data_p$), siendo η el intervalo de

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

tiempo transcurrido entre las mediciones de los sensores. Cuando un sensor *p* recibe un mensaje (I-AM-ALIVE, *q*, $counter_q$, $data_q$), resetea $timer_p$(q) para cuando se espera recibir el siguiente (I-AM-ALIVE, *q*, $counter_q$, $data_q$), actualiza acordemente su tabla $counter_p$ y llama al procedimiento *updateLeader()*. Si *p* es el líder, recoge los datos de *q* ($data_q$).

Si el $timer_p(q)$ expira antes de recibir un nuevo mensaje (I-AM-ALIVE, *q*, $counter_q$, $data_q$), entonces *p* incrementa $counter_p[q]$, incrementa $Timeout_p[q]$, resetea $timer_p(q)$ y además llama al procedimiento updateLeader(). Los siguientes mensajes enviados por *p* incluirán el incremento de $counter_p[q]$ y de esta forma el resto de los sensores conocerán las sospechas de *p* sobre *q*.

El algoritmo incluye un mecanismo para rechazar finalmente los sensores inestables que perturban la elección del líder. Este mecanismo se basa en el número de encarnaciones de los sensores. Durante la inicialización, cada sensor *p* inicializa sus timeouts con respecto a los otros sensores a $\eta + incarnation_p$ (Línea 4). Además, *p* incializa $counter_p[p]$ a $incarnation_p$ (Línea 7). Estas inicializaciones aseguran que al final:

1. Todo sensor *p* inestable nunca sospecha acerca de un sensor correcto *q* que puede comunicarse directamente con todo los demás sensores (como *p* es un sensor inestable, su $incarnation_p$ aumenta y en la inicialización el timeout de p con respecto de q ($Timeout_p[q]$) sigue aumentando constantemente y por tanto, al final $timer_p(q)$ nunca expira) y en consecuencia *p* no incrementa más $counter_p[q]$.
2. Todo sensor *p* inestable nunca va a ser elegido como líder en el procedimiento *updateLeader()* (debido a que $incarnation_p$ y por lo tanto, $counter_p[p]$, sigue aumentando constantemente).

Como en el anterior algoritmo, el número de mensajes enviados durante el período de adquisición de los datos (η) es lineal con el número de sensores de la región, ya que todo sensor emite un mensaje por la tarea 1.

La demostración de la corrección de este algoritmo puede se encuentra en [1].

3.3.3.- Tercer algoritmo

A continuación se presenta el tercer algoritmo de elección del agregador y agregación de datos para el nivel local. Contrariamente al algoritmo anterior, no requiere que ningún sensor se comunique directamente con el resto; sólo requiere la existencia de un camino bidireccional *multi-hop* desde algún sensor(es) correcto hasta el resto de los sensores. Además, como ocurre en al primer algoritmo, los sensores no necesitan conocer por anticipado los identificadores del resto de los sensores. Como ya hemos dicho, asumimos que existe un límite desconocido δ en el retraso de los mensajes y que la ejecución de cada línea de código del algoritmo requiere al menos σ unidades de tiempo.

El sensor elegido por *p* como agregador es el sensor con el mínimo valor asociado de $membership_p$ (min{ $membership_p$ }), usando el identificador del sensor para romper los lazos. El algoritmo utiliza la variable $membership_p$ para almacenar los identificadores de los diferentes sensores descubiertos hasta el momento y que contiene un conjunto de tuplas *(q, v)*, una para cada sensor conocido, donde *q* es el identificador del sensor y *v* es el número de veces que los sensores sospechan de q.

La figura 11 presenta el algoritmo en detalle. En la tarea 1, los sensores emiten mensajes periódicamente para intentar convertirse en agregador, así como para enviar sus datos, con una periodicidad de κ. Todo mensaje enviado por un sensor *p* contiene el conjunto $membership_p$ y los

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

datos $data_p$. En la tarea 2, si un sensor *p* recibe un mensaje (I-AM-ALIVE, q, $membership_q$, $data_q$) con $q \neq p$ por primera vez, re-emite el mensaje para intentar llegar a todos los sensores de la región, actualiza $membership_p$ basándose en $membership_q$ (líneas 13-21) y resetea $timer_p$ *(q)*. A continuación, *p* llama al procedimiento *updateLeader()*. Finalmente, si *p* es el agregador, recoge los datos $data_q$.

En la tarea 3, si $timer_p$ *(q)* expira antes de recibir un nuevo mensaje I-AM-ALIVE procedente de q, entonces *p* 'sospecha' de *q*. *P* reemplaza en $membership_p$ (q, v) por (q, v + 1), incrementa $Timeout_p[q]$, resetea el timer y llama al procedimiento *updateLeader()*. Observamos que, si *q* no ha fallado, a la recepción del siguiente mensaje de *p*, *q* incrementará su contador asociado en $membership_q$.

Every sensor *p* executes the following:

procedure *updateLeader*()
(1) $leader_p \leftarrow$ sensor in min{$membership_p$}
end procedure

Initialisation:
(2) increment $incarnation_p$ by 1 in stable storage
(3) read ($incarnation_p$) from stable storage
(4) $membership_p \leftarrow \{(p, incarnation_p)\}$
(5) $leader_p \leftarrow p$
(6) start tasks 1, 2 and 3

Task 1:
(7) loop forever
(8) $data_p \leftarrow$ acquire sensed data
(9) broadcast (I-AM-ALIVE, *p*, $membership_p$, $data_p$)
(10) wait(κ)

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

Task 2:

(11) upon reception of message (I-AM-ALIVE, q, $membership_q$, $data_q$)
with $q \neq p$ for the first time do

(12) broadcast (I-AM-ALIVE, q, $membership_q$, $data_q$)

(13) $\forall(r,-) \in membership_q$:

(14) if $(r,-) \notin membership_p$ then

(15) $membership_p \leftarrow membership_p \cup \{(r, v)\} : (r, v) \in membership_q$

(16) create $timer_p(r)$ and $Timeout_p[r]$

(17) $Timeout_p[r] \leftarrow \kappa + incarnation_p$

(18) reset $timer_p(r)$ to $Timeout_p[r]$

(19) else

(20) replace in $membership_p$ (r, v) by $(r, \max\{v, v'\}) : (r, v') \in membership_q$

(21) end if

(22) reset $timer_p(q)$ to $Timeout_p[q]$

(23) *updateLeader*()

(24) if $[leader_p = p]$ then

(25) collect $data_q$

(26) end if

Task 3:

(27) upon expiration of $timer_p(q)$ do

(28) replace in $membership_p$ (q, v) by $(q, v + 1)$

(29) $Timeout_p[q] \leftarrow Timeout_p[q] + 1$

(30) reset $timer_p(q)$ to $Timeout_p[q]$

(31) *updateLeader*()

Figura 11: *Elección de agregador y agregación de datos a nivel intra-regiona (Algoritmo 3)*

El número de mensajes enviados durante un período de adquisición de datos ( ) es cuadrático en el número de sensores de la región, ya que todos los sensores emiten un mensaje por la tarea 1 y los sensores re-emiten los mensajes recibidos.

En cuanto a la prueba de corrección de este algoritmo, se observará que es muy similar a la de la sección anterior. Las principales diferencias son:

1. La desconocida *membership*. En la variable membership, el sensor almacena el número de sospechas que tiene de cada uno del resto de los sensores.
2. A diferencia del algoritmo anterior, ahora sólo se requiere la existencia de un camino multi-hop bidireccional desde algunos sensor(es) correctos hasta el resto de los sensores.

La primera cuestión (1) se trata con un membership no decreciente($membership_p$) y temporizadores creados de forma dinámica, mientras que la segunda (2) es superada por la retransmisión del mensaje que cada sensor recibe por primera vez (Línea 12).

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

3.4.- El consumo de energía en la elección del agregador y en la agregación de datos.

En los algoritmos que hemos presentado hasta ahora, eventualmente el sensor agregador se mantiene como tal hasta el final de su batería. Teniendo en cuenta que la batería de un sensor que actúa como agregador decrece considerablemente más rápido que la batería de un sensor normal (ver figura 12), para prevenir el gasto completo de la batería del agregador, vamos a introducir en el algoritmo un umbral de batería. Cuando el agregador detecte un nivel bajo de batería, es decir, un nivel por debajo del umbral establecido, inducirá al sistema a elegir otro sensor como agregador.

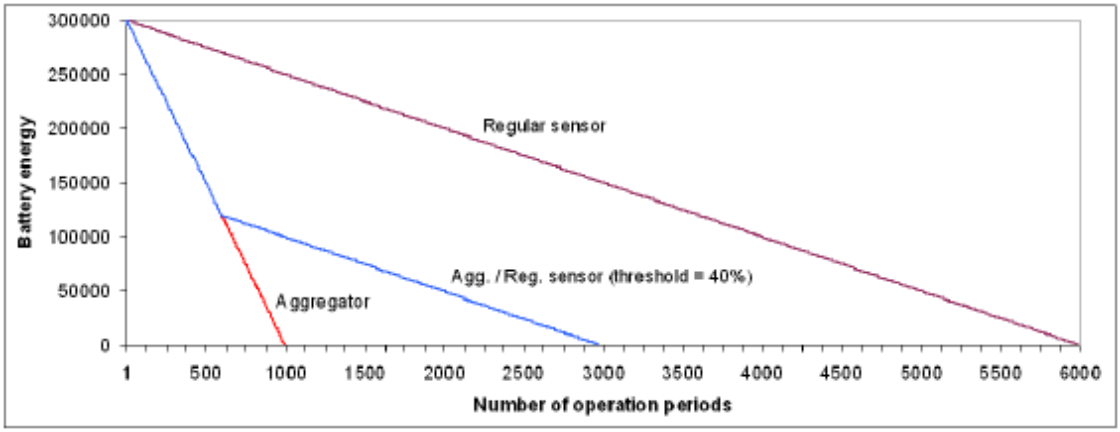


Figura 12: *Comparación de la vida de las baterías de los sensores.*

Para introducir el umbral de batería en los algoritmos anteriores, se introduce la modificación presentada en la figura 13. Esta modificación consiste en una nueva tarea que chequea si el sensor es el agregador de esa región y si su nivel de batería está por debajo del umbral establecido. Si este es el caso, entonces el sensor incrementa su número de encarnaciones (en el caso del primer algoritmo del nivel local) o su contador de sospechas (en el caso del segundo y del tercer algoritmo del nivel local).

```
Task 0:
(1)          loop forever
(2)               if [leader_p = p] and [getBatteryLevel() < THRESHOLD] then
(Alg. I)               incarnation_p ← incarnation_p + 1
(Alg. II)              counter_p[p] ← counter_p[p] + 1
(Alg. III)             replace in membership_p (p, υ) by (p, υ + 1)
(4)               wait(γ)
```

Figura 13: *Modificación del algoritmo para adaptar el umbral.*

La selección del umbral determina la calidad de servicio (QoS) de la red inalámbrica de sensores, medida en el número de sensores que permanecen activos durante un período dado de tiempo, proveyendo la existencia de un agregador común.

Para un número deseado de sensores activos durante un período de tiempo determinado y de acuerdo con el consumo de energía de los sensores, podemos determinar el umbral asociado. El riesgo potencial de esta estrategia se produce cuando debido a un nivel de la batería por debajo del umbral, todos los sensores comienzan a aumentar su número de encarnaciones o su contador de sospechas. Cuando esto ocurre, los sensores podrían establecer el umbral a 0% y la red podría

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

continuar funcionando correctamente (sin QoS como garantía) hasta el final de la batería de todos los sensores.

El uso de un umbral proporciona un mecanismo de elección de agregador y agregación de datos consciente del consumo de energía, ya que permite una reducción equilibrada de las pilas de los sensores bien comunicados o correctos de cada región. Como consecuencia, el número de sensores que se mantienen activos durante un período determinado de tiempo aumenta, lo cual es un factor relevante a la hora de determinar la QoS de una red de sensores.

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

4.- ANÁLISIS Y DISEÑO

En este capítulo abordamos el análisis y el diseño de las características de los tres algoritmos que vamos a desarrollar, recogidas en la especificación de requisitos de usuario. Además, ponemos de manifiesto las posibles restricciones al problema.

4.1. Particularidades

Antes de comenzar el análisis y diseño del problema, debemos tener en cuenta ciertas restricciones inherentes a las redes de sensores inalámbricas, que hacen de nuestro desarrollo una aplicación informática atípica. Es importante tener en cuenta estas particularidades para poder realizar un análisis y un diseño adaptado a sus necesidades.

A continuación enumeramos dichas restricciones:

1. Las aplicaciones de sensores inalámbricos deben tener en cuenta la limitación de estos dispositivos en lo que a memoria se refiere.

- Es necesario optimizar la aplicación para el consumo de energía, pues éste es un aspecto clave en el despliegue de una red con estas características.
- 3. No existirá ninguna interacción entre el usuario final y nuestra aplicación. La única intervención humana en este sentido será el tratamiento final de los datos que llegan al PC, ya sea para su posterior estudio, generación de informes, tratamiento estadístico, etc.
- 4. El perfil del usuario final de nuestra aplicación deberá ser el de una persona especializada a nivel técnico y capacitado para interpretar los datos recogidos, ya que no se presentan de manera legible ni existe una interfaz gráfica que facilite su interpretación.

Una vez mencionadas estas restricciones, pasamos a presentar el proyecto que hemos desarrollado, diferenciado en tres etapas: análisis, diseño e implementación. A continuación abordamos el análisis del problema que ya se introdujo en el c*apítulo 3* y cuyo escenario, como pudimos ver, puesta en marcha de un algoritmo de elección de agregador y agregación de datos en una red inalámbrica de sensores.

4.2. Especificación de Requisitos de Usuario

Se pretende desarrollar e implementar tres algoritmos de elección de agregador y agregación de datos y ponerlos en marcha en una red inalámbrica de sensores. En este sentido, definiremos la arquitectura del sistema desarrollado, así como los objetivos y requisitos funcionales de cada uno de los dispositivos que forman parte de dicho sistema. Recordemos que el objetivo primordial de este proyecto se centra en el estudio de los mecanismos para garantizar la autoconfiguración de redes inalámbricas de sensores y su tolerancia y recuperación frente a fallos. Es decir, pretendemos comprobar la tolerancia a fallos de cada uno de los nodos que componen la red y ver como gracias a los algoritmos implementados se da solución al problema y el funcionamiento de la red continúa siendo correcto.

4.2.1.- Arquitectura

En la figura 14 se muestra la arquitectura de una red inalámbrica de sensores similar a la que nosotros vamos a configurar para desarrollar nuestro proyecto. Esta red está formado por una serie de nodos que se encargan de obtener ciertos datos del entorno (en nuestro caso mediremos únicamente la luminosidad) para ser transmitidos hasta un líder común de la región. A su vez, este líder será el encargado de enviar los datos recogidos hasta un gateway o estación base conectada a

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

un PC que los hará accesibles desde el exterior. Todas las motas de nuestra red ejecutarán uno de los tres algoritmos que hemos estudiado anteriormente. Pese a que el funcionamiento global de la red será el mismo, la forma de elección de líder y agregación de los datos será diferente en función del algoritmo de agregación ejecutado.

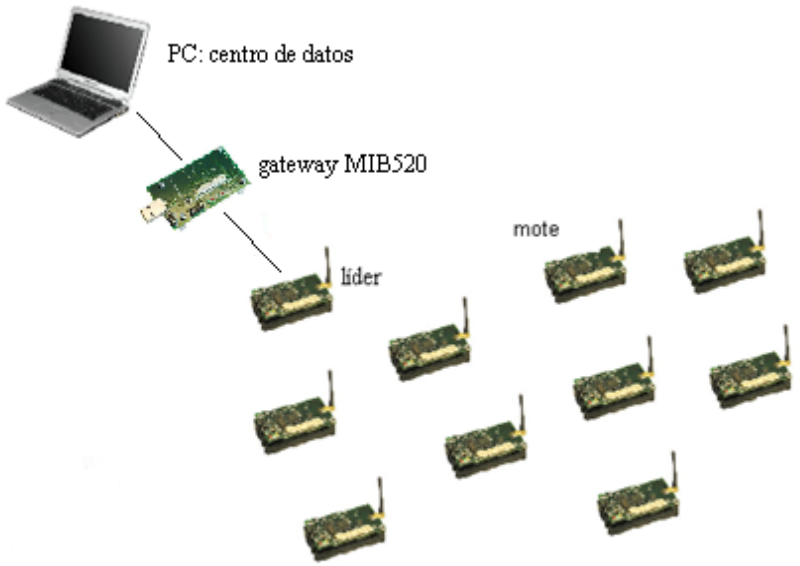


Figura 14: *arquitectura de red de nodos Iris usada en este proyecto*

Es importante señalar que la arquitectura de red de los tres algoritmos se diferencia en lo siguiente:

- El primer algoritmo asume que todo par de sensores en una región puede comunicarse directamente.
- El segundo algoritmo asume que algunos sensor(es) correctos pueden comunicarse directamente con el resto de los sensores de la región.
- El tercer algoritmo asume la existencia de un camino multi-hop bidireccional desde algunos sensores correctos hasta el resto de los sensores de la región.

La comunicación entre las motas es bidireccional, todas pueden actuar como emisor y como receptor. La comunicación entre el líder y el gateway es unidireccional. Este último siempre actúa como receptor de mensaje. Asimismo, la comunicación entre el gateway el PC también es unidireccional.

Como se puede apreciar en la figura 15, disponemos de varios nodos, unos que actuarán como nodos sensores (obtendrán la información sobre la luz), otro nodo sensor que desempeñará la función de líder y otro (el gateway) que desempeñará la función de pasarela de información entre los nodos que compondrán la red y el centro de datos, que en nuestro caso se trata de un PC.

La distribución de los nodos sensores dependerá de las hipótesis iniciales del algoritmo que ya hemos mencionado anteriormente.

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

Para construir nuestro sistema, es necesario identificar y definir dos roles bien diferenciados en la red:

- Aplicación en los nodos sensores o *motas:* código nesC de uno de los tres algoritmos de elección de agregador y agregación de datos. El algoritmo en cuestión, será cargado en todos los sensores de la red.
- Aplicación en el gateway para la recogida de los datos. En el caso del primer algoritmo utilizaremos la aplicación XMeshBase que recoge los datos de la XMesh y permite mostrarlos utilizando la herramienta MoteView. En el segundo y tercer algoritmo, hemos desarrollado una pequeña aplicación que simplemente se limita a recoger los datos enviados por el líder y mostrarlos por pantalla.

4.3.- Algoritmo 1

En esta sección explicaremos el análisis y diseño de nuestro primer algoritmo de elección de agregador y agregación de datos para el nivel local. Para ello vamos a basarnos en lo explicado en la sección 3.3.1 y en el pseudocódigo de la figura 8. A grandes rasgos, recordemos que en este algoritmo asumimos que los sensores se despiertan periódicamente para proveer los datos recogidos e hibernan el resto del tiempo. Además, es de suma importancia señalar que los sensores no necesitan conocer por adelantado los identificadores del resto de los sensores y que todos los sensores de una región pueden comunicarse directamente. Estos dos factores serán determinantes a la hora de diseñar el algoritmo y serán las dos principales características que lo diferencian de los otros dos algoritmos.

4.3.1.- Comportamiento de la aplicación

El funcionamiento del algoritmo es el siguiente. En primer lugar, al encender la mota se ejecuta el comando StdControl.init() que Inicializa los componentes y las variables globales. Después del init y cuando se reinicia la mota (ahorro de energía) se ejecuta el comando StdControl.start() el cual arranca la ejecución y establece la frecuencia del reloj del temporizador 1. Este temporizador se ejecutará hasta que el sensor falle o agote sus baterías y es el encargado de despertar al sensor del período de hibernación cada 10 segundos. A continuación, cuando salta el temporizador que habíamos iniciado anteriormente, el algoritmo ejecuta la tarea leer_variables() que se encarga de leer las variables almacenadas en memoria no volátil. Estas variables son *incarnation*, *leader*, *incarnation_leader* y *timeout* que guardan respectivamente el número de encarnaciones del sensor, el sensor elegido como agregador, el número de encarnaciones del líder y el tiempo de espera de un sensor al recibir el anuncio de un agregador. Una vez recuperado correctamente el valor de todas las variables, se inicia el algoritmo principal llamando a la tarea initialization(). En esta tarea, si el sensor se considera a sí mismo el actual agregador, el algoritmo empieza las tareas 1 y 2. Por otro lado, si el sensor no se considera a sí mismo agregador, el algoritmo resetea el *timerp* local y comienza las tareas 2 y 3. Otras variables importantes son *timerp* (control de tiempo local usado para detectar el desplome del sensor agregador), $scheduled\_wakeup_p$ (sirve para comprobar si un sensor ha hibernado o ha fallado), *inc_timeout* (determina el crecimiento del *timeout* para alcanzar el acuerdo).

En la tarea 1 se anuncia a los demás sensores quien es el agregador y se reciben los datos de los sensores. Al comienzo de la tarea, se espera un número e de unidades de tiempo y se construye el mensaje a enviar indicando el número de encarnaciones del sensor. Finalmente, se llama al comando SendMsg.send pasándole la dirección del nodo de destino, en este caso TOS_BCAST_ADDR y un puntero al mensaje actual que queremos enviar.

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

La tarea 2, que realizan todos los sensores, es utilizada para obtener los datos y enviarlos al agregador, y para cambiar el agregador si fuera requerido (si ha aumentado su número de encarnación). Para obtener sus datos, el sensor llama al comando Light.getData(). Cuando el dato medido está listo, salta el evento Light.dataReady en el que guardamos correctamente el dato en el paquete que va a ser enviado a través de la XMesh y llamamos a SendData(). En SendData(), si no hay ningún envío de paquete en proceso, y si el sensor en cuestión es el líder envía los datos recibidos a la estación base y si no lo es, envía el mensaje al líder. Para enviar los datos a la estación base o al líder, hemos utilizado el servicio de red multisalto *XMesh.* Para enviar estos paquetes utilizamos el comando Send.send.

Finalmente, en la tarea 3 se carga el temporizador 2 con el valor de la variable *timerp*. Cuando salta el evento correspondiente a este timer, si el sensor ha recibido un mensaje I-AM-ALIVE pero no lo ha recibido del líder, establece al emisor de ese mensaje como nuevo líder. Si no ha recibido ningún mensaje I-AM-ALIVE, se propone a sí mismo como agregador, y además incrementa el timeout en *inc_timeout* unidades de tiempo. Cada vez que actualizamos el valor de algunas de las variables almacenadas en memoria no volátil, escribimos su nuevo valor en memoria no volátil llamando a la tarea leer_variables(). Por último, en el evento ReceiveMsg.receive establecemos gracias a dos variables booleanas (recv1 y recv2) cuando ha llegado un paquete del tipo TOS_Msg. Estos paquetes son aquellos enviados por el líder durante la tarea 1 y que ya hemos comentado anteriormente.

4.3.2.- Estructura de la trama de datos

Es importante definir una estructura de los datos que se necesitan enviar a través de los motes.

En este primer algoritmo tenemos dos tipos diferentes de mensajes. En primer lugar, nos encontramos con los mensajes enviados por el agregador de la región para anunciarse al resto de los sensores. Para ello, hemos utilizado mensajes del tipo TosMsg. Se trata de la estructura estándar de los mensajes de TinyOS. Contiene los datos específicos de cabecera y encapsula la trama del mensaje de nuestra aplicación. La estructura del mensaje puede verse en la figura 15:

```
typedef struct TOS_Msg
{
/* The following fields are transmitted/received on the radio. */
uint16_t addr;
uint8_t type;
uint8_t group;
uint8_t length;
int8_t data[TOSH_DATA_LENGTH];
}

typedef TOS_Msg *TOS_MsgPtr;
```

Figura 15: *Estructura del paquete de datos TOS_Msg*

El significado de cada uno de los campos es el siguiente:

- *addr* , es la dirección a la que va destinado el paquete. En nuestro caso, la dirección de destino es la de broadcast (*TOS_BCAST_ADDR).*
- *type*, es el tipo de paquete que se está enviando, este tipo se utiliza en las interfaces paramétricas para determinar que instancia de la interfaz se va a hacer cargo de procesar dicho paquete (AM_ALVMSG para este algoritmo).

