Proyecto Fin de Carrera

Protocolo para la diseminación de información en una red vehicular mediante una estrategia unicast

Luis Pastor González

Director: Javier Vales Alonso

Universidad Politécnica de Cartagena

Escuela Técnica Superior de Ingeniería de Telecomunicación

Septiembre de 2009

Autor	Luis Pastor González
E-mail del autor	luispastor14@hotmail.com
Director(es)	Dr. Javier Vales Alonso
E-mail del director	javier.vales@upct.es
Título del PFC	Protocolo para la diseminación de información en una red vehicular mediante una estrategia unicast
Descriptores	Wireless Sensor Network, VANET, MICAz, TinyOS, NesC

Resumen

El proyecto se basa en la creación de un protocolo para el intercambio y difusión de bloques de información entre nodos de una red vehicular. Cada vehículo de la red incorporará un dispositivo llamado MICAz, con el sistema operativo TinyOS, que contendrá los bloques de información que se deseen difundir, así como un programa, escrito en el lenguaje NesC, que ejecutará un algoritmo encargado de controlar la transmisión/recepción de los distintos tipos de paquetes (control y datos), así como de su almacenamiento en la memoria de dicho dispositivo. La principal característica de la solución propuesta es que es unicast, es decir, sólo habrá transmisiones de información entre parejas de nodos: un emisor, un receptor.

Los objetivos principales del proyecto son el desarrollo y la implementación del protocolo mencionado, que, a su vez, tiene como objetivo llevar a cabo la mayor difusión posible de los bloques de información que contiene cada vehículo. Es decir, que en un escenario en el que haya varios vehículos en movimiento, se consiga la mayor dispersión de información antes de que salgan de las zonas de cobertura que tienen sus antenas (el caso ideal sería aquel en el que todos los vehículos acaben teniendo la misma información).

Titulación	Ingeniero Técnico de Telecomunicación, especialidad Telemática
Departamento	Tecnologías de la Información y las Comunicaciones
Fecha de presentación	29 de septiembre de 2009

ÍNDICE

1. INTRODUCCIÓN

El escenario inicial de este proyecto es una **red inalámbrica** de nodos, que pueden estar o no en movimiento, y un **bloque de información**, que estará en uno o más nodos. El objetivo es el desarrollo de un protocolo que, ejecutado en dichos nodos, les permita poner esa información en común, es decir, que haya una **distribución** efectiva y eficiente de los datos.

El tipo de redes que constituyen este escenario se conoce como redes inalámbricas de sensores (*Wireless Sensor Networks, WSNs*) [1] porque es habitual que los nodos [2], también conocidos como "motas" (*motes*, en inglés), que las integran estén equipados con sensores dirigidos a colaborar en una tarea común. Dichos nodos tienen ciertas capacidades de comunicación inalámbrica que les permiten formar redes **ad-hoc** [3], es decir, punto a punto, sin necesidad de que haya una infraestructura física preestablecida ni administración central.

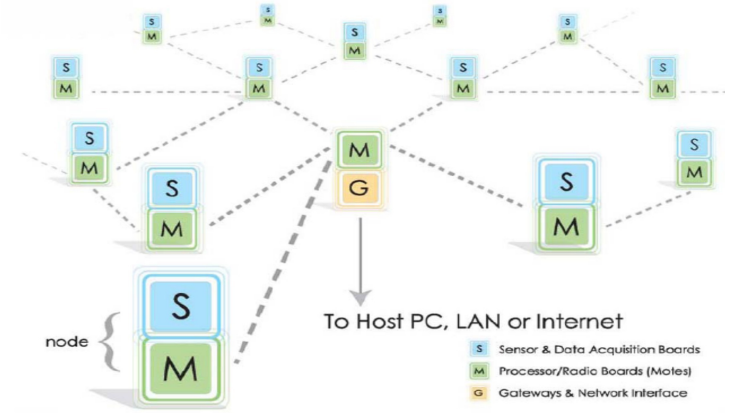


Figura 1: Topología habitual en una red de sensores

Figura 2: Placa de sensores MTS310

En el año 2003 el Instituto Tecnológico de Massachusetts, MIT, identificó las WSNs como una de las 10 tecnologías emergentes que cambiarían el mundo. [4]

Entonces las redes inalámbricas de sensores se limitaban a un conjunto de aplicaciones muy básicas pero de gran potencial ya que por primera vez era posible interactuar con el entorno. Así se iniciaron diversos desarrollos en campos como el medio ambiente (cambio climático), seguridad (vigilancia) y tracking (sistemas de rastreo y seguimiento), a la vez que evidenciaban un conjunto de desafíos muy apetecibles para la comunidad investigadora . [5]

El escenario habitual de estas aplicaciones es el siguiente: se realiza una serie de mediciones del entorno que se monitoriza (con sensores analógicos). Esta información analógica se transforma en digital en el propio nodo y se transmite fuera de la red de sensores a través de un elemento *gateway* (figura 4), donde pasa a una estación base o *stargate* (figura 3) que la almacena temporalmente antes de enviarla a un servidor de mayor capacidad para realizar un seguimiento o análisis de los datos. [6]

Figura 3: Stargate

Figura 4: MIB600 Ethernet Gateway

Algunas aplicaciones en las que se están usando las WSNs en la actualidad son:

- **Monitorización del entorno**: Por ejemplo, una aplicación donde un científico quiere recoger lecturas de un entorno inaccesible y hostil en un periodo de tiempo para detectar cambios, tendencias, etc. En este caso suele haber un gran número de nodos sincronizados, midiendo y transmitiendo periódicamente. La topología física es relativamente estable y no hay requerimientos de latencia estrictos ya que no es habitual procesar los datos en tiempo real. La monitorización del entorno es muy frecuente en aplicaciones relacionadas con la agricultura.

- **Monitorización de seguridad**: Suelen ser aplicaciones para la detección de anomalías o ataques en entornos monitorizados continuamente por sensores. En este caso los nodos no están transmitiendo datos continuamente. Se encontrarán en un estado que implique un consumo de energía mínimo hasta que detecten alguna anomalía. En ese momento será cuando transmitan la información correspondiente (*report by exception*). La principal característica de estos entornos es un aprovechamiento importante de la energía. Aquí sí existen requisitos de tiempo real y la latencia de las comunicaciones juega un papel importante. La monitorización de seguridad suele estar enfocada a la detección de incendios, al control de edificios inteligentes, aplicaciones militares, etc.

- **Tracking**: Aplicaciones para controlar objetos que están “etiquetados” con nodos sensores en una región determinada. A diferencia del resto de aplicaciones, la topología de la red es muy dinámica debido al movimiento de los nodos. Por ello, la WSN deber ser capaz de descubrir nuevos nodos y de formar nuevas topologías.

- **Redes híbridas**: En general, los escenarios de aplicación contienen aspectos de las tres categorías anteriores.

Teniendo en cuenta la limitación de recursos que tienen los nodos que integran estas redes, no es habitual encontrar aplicaciones destinadas exclusivamente a la diseminación de bloques de información relativamente grandes, como es el caso de este proyecto.

Sin embargo esta distribución de información se da en multitud de las redes de sensores, pero no como fin, sino como medio para lograr objetivos como los de las aplicaciones anteriormente mencionadas.

Esto es así por la existencia de los *gateways* antes mencionados. En una **arquitectura centralizada**, en la que todos los nodos transmiten su información al nodo *gateway*, existe un "cuello de botella" en ese nodo (figura 5):

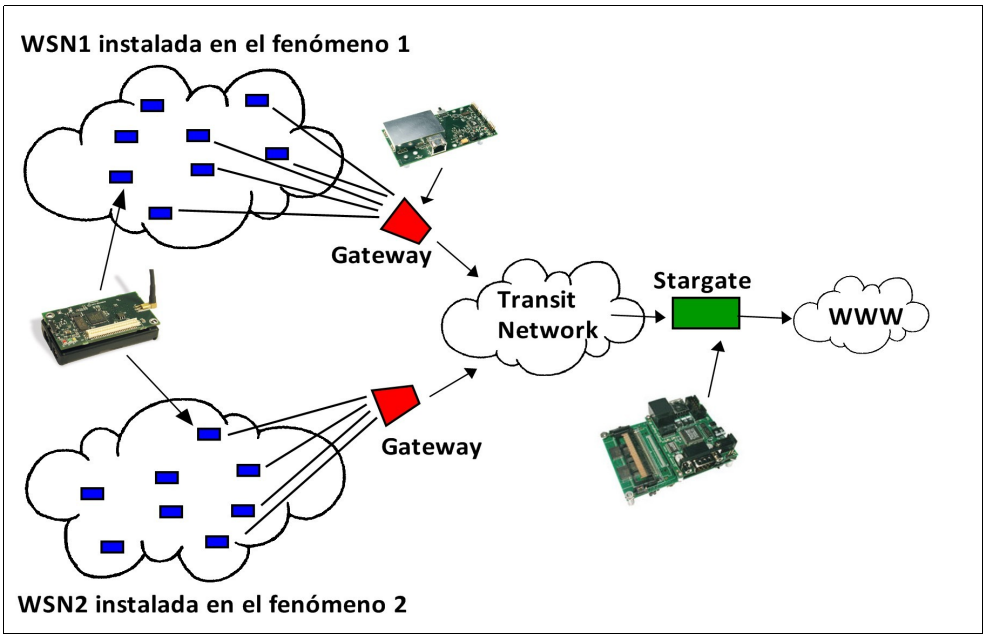


Figura 5: Arquitectura centralizada

Por otro lado, dada la naturaleza de las WSN se tiende a una **arquitectura distribuida**, es decir, los nodos sensores se comunican sólo con otros nodos "vecinos". Existen conjuntos de nodos, denominados *clusters*, en los que los nodos cooperan y ejecutan algoritmos distribuidos para obtener una única respuesta global que un nodo, llamado *cluster head*, se encargará de comunicar al *gateway*. De esta forma se reduce el tráfico que reciben éstos (figura 6):

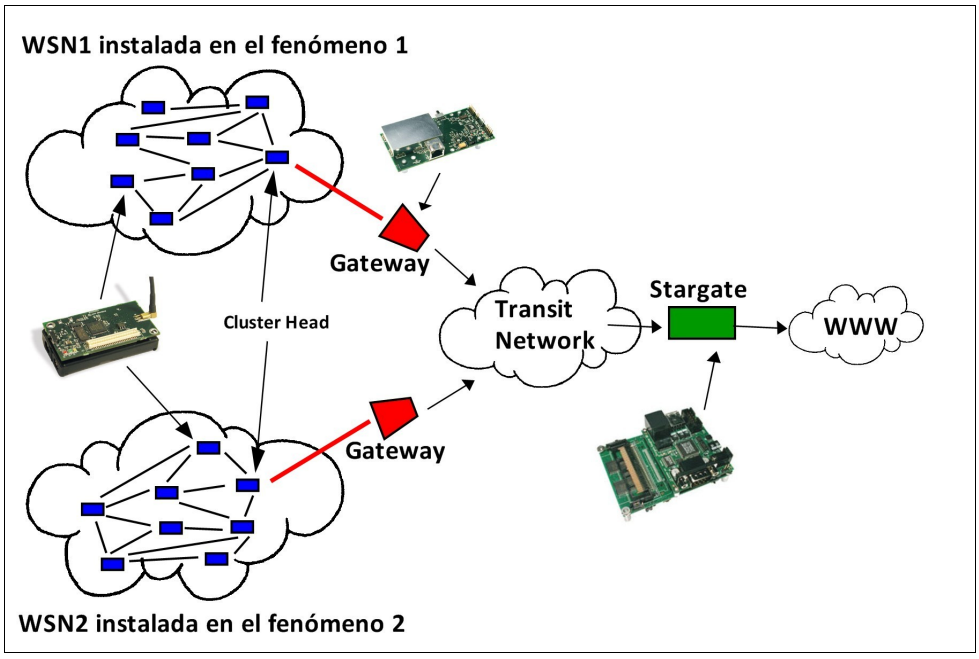


Figura 6: Arquitectura distribuida

En la arquitectura distribuida surge la necesidad de que los nodos se comuniquen entre sí, y de que pongan en común los datos que van obteniendo, ya sea de sus propios sensores o de otros nodos. Sabiendo que la topología de estas redes es dinámica, hacen falta implementaciones que permitan el descubrimiento de nuevos nodos que se vayan incorporando a la red, o al campo de visión de un nodo concreto.

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 siempre es cambiante y éstos tienen que adaptarse para poder comunicar nuevos datos adquiridos. En el intercambio de información que se produce en este proyecto, los nodos conocen nuevas fuentes de datos mediante el intercambio de mensajes que se describirán con detalle en apartados posteriores.