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

- *group*, es el grupo al que pertenece el sensor que está enviando el paquete, de manera que solo lo reciban aquellos sensores que pertenezcan al mismo grupo, de esta manera pueden coexistir dos redes diferentes de sensores sobre el mismo medio físico aéreo y sobre el mismo canal de radio.
- *lenght*, es la longitud total del paquete.
- *data*, este es el campo más importante y es el que nos posibilita jugar con los paquetes, en este campo se especifican los datos que se van a enviar, estos datos pueden ser una estructura que conforme un nuevo tipo de paquete de nivel superior.

En este proyecto, sólo nos interesan los bytes enviados en el campo *data* de los mensajes enviados por el líder. En este campo, introducimos una nueva estructura del siguiente tipo:

```
typedef struct AlvMsg {

  uint8_t  addr;

  uint8_t  board_id;

  uint8_t  packet_id;

  //uint8_t  node_id;

  uint8_t  inc;

} __attribute__ ((packed)) AlvMsg;
```

Figura 16: *Estructura del paquete de datos AlvMsg*

En esta estructura es donde nosotros vamos a introducir los datos que realmente queremos transmitir que van a ser necesarios para el correcto funcionamiento del algoritmo. En el campo addr introducimos la dirección del nodo que emite el mensaje (en este caso será la del líder), el campo board_id es el identificador de la placa sensora utilizada (la MDA100 en nuestro caso), packet_id recoge el identificador de paquete y por último, el campo inc es el número de encarnaciones del nodo emisor del mensaje.

El segundo tipo de mensajes enviados por la aplicación es el tipo XDataMsg. Estos mensajes son enviados por el líder a la estación base y por el resto de sensores al líder. La razón por la que hemos utilizado este tipo de mensajes es porque nos permiten utilizar la aplicación . Esta aplicación proporciona una interfaz gráfica intuitiva para monitorizar y gestionar la red de sensores inalámbrica. En el siguiente capítulo explicaremos más detalladamente su funcionamiento.

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

La estructura del mensaje es puede verse en la figura 17.

```
typedef struct XDataMsg {
  uint8_t  board_id;
  uint8_t  packet_id;
  //uint8_t  node_id;
  uint16_t  parent;       // 4
  uint16_t vref;
  uint16_t thermistor;
  uint16_t light;
  uint16_t mic;
  uint16_t accelX;
  uint16_t accelY;
  uint16_t magX;
  uint16_t magY;
} __attribute__ ((packed)) XDataMsg;
```

Figura 17: *Estructura del paquete de datos XDataMsg*

En nuestro algoritmo, únicamente utilizamos el campo light donde guardamos el dato de luz obtenido por el sensor.

4.3.3. Diagramas de secuencia

En las siguientes figuras se muestran los diagramas de secuencia de las interacciones de los nodos de la red y la estación base en diferentes situaciones.

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

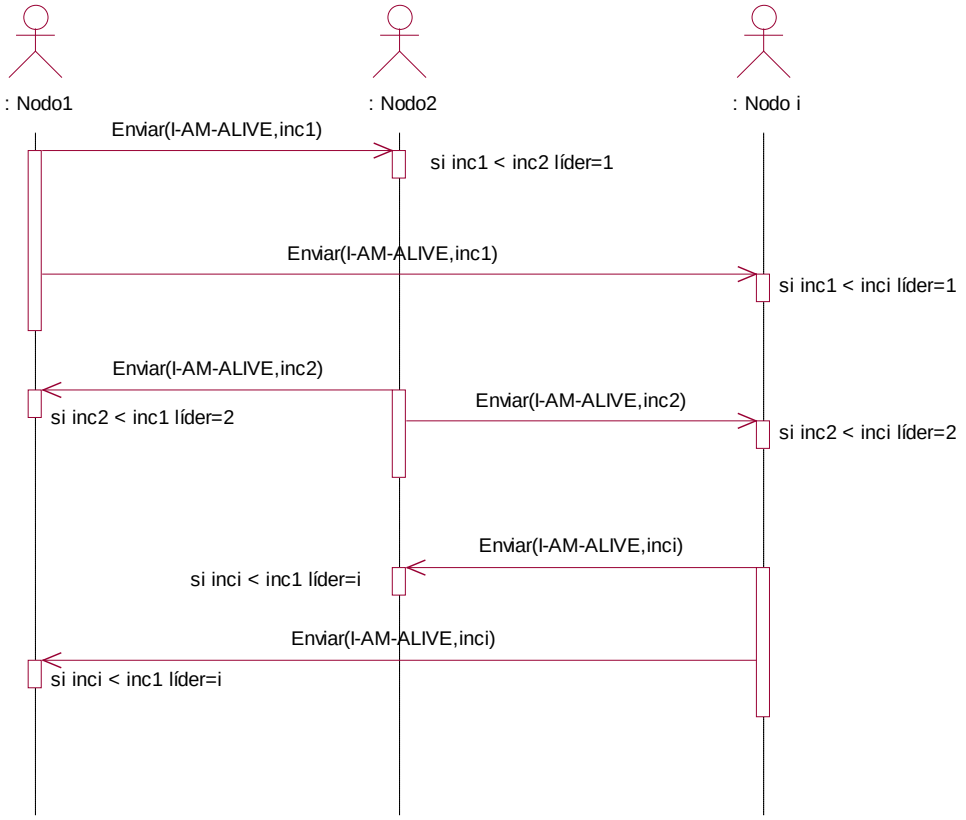


Figura 18: *elección del agregador común*

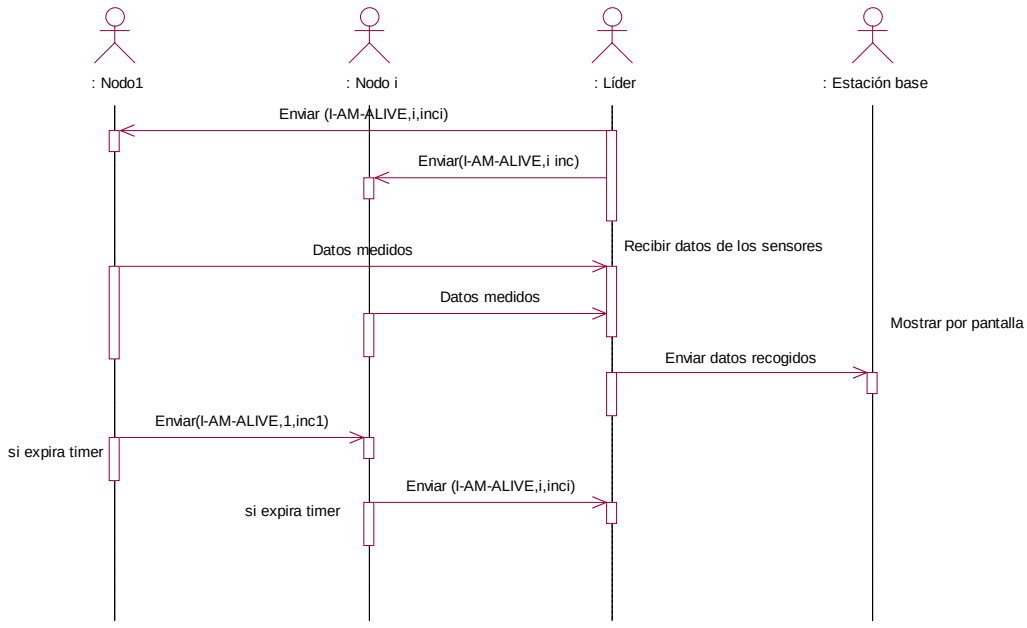


Figura 19: *agregación de datos y envío a la estación base*

4.3.4.- Estructura de la aplicación

Una vez realizado el análisis y el diseño del algoritmo, pasamos a implementar el código en el lenguaje de programación nesC con los eventos y tareas que se desea que lleven a cabo. Como explicamos en el capítulo, el desarrollo de las aplicaciones con TinyOS se basa en

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

componentes y en la unión de éstos a través de interfaces. La figura 20 muestra el diagrama de componentes de los que hace uso el primer algoritmo, y la manera en la que están conectados.

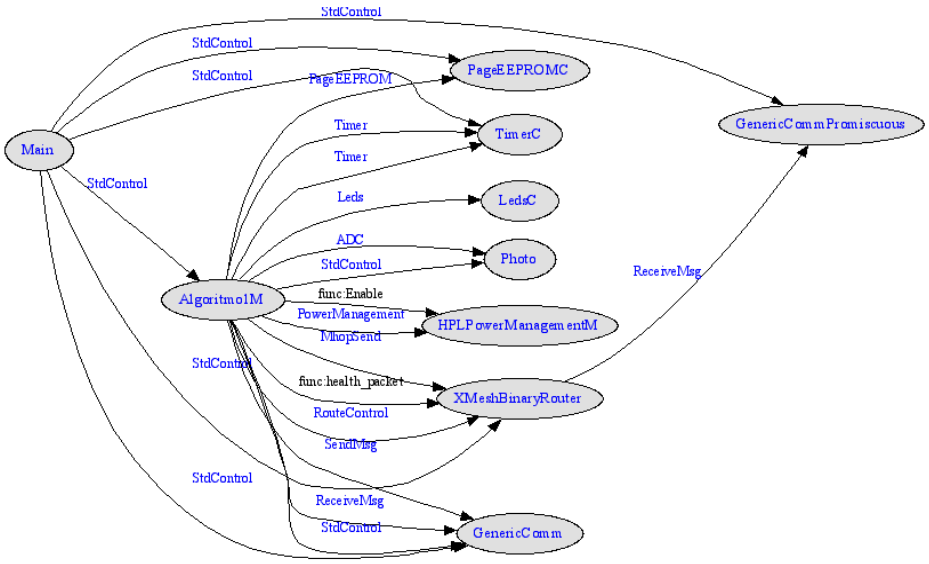


Figura 20: *diagrama de componentes del algoritmo 1*

Para entender mejor el diagrama de componentes del primer algoritmo, la tabla 1 muestra el significado de todos los elementos de comunicaciones.

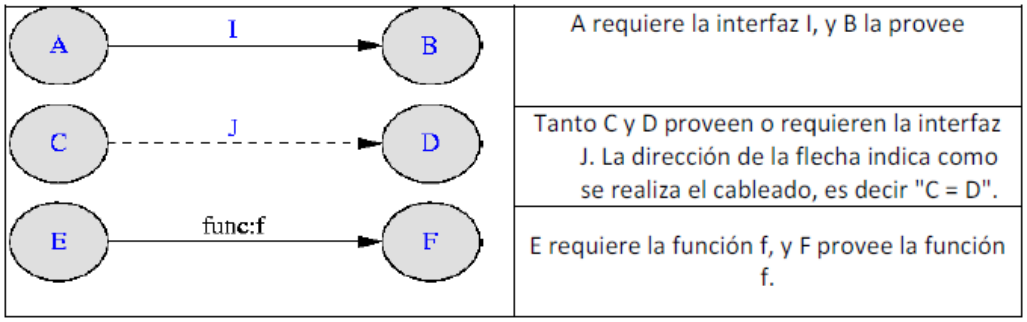


Tabla 1: *convenio establecido pos nesdoc*

El código nesC del algoritmo puede verse en el Anexo A.

4.4.- Algoritmo 2

En el segundo algoritmo de elección de agregador y agregación de datos para el nivel local rebajamos las hipótesis iniciales y ya no suponemos que todo par de sensores debe poder comunicarse directamente. Ahora simplemente suponemos que existe un subconjunto de sensores en la región que sí son capaces de alcanzar directamente al resto de los sensores de la región y además son también capaces de recibir mensajes emitidos por todos los sensores de la región. De este subconjunto de sensores, saldrá elegido el agregador o líder de la región. De nuevo, esta peculiaridad será totalmente determinante a la hora de diseñar e implementar el algoritmo.

4.4.1.- Comportamiento de la aplicación

El funcionamiento de este segundo algoritmo es el siguiente. Al encendido de la mota le sigue la ejecución de los comando StdControl.init() que Inicializa los componentes y las variables globales. Después del init y cuando se reinicia la mota (ahorro de energía) se ejecuta el comando

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

StdControl.start() el cual arranca la ejecución, establece la frecuencia del reloj del temporizador 3 y ejecuta la tarea leer_variables(). La única variable leída de memoria no volátil es el número de encarnaciones del sensor (*incarnation*). Este ejecutará cada k+2 segundos y servirá para que el líder envíe los datos recogidos a la estación base. A continuación, se ejecuta la tarea escribir_variables(). Esta tarea aumenta el valor del número de encarnaciones en una unidad, escribe el nuevo valor en memoria no volátil e inicia el algoritmo principal llamando a la tarea initialization(). En esta tarea, inicializamos las variables *counter*, *timer*, *timeout* y *leader*. La variable *counter* representa la estimación de *p* sobre el número de veces que el resto de sensores han sido sospechosos. En las variables *timer* y *timeout* el sensor almacena los timers y timeouts correspondientes al resto de los sensores. Al igual que en el algoritmo 1, la variable *leader* almacena el identificador del líder. Estas tres variables son declaradas como tablas siendo su tamaño el valor de la constante SENSORES. La constante SENSORES almacena el número total de sensores de la red. Por ejemplo, el sensor 1 almacena en la posición el número de veces que sospecha que 2 ha fallado. Además, en la inicialización se establece la frecuencia de reloj del temporizador 1 y se inician las tareas 2 y 3.

El temporizador 1 es el encargado de lanzar cada *n* unidades de tiempo la tarea 1. Por tanto, esta tarea será ejecutada por todos los sensores cada vez que se produzca una expiración del timer 1. En esta tarea, el sensor ejecuta el comando Light.getData() para obtener el dato de luz. Cuando el dato medido está listo, salta el evento Light.dataReady y se emite un mensaje del tipo AlvMsg al resto de sensores con el dato recogido.

Cuando un sensor *p* recibe un mensaje del tipo AlvMsg se ejecuta la tarea 2. Esta tarea comprueba que efectivamente el mensaje ha sido recibido correctamente y en caso afirmativo, resetea $timer_p$(q) para cuando se espera recibir el siguiente (I-AM-ALIVE, *q*, $counter_q$, $data_q$), actualiza acordemente su tabla $counter_p$ y llama al procedimiento *updateLeader()*. El procedimiento updateLeader() actualiza el líder de la red. El sensor *l* elegido como líder es aquel con el valor de $counter_p[l]$ más pequeño Si *p* es el líder, recoge los datos de *q* y los almacena en la posición correspondiente a q de tabla *datos* (la posición del sensor q sería q-1 ya que la tabla comienza en la posición 0). En la tabla *datos* estarán almacenados los datos más recientes medidos por los sensores.

Finalmente la tarea 3 establece la frecuencia de reloj del temporizador 2 gracias al cual controlamos si alguno de los timers ha expirado. En el caso de $timer_p(q)$ expire antes de recibir un nuevo mensaje (I-AM-ALIVE, *q*, $counter_q$, $data_q$), entonces *p* incrementa $counter_p[q]$, incrementa $Timeout_p[q]$, resetea $timer_p(q)$ y además llama al procedimiento updateLeader(). Los siguientes mensajes enviados por *p* incluirán el incremento de $counter_p[q]$ y de esta forma el resto de los sensores conocerán las sospechas de *p* sobre *q*.

En este algoritmo, empleamos un nuevo temporizador 3. Cada vez que se produce una expiración de este temporizador, si el sensor en cuestión es el líder, envía los datos recogidos hasta el momento a la estación base.

4.4.2.- Estructura de la trama de datos

En este segundo algoritmo tenemos dos tipos diferentes de mensajes: mensajes del tipo AlvMsg y mensajes del tipo DataMsg. La estructura del paquete AlvMsg es similar a la utilizada en el algoritmo 1 y se muestra en la figura 21. La única diferencia es que hemos cambiado el campo *inc* (indicaba el número de encarnaciones del sensor) por el campo *counter* (ya explicado anteriormente) y hemos añadido un nuevo campo *data* para poder enviar los datos de luz recogidos a todos los demás sensores que configura la red.

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

```
typedef struct AlvMsg {
uint8_t  addr;
uint8_t  board_id;
int  packet_id;
//uint8_t  node_id;
int counter[SENSORES];
uint16_t data;
} __attribute__ ((packed)) AlvMsg;
```

Figura 21: *estructura del paquete de datos AlvMsg*

Los datos enviados por el líder de la región hacia la estación base se encapsulan dentro de un mensaje del tipo DataMsg. La estructura de este mensaje puede verse en la figura 22. El campo más importante es *data* y consiste en una tabla de tantas posiciones como sensores tenga la red en la que se almacenan los datos más recientes medidos por los sensores. Este mensaje será recibido por la estación base y ella será la encargada de procesar y mostrar los datos.

```
typedef struct DataMsg {
  uint8_t  addr;
  uint8_t  board_id;
  int  packet_id;
  //uint8_t  node_id;
  uint16_t data[SENSORES];
} __attribute__ ((packed)) DataMsg;
```

Figura 22: *estructura del paquete de datos DataMsg*

4.4.3.- Diagramas de secuencia

En el siguiente diagrama de secuencia (figura 23) podemos observar las interacciones entre los nodos de la red y la estación base.

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

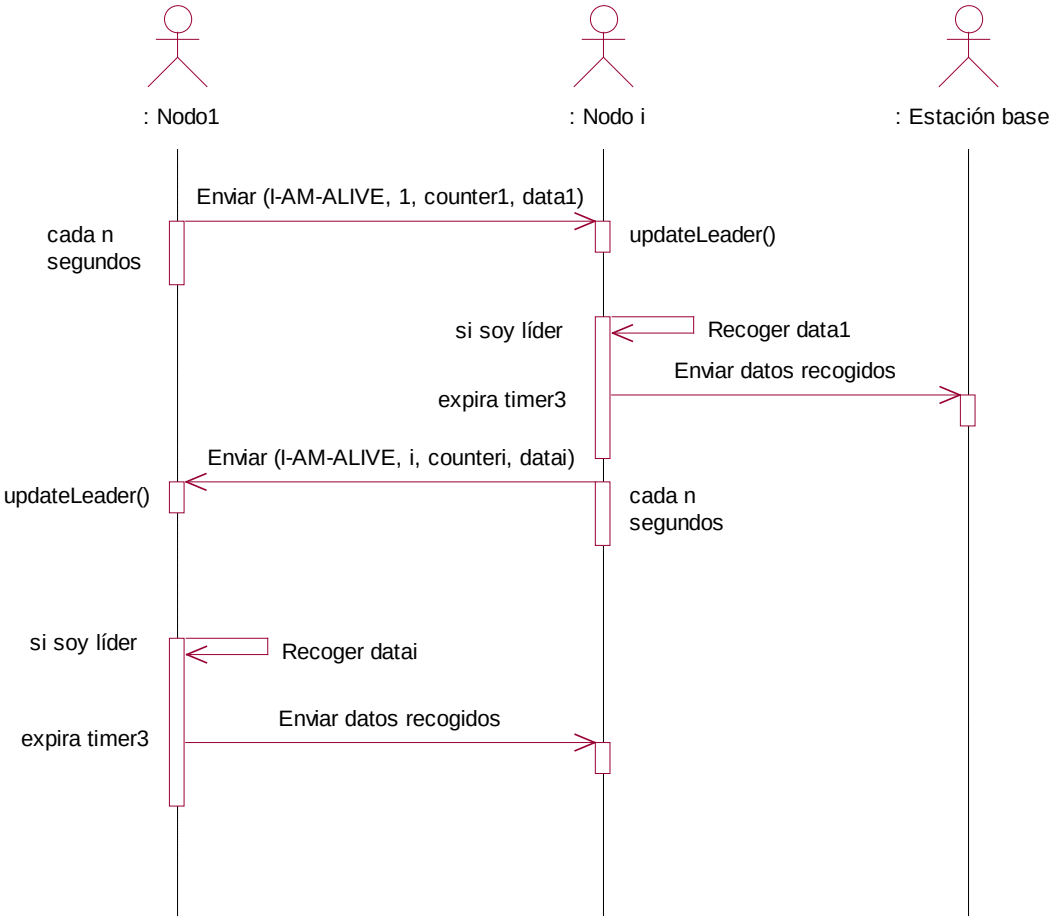


Figura 23: *diagrama de secuencia algoritmo 2*

4.4.4.- Estructura de la aplicación

En cuanto a la implementación del algoritmo en el lenguaje de programación, la figura 24 muestra el diagrama de componentes de los que hace uso este segundo algoritmo, y la manera en la que están conectados.

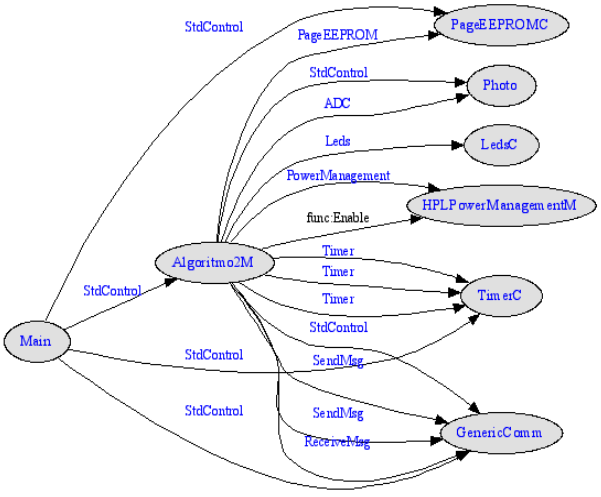


Figura 24: *diagrama de componentes del algoritmo 2*

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

El código nesC completo de este algoritmo puede verse detalladamente en el Anexo B.

4.5.- Algoritmo 3

Por último, comentaremos el análisis y diseño del tercer algoritmo de elección del agregador y agregación de datos para el nivel local. A diferencia del algoritmo anterior, en este volvemos a rebajar las hipótesis iniciales y ya no exigimos que ningún sensor se comunique directamente con el resto; sólo requiere la existencia de un camino bidireccional *multi-hop* desde algún sensor(es) correcto hasta el resto de los sensores. Este factor, será fundamental a la hora de diseñar el algoritmo e implementarlo posteriormente. Pese a que en la presentación del algoritmo en el capítulo 3 indicábamos que en este algoritmo los sensores no necesitaban conocer por anticipado los identificadores del resto de los sensores, por restricciones hardware y del lenguaje de programación elegido, vamos a suponer que los sensores conocen como mínimo el número máximo de sensores que albergará la red. Este valor lo guardaremos en la constante SENSORES.