- **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 e interferencias que pueden producir errores en las transmisiones. Más aún, como es habitual, en la banda de frecuencias de 2.4 Ghz (ISM). En este aspecto se usará un sistema muy simple de asentimientos (ACK's) a nivel MAC para solucionar problemas relacionados con la pérdida de paquetes.

- **No se utiliza infraestructura de red**: Una red sensora no tiene necesidad alguna de infraestructura para poder operar, ya que sus nodos pueden actuar de emisores, receptores o enrutadores de la información en cualquier momento. Una de las funciones más importantes que tienen que implementar estas redes es precisamente esa capacidad de auto-configuración y adaptación a los cambios de topología.

- **Tolerancia a errores**: Un dispositivo sensor dentro de una red sensora tiene que ser capaz de seguir funcionando a pesar de tener errores en el sistema propio.

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

- **Consumo energético**: Es uno de los factores más sensibles debido a que los nodos 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 poder conseguir un consumo ajustado, se hace indispensable que el hardware sea lo más sencillo posible, así como su transceptor radio, esto nos deja una capacidad de proceso limitada.

- **Costes de producción**: Dado que la cantidad de nodos de una red de sensores tiene que ser

elevada para poder obtener datos con fiabilidad, los nodos sensores, una vez definida su aplicación, son económicos de hacer si son fabricados en grandes cantidades.

El escenario de estudio en que se basa este proyecto es un caso particular de las redes de sensores, con las mismas características que éstas: **las redes vehiculares.**

1.2. REDES VEHICULARES

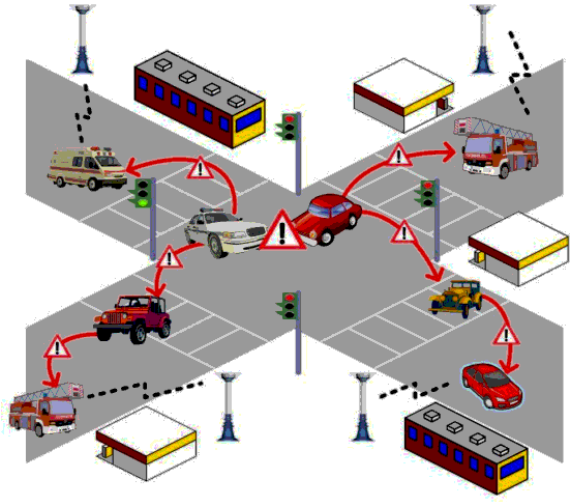


Figura 7: Ejemplo de VANET

Comúnmente conocidas como ***VANET (Vehicular Ad-hoc Network)*** [7], las redes vehiculares son redes ad-hoc móviles capaces de comunicar información entre diversos vehículos colindantes y múltiples puntos emisores/receptores de información distribuidos cerca de las vías de circulación de los vehículos.

Las características de las WSNs más presentes en este tipo de redes son la topología dinámica y la inexistencia de una infraestructura de red.

El objetivo principal de una VANET es proporcionar seguridad y confort a los conductores, dándoles un mejor conocimiento de las condiciones de las carreteras para reducir así el número de accidentes y que la conducción sea más cómoda y fluida. Estas redes implementan servicios como la compartición de archivos multimedia entre vehículos.

Las VANETs son un subconjuntos de las redes ***MANET (Mobile Ad-hoc Network)*** [8]. Éstas se definen como redes autoconfigurables de dispositivos móviles conectados por enlaces inalámbricos. En estas redes, cada dispositivo es libre de moverse independientemente en cualquier dirección.

Conociendo la definición de MANET, se puede definir una red vehicular, VANET, como una red MANET cuyos nodos son vehículos.

En las VANETs, los vehículos que las integran deben incorporar un dispositivo electrónico que les proporcione una conectividad ad-hoc dentro de la red. En este proyecto en particular, los dispositivos encargados de esa tarea serán los dispositivos MICAz, de Crossbow Technologies [9], el mayor fabricante de nodos sensores de la actualidad.

1.3. Entorno de trabajo

1.3.1. MICAz

El MICAz es un dispositivo electrónico que permite la comunicación inalámbrica en la banda de los 2,4 Ghz basada en el estándar IEEE 802.15.4/ZigBee. Este estándar está dirigido a las aplicaciones que requieren comunicaciones *wireless* seguras con baja tasa de envío de datos y maximización de la vida útil de las baterías de los módulos en que se ejecutan .

Figura 8: MICAz

Estas son algunas de las características del MICAz [10]:

- Memoria flash de programa de 128 Kbytes
- Memoria flash de datos de 512 Kbytes
- Memoria RAM de 4 Kbytes
- Tasa de envío de 250 Kbps
- Transceptor radio de 2,4 Ghz (entre 2400 y 2483,5 Mhz)
- 2 baterías de tipo AA para la alimentación

1.3.2. TinyOS y NesC

Con dispositivos como el MICAz como hardware de referencia, la Universidad de Berkeley tiene la iniciativa más importante y la que más repercusión ha tenido: el desarrollo en 2003 de un sistema operativo capaz de gestionar los recursos de las motas y permitir la ejecución de aplicaciones. Este sistema operativo, denominado **TinyOS** es considerado en la actualidad el estándar de facto. [11]

Figura 9: Logo de TinyOS

TinyOS [12] es un sistema operativo basado en componentes diseñado específicamente para redes inalámbricas de sensores. La versión del sistema operativo usada en este proyecto es la 2.0. Está escrito en el lenguaje de programación **NesC (Network Embedded Systems C)** y está pensado para funcionar bajo las importantes restricciones de memoria que se dan en dichas redes.

NesC es una extensión del lenguaje C orientado a componentes y conducido por eventos, optimizado para las limitaciones de memoria y especialmente diseñado para programar aplicaciones sobre redes de sensores. [13]

Un programa en NesC está **estructurado en componentes** de forma que una aplicación

consiste en uno o más componentes ensamblados (*wired*) formando un ejecutable. Éstos a su vez pueden proporcionar o usar **interfaces**.

Las interfaces que proporciona el componente representan la funcionalidad que éste ofrece a su usuario, mientras que las interfaces que usa representan la funcionalidad que el componente necesita para llevar a cabo su trabajo. Las interfaces son bidireccionales ya que, por un lado especifican un conjunto de comandos, que son funciones que debe implementar el componente que proporciona la interfaz, y por otro una serie de eventos, que son funciones que debe implementar el componente que usa la interfaz.