4.5.1.- Comportamiento de la aplicación

El algoritmo 3 es prácticamente idéntico al anterior salvo las siguientes diferencias. En el algoritmo 3 hemos sustituido la variable *counter* por la riable *membership* para almacenar los identificadores de los diferentes sensores descubiertos hasta el momento y que contiene un conjunto de tuplas *(q, v)*, una para cada sensor conocido, donde *q* es el identificador del sensor y *v* es el número de veces que los sensores sospechan de q. Cuando recibamos noticas de un nuevo nodo, agregaremos ese nodo a nuestra variable *membership* y crearemos sus timers y timeouts correspondientes. La diferencia más importante nos la encontramos en la tarea 2. En la tarea 2, si un sensor *p* recibe un mensaje (I-AM-ALIVE, q, $membership_q$, $data_q$) con $q \neq p$ por primera vez, re-emite el mensaje para intentar llegar a todos los sensores de la región, actualiza $membership_p$ basándose en $membership_q$ y resetea $timer_p$ *(q)*. De esta forma, reenviando los mensajes recibidos, conseguimos crear un camino multisalto desde un sensor correcto hacia el resto de los sensores de la red y nos ajustamos a la hipótesis inicial. A continuación, *p* llama al procedimiento *updateLeader()*. Finalmente, si *p* es el agregador, recoge los datos $data_q$. El procedimiento updateLeader() elige como líder aquel sensor con el mínimo valor asociado de *membership* (min{ *membership* }).

4.5.2.- Estructura de la trama de datos

La estructura de los paquetes enviados por los sensores son del tipo AlvMsg es similar a la utilizada en el algoritmo 2 y se muestra en la figura 25. La única diferencia es que hemos cambiado el campo *counter* por el campo *membership* (ya explicado anteriormente) y hemos añadido un nuevo campo *data* para poder enviar los datos de luz recogidos a todos los demás sensores que configura la red.

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

```
typedef struct AlvMsg {
        uint8_t  addr;
        uint8_t  board_id;
        int  packet_id;
         //uint8_t  node_id;
         int membership[SENSORES];
         uint16_t data;
} __attribute__ ((packed)) AlvMsg;
```

Figura 25: *estructura del paquete de datos AlvMsg*

Para enviar los datos recogidos por el líder hacia la estación base, utilizamos paquetes del tipo DataMsg. La estructura de estos paquetes ya fue explicada en el apartado anterior y podemos verla en la figura 22.

4.5.3.- Diagramas de secuencia

En el siguiente diagrama de secuencia (figura 26) podemos observar las interacciones entre los nodos de la red y la estación base.

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

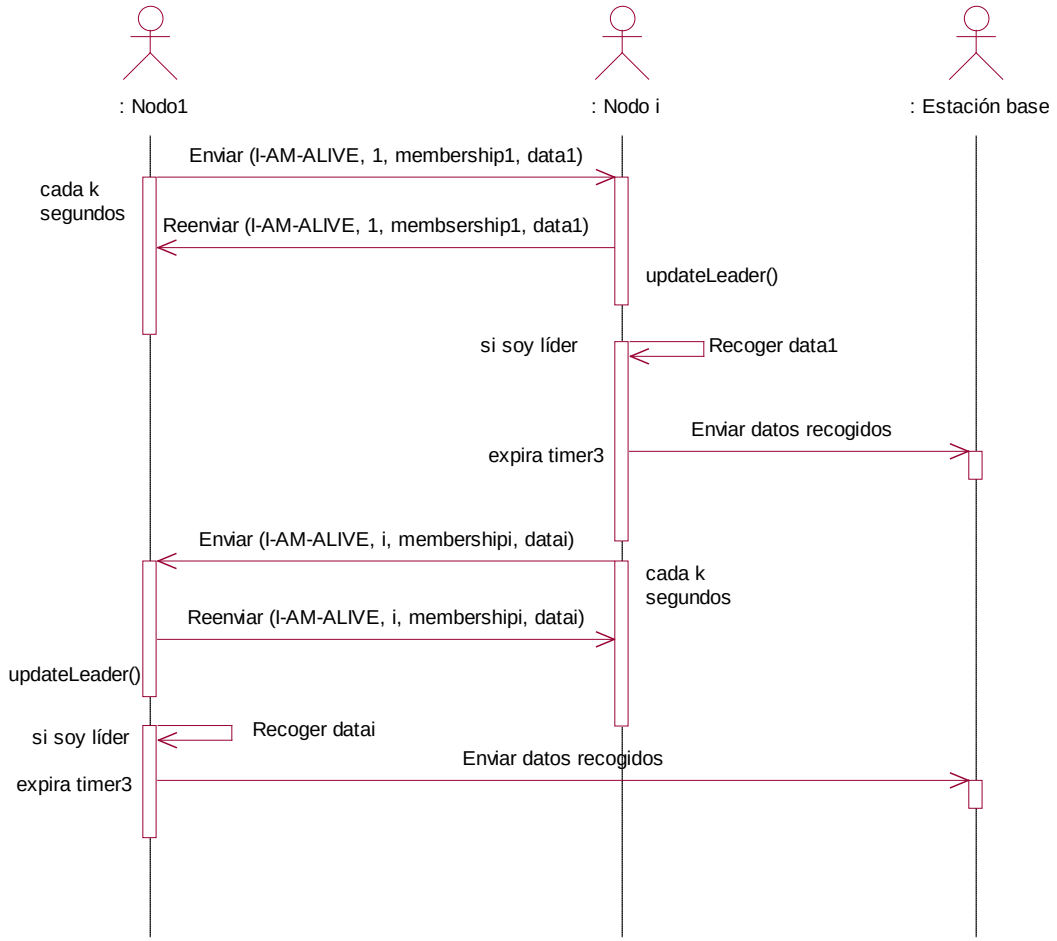


Figura 26: *diagrama de secuencia algoritmo 3*

4.5.4 Estructura de la aplicación

En cuanto a la implementación en nesC, la figura 27 muestra el diagrama de componentes de los que hace uso este tercer algoritmo, y la manera en la que están conectados.

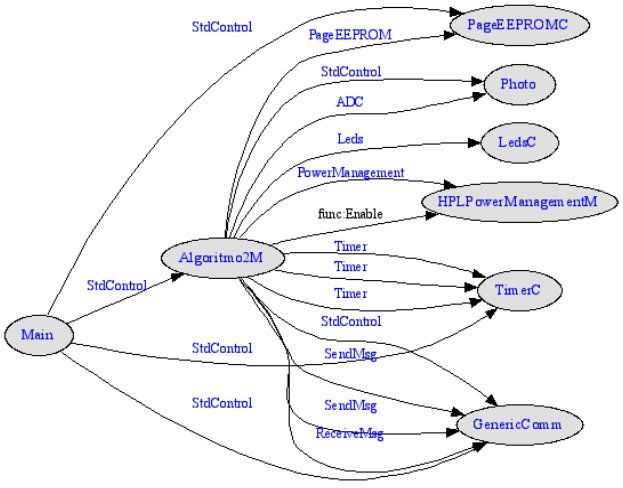


Figura 27: *diagrama de componentes del algoritmo 3*

El código completo del algoritmo puede verse al detalle en el Anexo C.

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

5.- RESULTADOS COMPARATIVOS

En este capítulo se presentan las pruebas realizadas y se comprueba el correcto funcionamiento de los tres algoritmos implementados. Describimos los escenarios elegidos para la ejecución de los tres algoritmos y se explican los pasos a seguir en la evaluación de dichos escenarios. Las pruebas realizadas se centran más que en la simple obtención y monitorización de datos del entorno, en ver como los tres algoritmos reaccionan ante posibles fallos del sistema. Una vez configurada la red con uno de los algoritmos, seremos nosotros mismos los que forzaremos los fallos en el sistema y veremos que ocurre.

5.1 Resultados para el algoritmo 1

En este primer escenario vamos a probar el correcto funcionamiento del algoritmo 1 y vamos a evaluar los resultados obtenidos. Una vez configurada correctamente la red con los diferentes sensores y con el gateway, vamos a ayudarnos de la herramienta MoteView que nos proporciona el software suministrado por el fabricante para mostrar los resultados obtenidos.

La figura 28 muestra de manera esquemática la estructura de dicho escenario:

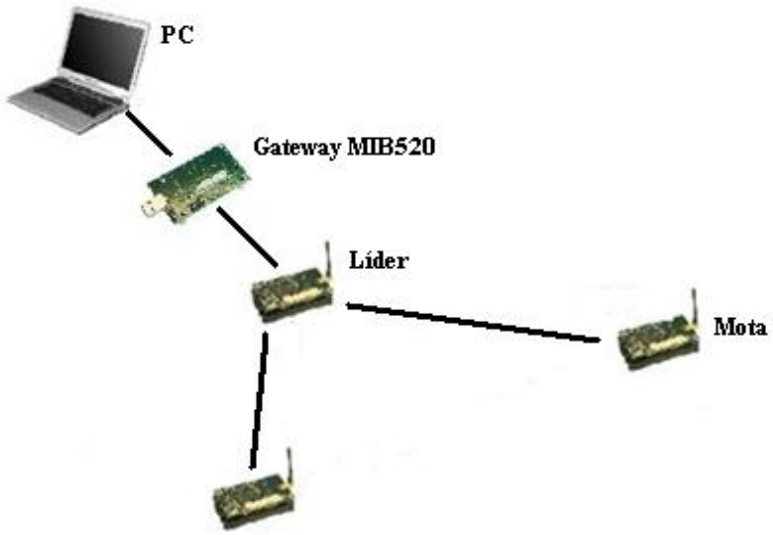


Figura 28: *arquitectura de red escenario 1*

Comenzamos configurando una red de sensores con el material suministrado para la realización de este proyecto. La red sobre la que vamos a evaluar el algoritmo consta de tres motas IRIS, un gateway MIB520 y un PC para mostrar los resultados obtenidos. Encendemos las motas y recogemos los resultados con la herramienta MoteView. La evolución del proceso y los resultados obtenidos son los siguientes.

En primer lugar, encendemos la mota 1 en solitario. En los instantes iniciales, la mota 1 se considera líder asimismo, envía mensajes BROADCAST al resto de sensores anunciándose como agregador y envía los datos medidos a la estación base. La mota permanece en este estado hasta que no encendamos las demás motas. En la figura 29 se muestra esta situación inicial.

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

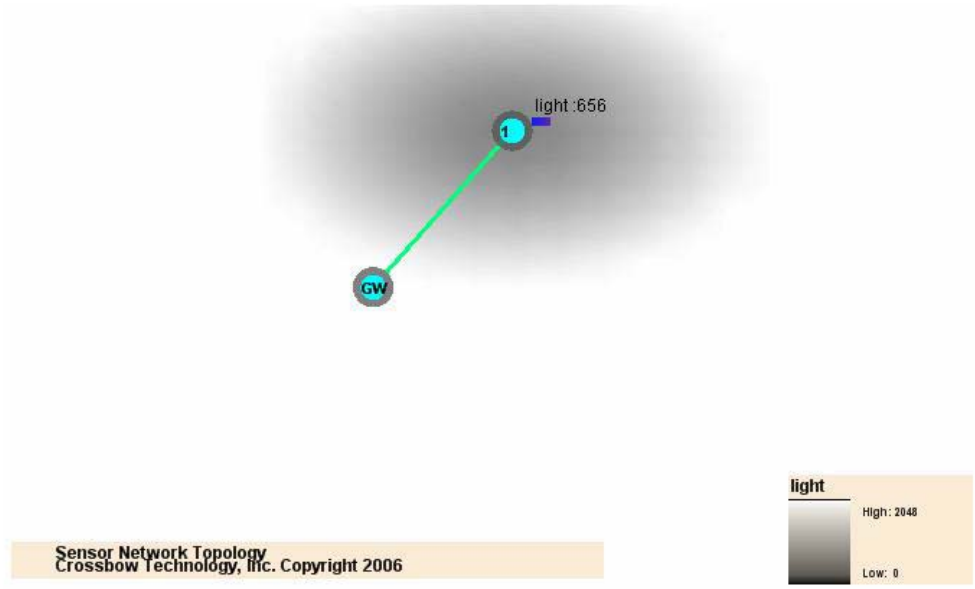


Figura 29: *situación inicial*

A continuación encendemos el resto de las motas. Tras unos instantes, el algoritmo nos proporciona un acuerdo en la agregación y la mota 1 es elegida como agregador común. La razón de esta elección se debe a que la mota con el identificador más pequeño entre las que tiene el mínimo número de encarnaciones. En la figura 30 se muestra la situación que acabamos de describir. La mota 1 es el líder de la red y es la encargada de recoger los datos medidos por las motas 2 y 3 y enviarlos a la estación base.

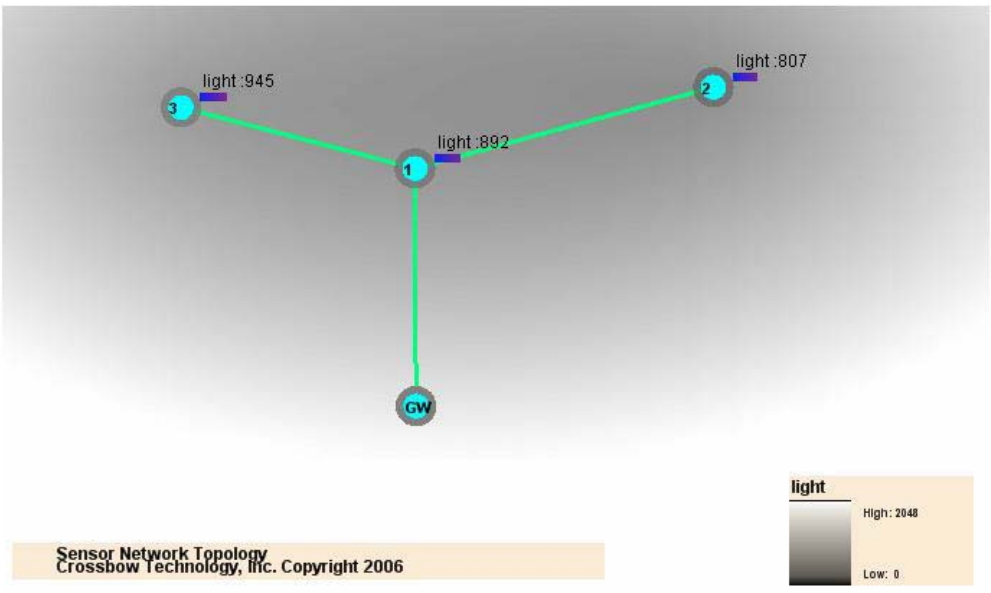


Figura 30: *elección de agregador común*

Hasta el momento hemos comprobado que el sistema funciona correctamente y la elección inicial del agregador común ha sido correcta. Vamos a dar un paso más en la verificación del algoritmo y vamos a forzar fallos en el sistema para ver que efectivamente la red tolera los fallos de la forma esperada. Para forzar que la mota 1 deje de ser líder apagamos el líder actual para aumentar su número de encarnaciones. En la figura 31 vemos el estado de la red antes de volver a encender la mota 1. La herramienta MoteView muestra la mota 1 en color gris para indicar que no se están recibiendo datos de él.

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

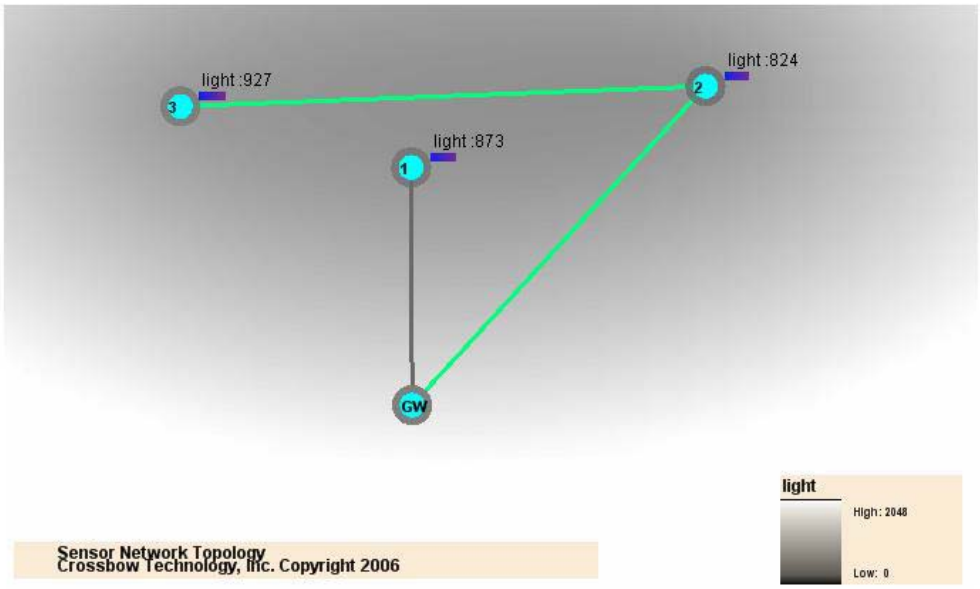


Figura 31: *fallo del agregador común*

Seguidamente volvemos a encender la mota 1. Tras unos instantes en los que no hay elegido ningún agregador (figura 32), se produce un nuevo acuerdo de agregación. Tal y como habíamos supuesto, la mota 1 deja de ser el líder de la región al haberse desplomado y es ahora la mota 2 la que ostenta el cargo. Esta nueva situación queda reflejada en la figura 33.

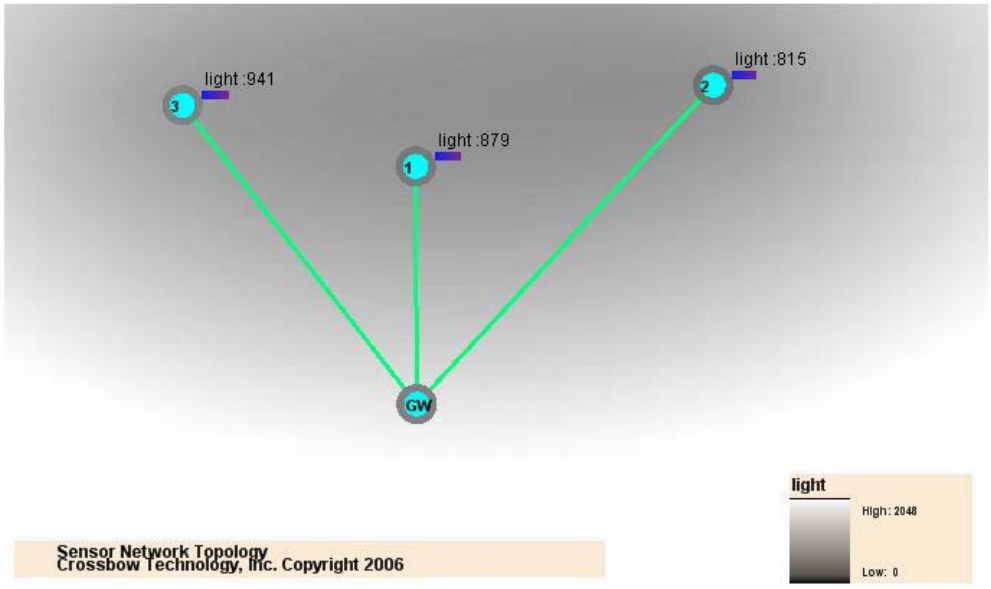


Figura 32: *agregador común no elegido*

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

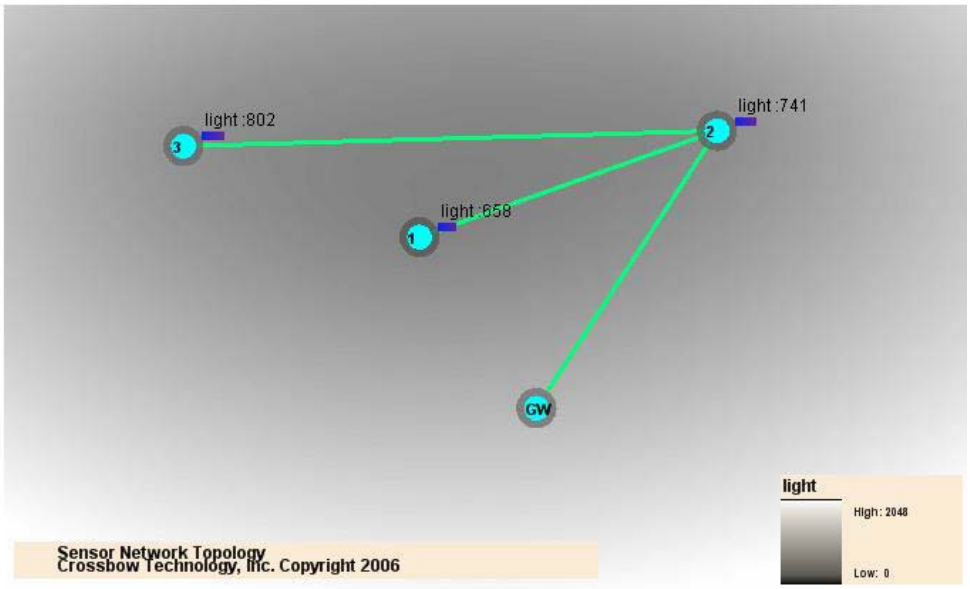


Figura 33: *elección de un nuevo agregador común al fallar el actual*

Si ahora provocamos varios desplomes en las motas 1 y 2 (apagándolas y encendiéndolas repetidamente), aumentamos su número de encarnaciones por encima del número de encarnaciones de la mota 3 originando de esta forma la elección de la mota 3 como líder o agregador común. En las figura 34 y 35, vemos como al apagar y encender la motas 1 y 2, la mota 3 acaba convirtiéndose en líder de la región.

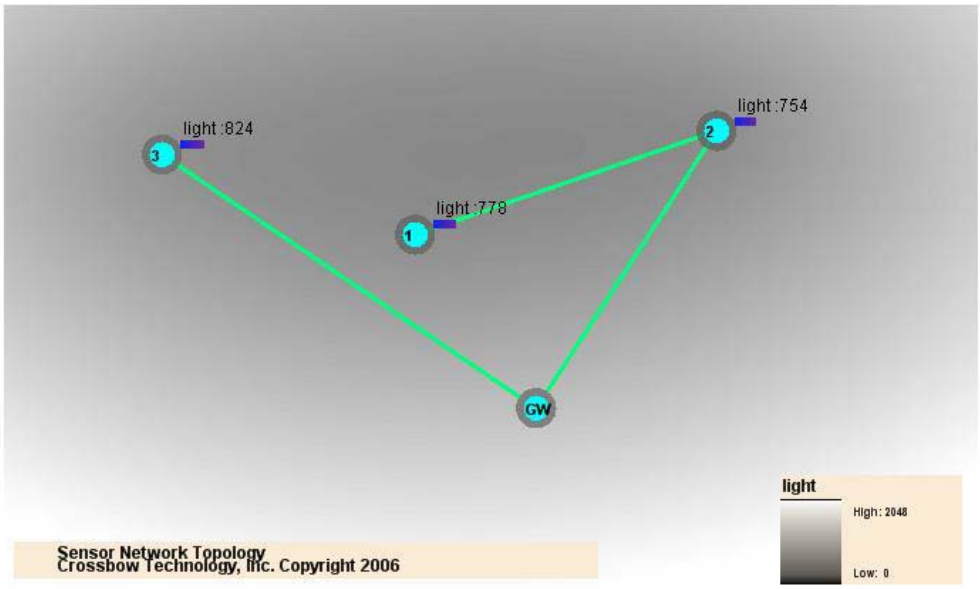


Figura 34: *la mota 3 comienza a comunicarse con la estación base*

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

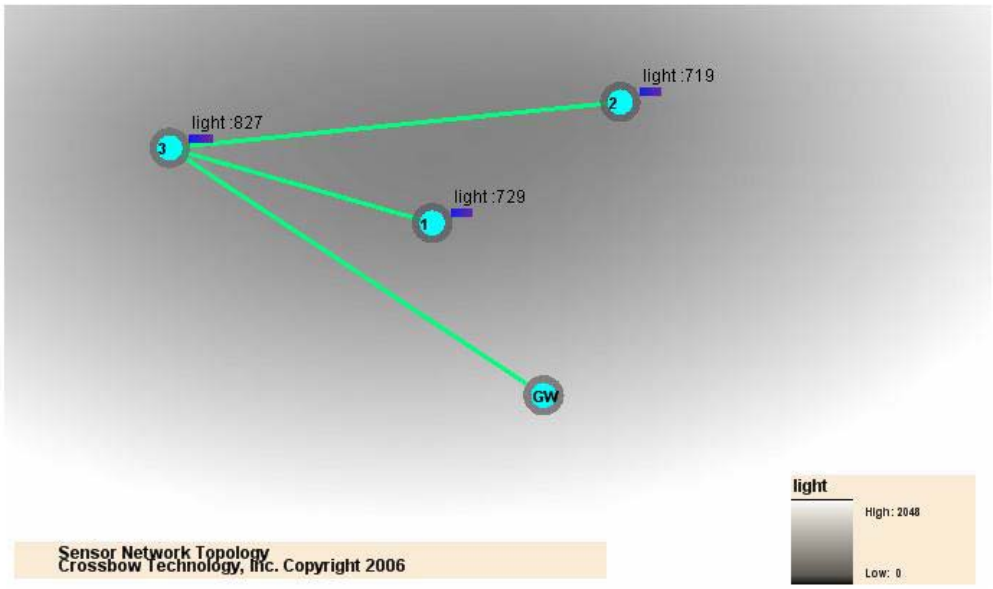
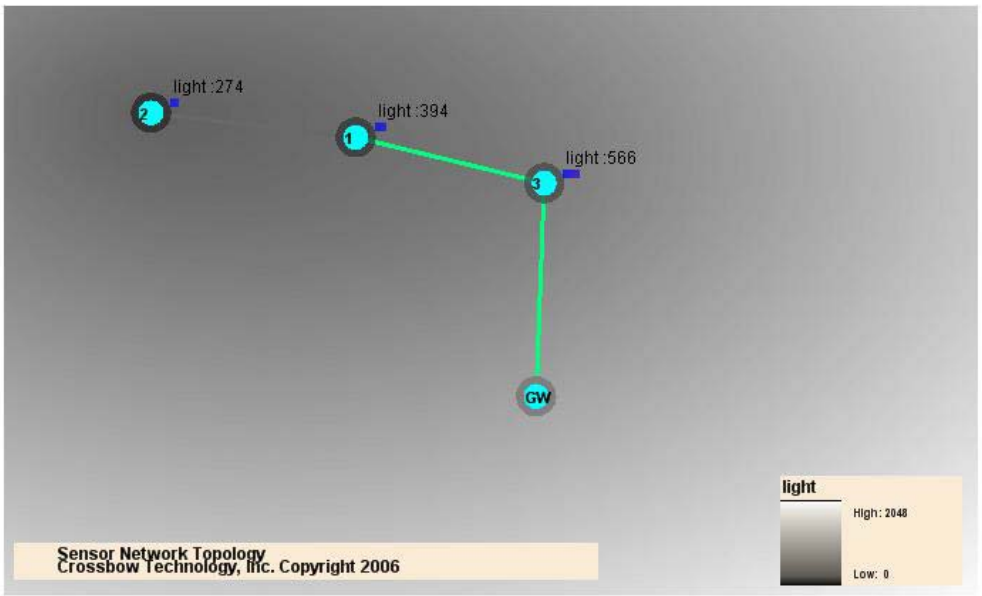


Figura 35: *la mota 3 acaba convirtiéndose en el líder de la región*

Las pruebas anteriores demuestran el correcto funcionamiento del algoritmo y su tolerancia a posibles fallos de los sensores.

Recordemos que en el algoritmo 1 inicialmente asumíamos que todo par de sensores de una región pueden comunicarse directamente. A continuación, vamos a poner a prueba al sistema para observar y estudiar qué sucede si dejamos de satisfacer esta hipótesis inicial. Siguiendo con la red anterior, alejamos lo suficiente la mota 2 para evitar de esta forma la comunicación directa con la mota 3. Experimentamos lo siguiente. La mota 2 ya no puede comunicarse directamente con el líder por lo que ya no recibe los correspondientes I-AM-ALIVE, expira su timer y se propone como agregador al resto de sensores. En nuestro caso, su radio de acción le permite comunicarse con la mota 1 (ver figura 36) pero no es elegido como líder por esta. Existe un camino multisalto entre la mota 2 y el líder (a través de la mota 1) y sin embargo los datos de la mota 2 no son recogidos y la red no funciona correctamente. Comprobamos que en cuanto un par de sensores de la red no pueden comunicarse directamente el algoritmo no funciona correctamente, el sistema falla y no se comporta de manera correcta.



upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

Figura 36: *fallo del sistema al encontrar un par de sensores que no se comunican directamente*

Por último y para concluir este apartado, vamos a realizar una última prueba orientada estudiar el consumo de batería por parte de los sensores. Para ello nos hemos ayudado de la herramienta MoteView que nos permite monitorizar el nivel de las baterías de las motas. En la tabla 2 se muestra el nivel de las baterías al inicio de las pruebas. Configuramos la red y la dejamos funcionando durante varias horas.

Id	battery
1	2.4v
2	2.9v
3	2.8v

Tabla 2: nivel inicial de las baterías

Al finalizar la prueba, el resultado obtenido puede verse en la tabla 3:

Id	battery
1	1,6v
2	2,7v
3	2,6

Tabla 3: nivel inicial de las baterías

Como se puede observar en las tablas anteriores, el consumo de batería del sensor 1 es mayor que el del resto de los sensores ya qué ha ejercido de líder la mayor parte del tiempo. Gracias a este algoritmo de elección de líder, logramos uno de nuestros objetivos fundamentales, es decir, que muchos sensores ahorren la energía de enviar los datos a la estación.

A modo de ejemplo, en las figuras 37 y 38 pueden verse los datos de luz obtenidos por los sensores durante la realización de la prueba.

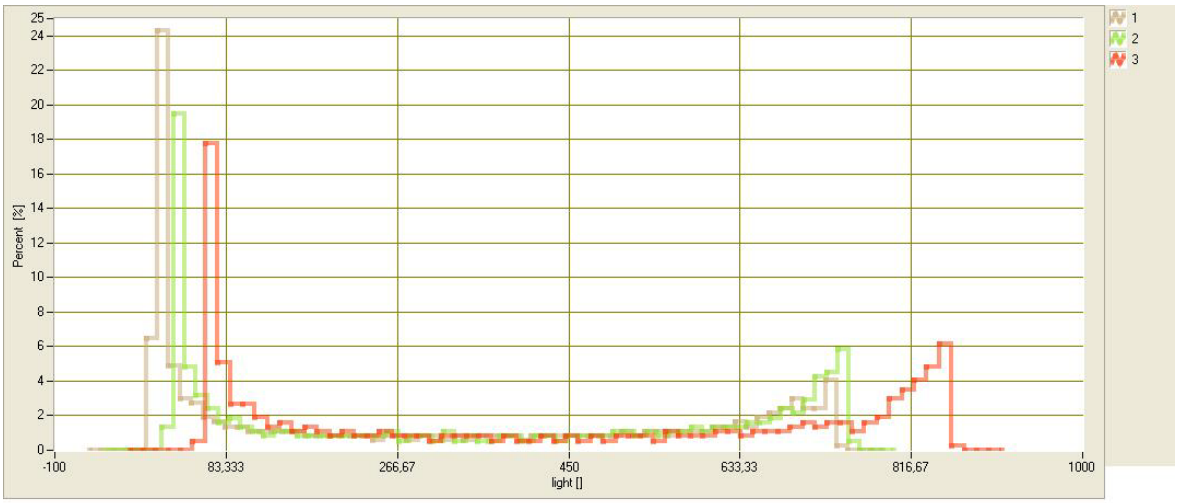


Figura 37: *histograma de los datos de luz obtenidos por los sensores*

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

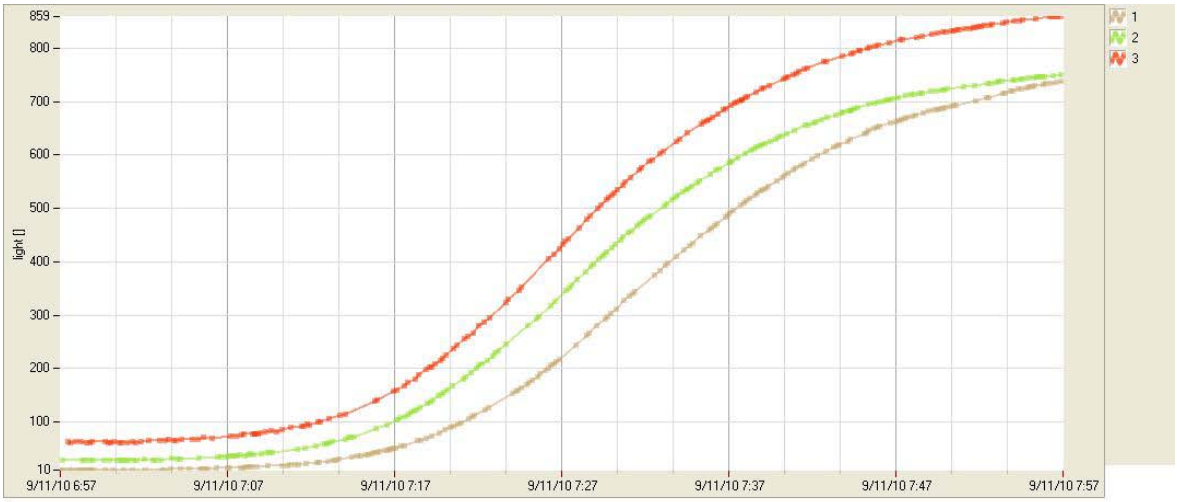


Figura 38: *evolución en el tiempo de la luz medida por los sensores*

5.2 Resultados para el algoritmo 2

En este segundo apartado vamos a probar el correcto funcionamiento del algoritmo 2 y vamos a evaluar los resultados obtenidos. Al igual que anteriormente, configuramos la red correctamente con los sensores y la estación base. La implementación del propio algoritmo no nos deja utilizar la herramienta MoteView ya que para poder utilizarla debíamos utilizar mensajes no acordes con nuestra aplicación.

La premisa previa fundamental de este algoritmo radica en que ahora simplemente suponemos que existe un subconjunto de sensores en la región que sí son capaces de alcanzar directamente al resto de los sensores de la región y además son también capaces de recibir mensajes emitidos por todos los sensores de la región.

Las pruebas de evaluación que vamos a ver a continuación tienen como principal objetivo comprobar que el algoritmo funciona correctamente y observar y analizar las principales diferencias con respecto al resto de algoritmos.

En primer lugar, vamos a realizar una primera prueba para comprobar que ante posibles fallos en los nodos el sistema se reorganiza y recupera correctamente. Al igual que en el apartado anterior, configuramos la red y la ponemos en funcionamiento. En la figura 39 vemos como los datos son correctamente recogidos por la estación base. Apagamos y encendemos las motas repetida y aleatoriamente para provocar cambios de líderes. Es importante señalar aquí que debido a problemas de cobertura y de colisiones al trabajar en un espacio cerrado, numerosos paquetes se pierden en el camino. Esto provoca que en ocasiones, las motas aumenten su número de sospechas sobre otras motas al no haber recibido sus mensajes por problemas de este tipo.

Terminal log file

Iteración 1
Líder: 3
Datos recibidos nodo 1: 785
Datos recibidos nodo 2: 725
Datos recibidos nodo 3: 848

Iteración 2
Líder: 3
Datos recibidos nodo 1: 797
Datos recibidos nodo 2: 745
Datos recibidos nodo 3: 874

Iteración 3
Líder: 3
Datos recibidos nodo 1: 771
Datos recibidos nodo 2: 733
Datos recibidos nodo 3: 876

Iteración 4
Líder: 3
Datos recibidos nodo 1: 764
Datos recibidos nodo 2: 738
Datos recibidos nodo 3: 870

Iteración 5
Líder: 3
Datos recibidos nodo 1: 770
Datos recibidos nodo 2: 739
Datos recibidos nodo 3: 871

Iteración 6
Líder: 3
Datos recibidos nodo 1: 770
Datos recibidos nodo 2: 741

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

Datos recibidos nodo 3: 872

Iteración 7
Líder: 3
Datos recibidos nodo 1: 764
Datos recibidos nodo 2: 742
Datos recibidos nodo 3: 869

Iteración 8
Líder: 3
Datos recibidos nodo 1: 767
Datos recibidos nodo 2: 723
Datos recibidos nodo 3: 868

Iteración 9
Líder: 1
Datos recibidos nodo 1: 784
Datos recibidos nodo 2: 728
Datos recibidos nodo 3: 870

Iteración 10
Líder: 1
Datos recibidos nodo 1: 797
Datos recibidos nodo 2: 704
Datos recibidos nodo 3: 847

Iteración 11
Líder: 1
Datos recibidos nodo 1: 767
Datos recibidos nodo 2: 725
Datos recibidos nodo 3: 842

Iteración 12
Líder: 1
Datos recibidos nodo 1: 460
Datos recibidos nodo 2: 737
Datos recibidos nodo 3: 654

Iteración 13
Líder: 1
Datos recibidos nodo 1: 473
Datos recibidos nodo 2: 223
Datos recibidos nodo 3: 195

Iteración 14
Líder: 1
Datos recibidos nodo 1: 480
Datos recibidos nodo 2: 407
Datos recibidos nodo 3: 178

Iteración 15
Líder: 1
Datos recibidos nodo 1: 476
Datos recibidos nodo 2: 404
Datos recibidos nodo 3: 327

Iteración 16
Líder: 1
Datos recibidos nodo 1: 477
Datos recibidos nodo 2: 403
Datos recibidos nodo 3: 394

Iteración 17
Líder: 2
Datos recibidos nodo 1: 568
Datos recibidos nodo 2: 398
Datos recibidos nodo 3: 404

Iteración 18
Líder: 2
Datos recibidos nodo 1: 568
Datos recibidos nodo 2: 489
Datos recibidos nodo 3: 416

Iteración 19
Líder: 2
Datos recibidos nodo 1: 568
Datos recibidos nodo 2: 490
Datos recibidos nodo 3: 417

Iteración 20
Líder: 2
Datos recibidos nodo 1: 569
Datos recibidos nodo 2: 490
Datos recibidos nodo 3: 417

Iteración 21
Líder: 2
Datos recibidos nodo 1: 570
Datos recibidos nodo 2: 489
Datos recibidos nodo 3: 416

Iteración 22
Líder: 2
Datos recibidos nodo 1: 567
Datos recibidos nodo 2: 487
Datos recibidos nodo 3: 413

Iteración 23
Líder: 2
Datos recibidos nodo 1: 552
Datos recibidos nodo 2: 486
Datos recibidos nodo 3: 413

Iteración 24
Líder: 3
Datos recibidos nodo 1: 552
Datos recibidos nodo 2: 487
Datos recibidos nodo 3: 417

Iteración 25
Líder: 3
Datos recibidos nodo 1: 551
Datos recibidos nodo 2: 491
Datos recibidos nodo 3: 416

Iteración 26
Líder: 3
Datos recibidos nodo 1: 550
Datos recibidos nodo 2: 491
Datos recibidos nodo 3: 416

Iteración 27
Líder: 3
Datos recibidos nodo 1: 550
Datos recibidos nodo 2: 491
Datos recibidos nodo 3: 416

Iteración 28
Líder: 3
Datos recibidos nodo 1: 548
Datos recibidos nodo 2: 490
Datos recibidos nodo 3: 416

Iteración 29
Líder: 3
Datos recibidos nodo 1: 540
Datos recibidos nodo 2: 491
Datos recibidos nodo 3: 414

Iteración 30
Líder: 3
Datos recibidos nodo 1: 544
Datos recibidos nodo 2: 491
Datos recibidos nodo 3: 416

Iteración 31
Líder: 3
Datos recibidos nodo 1: 546
Datos recibidos nodo 2: 492
Datos recibidos nodo 3: 415

Iteración 32
Líder: 3
Datos recibidos nodo 1: 544
Datos recibidos nodo 2: 490
Datos recibidos nodo 3: 414

Iteración 33
Líder: 3
Datos recibidos nodo 1: 546
Datos recibidos nodo 2: 490
Datos recibidos nodo 3: 414

Iteración 34
Líder: 3
Datos recibidos nodo 1: 546
Datos recibidos nodo 2: 491
Datos recibidos nodo 3: 414

Iteración 35
Líder: 3
Datos recibidos nodo 1: 549
Datos recibidos nodo 2: 490
Datos recibidos nodo 3: 414

Iteración 36
Líder: 1
Datos recibidos nodo 1: 539
Datos recibidos nodo 2: 398

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

Datos recibidos nodo 3: 404

Iteración 37
Líder: 1
Datos recibidos nodo 1: 540
Datos recibidos nodo 2: 490
Datos recibidos nodo 3: 415

Iteración 38
Líder: 1
Datos recibidos nodo 1: 542
Datos recibidos nodo 2: 489
Datos recibidos nodo 3: 414

Iteración 39
Líder: 1
Datos recibidos nodo 1: 537
Datos recibidos nodo 2: 488
Datos recibidos nodo 3: 414

Iteración 40
Líder: 1
Datos recibidos nodo 1: 537
Datos recibidos nodo 2: 485
Datos recibidos nodo 3: 407

Iteración 41
Líder: 2
Datos recibidos nodo 1: 513
Datos recibidos nodo 2: 480
Datos recibidos nodo 3: 396

Iteración 42
Líder: 2
Datos recibidos nodo 1: 518
Datos recibidos nodo 2: 477
Datos recibidos nodo 3: 396

Iteración 43
Líder: 2
Datos recibidos nodo 1: 534
Datos recibidos nodo 2: 480
Datos recibidos nodo 3: 407

Iteración 44
Líder: 2
Datos recibidos nodo 1: 536
Datos recibidos nodo 2: 478
Datos recibidos nodo 3: 408

Iteración 45
Líder: 2
Datos recibidos nodo 1: 540
Datos recibidos nodo 2: 479
Datos recibidos nodo 3: 414

Iteración 46
Líder: 2
Datos recibidos nodo 1: 541
Datos recibidos nodo 2: 478
Datos recibidos nodo 3: 389

Iteración 47
Líder: 2
Datos recibidos nodo 1: 341
Datos recibidos nodo 2: 468
Datos recibidos nodo 3: 358

Iteración 48
Líder: 2
Datos recibidos nodo 1: 352
Datos recibidos nodo 2: 502
Datos recibidos nodo 3: 332

Iteración 49
Líder: 2
Datos recibidos nodo 1: 358
Datos recibidos nodo 2: 496
Datos recibidos nodo 3: 351

Iteración 50
Líder: 2
Datos recibidos nodo 1: 357
Datos recibidos nodo 2: 499
Datos recibidos nodo 3: 356

Iteración 51
Líder: 2
Datos recibidos nodo 1: 407
Datos recibidos nodo 2: 492
Datos recibidos nodo 3: 353

Iteración 52
Líder: 2
Datos recibidos nodo 1: 724
Datos recibidos nodo 2: 487
Datos recibidos nodo 3: 323

Iteración 53
Líder: 2
Datos recibidos nodo 1: 720
Datos recibidos nodo 2: 491
Datos recibidos nodo 3: 325

Iteración 54
Líder: 3
Datos recibidos nodo 1: 685
Datos recibidos nodo 2: 491
Datos recibidos nodo 3: 347

Iteración 55
Líder: 3
Datos recibidos nodo 1: 685
Datos recibidos nodo 2: 493
Datos recibidos nodo 3: 348

Iteración 56
Líder: 3
Datos recibidos nodo 1: 685
Datos recibidos nodo 2: 717
Datos recibidos nodo 3: 348

Iteración 57
Líder: 3
Datos recibidos nodo 1: 765
Datos recibidos nodo 2: 714
Datos recibidos nodo 3: 780

Iteración 58
Líder: 3
Datos recibidos nodo 1: 766
Datos recibidos nodo 2: 722
Datos recibidos nodo 3: 780

Iteración 59
Líder: 3
Datos recibidos nodo 1: 765
Datos recibidos nodo 2: 722
Datos recibidos nodo 3: 779

Date: 09/09/2010 - 19:17:53
End log file Interacción 56
Líder: 3
Datos recibidos nodo 1: 685
Datos recibidos nodo 2: 717
Datos recibidos nodo 3: 348

Interacción 57
Líder: 3
Datos recibidos nodo 1: 765
Datos recibidos nodo 2: 714
Datos recibidos nodo 3: 780

Interacción 58
Líder: 3
Datos recibidos nodo 1: 766
Datos recibidos nodo 2: 722
Datos recibidos nodo 3: 780

Interacción 59
Líder: 3
Datos recibidos nodo 1: 765
Datos recibidos nodo 2: 722
Datos recibidos nodo 3: 779

End log file

Figura 39: *datos recogidos por la estación base (algoritmo 2).*

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

Comprobamos que el algoritmo funciona correctamente ante posibles errores de los nodos que componen la red y en cada momento es elegido como agregador común el sensor menos sospechoso entre los que pueden comunicarse directamente con el resto de los sensores de la región.

Recordemos que a diferencia del algoritmo anterior, en este asumimos que no todo par de sensores de una región puede comunicarse directamente. Sin embargo, existe un subconjunto de sensores en la región que sí son capaces de alcanzar directamente al resto de los sensores de la región y además son también capaces de recibir mensajes emitidos por todos los sensores de la región. Es muy importante hacer hincapié en este aspecto ya que se trata de la principal de la diferencia respecto al anterior algoritmo.

Vamos a realizar una serie de pruebas para comprobar que se cumple lo citado en el párrafo anterior.

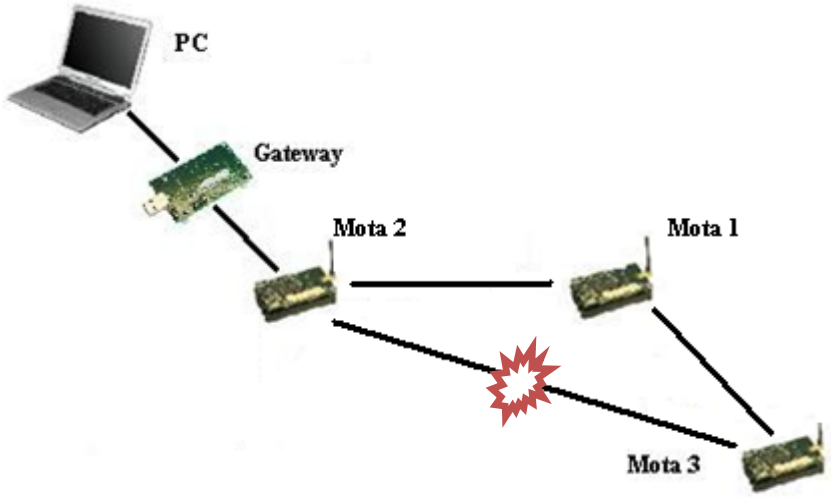


Figura 40: *arquitectura de red escenario 2*

Colocamos los tres sensores en la misma área de acción y los encendemos. Hasta ahora, el funcionamiento es el esperado. Siguiendo los criterios establecidos, los tres se ponen de acuerdo en la elección de un líder encargado de la agregación de los datos y del envío de estos hasta la estación base. Siguiendo la figura 40, alejamos los sensores 2 y 3 de forma que no puedan comunicarse entre ellos, vemos como el algoritmo sigue funcionando correctamente ya que existe el sensor 1 es capaz de alcanzar directamente a los sensores 2 y 3 y además es también capaz de recibir mensajes emitidos ambos dos. El sensor 3 no puede comunicarse directamente con el sensor 2, por tanto, aumenta constantemente sus sospechas sobre ella. Asimismo, el sensor 2 no puede comunicarse directamente con el sensor 3 y aumenta constantemente sus sospechas sobre ella. Esta información, es captada por el sensor 1 que actualiza su tabla de sospechas (counter) y acaba convirtiéndose en el líder de la red. De esta forma, hemos comprobado que en el mismo escenario en el que el algoritmo 1 no funcionaba, este lo hace de manera correcta.