Existen dos tipos de componentes en NesC: módulos y configuraciones. Los **módulos** proporcionan la implementación de una o más interfaces y son el código ejecutable de la aplicación, mientras que las **configuraciones** se usan para ensamblar otros componentes, conectando las interfaces que proporcionan unos componentes con las que usan otros. Por eso es común usar el término *wiring* (cableado) para referirse al proceso de configuración.

Como se ha mencionado antes, NesC es un lenguaje **conducido por eventos**. Un componente que proporcione una interfaz con eventos señalizará a otro componente que use esa interfaz cada vez que ocurra uno de los eventos, de forma que éste tendrá que "controlarlo". Un ejemplo de evento es la recepción de un paquete. Habrá un componente que esté usando una interfaz que controla la recepción. El componente que esté proporcionando esa interfaz notificará al primero que ha llegado un paquete. En ese momento se ejecutará la sección del código que controle dicho evento en el componente que usa la interfaz. Así, el receptor decidirá qué hacer con el mensaje recibido.

La existencia de estos eventos hace que el código de un módulo en NesC no sea puramente secuencial, si no que se ejecutarán pedazos de código en función del evento que se señalice, dando lugar a una cierta concurrencia en su ejecución.

TinyOS proporciona interfaces, módulos y configuraciones específicas que permiten a los programadores construir programas como una serie de módulos que hacen tareas específicas. Estas interfaces permiten interactuar con el hardware y el software del MICAz (por ejemplo, controlar el encendido/apagado de los leds, el envío/recepción de paquetes o el uso de temporizadores).

1.3.3. XubunTOS

Figura 10: Logo de XubunTOS

La programación de los nodos se llevará a cabo en el ordenador usando la versión 2.1.4 del sistema operativo **XubunTOS**, que simplifica la instalación de TinyOS mediante un *live CD* de Linux. El disco contiene el entorno de trabajo de TinyOS y ofrece la posibilidad de ejecutar una instalación completa en el PC.

Xubuntos se basa en la distribución Ubuntu con el entorno gráfico XCFE e incorpora todos los paquetes de TinyOS 2.x (más los repositorios de TinyOS 1.x). [14]

En este entorno de programación se llevará a cabo el desarrollo de un **protocolo** que permita la distribución de los bloques de información de una forma organizada y eficiente. Se basará en un algoritmo que se encargará, entre otras cosas, de gestionar la memoria flash del dispositivo. La memoria flash de datos tiene un tamaño de 512 Kbytes, por lo que el tamaño del bloque de datos que se distribuirá en las pruebas será de 400 Kbytes, para no llevar al límite la capacidad de la memoria.

Sin entrar en profundidad en la descripción del protocolo, cabe destacar que será ***unicast***, es decir, las transmisiones de información se llevarán a cabo entre parejas de nodos, habiendo un único emisor y un único receptor. En un momento dado, un nodo transmisor sólo podrá enviar datos a un único nodo receptor, sin que aquel pueda recibir información mientras realiza el envío. De la misma forma, el nodo receptor no podrá recibir información de otros, ni transmitirla simultáneamente.

La **coordinación entre nodos** será uno de los puntos más importantes en el planteamiento del algoritmo, pues es necesario dotar a los nodos de cierta "inteligencia" que les permita desenvolverse de forma autónoma dentro de la red. Esa coordinación será posible mediante un mecanismo de intercambio de mensajes de distintos tipos (anuncios con la información disponible, solicitudes como respuesta a los anuncios, etc.) que harán posible que los dispositivos se comuniquen con sus vecinos y sepan cuál es su estado en todo momento.

El funcionamiento del protocolo se describe con más detalle en el tercer apartado. En él se incluye una descripción en SDL del algoritmo empleado, así como una lista de los tipos de mensajes que existen, su longitud y utilidad, etc.

En el segundo apartado se comentan algunos trabajos relacionados con éste y, en general, con redes vehiculares y de sensores. Destacando trabajos en el mismo campo de los propios desarrolladores de TinyOS.

En el apartado cuarto se detalla todo lo relacionado con las pruebas de laboratorio para ver y corregir los fallos del protocolo, así como sus limitaciones en distintos escenarios, variando el número de MICAz, su posición, la cantidad de "semillas" (nodos con información), etc. Con cada escenario se medirá el tiempo que tardan todos los nodos en tener la totalidad del bloque de información. Para cada una de las pruebas se llevarán a cabo hasta cinco mediciones para poder obtener resultados más o menos fiables en escenarios donde haya colisiones y pérdidas.

En el quinto y último apartado se extraen conclusiones de los resultados obtenidos en el apartado anterior, en relación con la viabilidad del proyecto en una hipotética implantación real, vulnerabilidades, posibles mejoras, etc.

2. Trabajos relacionados

En este apartado se describen algunos protocolos para redes inalámbricas de sensores que se han desarrollado o que se siguen desarrollando en la actualidad. También se dan algunos ejemplos de aplicaciones de WSNs y algunas pinceladas sobre proyectos de redes vehiculares.

2.1. Protocolos de TinyOS

A continuación se detallan varios protocolos llevados a cabo por los propios desarrolladores de TinyOS. Por un lado, protocolo relacionados con el concepto de diseminación, y por otro, se dan algunas características de protocolos de encaminamiento, calidad de enlace...

2.1.1. Protocolos de diseminación de TinyOS

Los mismos creadores de TinyOS han desarrollado protocolos para la diseminación de bloques de información, que van desde pequeños fragmentos de datos (comandos o parámetros), de menor tamaño que el del campo de datos de un paquete, hasta imágenes binarias para reprogramar los nodos.

Los protocolos de diseminación pueden diferir considerablemente según el tamaño de los datos que se quieran compartir, es decir, una diseminación eficiente de decenas de kilobytes requiere un protocolo distinto que la diseminación de una variable de dos bytes. Esto no quiere decir que no haya similitudes: si se separa un protocolo de diseminación en dos partes, tráfico de control y tráfico de datos, se observa que el tráfico de datos es muy dependiente del tamaño de la información que se distribuya, mientras que la parte de control es la misma en todos los casos, o muy similar.

2.1.1.1. Diseminación de pequeños fragmentos de información

Uno de los usos principales que los desarrolladores dan a los protocolos de diseminación es el de mantener la consistencia cuando existe una variable compartida por varios nodos [15]. En este ámbito existen dos implementaciones: ***Drip*** y ***DIP***.