Para demostrar todo lo explicado, hemos elegido al azar el sensor 2 y lo hemos conectado directamente a la estación base. Gracias a la herramienta Terminal (figura 41), obtenemos los valores de algunas de las variables más importantes de la red almacenadas en el sensor 2. Forzamos los desplomes de los sensores para ver con más claridad que el algoritmo funciona correctamente.

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

Terminal log file
Date: 09/09/2010 - 23:19:10

Expira el timer1
counter: 0 | 0 | 0 |
timer: 10000 | 10000 | -1 |
timeout: 10000 | 10000 | -1 |
counter: 0 | 0 | 0 |
Líder: 3

Datos recogidos por el líder
Datos del sensor 1: -1
Datos del sensor 2: -1
Datos del sensor 3: 180

Expira el timer1
counter: 2 | 2 | 0 |
timer: 10100 | 10100 | -1 |
timeout: 10100 | 10100 | -1 |
counter: 2 | 2 | 0 |
Líder: 3

Datos recogidos por el líder
Datos del sensor 1: 828
Datos del sensor 2: 783
Datos del sensor 3: 179

Expira el timer1
counter: 2 | 2 | 0 |
timer: 10100 | 10100 | -1 |
timeout: 10100 | 10100 | -1 |
counter: 2 | 2 | 0 |
Líder: 3

Datos recogidos por el líder
Datos del sensor 1: 816
Datos del sensor 2: 786
Datos del sensor 3: 182

Expira el timer1
counter: 2 | 2 | 0 |
timer: 10100 | 10100 | -1 |
timeout: 10100 | 10100 | -1 |
counter: 2 | 2 | 0 |
Líder: 3

Datos recogidos por el líder
Datos del sensor 1: 807
Datos del sensor 2: 777
Datos del sensor 3: 181

Expira el timer1
counter: 2 | 2 | 0 |
timer: 10100 | 10100 | -1 |
timeout: 10100 | 10100 | -1 |
counter: 2 | 2 | 0 |
Líder: 3

Expira el timer1
counter: 2 | 2 | 0 |
timer: 10100 | 10100 | -1 |
timeout: 10100 | 10100 | -1 |
counter: 2 | 2 | 0 |
Líder: 3

Datos recogidos por el líder
Datos del sensor 1: 833
Datos del sensor 2: 769
Datos del sensor 3: 182

Expira el timer1
counter: 2 | 2 | 0 |
timer: 10100 | 10100 | -1 |
timeout: 10100 | 10100 | -1 |
counter: 2 | 2 | 0 |
Líder: 3

Datos recogidos por el líder
Datos del sensor 1: 825
Datos del sensor 2: 788
Datos del sensor 3: 181

Expira el timer1
counter: 2 | 2 | 0 |
timer: 10100 | 10100 | -1 |
timeout: 10100 | 10100 | -1 |
counter: 2 | 2 | 0 |
Líder: 3

Datos recogidos por el líder
Datos del sensor 1: 824
Datos del sensor 2: 786
Datos del sensor 3: 183

Expira el timer1
counter: 2 | 2 | 0 |
timer: 10100 | 10100 | -1 |
timeout: 10100 | 10100 | -1 |
counter: 2 | 2 | 0 |
Líder: 3

Datos recogidos por el líder
Datos del sensor 1: 832
Datos del sensor 2: 769
Datos del sensor 3: 183

Expira el timer1
counter: 2 | 2 | 1 |
timer: 10100 | 10100 | -1 |
timeout: 10100 | 10100 | -1 |
counter: 2 | 2 | 1 |
Líder: 3

Datos recogidos por el líder
Datos del sensor 1: 824
Datos del sensor 2: 774
Datos del sensor 3: 182

Expira el timer1
counter: 2 | 2 | 1 |
timer: 10100 | 10100 | -1 |
timeout: 10100 | 10100 | -1 |
counter: 2 | 2 | 1 |
Líder: 3

Datos recogidos por el líder
Datos del sensor 1: 833
Datos del sensor 2: 783
Datos del sensor 3: 181

Expira el timer1
counter: 2 | 2 | 1 |
timer: 10100 | 10100 | -1 |
timeout: 10100 | 10100 | -1 |
counter: 2 | 2 | 1 |
Líder: 3

Datos recogidos por el líder
Datos del sensor 1: 836
Datos del sensor 2: 769
Datos del sensor 3: 180

Expira el timer1
counter: 2 | 2 | 1 |
timer: 10100 | 10100 | -1 |
timeout: 10100 | 10100 | -1 |
counter: 2 | 2 | 1 |
Líder: 3

Datos recogidos por el líder
Datos del sensor 1: 831
Datos del sensor 2: 782
Datos del sensor 3: 182

Expira el timer1
counter: 2 | 2 | 1 |
timer: 10100 | 10100 | -1 |
timeout: 10100 | 10100 | -1 |
counter: 2 | 2 | 1 |
Líder: 3

Expira el timer1
counter: 2 | 2 | 1 |
timer: 10100 | 10100 | -1 |
timeout: 10100 | 10100 | -1 |
counter: 2 | 2 | 1 |
Líder: 3

Datos recogidos por el líder
Datos del sensor 1: -1
Datos del sensor 2: -1
Datos del sensor 3: 182

Expira el timer1
counter: 3 | 3 | 1 |
timer: 10200 | 10200 | -1 |
timeout: 10200 | 10200 | -1 |
counter: 3 | 3 | 1 |
Líder: 3

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

Datos recogidos por el líder
Datos del sensor 1: -1
Datos del sensor 2: -1
Datos del sensor 3: 181

Expira el timer1
counter: 4 | 5 | 1 |
timer: 10300 | 10300 | -1 |
timeout: 10300 | 10300 | -1 |
counter: 4 | 5 | 1 |
Líder: 3

Datos recogidos por el líder
Datos del sensor 1: 816
Datos del sensor 2: 765
Datos del sensor 3: 181

Expira el timer1
counter: 6 | 5 | 1 |
timer: 10300 | 10300 | -1 |
timeout: 10300 | 10300 | -1 |
counter: 6 | 5 | 1 |
Líder: 3

Datos recogidos por el líder
Datos del sensor 1: 827
Datos del sensor 2: 771
Datos del sensor 3: 181

Expira el timer1
counter: 6 | 5 | 1 |
timer: 10300 | 10300 | -1 |
timeout: 10300 | 10300 | -1 |
counter: 6 | 5 | 1 |
Líder: 3

Datos recogidos por el líder
Datos del sensor 1: 819
Datos del sensor 2: 783
Datos del sensor 3: 181

Expira el timer1
counter: 6 | 5 | 1 |
timer: 10300 | 10300 | -1 |
timeout: 10300 | 10300 | -1 |
counter: 6 | 5 | 1 |
Líder: 3

Expira el timer1
counter: 6 | 5 | 1 |
timer: 10300 | 10300 | -1 |
timeout: 10300 | 10300 | -1 |
counter: 6 | 5 | 1 |
Líder: 3

Datos recogidos por el líder
Datos del sensor 1: 823
Datos del sensor 2: 779
Datos del sensor 3: 181

Expira el timer1
counter: 6 | 5 | 1 |
timer: 10300 | 10300 | -1 |
timeout: 10300 | 10300 | -1 |
counter: 6 | 5 | 1 |
Líder: 3

Datos recogidos por el líder
Datos del sensor 1: 830
Datos del sensor 2: 777
Datos del sensor 3: 182

Expira el timer1
counter: 6 | 5 | 1 |
timer: 10300 | 10300 | -1 |
timeout: 10300 | 10300 | -1 |
counter: 6 | 5 | 1 |
Líder: 3

Datos recogidos por el líder
Datos del sensor 1: 821
Datos del sensor 2: 781
Datos del sensor 3: 183

Expira el timer1
counter: 6 | 5 | 1 |
timer: 10300 | 10300 | -1 |
timeout: 10300 | 10300 | -1 |
counter: 6 | 5 | 1 |
Líder: 3

Datos recogidos por el líder
Datos del sensor 1: 809
Datos del sensor 2: 783
Datos del sensor 3: 181

Expira el timer1
counter: 6 | 5 | 1 |
timer: 10300 | 10300 | -1 |
timeout: 10300 | 10300 | -1 |
counter: 6 | 5 | 1 |
Líder: 3

Datos recogidos por el líder
Datos del sensor 1: 808
Datos del sensor 2: 785
Datos del sensor 3: 181

Expira el timer1
counter: 6 | 5 | 1 |
timer: 10300 | 10300 | -1 |
timeout: 10300 | 10300 | -1 |
counter: 6 | 5 | 1 |
Líder: 3

Expira el timer1
counter: 6 | 5 | 1 |
timer: 10300 | 10300 | -1 |
timeout: 10300 | 10300 | -1 |
counter: 6 | 5 | 1 |
Líder: 3

Datos recogidos por el líder
Datos del sensor 1: 816
Datos del sensor 2: 783
Datos del sensor 3: 180

Expira el timer1
counter: 6 | 5 | 1 |
timer: 10300 | 10300 | -1 |
timeout: 10300 | 10300 | -1 |
counter: 6 | 5 | 1 |
Líder: 3

Datos recogidos por el líder
Datos del sensor 1: 835
Datos del sensor 2: 783
Datos del sensor 3: 181

Expira el timer1
counter: 6 | 5 | 1 |
timer: 10300 | 10300 | -1 |
timeout: 10300 | 10300 | -1 |
counter: 6 | 5 | 1 |
Líder: 3

Datos recogidos por el líder
Datos del sensor 1: 836
Datos del sensor 2: 778
Datos del sensor 3: 181

Expira el timer1
counter: 6 | 5 | 1 |
timer: 10300 | 10300 | -1 |
timeout: 10300 | 10300 | -1 |
counter: 6 | 5 | 1 |
Líder: 3

Datos recogidos por el líder
Datos del sensor 1: 829
Datos del sensor 2: 772
Datos del sensor 3: 179

Expira el timer1
counter: 6 | 5 | 1 |
timer: 10300 | 10300 | -1 |
timeout: 10300 | 10300 | -1 |
counter: 6 | 5 | 1 |
Líder: 3

Datos recogidos por el líder
Datos del sensor 1: 813
Datos del sensor 2: 780
Datos del sensor 3: 178

Expira el timer1
counter: 6 | 5 | 1 |
timer: 10300 | 10300 | -1 |
timeout: 10300 | 10300 | -1 |

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

counter: 6 | 5 | 1 |
Líder: 3

Expira el timer1
counter: 6 | 5 | 1 |
timer: 10300 | 10300 | -1 |
timeout: 10300 | 10300 | -1 |
counter: 6 | 5 | 1 |
Líder: 3

Datos recogidos por el líder
Datos del sensor 1: 808
Datos del sensor 2: 784
Datos del sensor 3: 178

Expira el timer1
counter: 6 | 5 | 1 |
timer: 10300 | 10300 | -1 |
timeout: 10300 | 10300 | -1 |
counter: 6 | 5 | 1 |
Líder: 3

Datos recogidos por el líder
Datos del sensor 1: 807
Datos del sensor 2: 783
Datos del sensor 3: 179

Expira el timer1
counter: 6 | 6 | 1 |
timer: 10300 | 10400 | -1 |
timeout: 10300 | 10400 | -1 |
counter: 6 | 6 | 1 |
Líder: 3

Datos recogidos por el líder
Datos del sensor 1: 429
Datos del sensor 2: -1
Datos del sensor 3: 178

Expira el timer1
counter: 6 | 8 | 1 |
timer: 10300 | 10500 | -1 |
timeout: 10300 | 10500 | -1 |
counter: 6 | 8 | 1 |
Líder: 3

Datos recogidos por el líder
Datos del sensor 1: -1
Datos del sensor 2: 772
Datos del sensor 3: 178

Expira el timer1
counter: 7 | 8 | 1 |
timer: 10400 | 10500 | -1 |
timeout: 10400 | 10500 | -1 |
counter: 7 | 8 | 1 |
Líder: 3

Datos recogidos por el líder
Datos del sensor 1: 816
Datos del sensor 2: 777
Datos del sensor 3: 180

Expira el timer1
counter: 12 | 8 | 1 |
timer: 10400 | 10500 | -1 |
timeout: 10400 | 10500 | -1 |
counter: 12 | 8 | 1 |
Líder: 3

Expira el timer1
counter: 12 | 8 | 1 |
timer: 10400 | 10500 | -1 |
timeout: 10400 | 10500 | -1 |
counter: 12 | 8 | 1 |
Líder: 3

Datos recogidos por el líder
Datos del sensor 1: 810
Datos del sensor 2: 784
Datos del sensor 3: 179

Expira el timer1
counter: 12 | 8 | 2 |
timer: 10400 | 10500 | -1 |
timeout: 10400 | 10500 | -1 |
counter: 12 | 8 | 2 |
Líder: 3

Datos recogidos por el líder
Datos del sensor 1: 812
Datos del sensor 2: 782
Datos del sensor 3: 180

Expira el timer1
counter: 12 | 8 | 2 |
timer: 10400 | 10500 | -1 |
timeout: 10400 | 10500 | -1 |
counter: 12 | 8 | 2 |
Líder: 3

Datos recogidos por el líder
Datos del sensor 1: 826
Datos del sensor 2: 727
Datos del sensor 3: 175

Expira el timer1
counter: 12 | 8 | 2 |
timer: 10400 | 10500 | -1 |
timeout: 10400 | 10500 | -1 |
counter: 12 | 8 | 2 |
Líder: 3

Datos recogidos por el líder
Datos del sensor 1: 261
Datos del sensor 2: -1
Datos del sensor 3: 177

Date: 09/09/2010 - 23:27:27
End log file

Figura 41: *datos recogidos por la herramienta Terminal referentes al sensor 2*

Por el momento, el líder de la red es el sensor 3 ya que es el menos sospechoso entre los que pueden comunicarse directamente con el resto de los sensores de la región (por ahora todos). Según lo explicado, alejamos ahora los sensores 2 y 3 e interrumpimos su comunicación directa. En la figura 42, vemos como evoluciona la red:

Terminal log file
Date: 09/09/2010 - 23:27:30

Datos recogidos por el líder
Datos del sensor 1: 270
Datos del sensor 2: -1
Datos del sensor 3: 180

Expira el timer1
counter: 12 | 10 | 3 |
timer: 10400 | 10700 | -1 |
timeout: 10400 | 10700 | -1 |
counter: 12 | 10 | 3 |
Líder: 3

Expira el timer1
counter: 12 | 11 | 4 |
timer: 10400 | 10800 | -1 |
timeout: 10400 | 10800 | -1 |
counter: 12 | 11 | 4 |
Líder: 3

Datos recogidos por el líder
Datos del sensor 1: 270
Datos del sensor 2: -1
Datos del sensor 3: 180

Expira el timer1
counter: 12 | 12 | 6 |
timer: 10400 | 10900 | -1 |
timeout: 10400 | 10900 | -1 |

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

counter: 12 | 12 | 6 |
Líder: 3

Datos recogidos por el líder
Datos del sensor 1: 271
Datos del sensor 2: -1
Datos del sensor 3: 177

Expira el timer1
counter: 12 | 13 | 7 |
timer: 10400 | 11000 | -1 |
timeout: 10400 | 11000 | -1 |
counter: 12 | 13 | 7 |
Líder: 3

Datos recogidos por el líder
Datos del sensor 1: 230
Datos del sensor 2: -1
Datos del sensor 3: 177

Expira el timer1
counter: 12 | 14 | 7 |
timer: 10400 | 11100 | -1 |
timeout: 10400 | 11100 | -1 |
counter: 12 | 14 | 7 |
Líder: 3

Datos recogidos por el líder
Datos del sensor 1: 239
Datos del sensor 2: -1
Datos del sensor 3: 177

Expira el timer1
counter: 12 | 15 | 8 |
timer: 10400 | 11200 | -1 |
timeout: 10400 | 11200 | -1 |
counter: 12 | 15 | 8 |
Líder: 3

Datos recogidos por el líder
Datos del sensor 1: 252
Datos del sensor 2: -1
Datos del sensor 3: 178

Expira el timer1
counter: 12 | 16 | 9 |
timer: 10400 | 11300 | -1 |
timeout: 10400 | 11300 | -1 |
counter: 12 | 16 | 9 |
Líder: 3

Expira el timer1
counter: 12 | 17 | 10 |
timer: 10400 | 11400 | -1 |
timeout: 10400 | 11400 | -1 |
counter: 12 | 17 | 10 |
Líder: 3

Datos recogidos por el líder
Datos del sensor 1: 253
Datos del sensor 2: -1
Datos del sensor 3: 179

Expira el timer1
counter: 12 | 17 | 12 |
timer: 10400 | 11400 | -1 |
timeout: 10400 | 11400 | -1 |
counter: 12 | 17 | 12 |
Líder: 1

Expira el timer1
counter: 12 | 18 | 12 |
timer: 10400 | 11500 | -1 |
timeout: 10400 | 11500 | -1 |
counter: 12 | 18 | 12 |
Líder: 1

Expira el timer1
counter: 12 | 19 | 13 |
timer: 10400 | 11600 | -1 |
timeout: 10400 | 11600 | -1 |
counter: 12 | 19 | 13 |
Líder: 1

Expira el timer1
counter: 12 | 20 | 13 |
timer: 10400 | 11700 | -1 |
timeout: 10400 | 11700 | -1 |
counter: 12 | 20 | 13 |
Líder: 1

Expira el timer1
counter: 12 | 21 | 14 |
timer: 10400 | 11800 | -1 |
timeout: 10400 | 11800 | -1 |
counter: 12 | 21 | 14 |
Líder: 1

Expira el timer1
counter: 12 | 22 | 15 |
timer: 10400 | 11900 | -1 |
timeout: 10400 | 11900 | -1 |
counter: 12 | 22 | 15 |
Líder: 1

Expira el timer1
counter: 12 | 23 | 16 |
timer: 10400 | 12000 | -1 |
timeout: 10400 | 12000 | -1 |
counter: 12 | 23 | 16 |
Líder: 1

Expira el timer1
counter: 12 | 23 | 17 |
timer: 10400 | 12000 | -1 |
timeout: 10400 | 12000 | -1 |
counter: 12 | 23 | 17 |
Líder: 1

Expira el timer1
counter: 12 | 24 | 18 |
timer: 10400 | 12100 | -1 |
timeout: 10400 | 12100 | -1 |
counter: 12 | 24 | 18 |
Líder: 1

Date: 09/09/2010 - 23:30:19
End log file

Figura 42: *datos recogidos por la herramienta Terminal referentes al sensor 2*

Evidenciamos que nuestras suposiciones eran ciertas y finalmente el sensor 1 (el única capaz de comunicarse directamente con el resto de sensores) termina convirtiéndose en líder de la red.

No vamos a realizar pruebas para comprobar que el consumo de batería de los líderes es mayor que el de los sensores normales porque ya lo hicimos en la sección anterior y porque no es posible utilizar la herramienta MoteView con este algoritmo. De todas formas, los resultados que obtendríamos vendrían a corroborar lo dicho en la sección anterior sobre el consumo de batería por parte de los líderes.

5.3 Resultados para el algoritmo 3.

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

Finalmente, para acabar con el capítulo correspondiente a resultados comparativos vamos a evaluar el algoritmo 3. Consideramos que no es necesario realizar pruebas encaminadas a obtener simples datos del entorno o para ver que efectivamente la elección del agregador común y la agregación de datos es correcta. Vamos a centrarnos exclusivamente en demostrar que lo novedoso de este algoritmo funciona correctamente.

Recordemos que en este algoritmo vamos a dar un paso más hacia adelante y vamos a rebajar más si caben las hipótesis iniciales. Este algoritmo no requiere que haya ningún sensor conectado directamente con el resto, pero solamente requiere la existencia de un camino multisalto bidireccional desde algunos sensores correctos hasta el resto de los sensores.

Vamos a comprobar que efectivamente esto funciona montando la red de la figura 43.

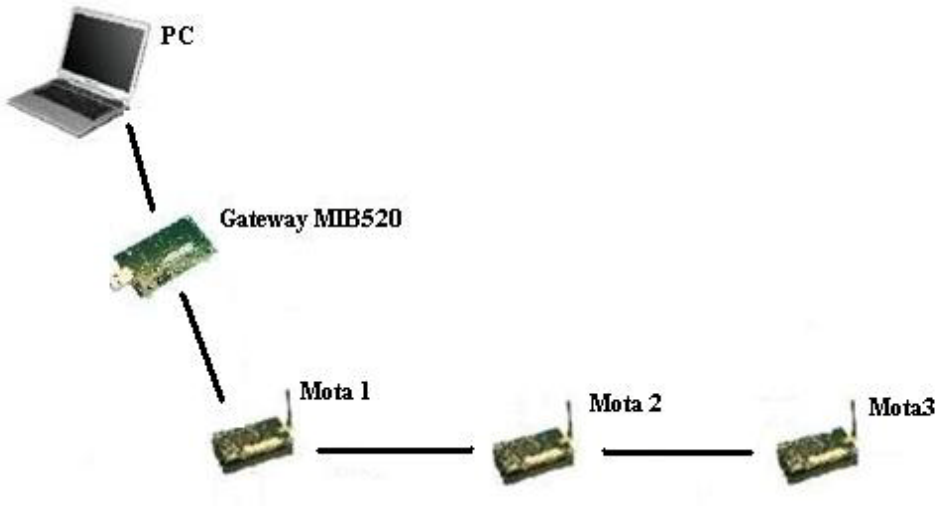


Figura 43: *arquitectura de red escenario 3*

De esta forma, la mota 1 puede comunicarse directamente con la mota 2 y esta a su vez puede comunicarse directamente con la mota 3. Sin embargo, no existe comunicación directa entre la mota 1 y la mota 3. Cargamos los sensores con el algoritmo 3 y los encendemos. Para poder utilizar la herramienta Terminal y ver por pantalla el valor de las variables más importantes de un sensor, lo conectamos directamente al gateway. En este caso conectamos el sensor 1. Los resultados obtenidos se muestran en la figura 44:

Terminal log file
Date: 10/09/2010 - 0:20:51

Expira el timer1
membership: 1 | -1 | -1 |
identificadores: -1 | -1 | -1 |
timer: -1 | -1 | -1 |
timeout: -1 | -1 | -1 |
Líder: 1

Datos recogidos por el líder
Datos del sensor 1: 80
Datos del sensor 2: -1
Datos del sensor 3: -1

Paquete recibido de: 1 con código: 17

Paquete recibido de: 2 con código: 1

Expira el timer1
membership: 1 | 1 | -1 |
identificadores: -1 | 1 | -1 |
timer: -1 | 6100 | 6100 |
timeout: -1 | 6100 | 6100 |
Líder: 1

Paquete recibido de: 1 con código: 19

Paquete recibido de: 1 con código: 23

Paquete recibido de: 2 con código: 7

Expira el timer1
membership: 1 | 1 | 5 |
identificadores: -1 | 7 | -1 |
timer: -1 | 6100 | 6400 |
timeout: -1 | 6100 | 6400 |
Líder: 1

Datos recogidos por el líder
Datos del sensor 1: 82
Datos del sensor 2: 346
Datos del sensor 3: -1

Paquete recibido de: 1 con código: 25

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

Paquete recibido de: 2 con código: 9

Expira el timer1
membership: 1 | 1 | 7 |
identificadores: -1 | 9 | -1 |
timer: -1 | 6100 | 6500 |
timeout: -1 | 6100 | 6500 |
Líder: 1

Paquete recibido de: 1 con código: 27

Datos recogidos por el líder
Datos del sensor 1: 82
Datos del sensor 2: 350
Datos del sensor 3: -1

Paquete recibido de: 2 con código: 11

Paquete recibido de: 2 con código: 26

Expira el timer1
membership: 2 | 1 | 20 |
identificadores: -1 | 26 | -1 |
timer: -1 | 6100 | 7300 |
timeout: -1 | 6100 | 7300 |
Líder: 2

Paquete recibido de: 1 con código: 44

Paquete recibido de: 2 con código: 28

Expira el timer1
membership: 2 | 1 | 21 |
identificadores: -1 | 28 | -1 |
timer: -1 | 6100 | 7400 |
timeout: -1 | 6100 | 7400 |
Líder: 2

Paquete recibido de: 1 con código: 46

Paquete recibido de: 2 con código: 30

Expira el timer1
membership: 2 | 1 | 21 |
identificadores: -1 | 30 | -1 |
timer: -1 | 6100 | 7400 |
timeout: -1 | 6100 | 7400 |
Líder: 2

Paquete recibido de: 1 con código: 48

Expira el timer1
membership: 2 | 2 | 41 |
identificadores: -1 | 69 | -1 |
timer: -1 | 6200 | 8900 |
timeout: -1 | 6200 | 8900 |
Líder: 1

Datos recogidos por el líder
Datos del sensor 1: 83
Datos del sensor 2: 332
Datos del sensor 3: -1

Paquete recibido de: 2 con código: 70

Expira el timer1
membership: 2 | 2 | 43 |
identificadores: -1 | 70 | -1 |
timer: -1 | 6200 | 9000 |
timeout: -1 | 6200 | 9000 |
Líder: 1

Paquete recibido de: 1 con código: 88

Datos recogidos por el líder
Datos del sensor 1: 83
Datos del sensor 2: 332
Datos del sensor 3: -1

Paquete recibido de: 2 con código: 72

Expira el timer1
membership: 3 | 2 | 43 |
identificadores: -1 | 72 | -1 |
timer: -1 | 6200 | 9000 |
timeout: -1 | 6200 | 9000 |
Líder: 2

Paquete recibido de: 1 con código: 90

Paquete recibido de: 2 con código: 74

Expira el timer1
membership: 3 | 2 | 44 |
identificadores: -1 | 74 | -1 |
timer: -1 | 6200 | 9100 |
timeout: -1 | 6200 | 9100 |
Líder: 2

Paquete recibido de: 1 con código: 92

Paquete recibido de: 2 con código: 76

Expira el timer1
membership: 3 | 2 | 46 |
identificadores: -1 | 76 | -1 |
timer: -1 | 6200 | 9200 |
timeout: -1 | 6200 | 9200 |
Líder: 2

Paquete recibido de: 1 con código: 94

Paquete recibido de: 3 con código: 2

Paquete recibido de: 2 con código: 79

Expira el timer1
membership: 3 | 2 | 46 |
identificadores: -1 | 79 | 2 |
timer: -1 | 6200 | 9200 |
timeout: -1 | 6200 | 9200 |
Líder: 2

Paquete recibido de: 1 con código: 97

Paquete recibido de: 2 con código: 81

Expira el timer1
membership: 3 | 2 | 46 |
identificadores: -1 | 81 | 2 |
timer: -1 | 6200 | 9200 |
timeout: -1 | 6200 | 9200 |
Líder: 2

Paquete recibido de: 1 con código: 99

Paquete recibido de: 2 con código: 83

Expira el timer1
membership: 3 | 2 | 48 |
identificadores: -1 | 83 | 2 |
timer: -1 | 6200 | 9300 |
timeout: -1 | 6200 | 9300 |
Líder: 2

Paquete recibido de: 1 con código: 101

Paquete recibido de: 1 con código: 116

Paquete recibido de: 2 con código: 100

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

Expira el timer1
membership: 4 | 3 | 53 |
identificadores: -1 | 100 | 21 |
timer: -1 | 6200 | 9500 |
timeout: -1 | 6200 | 9500 |
Líder: 2

Paquete recibido de: 1 con código: 118

Paquete recibido de: 3 con código: 27

Paquete recibido de: 2 con código: 103

Expira el timer1
membership: 6 | 6 | 53 |
identificadores: -1 | 103 | 27 |
timer: -1 | 6200 | 9500 |
timeout: -1 | 6200 | 9500 |
Líder: 1

Datos recogidos por el líder
Datos del sensor 1: 82
Datos del sensor 2: 352
Datos del sensor 3: -1

Paquete recibido de: 1 con código: 121

Paquete recibido de: 2 con código: 105

Expira el timer1
membership: 6 | 6 | 53 |
identificadores: -1 | 105 | 27 |
timer: -1 | 6200 | 9500 |
timeout: -1 | 6200 | 9500 |
Líder: 1

Paquete recibido de: 1 con código: 123

Datos recogidos por el líder
Datos del sensor 1: 83
Datos del sensor 2: 349
Datos del sensor 3: -1

Paquete recibido de: 2 con código: 107

Expira el timer1
membership: 6 | 6 | 55 |
identificadores: -1 | 107 | 27 |
timer: -1 | 6200 | 9600 |
timeout: -1 | 6200 | 9600 |
Líder: 1

Paquete recibido de: 1 con código: 125

Datos recogidos por el líder
Datos del sensor 1: 81
Datos del sensor 2: 339
Datos del sensor 3: -1

Expira el timer1
membership: 6 | 7 | 55 |
identificadores: -1 | 107 | 27 |
timer: -1 | 6300 | 9600 |
timeout: -1 | 6300 | 9600 |
Líder: 1

Expira el timer1
membership: 6 | 8 | 56 |
identificadores: -1 | 107 | 27 |
timer: -1 | 6400 | 9700 |
timeout: -1 | 6400 | 9700 |
Líder: 1

Datos recogidos por el líder
Datos del sensor 1: 82
Datos del sensor 2: -1
Datos del sensor 3: -1

Expira el timer1
membership: 6 | 9 | 57 |
identificadores: -1 | 107 | 27 |
timer: -1 | 6500 | 9800 |
timeout: -1 | 6500 | 9800 |
Líder: 1

Datos recogidos por el líder
Datos del sensor 1: 82
Datos del sensor 2: -1
Datos del sensor 3: -1

Expira el timer1
membership: 6 | 9 | 57 |
identificadores: -1 | 107 | 27 |
timer: -1 | 6500 | 9800 |
timeout: -1 | 6500 | 9800 |
Líder: 1

Datos recogidos por el líder
Datos del sensor 1: 83
Datos del sensor 2: -1
Datos del sensor 3: -1

Expira el timer1
membership: 6 | 10 | 58 |
identificadores: -1 | 107 | 27 |
timer: -1 | 6600 | 9900 |
timeout: -1 | 6600 | 9900 |
Líder: 1

Expira el timer1
membership: 9 | 9 | 1 |
identificadores: -1 | -1 | 12 |
timer: -1 | 10000 | 10000 |
timeout: -1 | 10000 | 10000 |
Líder: 3

Paquete recibido de: 1 con código: 0

Paquete recibido de: 3 con código: 14

Expira el timer1
membership: 10 | 10 | 1 |
identificadores: -1 | -1 | 14 |
timer: -1 | 10100 | 10000 |
timeout: -1 | 10100 | 10000 |
Líder: 3

Paquete recibido de: 1 con código: 2

Paquete recibido de: 3 con código: 16

Expira el timer1
membership: 1 | -1 | -1 |
identificadores: -1 | -1 | -1 |
timer: -1 | -1 | -1 |
timeout: -1 | -1 | -1 |
Líder: 1

Datos recogidos por el líder
Datos del sensor 1: 150
Datos del sensor 2: -1
Datos del sensor 3: -1

Expira el timer1
membership: 1 | -1 | -1 |
identificadores: -1 | -1 | -1 |
timer: -1 | -1 | -1 |
timeout: -1 | -1 | -1 |
Líder: 1

Datos recogidos por el líder
Datos del sensor 1: 143
Datos del sensor 2: -1
Datos del sensor 3: -1

Expira el timer1
membership: 7 | 5 | 9 |
identificadores: -1 | 34 | 21 |
timer: -1 | 10400 | 10900 |
timeout: -1 | 10400 | 10900 |
Líder: 2

Paquete recibido de: 1 con código: 38

Paquete recibido de: 2 con código: 37

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

Paquete recibido de: 2 con código: 37

Paquete recibido de: 3 con código: 24

Expira el timer1
membership: 7 | 5 | 9 |
identificadores: -1 | 37 | 24 |
timer: -1 | 10400 | 10900 |
timeout: -1 | 10400 | 10900 |
Líder: 2

Paquete recibido de: 1 con código: 41

Paquete recibido de: 3 con código: 26

Expira el timer1
membership: 7 | 7 | 9 |
identificadores: -1 | 37 | 26 |
timer: -1 | 10500 | 10900 |
timeout: -1 | 10500 | 10900 |
Líder: 1

Paquete recibido de: 1 con código: 43

Paquete recibido de: 2 con código: 42

Paquete recibido de: 3 con código: 29

Datos recogidos por el líder
Datos del sensor 1: 134
Datos del sensor 2: 344
Datos del sensor 3: 866

Expira el timer1
membership: 7 | 7 | 9 |
identificadores: -1 | 42 | 29 |
timer: -1 | 10500 | 10900 |
timeout: -1 | 10500 | 10900 |
Líder: 1

Paquete recibido de: 1 con código: 46

Paquete recibido de: 2 con código: 45

Paquete recibido de: 3 con código: 32

Expira el timer1
membership: 7 | 7 | 9 |
identificadores: -1 | 45 | 32 |
timer: -1 | 10500 | 10900 |
timeout: -1 | 10500 | 10900 |
Líder: 1

Datos recogidos por el líder
Datos del sensor 1: 134
Datos del sensor 2: 344
Datos del sensor 3: 870

Paquete recibido de: 1 con código: 49

Paquete recibido de: 2 con código: 48

Paquete recibido de: 2 con código: 48

Paquete recibido de: 3 con código: 35

Paquete recibido de: 3 con código: 35

Expira el timer1
membership: 7 | 7 | 9 |
identificadores: -1 | 48 | 35 |
timer: -1 | 10500 | 10900 |
timeout: -1 | 10500 | 10900 |
Líder: 1

Paquete recibido de: 1 con código: 52

Paquete recibido de: 2 con código: 51

Paquete recibido de: 2 con código: 51

Datos recogidos por el líder
Datos del sensor 1: 137
Datos del sensor 2: 345
Datos del sensor 3: 876

Paquete recibido de: 3 con código: 38

Paquete recibido de: 3 con código: 38

Expira el timer1
membership: 7 | 7 | 9 |
identificadores: -1 | 51 | 38 |
timer: -1 | 10500 | 10900 |
timeout: -1 | 10500 | 10900 |
Líder: 1

Paquete recibido de: 2 con código: 54

Paquete recibido de: 2 con código: 54

Datos recogidos por el líder
Datos del sensor 1: 137
Datos del sensor 2: 344
Datos del sensor 3: 882

Paquete recibido de: 3 con código: 41

Expira el timer1
membership: 7 | 7 | 9 |
identificadores: -1 | 54 | 41 |
timer: -1 | 10500 | 10900 |
timeout: -1 | 10500 | 10900 |
Líder: 1

Paquete recibido de: 1 con código: 58

Paquete recibido de: 2 con código: 57

Paquete recibido de: 3 con código: 44

Paquete recibido de: 3 con código: 44

Datos recogidos por el líder
Datos del sensor 1: 135
Datos del sensor 2: 343
Datos del sensor 3: 863

Expira el timer1
membership: 7 | 7 | 9 |
identificadores: -1 | 57 | 44 |
timer: -1 | 10500 | 10900 |
timeout: -1 | 10500 | 10900 |
Líder: 1

Paquete recibido de: 1 con código: 61

Paquete recibido de: 2 con código: 60

Paquete recibido de: 3 con código: 47

Datos recogidos por el líder
Datos del sensor 1: 135
Datos del sensor 2: 346
Datos del sensor 3: 868

Expira el timer1
membership: 7 | 7 | 9 |
identificadores: -1 | 60 | 47 |

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

timer: -1 | 10500 | 10900 |
timeout: -1 | 10500 | 10900 |
Líder: 1

Paquete recibido de: 1 con código: 64

Paquete recibido de: 2 con código: 63
membershipq[1]: 7

Paquete recibido de: 3 con código: 50

Expira el timer1
membership: 7 | 7 | 9 |
identificadores: -1 | 63 | 50 |
timer: -1 | 10500 | 10900 |
timeout: -1 | 10500 | 10900 |
Líder: 1

Datos recogidos por el líder
Datos del sensor 1: 135
Datos del sensor 2: 347
Datos del sensor 3: 868

Paquete recibido de: 2 con código: 65

Expira el timer1
membership: 7 | 7 | 9 |
identificadores: -1 | 65 | 50 |
timer: -1 | 10500 | 10900 |
timeout: -1 | 10500 | 10900 |
Líder: 1

Datos recogidos por el líder
Datos del sensor 1: 137
Datos del sensor 2: 349
Datos del sensor 3: -1

Expira el timer1
membership: 7 | 7 | 10 |
identificadores: -1 | 65 | 50 |
timer: -1 | 10500 | 11000 |
timeout: -1 | 10500 | 11000 |
Líder: 1

Datos recogidos por el líder
Datos del sensor 1: 138
Datos del sensor 2: -1
Datos del sensor 3: -1

Expira el timer1
membership: 7 | 8 | 11 |
identificadores: -1 | 65 | 50 |
timer: -1 | 10600 | 11100 |
timeout: -1 | 10600 | 11100 |
Líder: 1

Datos recogidos por el líder
Datos del sensor 1: 161
Datos del sensor 2: -1
Datos del sensor 3: -1

Expira el timer1
membership: 7 | 9 | 12 |
identificadores: -1 | 65 | 50 |
timer: -1 | 10700 | 11200 |
timeout: -1 | 10700 | 11200 |
Líder: 1

Datos recogidos por el líder
Datos del sensor 1: 148
Datos del sensor 2: -1
Datos del sensor 3: -1

Expira el timer1
membership: 7 | 10 | 13 |
identificadores: -1 | 65 | 50 |
timer: -1 | 10800 | 11300 |
timeout: -1 | 10800 | 11300 |
Líder: 1

Expira el timer1
membership: 7 | 11 | 14 |
identificadores: -1 | 65 | 50 |
timer: -1 | 10900 | 11400 |
timeout: -1 | 10900 | 11400 |
Líder: 1

Datos recogidos por el líder
Datos del sensor 1: 144
Datos del sensor 2: -1
Datos del sensor 3: -1

Paquete recibido de: 1 con código: 73

Paquete recibido de: 2 con código: 72

Paquete recibido de: 3 con código: 54

Paquete recibido de: 3 con código: 54

Expira el timer1
membership: 13 | 11 | 14 |
identificadores: -1 | 72 | 54 |
timer: -1 | 10900 | 11400 |
timeout: -1 | 10900 | 11400 |
Líder: 2

Paquete recibido de: 1 con código: 76

Paquete recibido de: 2 con código: 75

Paquete recibido de: 3 con código: 57

Expira el timer1
membership: 13 | 11 | 14 |
identificadores: -1 | 75 | 57 |
timer: -1 | 10900 | 11400 |
timeout: -1 | 10900 | 11400 |
Líder: 2

Paquete recibido de: 1 con código: 79

Paquete recibido de: 2 con código: 78

Paquete recibido de: 3 con código: 60

Datos recogidos por el líder
Datos del sensor 1: 128
Datos del sensor 2: -1
Datos del sensor 3: -1

Expira el timer1
membership: 14 | 17 | 18 |
identificadores: -1 | 113 | 95 |
timer: -1 | 11400 | 11800 |
timeout: -1 | 11400 | 11800 |
Líder: 1

Paquete recibido de: 3 con código: 101

Paquete recibido de: 2 con código: 117

Expira el timer1
membership: 20 | 19 | 18 |
identificadores: -1 | 117 | 101 |
timer: -1 | 11500 | 11800 |
timeout: -1 | 11500 | 11800 |
Líder: 3

Paquete recibido de: 3 con código: 103

Paquete recibido de: 2 con código: 119

Expira el timer1
membership: 22 | 19 | 18 |
identificadores: -1 | 119 | 103 |
timer: -1 | 11500 | 11800 |
timeout: -1 | 11500 | 11800 |
Líder: 3

Paquete recibido de: 1 con código: 130

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

Expira el timer1
membership: 22 | 20 | 19 |
identificadores: -1 | 119 | 103 |
timer: -1 | 11600 | 11900 |
timeout: -1 | 11600 | 11900 |
Líder: 3

Paquete recibido de: 3 con código: 107

Paquete recibido de: 2 con código: 123

Expira el timer1
membership: 23 | 20 | 19 |
identificadores: -1 | 123 | 107 |
timer: -1 | 11600 | 11900 |
timeout: -1 | 11600 | 11900 |
Líder: 3

Paquete recibido de: 1 con código: 134

Paquete recibido de: 3 con código: 110

Paquete recibido de: 2 con código: 126

Expira el timer1
membership: 23 | 20 | 19 |
identificadores: -1 | 126 | 110 |
timer: -1 | 11600 | 11900 |
timeout: -1 | 11600 | 11900 |
Líder: 3

Paquete recibido de: 1 con código: 137

Paquete recibido de: 3 con código: 113

Paquete recibido de: 2 con código: 129

Expira el timer1
membership: 23 | 20 | 19 |
identificadores: -1 | 129 | 113 |
timer: -1 | 11600 | 11900 |
timeout: -1 | 11600 | 11900 |
Líder: 3

Paquete recibido de: 1 con código: 140

Paquete recibido de: 3 con código: 116

Paquete recibido de: 2 con código: 132

Expira el timer1
membership: 23 | 20 | 19 |
identificadores: -1 | 132 | 116 |
timer: -1 | 11600 | 11900 |
timeout: -1 | 11600 | 11900 |
Líder: 3

Expira el timer1
membership: 23 | 20 | 19 |
identificadores: -1 | 132 | 116 |
timer: -1 | 11600 | 11900 |
timeout: -1 | 11600 | 11900 |
Líder: 3

Expira el timer1
membership: 23 | 21 | 20 |
identificadores: -1 | 132 | 116 |
timer: -1 | 11700 | 12000 |
timeout: -1 | 11700 | 12000 |
Líder: 3

Paquete recibido de: 3 con código: 123

Expira el timer1
membership: 24 | 22 | 20 |
identificadores: -1 | 132 | 123 |
timer: -1 | 11800 | 12000 |
timeout: -1 | 11800 | 12000 |
Líder: 3

Expira el timer1
membership: 24 | 23 | 21 |
identificadores: -1 | 132 | 123 |
timer: -1 | 11900 | 12100 |
timeout: -1 | 11900 | 12100 |
Líder: 3

Paquete recibido de: 3 con código: 127

Expira el timer1
membership: 26 | 24 | 21 |
identificadores: -1 | 132 | 127 |
timer: -1 | 12000 | 12100 |
timeout: -1 | 12000 | 12100 |
Líder: 3

Expira el timer1
membership: 26 | 24 | 21 |
identificadores: -1 | 132 | 127 |
timer: -1 | 12000 | 12100 |
timeout: -1 | 12000 | 12100 |
Líder: 3

Paquete recibido de: 3 con código: 131

Expira el timer1
membership: 2 | -1 | -1 |
identificadores: -1 | -1 | -1 |
timer: -1 | -1 | -1 |
timeout: -1 | -1 | -1 |
Líder: 1

Date: 10/09/2010 - 0:52:29
End log file

Figura 44: *datos recogidos por la herramienta Terminal referentes al sensor 1*

A la vista de los resultados, llegamos a la conclusión de que el algoritmo funciona correctamente y que únicamente requiere que exista un camino multisalto bidireccional desde algunos sensores correctos hasta el resto de los sensores ya que los mensajes aceptados por los sensores son continuamente reenviados al resto. Observamos que a diferencia del algoritmo 2, cualquier sensor puede ser elegido como líder y que pese a no haber comunicación directa entre el sensor 1 y el sensor 3, ambos reciben mutuamente sus paquetes. Hemos solucionado por tanto, el problema que se nos planteaba en las pruebas realizadas en el algoritmo 2.

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

6.-CONCLUSIONES Y LÍNEAS FUTURAS

En este capítulo exponemos las conclusiones a las que hemos llegado tras analizar los resultados comparativos obtenidos con los tres algoritmos implementados. También hacemos referencia a posibles mejoras que serían aplicables al desarrollo que hemos llevado a cabo y realizamos unas observaciones para un futuro perfeccionamiento del actual trabajo.

Hemos centrado el trabajo en dos aspectos importantes dentro de las comunicaciones de redes de sensores inalámbricos, como son el consumo de batería y la tolerancia a fallos de cada uno de los nodos que la componen. Para ello, hemos implementado tres algoritmos de elección de agregador y agregación de datos.

En los tres algoritmos implementados, hemos implantado un sistema de agregación de datos mediantes el cual es un único sensor el encargado de transmitir toda la información hacia la estación base. Los sensores de una red eligen a sus líderes siguiendo un criterio de fiabilidad, basado en su número de encarnación.

Los resultados obtenidos en la Sección 5.1 y recogidos en las Tablas 2 y 3 demuestran que el consumo de batería de los líderes de la red es mayor que el de los sensores normales. Gracias a este sistema conseguimos que se distribuya mejor el consumo asociado a actuar como agregador entre todos los sensores de la red. Así se logra una mayor longevidad de la red y una mejor calidad de servicio, puesto que los sensores que actúan como agregadores cesan en dicha actividad cuando alcanzan un determinado umbral de batería y así el consumo energético asociado a la tarea de agregación no produce su muerte por agotamiento de la batería. El establecimiento de un umbral de batería nos permite que una vez un líder supere un límite en su batería, ceda su condición a otro sensor susceptible de ser líder por su fiabilidad. Logramos de esta forma una reducción equilibrada de las pilas de los sensores bien comunicados o correctos de cada región, alargando así su vida.

En la toma de resultados y conclusiones nos hemos centrado en garantizar la tolerancia a fallos de cada red sin comparar los algoritmos en cuanto a tiempo empleado en la elección del agregador común o en otras cuestiones similares, lo que podría ser una línea de trabajo futura.

Los algoritmos basados en el detector de fallos Ω nos proporciona n un acuerdo en la agregación incluso si han existido fallos. Sin embargo, la manera de alcanzar este acuerdo dista de unos a otros. A la vista de los resultados obtenidos en la Sección 5 podemos concluir que la topología y la finalidad de la red inalámbrica condicionarán la elección de uno u otro algoritmo.

El primero de ellos exige que todo par de sensores de la red puedan comunicarse directamente. Hemos descubierto que en el momento que deja de cumplirse esa hipótesis el sistema no sabe auto configurarse y se vuelve inestable. La tolerancia a fallos de la red queda totalmente en entredicho por el simple hecho de que cierto par de sensores ya no sean capaces de comunicarse directamente. Deberemos tener en cuenta qué tipo de red tengamos y su distribución para evitar que problemas de este tipo no permitan el correcto funcionamiento del algoritmo.

La implementación del segundo algoritmo soluciona gran parte de los problemas del primero pero sigue estableciendo como requisito inicial la existencia de un subconjunto de sensores en la región que sean capaces de alcanzar directamente al resto de los sensores de la región y además que sean también capaces de recibir mensajes emitidos por todos los sensores de la región. La aplicación de este algoritmo en un escenario real dependerá de si podemos garantizar previamente la existencia de este tipo de sensores ya que si dejan de existir, los problemas de la red acabarán por volverla inadecuada.