Ambas se basan en el mismo algoritmo [16]. *Drip* es una implementación básica de dicho algoritmo, mientras que *DIP* usa una aproximación más compleja del protocolo de diseminación. Éste es más rápido, especialmente cuando el servicio de diseminación mantiene muchas variables. Esta complejidad implica que *DIP* use más memoria de programa que *Drip*. [17]

En cualquier implementación cada nodo almacena una copia de esa variable y el protocolo avisa cada vez que su valor cambia. Es posible que, en un momento dado, dos nodos tengan valores distintos de la variable, pero con el paso del tiempo el número de diferencias entre nodos irá disminuyendo hasta que en la red sólo exista un valor de la variable. Para actualizar su valor, un nodo puede “proponerlo” en la red indicando que es nuevo. Si varios nodos deciden actualizar su variable, entonces el protocolo se asegurará de que toda la red converja en una de las

actualizaciones propuestas.

La consistencia no implica que cada nodo vea el posible valor que tomará la variable. Simplemente la red llegará eventualmente a un acuerdo sobre cuál es el valor más reciente. Es decir, si un nodo se desconecta de la red y tienen lugar ocho actualizaciones en su ausencia, cuando éste vuelva sabrá únicamente cuál es el valor actual de la variable.

2.1.1.2. Diseminación de imágenes binarias: Deluge

Los mecanismos de diseminación se usan con frecuencia con el objetivo de reconfigurar o reprogramar una red. El crecimiento de las WSNs así como el del número de nodos que las integran hacen que la propagación del código de programa a través de la red sea una necesidad para las pruebas y la depuración de las aplicaciones.

El protocolo ***Deluge*** es un servicio de reprogramación que difunde de forma fiable metadatos de la imagen binaria completa de una aplicación de TinyOS sobre una red multi-salto. Unido a lo que se conoce como *bootloader* (TinyOS proporciona el componente *TOSBoot*) permite la **programación de los nodos en red**. [18]

La programación de red implica varios **problemas**:

1. Las imágenes de programa ocupan mucho más que los datos que difundían protocolos de diseminación anteriores (la memoria de programa es de 128 Kbytes, mientras que el tamaño de la memoria RAM es de 4 Kbytes). Los mensajes que no suponen un problema en este aspecto (los de control) constituyen únicamente un 18% de la totalidad.

2. El protocolo de diseminación debe tolerar, además, una densidad nodal que puede sufrir variaciones del orden de cientos de motas.

3. Hace falta la máxima fiabilidad, ya que cada byte debe ser recibido correctamente por todos los nodos que tienen que reprogramarse, incluso en presencia de altas tasas de pérdidas como las que se dan en las redes inalámbricas.

4. La propagación debe ser un esfuerzo contínuo para asegurarse de que todos los nodos reciben el código más reciente, ya que los nodos van y vienen debido al dinamismo de la topología de estas redes: desconexiones temporales, fallos, etc.

5. La transmisión de mensajes broadcast sin ningún mecanismo de contención pueden provocar lo que se conoce como ***broadcast storm problem*** [19].Esta “tormenta” de mensajes pueden inundar el medio radio influyendo notablemente en la eficiencia y la fiabilidad del protocolo. Para evitarlo, *Deluge* implementa técnicas de supresión de mensajes para minimizar la congestión de la red.

Deluge es un protocolo que funciona como una máquina de estados en la que cada nodo sigue un conjunto de normas estrictamente locales para conseguir el comportamiento global deseado. En su versión más básica, cada nodo anuncia ocasionalmente a los nodos que están dentro de su campo broadcast cuál es la versión del programa más actualizada que tiene. El nodo

que “escuche” ese anuncio solicitará las partes que necesiten actualizarse al nodo emisor, el cual las enviará broadcast. Cuando un nodo reciba nuevos fragmentos, los anunciará de la misma forma para que continúe la propagación de datos.

Ya que está versión es bastante simple, los desarrolladores consideran bastantes aspectos que pueden aumentar la eficiencia del protocolo:

- El primero tiene que ver con el incremento de la densidad de nodos, donde se suprimen anuncios y solicitudes redundantes para minimizar el número de colisiones. Esta solución tiene como principal inconveniente el aumento de la latencia en el intercambio de mensajes en la red.

- Los protocolos de las WSNs deben protegerse de que haya enlaces asimétricos, o sea, la existencia de un enlace en un único sentido provocará una alta tasa de pérdidas. El protocolo *three-phase handsake* de *Deluge* (literalmente sería un saludo en tres fases) se asegura de que exista un enlace bidireccional antes de comenzar la transmisión de datos.

- Se ajusta dinámicamente la tasa de anuncios para permitir una rápida propagación cuando sea necesario y evitar la congestión cuando haya mucho tráfico.

- Se intenta minimizar el número de nodos que están enviando sus anuncios concurrentemente en un área determinada.

- Se enfatiza el uso de multiplexación espacial para permitir transferencias de datos paralelas.

2.1.2. Otros protocolos de TinyOS

2.1.2.1. Recolección: *Collection Tree Protocol, CTP*

La recolección es el proceso inverso a la diseminación. Se trata de un proceso muy común en las WSNs con arquitecturas distribuidas (ver figura 6, página 7). El mecanismo de recolección de TinyOS proporciona un servicio de entrega de paquetes en una red multisalto al nodo raíz de una topología en árbol. En esta topología puede haber más de un nodo raíz. En ese caso, el algoritmo se encarga de que al menos una de ellas reciba todos los datos (un nodo que envía un paquete no especifica a qué raíz está destinado) sin que existan garantías en cuanto a duplicados o desorden de mensajes. [20]

Recoger información de una red en una estación base suele ser común en las WSNs. En general, se parte de uno o más “árboles” de recolección, cada uno de los cuales tiene como raíz un nodo que actúa como estación base. Cuando un nodo tiene datos para transmitir, los envía “árbol abajo” y continúa la recolección de los datos que recibe de otros nodos. En algunos casos el sistema debe ser capaz de inspeccionar el contenido de los paquetes (mantener una estadística, cálculos agregados, supresión de mensajes redundantes...).

Los protocolos de recolección cuentan con varios **problemas** propios de los protocolos para redes de sensores:

- **Detección de bucles**: detectar cuándo un nodo elige uno de sus descendientes como un nuevo padre.