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

Por último, en el tercer algoritmo dábamos un paso más y solucionamos la gran restricción del anterior. Ahora solamente requerimos la existencia de un camino multisalto desde algunos sensores correctos hasta el resto de los sensores. Esta exigencia es considerablemente más fácil de garantizar en un escenario real que las dos anteriores. Es obvio que resulta mucho más sencillo encontrar un camino multisalto desde algunos sensores correctos hasta el resto de los sensores que garantizar que en una red de sensores de cientos de nodos, todo par de ellos va a poder comunicarse directamente (algoritmo 1) o que van a existir nodos visibles por todos los demás.

Sin embargo, no debemos dejarnos engañar y asegurar que el mejor algoritmo de los tres es el tercero ya que, como bien hemos dicho, dependerá de los objetivos buscados y de las posibilidades hardware disponible.

En cuanto a posibles mejoras en la implementación de los algoritmos, uno de los aspectos que podríamos mejorar sería el de introducir en los mecanismos para evitar la colisión de mensajes enviados por los nodos y para mejorar la cobertura de red de los sensores. Otra de las opciones contempladas, es la de adaptar los algoritmos 2 y 3 a posibles ampliaciones en la red de sensores en las que estén funcionando para no mantener fijo el número de sensores que la conforman desde el inicio hasta el final del algoritmo.

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

BIBLIOGRAFÍA

[1] M. Larrea, C. Martín, and J.J. Astrain, "Fault-Tolerant Aggregator Election and Data Aggregation in Wireless Sensor Networks", *International Journal of Communication Networks and Distributed Systems*, Vol. 3, No. 2, pp. 93-115, 2009. DOI: 10.1504/IJCNDS.2009.026821

[2] D. Culler, D. Estrin, M. Srivastava, "Overview of Sensor Networks", *IEEE Computer Magazine*, 37(9): 41-49, August, 2004.

[3] http://avances-tecnologicos.euroresidentes.com/2004/06/redes-de-sensores-sin-cable.html

[4] R. Dunlap, *In-Network Aggregation in Wireless Sensor Networks*. College of Computing. Georgia Institute of Technology. USA, 2004

[5] Karl H.,Willig A. *Protocols and Architectures for Wireless Sensor Networks*. John Wiley & Sons. ISBN 0-470-09510-5, 2005.

[6] N. Shrivastava, C. Buragohain, D. Agrawal, "Medians and Beyond: New aggregation Techniques for Sensor Networks", *Proceedings of the Second ACM Conference on Embedded Networked Sensor Systems (SenSys 2004)*, 2004. http://www.cs.virginia.edu/~son/cs851/papers/ucsb.sensys04.pdf

[7] I. Solis and K. Obraczka, "In-network aggregation trade-offs for data collection in wireless sensor networks". *International Journal of Sensor Networks*, 1(3-4): 200 – 212, 2006.

[8] J.N. Al-Karaki, A.E. Kamal, "Routing techniques in wireless sensor networks: a survey". *IEEE Wireless Communications*, 2004.

[9] B. Prydatek, D. X. Song and A. Perrig. "Sia: secure information aggregation in sensor networks", *Proceedings of the 1st International Conference in Embedded Network Sensor Systems, SenSys 2003*, pages 255-256, ACM. Los Angeles, California(USA), November 2003.

[10] Y. Chen, A. Liestman and J. Liu, "Energy-efficient data aggregation hierarchy for wireless sensor networks", *Proceedings of the 2nd International Conference on Quality of Service in Heterogeneous Wired/Wireless Networks (QSHINE'2005)*, IEEE Computer Society Press, Lake Buena Vista, Florida (USA), 2005.

[11] F. Hu, X. Cao and C. May, "Optimized scheduling for data aggregation in wireless sensor networks", *Proceedings of the International Conference on Information Technology: Coding and Computing (ITCC'2005)*, IEEE Computer Society, Las Vegas, Nevada (USA), Vol. 2, pp.557–561, 2005.

[12] N. Lynch, *Distributed Algorithms*, Boston, MT, USA, 1996.

[13] "TinyOS", http://www.tinyos.net

[14] T. Chandra, V. Hadzilacos and S. Toueg, "The weakest failure detector for solving consensus", *Journal of the ACM*, 43(4):685–722, 1996.

[16] "Wikipedia", http://es.wikipedia.org/"

[17] K. Vaidyanathan, S. Sur, S. Narravula and P. Sinha. *Data Aggregation Techniques in Sensor Networks*, Technical Report 11/04-TR60, The Ohio State University, 2004.

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

[18] M. Kumar, L. Schwiebert and M. Brockmeyer, “Efficient data aggregation middleware for wireless sensor networks”, *Proceedings of the First International Conference on Mobile, Ad-Hoc, and Sensor Systems (MASS’2004)*, IEEE Computer Society Press, Ft. Lauderdale, Florida, USA, pp.579–581, 2004.

[19] E. Jiménez, S. Arévalo and A. Fernández, “Implementing unreliable failure detectors with unknown membership”, *Information Processing Letters*, 100(2): 60–63, 2006.

[20] W. Heinzelman, A. Chandrakasan, and H. Balakrishnan, “Energy-efficient communication protocol for wireless micronsensor networks”, *Proceedings of the 33rd Hawaiian International Conference on Systems Science (HICSS’2000)*, Maui, Hawaii, USA, 2000.

[21] P. Tsuchiya, “The landmark hierarchy: A new hierarchy for routing in very large networks”, *Proceedings of the ACM Symposium on Communications Architectures and Protocols (SIGCOMM’88)*, Stanford, CA, USA, pp.35–42, 1988.

[22] G. Chockler, S. Gilbert and B. Patt-Shamir, “Communication-efficient probabilistic quorum systems for sensor networks”, *Proceedings of the 4th IEEE Conference on Pervasive Computing and Communications Workshops (Per-Com’2006 Workshops)*, IEEE Computer Society Press, Pisa, Italy, pp.111–117, 2006.

[23] K. Hatzis, G. Pentaris, P. Spirakis, V. Tampakas and R. Tan, “Fundamental control algorithms in mobile networks”, *Proceedings of the 11th ACM Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA’99)*, ACM Press, Saint-Malo, France, pp.251–260, 1999.

[24] N. Malpani, J. Welch and N. Vaidya, “Leader election algorithms for mobile ad hoc Networks”, *Proceedings of the 4th International Workshop on Algorithms and Methods for Mobile Computing and Communications*, pp.96–103, 2000.

[25] V. Park and M. Corson, “A highly adaptative distributed routing algorithm for mobile wireless networks”, *Proceedings of the 16th IEEE INFOCOM*, pp.1405–1413, 1997.

[26] S. Vasudevan, J. Kurose and D. Towsley, “Design and analysis of a leader election algorithm for mobile ad hoc networks”, *Proceedings of the 12th International Conference on Network Protocols (ICNP’2004)*, IEEE Computer Society Press, Berlin, Germany, pp.350–360, 2004.

[27] S. Masum, A. Ali and M. Bhuiyan, “Asynchronous leader election in mobile ad hoc networks”, *Proceedings of the 20th International Conference on Advanced Information Networking and Applications (AINA’06)*, IEEE Computer Society Press, Vienna, Austria, Vol. 2, pp.827–831, 2006.

[28] F. Zhao, L. Guibas, “Wireless Sensor Networks an information Processing Approach”, Edit. Morgan Kaufman, 2004.

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

Tolerancia a fallos y autoconfiguración en redes de sensores

Realizado por: Pablo Albizu Balerdi

Dirigido por: José Javier Astrain Escola

Dpto. Ingeniería Matemática e Informática - UNIVERSIDAD PÚBLICA DE NAVARRA

PAMPLONA, 15 de Septiembre de 2010

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa

Todos los derechos reservados
Eskubide guztiak erresalbatu dira

Contenido de la presentación

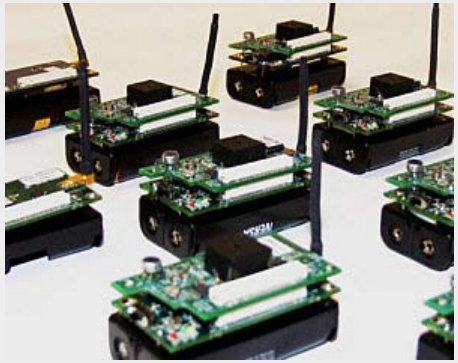
- Introducción: objetivos y motivaciones
- Entorno de trabajo
- Algoritmos de agregación implementados
- Análisis y diseño
- Pruebas realizadas
- Conclusiones
- Perspectivas futuras

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

Introducción

- Red de sensores
  - Red de dispositivos de reducidas capacidades de cómputo y almacenamiento (<<nodos o motas>>) equipados con sensores.

  - Misión principal: obtención de datos del entorno (temperatura, sonido, vibración, presión, movimiento, contaminantes, luminosidad, etc.).



upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

Nuestro objetivo

- ✓ Estudio de los mecanismos para garantizar la autoconfiguración de redes inalámbricas de sensores y su tolerancia y recuperación frente a fallos.
- ✓ Minimización del consumo de energía
- ✓ Implementación y comparación de tres algoritmos de elección de agregador y agregación de datos

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

Entorno de trabajo

- Hardware: Kit Crossbow
  - Motas IRIS
  - Placas sensoras MDA100CA
  - Estación base USB MIB520

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

Entorno de trabajo

- Software
  - Sistema operativo TinyOS
    - Código libre.
    - Basado en componentes y eventos.
    - Destinado para utilizarse en sensores.
  - Lenguaje de programación Nesc.
    - Dialecto de C .
    - Optimizado para las limitaciones de memoria de las redes de sensores.
    - Orientado a componentes.

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

Algoritmos de agregación

- Antecedentes: Detector de fallos Omega propuesto por Chandra en 1996.
- Cambio en el tiempo del sensor agregador o líder
- Gracias a los algoritmos garantizamos que todos los sensores de una región están de acuerdo en un agregador común.
- Cada algoritmo está basado en una hipótesis de conectividad diferente

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

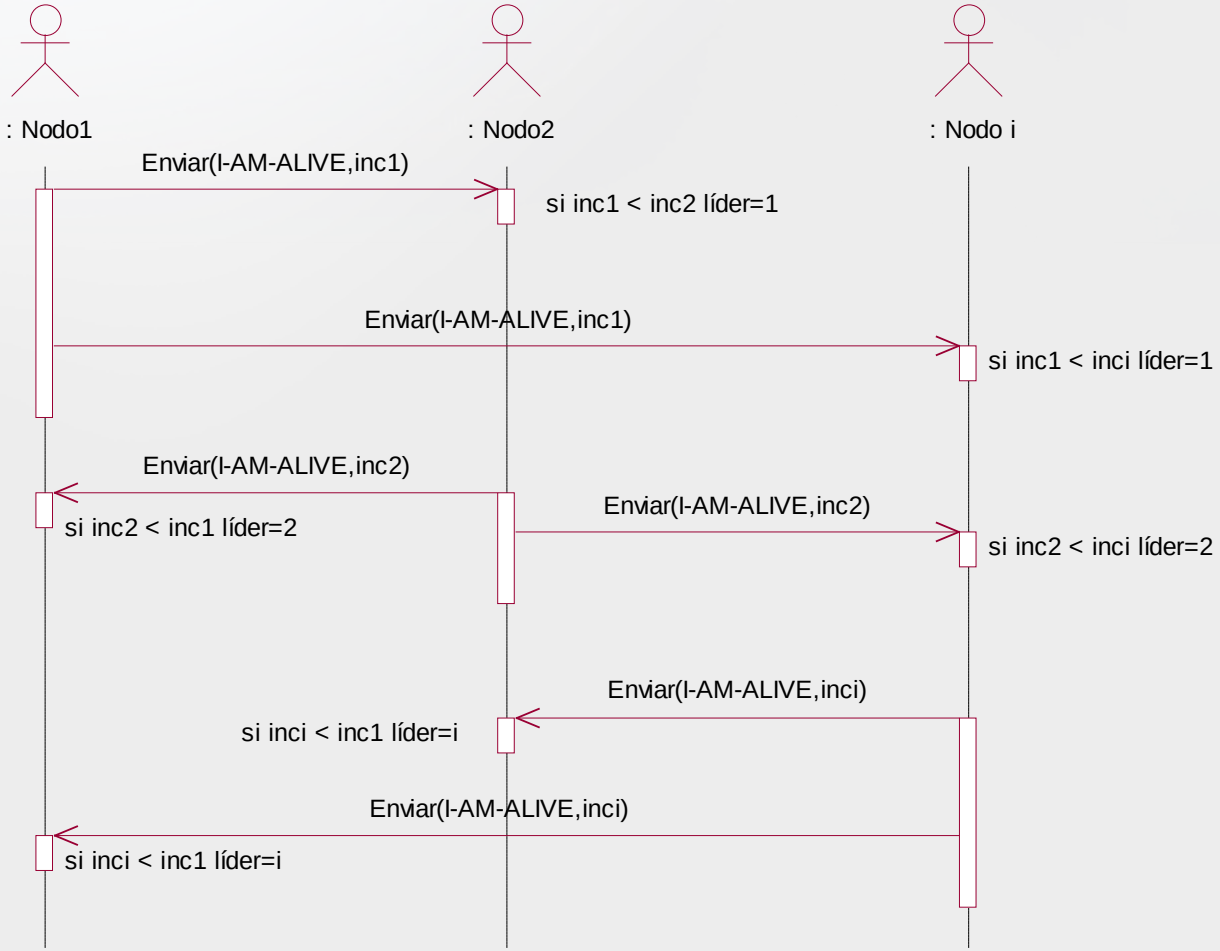
Primer algoritmo de agregación

- Primer algoritmo
  - Hipótesis de conectividad: todo par de sensores en una región puede comunicarse directamente.
  - Los sensores se despiertan periódicamente para proveer sus datos recogidos y enviárselos al agregador común o líder.
  - Hibernan el resto del tiempo.
  - Criterio de fiabilidad: número de encarnaciones

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

Primer algoritmo de agregación

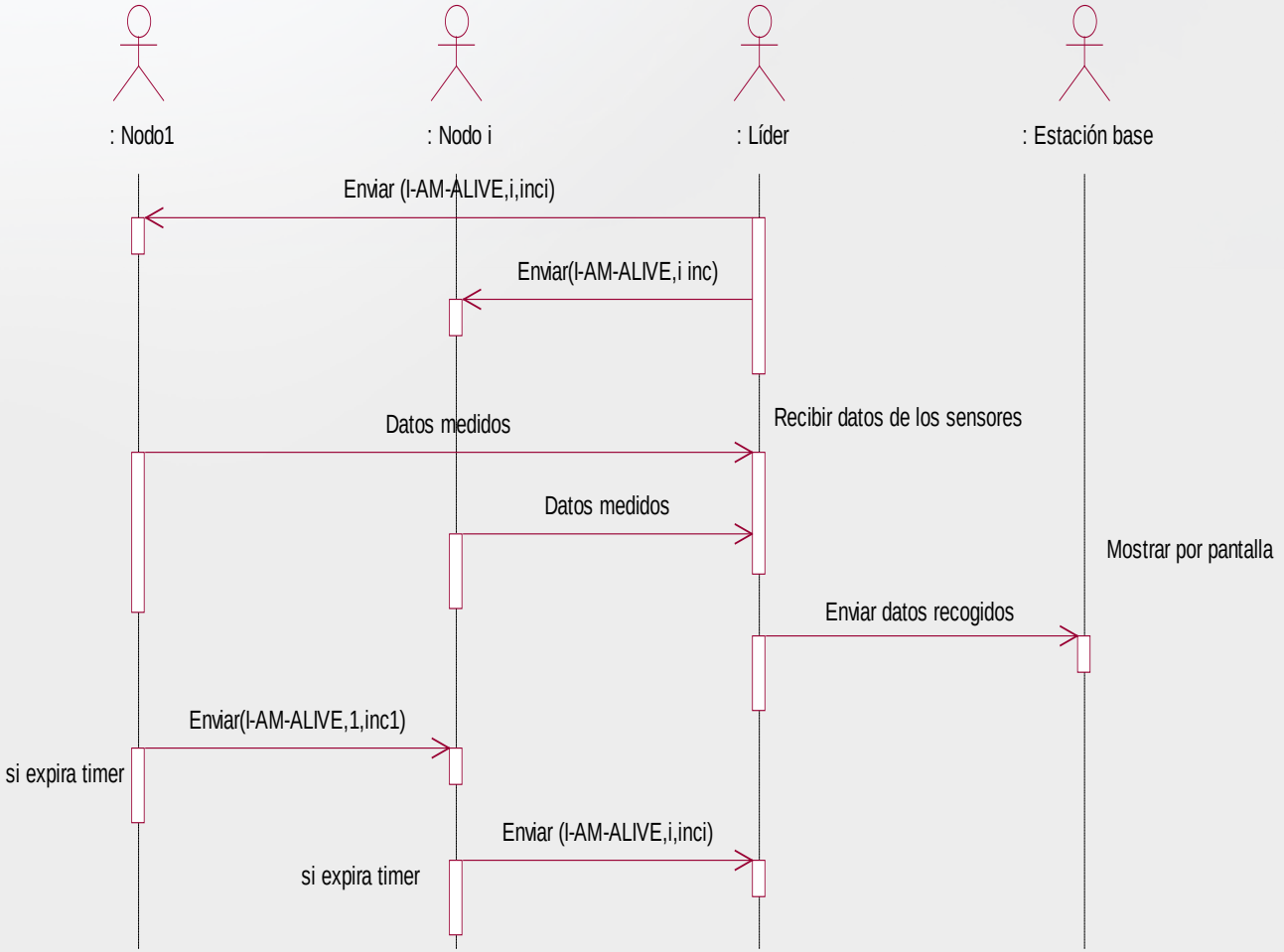
- Diagrama de secuencia: elección de líder



upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

Primer algoritmo de agregación

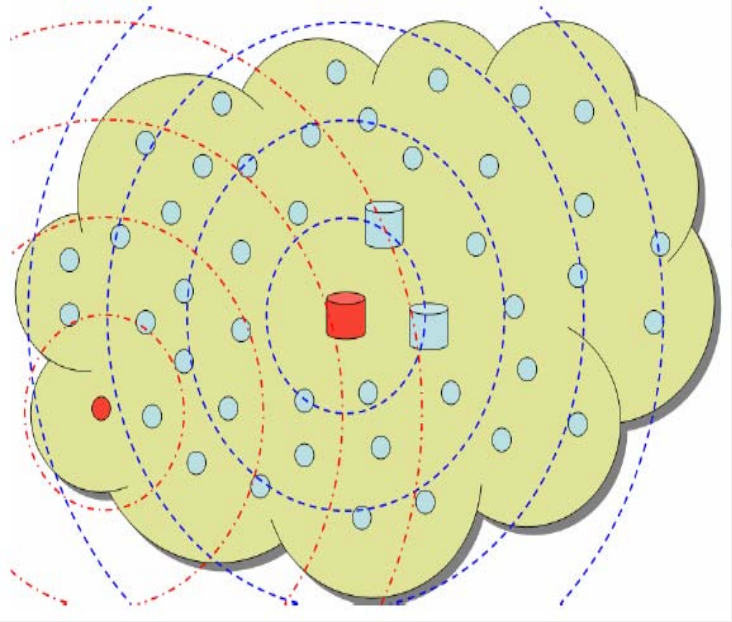
- Diagrama de secuencia: agregación de datos y envío a la estación base



upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

Segundo algoritmo de agregación

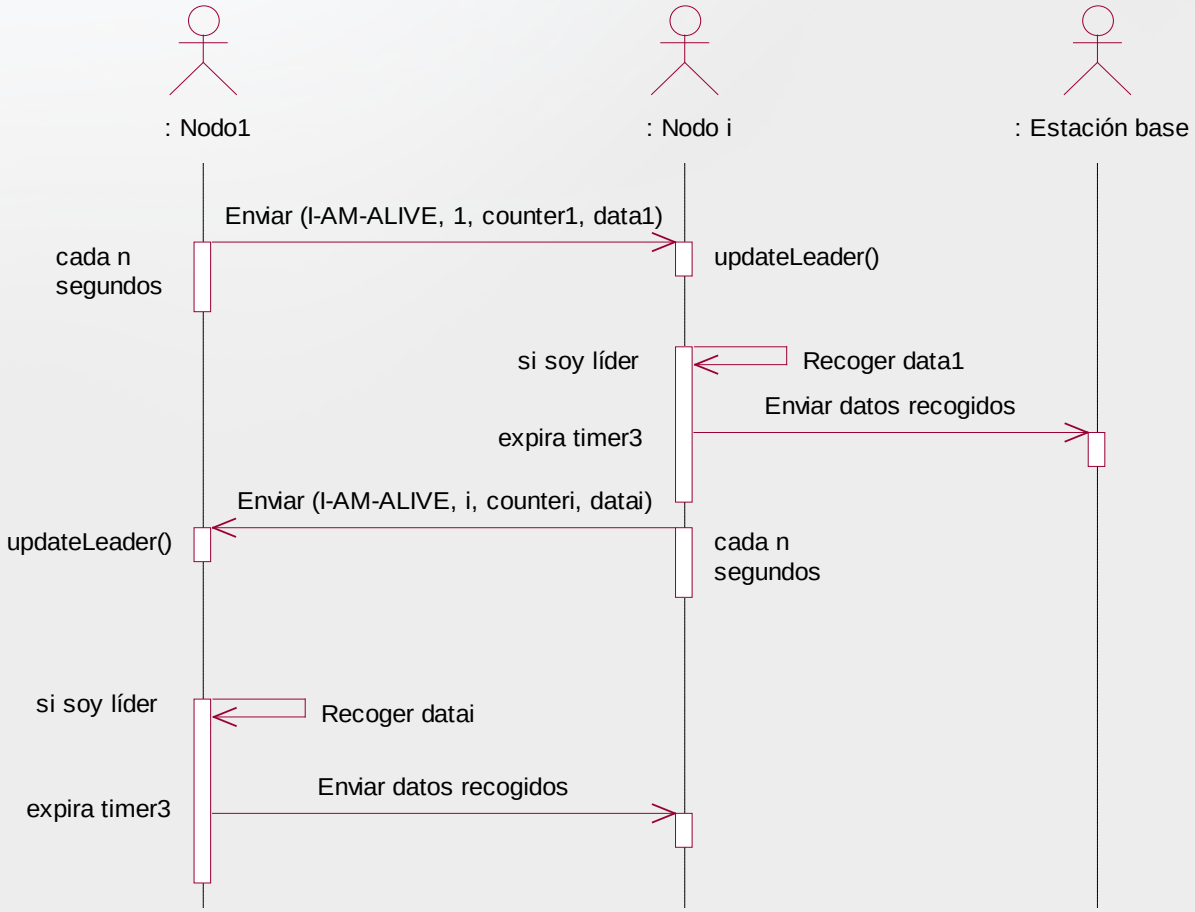
- Segundo algoritmo
  - Hipótesis de conectividad:
    - No todo par de sensores de una región puede comunicarse directamente.
    - Existe un subconjunto de sensores capaces de alcanzar directamente al resto (cilindros).
  - Criterio de fiabilidad: número de veces que el resto de sensores han sido sospechosos



upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

Segundo algoritmo de agregación

- Diagrama de secuencia



upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

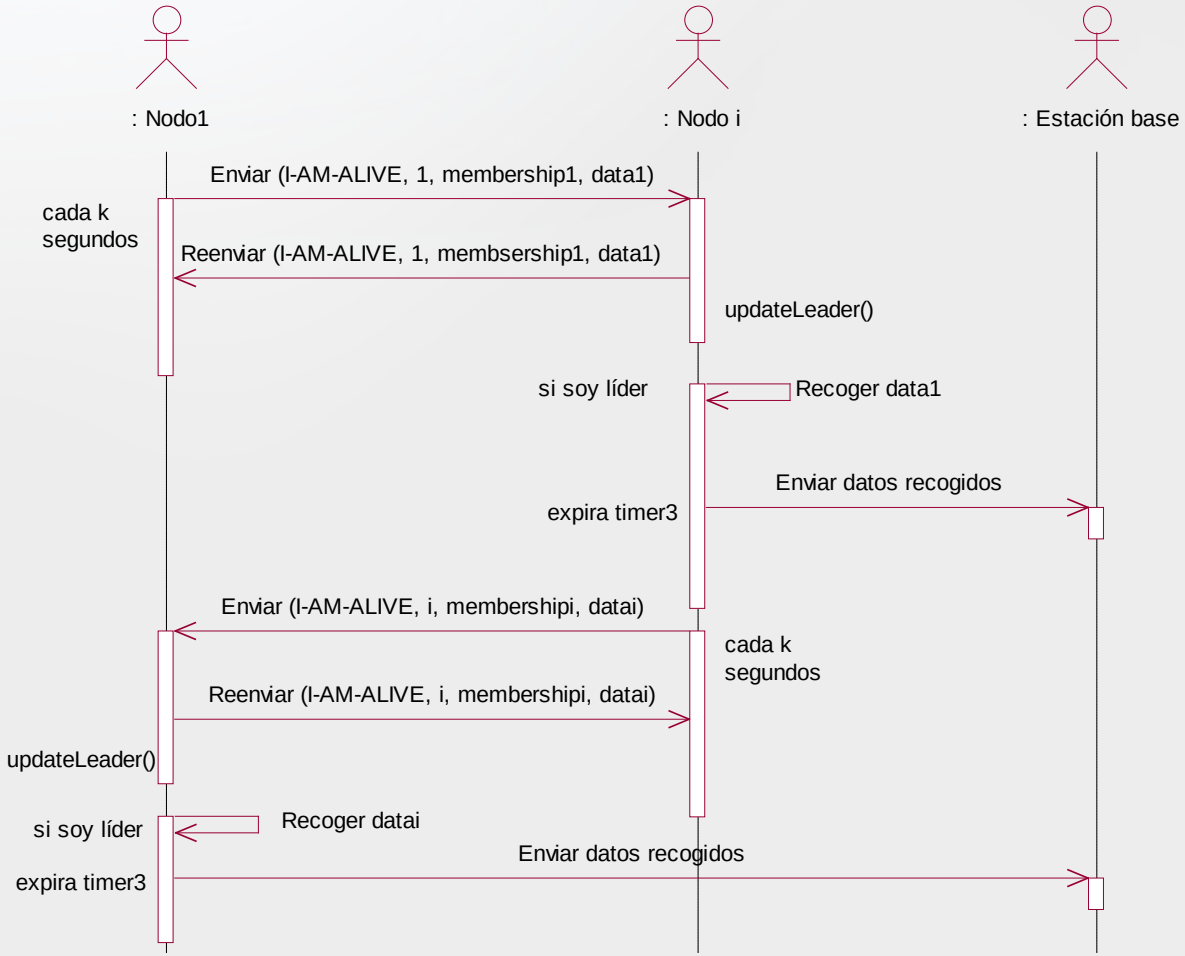
Tercer algoritmo de agregación

- Tercer algoritmo
  - Hipótesis de conectividad
    - No requiere que ningún sensor se comunique directamente con el resto.
    - Requisito mínimo: la existencia de un camino multisalto desde algún sensor correcto hasta el resto de los sensores.
  - Mismo criterio de fiabilidad que el algoritmo 2

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

Tercer algoritmo de agregación

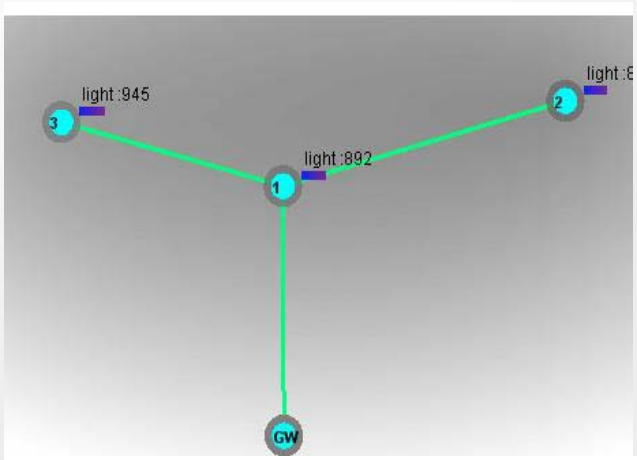
- Diagrama de secuencia

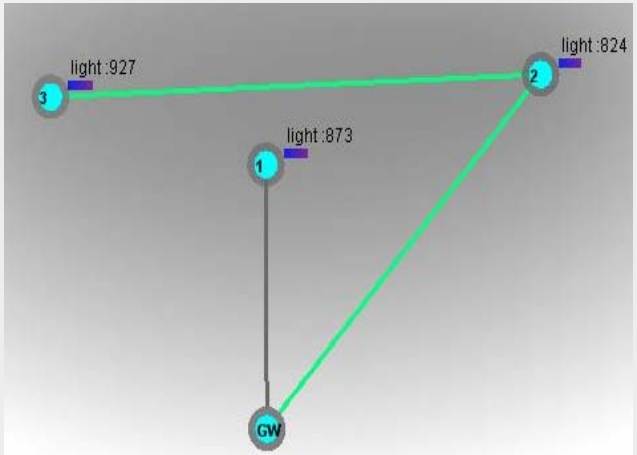


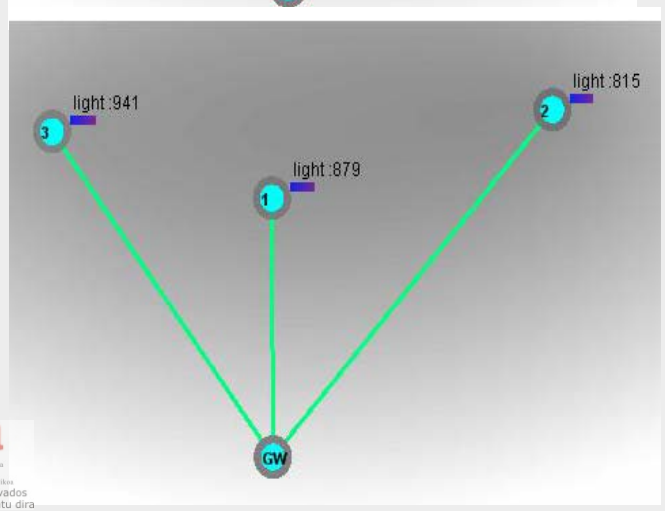
upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erreserbatu dira

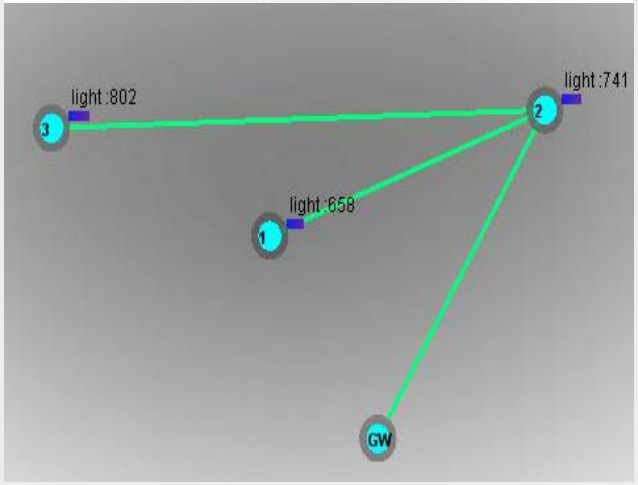
Resultados primer algoritmo

- Funcionamiento básico





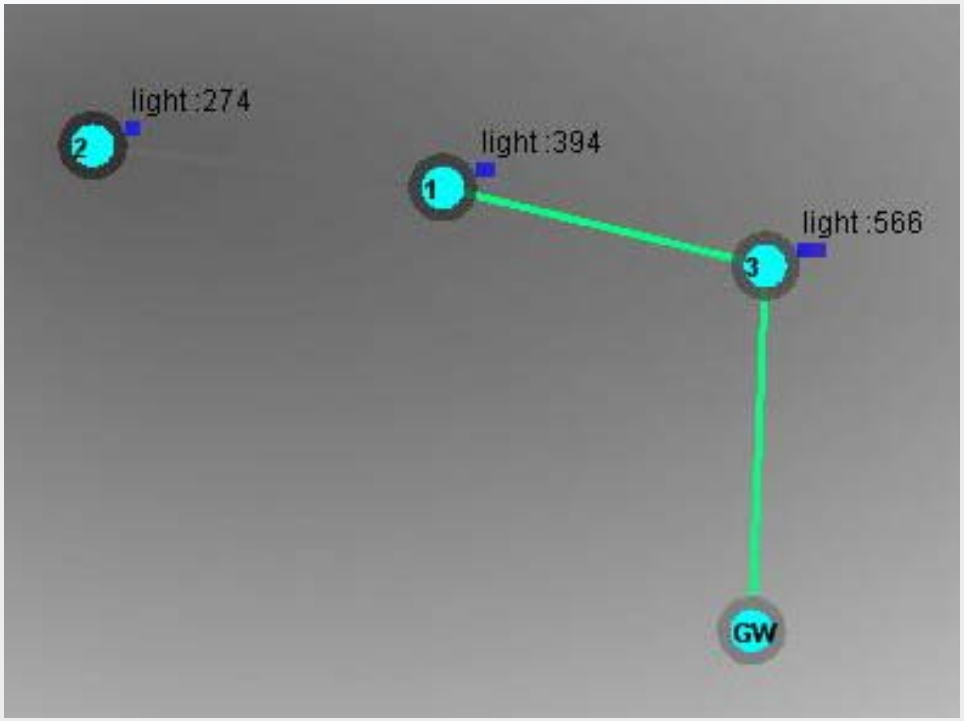




upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

Resultados primer algoritmo

- Incumplimos la hipótesis de conectividad

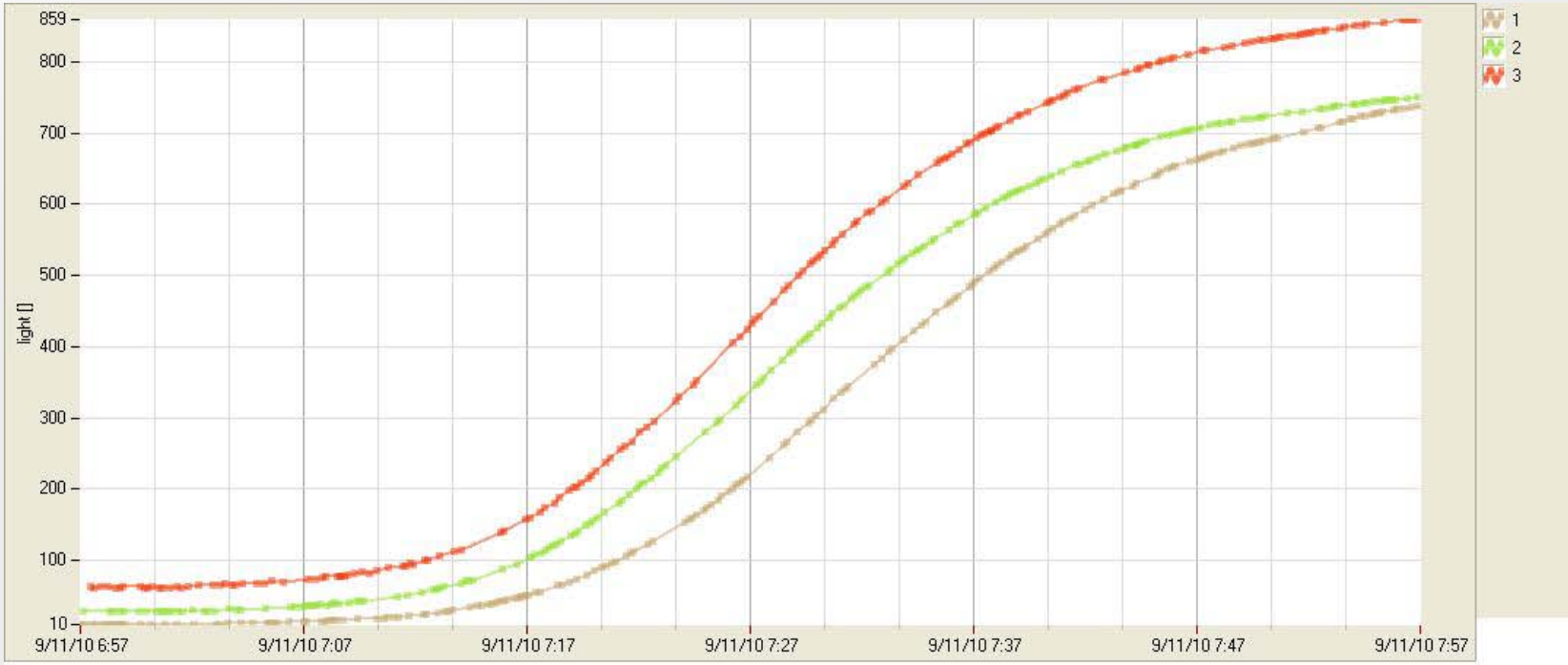


1. Alejamos la mota 2 y la mota 3
2. La mota 2 ya no puede comunicarse directamente con el líder.
3. Existe un camino multisalto entre la mota 2 y el líder (a través de la mota 1) y sin embargo los datos de la mota 2 no son recogidos.
4. El sistema falla. La red no funciona correctamente.

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

Resultados primer algoritmo

- Datos de luminosidad recogidos durante una simulación al azar



upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

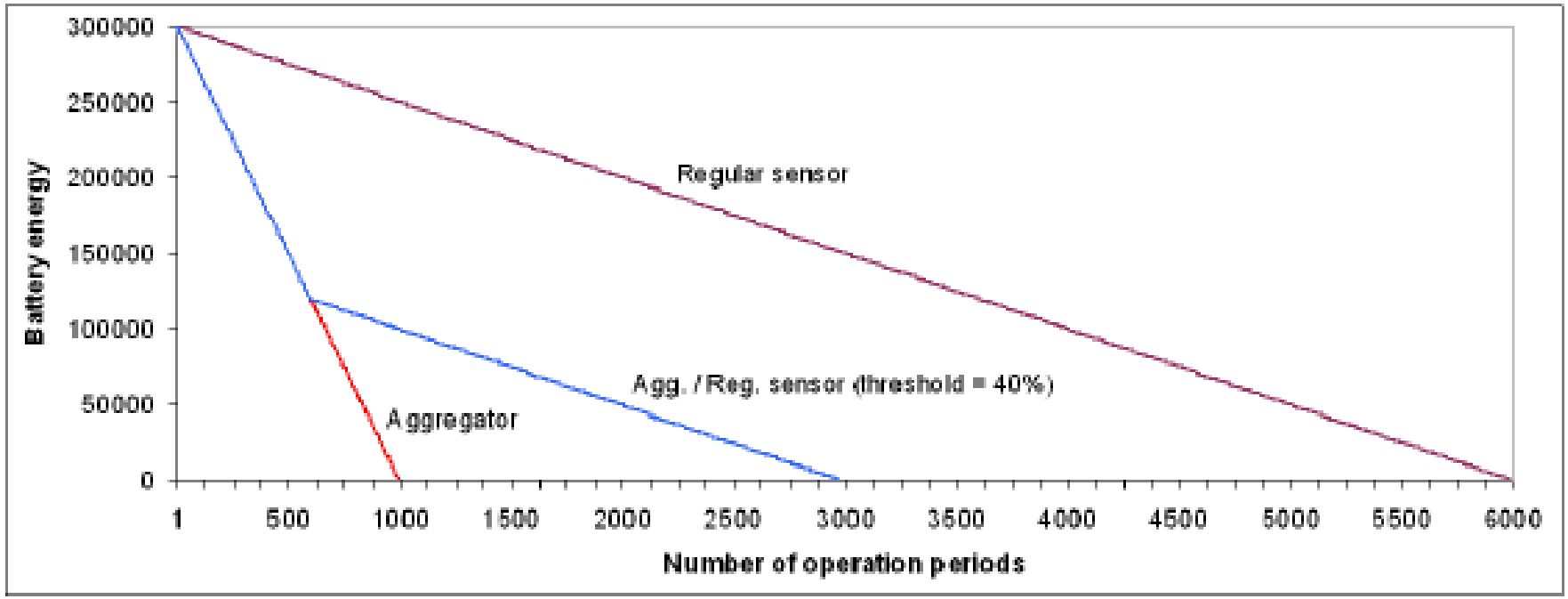
Resultados primer algoritmo

- Niveles de batería antes de la prueba
  - Sensor 1 : 2.4 v. (líder)
  - Sensor 2: 2.9 v.
  - Sensor 3: 2.8 v
- Niveles de batería tras la prueba
  - Sensor 1: 1.6 v.
  - Sensor 2: 2.7 v.
  - Sensor 3: 2.8 v
- El consumo de batería del líder de la red es mayor que el de los sensores normales

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

Resultados primer algoritmo

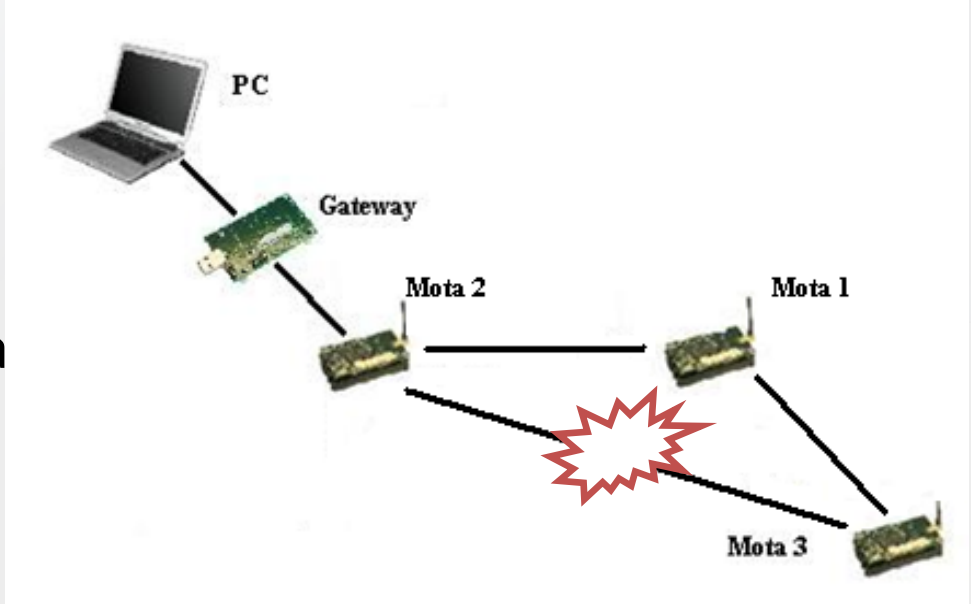
- Consumo de energía en la elección del agregador y en la agregación de datos.



upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

Resultados segundo algoritmo

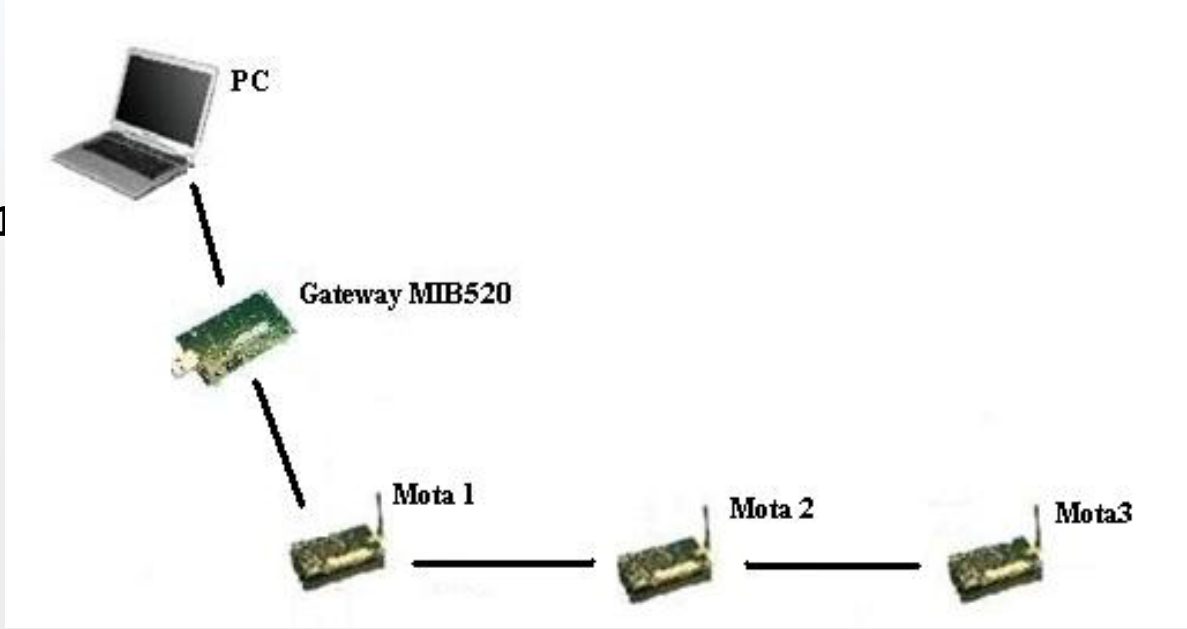
- Escenario inicial
  - Rompemos la comunicación directa entre 2 y 3.
  - El sensor 1 es capaz de alcanzar directamente a los dos sensores.
  - El algoritmo sigue funcionando correctamente.
  - En el mismo escenario en el que el algoritmo 1 no funcionaba, este lo hace de manera correcta.



upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

Resultados tercer algoritmo

- Escenario inicial:
  - Comunicación directa sólo entre 1 y 2 y 2 y 3.
  - Camino multisalto
  - Cualquiera puede ser elegido como líder.
  - El algoritmo funciona correctamente



upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

Conclusiones

- Implementación de tres algoritmos de agregación de datos:
  - Un único sensor es el encargado de transmitir toda la información hacia la estación base.
  - Los sensores eligen a sus líderes siguiendo un criterio de fiabilidad.
- El consumo de batería de los líderes es mayor que el de los sensores normales.
- Este consumo asociado a actuar como agregador se distribuye entre todos los sensores.
- Aumenta la longevidad de la red y mejora la calidad de servicio.

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

Conclusiones

- Primer algoritmo: muy vulnerable frente a posibles fallos ya que la hipótesis inicial es muy difícil de garantizar. Falla en el momento en algún par de sensores ya no son capaces de comunicarse directamente.

- Segundo algoritmo: mejora notablemente al anterior pero exige la existencia de un subconjunto de sensores en la región que sean capaces de alcanzar directamente al resto de los sensores de la región.

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

Conclusiones

- Tercer algoritmo: sus requisitos iniciales son mucho más fáciles de garantizar que la de los dos anteriores. Basta con un camino multisalto desde algunos sensores correctos hasta el resto de los sensores.

- Posibles mejoras:
  - Introducir mecanismos para evitar la colisión de mensajes y mejorar la cobertura de red.
  - Aumentar la flexibilidad de los algoritmos ante el aumento de la red.

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira

FIN

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa
Todos los derechos reservados
Eskubide guztiak erresalbatu dira