- **Supresión de duplicados**: detectar las pérdidas de ACKs que pueden provocar que haya réplicas de un mensaje circulando por la red y consumiendo ancho de banda.

- **Estimación del enlace**: evaluación de la calidad del enlace entre vecinos (nodos con visión directa). En este aspecto se han desarrollado protocolos con la única función de medir la calidad del enlace entre nodos (ver *Link Estimation Exchange Protocol*, *LEEP*, página 16).

Un ejemplo de protocolo de recolección es *CTP* (*Collection Tree Protocol*). En él, algunos nodos de la red se anuncian como raíces del árbol, de forma que le resto de nodos pasan a actuar como routers para que la información llegue a las raíces. Es un mecanismo *adress-free*, es decir, los nodos no envían sus paquetes a una dirección de memoria concreta, en su lugar, eligen una raíz simplemente enviando los datos al nodo más cercano (un sólo salto). [21]

El protocolo *CTP* asume que la capa de enlace proporciona cuatro servicios básicos:

1. Proporciona una dirección de broadcast local.
2. Proporciona asentimientos síncronos para paquetes unicast.
3. Proporciona un campo de datos para dar soporte a protocolos de capas superiores.
4. Ofrece campos de origen y destino para transmisiones unicast.

CTP cuenta con varios mecanismos para mejorar la fiabilidad en la entrega de paquetes, si bien no la garantiza totalmente y está diseñado para tasas de tráfico relativamente bajas.

2.1.2.2. *The Link Estimation Exchange Protocol, LEEP*

LEEP (*Link Estimation Exchange Protocol*) es un protocolo que los nodos de una red usan para estimar e intercambiar información sobre la **calidad del enlace** con sus vecinos. [22]

Los protocolos de encaminamiento necesitan a menudo valores que indiquen la calidad del enlace entre dos nodos en ambos sentidos para configurar las rutas. Los nodos pueden estimar la calidad del enlace "entrante" (en una transmisión entre dos nodos, A y B, y tomando como referencia B, es la calidad del enlace que va de A a B) calculando el ratio de paquetes recibidos frente al número de paquetes enviados o usando algunos indicadores de la calidad del enlace que proporciona el componente de la radio (LQI, RSSI, etc.).

Para calcular la calidad del enlace entre dos nodos en ambas direcciones, éstos deben intercambiar la calidad del enlace entrante con el otro nodo.

2.1.2.3. *TYMO*

TYMO es la implementación del protocolo *DYMO* en TinyOS. *DYMO (Dynamic MANET On-Demand*) es un **protocolo de encaminamiento** punto a punto para redes ad-hoc móviles. Éste protocolo no almacena explícitamente la topología de la red. En su lugar, un nodo calcula una ruta hacia el destino deseado únicamente cuando es necesario. El resultado es que se intercambian pequeños fragmentos de información de encaminamiento, lo cual permite ahorrar ancho de banda y energía. [23]

Los desarrolladores de TinyOS han desarrollado un protocolo de transporte llamado *MH* (*MultiHop*) que no hace más que reenviar los paquetes hasta que lleguen al nodo destino. *DYMO* está pensado para ejecutarse en dispositivos con poca memoria, por ese motivo la información de encaminamiento que se almacena es mínima.

Cuando un nodo necesita establecer una ruta, difunde un mensaje del tipo ***Route Request*** (*RREQ*), que es un paquete que pregunta por el camino entre el emisor y el destinatario. Cuando el paquete alcanza su destino (o un nodo que tenga una ruta reciente hasta el destino), éste responde con un mensaje del tipo ***Route Reply*** *(RREP)*.

Cuando un nodo recibe un RREQ o un RREP, almacena en caché información sobre el emisor del paquete a nivel de enlace (su vecino) y sobre el nodo origen del mismo. De esta forma conoce una ruta hasta el nodo origen del mensaje que puede usar más tarde (mientras se mantenga el enlace) sin necesidad de enviar RREQ. Los nodos tienen la posibilidad de integrar la ruta seguida del paquete que envían, en el paquete propiamente dicho. Así, cuando un nodo disemina un RREQ o un RREP, se puede obtener mucha más información del paquete a parte de la ruta entre dos nodos.

Cuando las rutas dejan de usarse durante un tiempo relativamente largo, son eliminadas. Si un nodo recibe una solicitud para encaminar información a través de una ruta eliminada, genera un mensaje de tipo ***Route Error*** *(RRER)* para avisar al nodo origen (y a otros nodos) de que esa ruta ya no está disponible. Como mecanismo de mantenimiento, *DYMO* usa números de secuencia y número de saltos en las rutas para determinar su utilidad y calidad.

TYMO todavía no es un protocolo estable y aún está siendo sometido a pruebas para garantizar su funcionalidad en distintos tipos de escenarios.

2.2. Proyectos de VANETs

Las redes vehiculares constituyen un campo muy activo de la investigación, y así lo demuestra una amplia variedad de proyectos desarrollados, o en fase de desarrollo, en esta materia, todos ellos dirigidos a la seguridad en carretera. Algunos ejemplos de este tipo de proyectos son: *SeVeCom (Secure Vehicular Communication)*, *NOW (Network On Wheels), WILL WARN, Watch-Over, Drive-In, AutoNomos,* y muchos otros.

2.2.1. *WATCH-OVER*

Watch-Over es un proyecto iniciado en enero de 2006 y finalizado casi tres años más tarde, en diciembre de 2008. La meta de este proyecto es el diseño y desarrollo de un sistema cooperativo de prevención de accidentes en áreas urbanas.

Figura 11: Comunicación entre usuarios de la vía pública

Lo que se pretende es detectar a usuarios de la vía pública que sean propensos a sufrir un accidente en complejos escenarios de tráfico, en los que los peatones y motoristas caminan o se mueven junto a coches y otros vehículos. [24]

2.2.2. WILLWARN

Se trata de un proyecto de desarrollo e integración de una aplicación para la seguridad en carretera que avisa con antelación al conductor del vehículo sobre alguna situación que pueda poner en peligro su seguridad: detección de obstáculos en la calzada (avisando si el propio vehículo supone un obstáculo), avisos sobre vehículos de emergencia, detección de condiciones climatológicas adversas y aviso sobre puntos peligrosos, como zonas en obras.

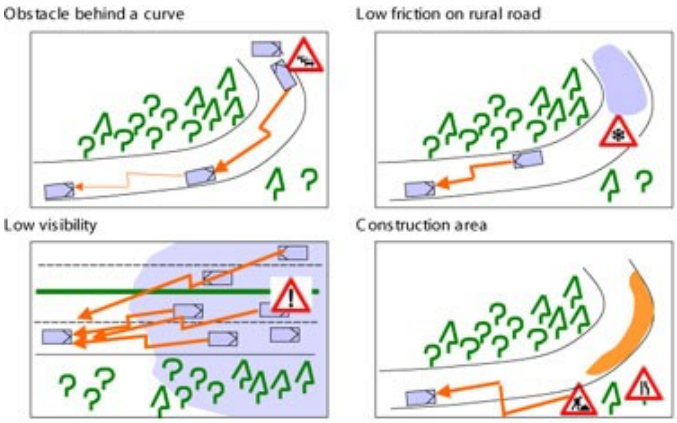


Figura 12: Escenarios de riesgo para el conductor

WILLWARN es un ejemplo típico de aplicación de las redes vehiculares, con el objetivo de la seguridad del conductor. Su desarrollo comenzó en enero de 2005 y finalizó dos años más tarde, en enero de 2007. [25]

2.3. Otros trabajos relacionados

2.3.1. *CitySense*

Figura 13: Nodo montado en una farola de Cambridge.

CitySense es una red de sensores a escala urbana que está siendo desarrollada por investigadores de la universidad de Harvard y de BBN Technologies. Consistirá en 100 sensores wireless desplegados en la ciudad de Cambridge, MA, en lugares como postes de electricidad o edificios (figura 13). Cada nodo monitorizará datos como la polución del aire y otros datos atmosféricos. [26]

CitySense es una iniciativa muy atractiva porque pretende ser un escenario de pruebas totalmente abierto, de forma que investigadores de todo el mundo puedan usarla para probar aplicaciones relacionadas con las redes de sensores en una configuración a escala urbana.

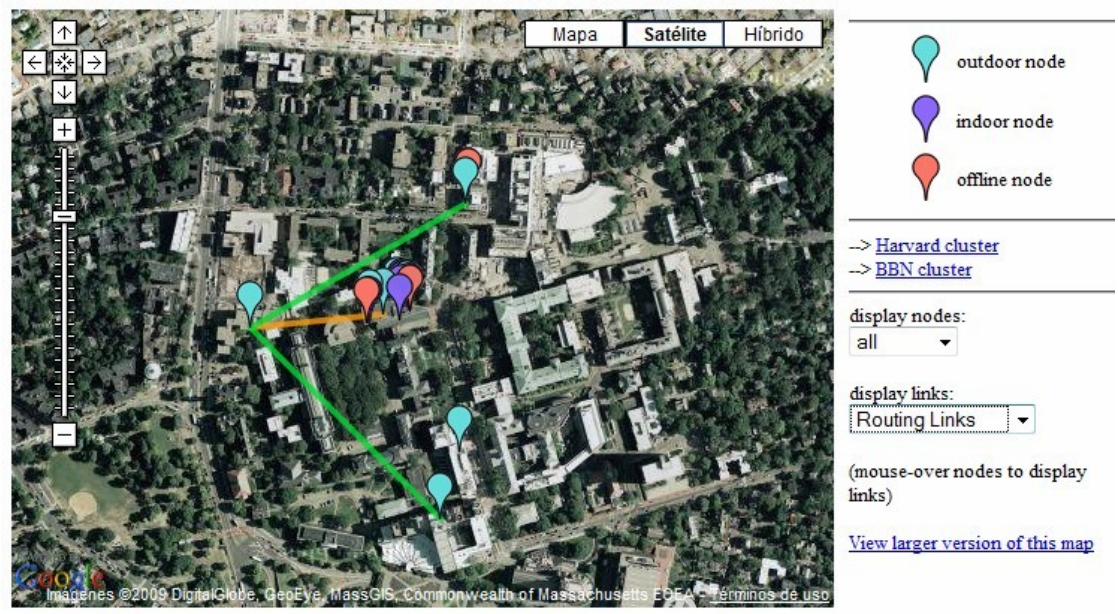


Figura 14: Foto de Cambridge. En la imagen se ve la posición de los nodos desplegados en el cluster de la universidad de Harvard.

2.3.2. Protocolo para la diseminación de información en una red vehicular mediante una estrategia broadcast

Éste es el trabajo más relacionado con el este proyecto. Desarrollado paralelamente como Proyecto de Fin de Carrera por Mario Torrecillas Rodríguez, el *Protocolo para diseminación de información en una red vehicular mediante una estrategia broadcast* tiene exactamente el mismo objetivo que éste trabajo, sólo que la implementación difiere bastante del que se presentará en el apartado 3. [27]

La primera diferencia que salta a la vista viendo sólo el título es que el protocolo tiene un enfoque **broadcast**. Los nodos solicitarán los fragmentos que les falten a sus vecinos y éstos los enviarán broadcast, de forma que otros nodos también podrán aprovechar esa transmisión si están lo suficientemente cerca.

Debido a que se propone una solución del tipo broadcast para este problema, se ha de prescindir de asentimientos por parte de los receptores (no se puede saber cuándo un nodo concreto está recibiendo correctamente la información o no, cuándo el nodo escapa a la visibilidad del emisor en medio de una transmisión, etc.).

Esto obliga, en cierto modo, a tratar de reducir la probabilidad de que una transmisión se realice de forma incorrecta o, lo que es lo mismo, tratar de reducir el tiempo en que permanece dicha transmisión activa entre el portador de la información y uno o varios nodos receptores en un momento dado. La solución que se propone en este trabajo es la de dividir el contenido en bloques o partes, de un tamaño fijado P_s, lo suficientemente grande como para que el número de partes de dicho contenido sea manejable, sin llegar a comprometer la eficiencia del protocolo.

Para comunicar de forma efectiva los nodos, se han planteado tres tipos de mensaje:

1. ANUNCIO
2. SOLICITUD
3. PARTE

El mensaje de tipo **ANUNCIO** se enviará periódicamente al medio para informar de las partes que un nodo está dispuesto a transmitir en un momento dado. Su formato se muestra a continuación:

Número de partes (1 byte)	Partes completas (vector de booleanos, tamaño variable)

Figura 15: Formato del mensaje de tipo ANUNCIO

Una vez que los nodos vecinos del portador de la información saben que existe ésta, y conocen las partes que pueden obtener, es necesario un tipo de mensaje, **SOLICITUD**, que sirva para indicar al portador que se desea recibir una parte concreta. La estructura de este tipo de mensaje es la siguiente:

Número de parte (1 byte)

Figura 16: Formato del mensaje de tipo SOLICITUD

Al contrario que el resto de tipos de mensaje, éste se enviará de forma unicast al anunciante, ya que de otra forma el protocolo funcionaría de forma incorrecta al recibir solicitudes en cualquier instante de tiempo de forma inesperada: es importante que sólo el anunciante reciba cada una de las solicitudes.

Por su parte, el anunciante, recibirá una o varias solicitudes por parte de los nodos receptores, y decidirá la parte a enviar; para esto se ha optado por un algoritmo que seleccione la parte más solicitada. La estructura del tipo de mensaje **PARTE**, sería la siguiente:

Número de parte (1 byte)	Información (6400 bytes)

Figura 17: Formato del mensaje de tipo PARTE

Es importante mencionar que los receptores han de analizar la parte recibida, y aceptarla o no en función de si ya se tiene almacenada; es decir, independientemente de la parte solicitada anteriormente, se aceptará la parte recibida (aun no siendo ésta la solicitada) siempre y cuando no se tenga.

El funcionamiento del protocolo, sin entrar en detalles más concretos, se representa en el siguiente esquema en SDL:

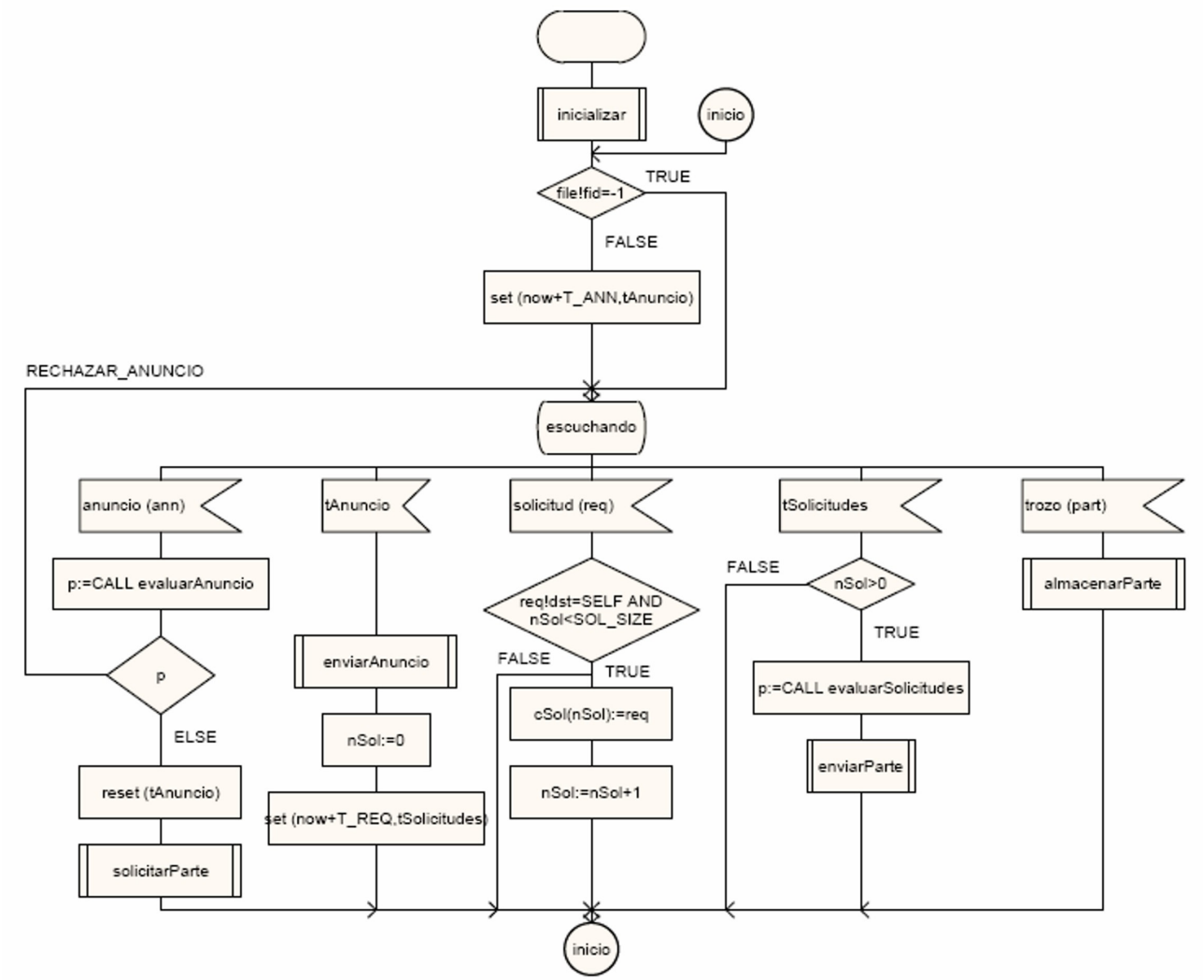


Figura 18: Descripción del protocolo en SDL

3. Descripción del protocolo

El objetivo del proyecto es llevar a cabo el desarrollo y la posterior implementación de un protocolo de diseminación de bloques relativamente grandes de información. Para que sea viable la transmisión, el bloque de información que se difunda se divide en fragmentos que se envían siempre de forma secuencial, sin que haya desorden entre éstos.

El escenario en el que tiene que ejecutarse este protocolo es una red inalámbrica de nodos con las características y **limitaciones** de las *Wireless Sensor Networks*, *WSNs*:

- **Topología dinámica**: Una de las premisas del protocolo es el hecho de que los nodos puedan estar en movimiento. Esto implica una topología cambiante, con nodos incorporándose a la red en cualquier momento. De ahí que sea necesario un **mecanismo de anuncios de información periódicos**, de forma que cuando un nodo entre en el área broadcast de otro, sepa, en un periodo mínimo de tiempo, qué información se está distribuyendo por la red.

- **Variabilidad del canal**: También tienen que estar preparados para afrontar desconexiones temporales y pérdidas de información por la existencia de colisiones en transmisiones paralelas. En este aspecto se implementa un **sistema de asentimientos y reintentos** muy simple en las transmisiones unicast para garantizar que no haya desorden en los paquetes que se transmiten.

- **Conexiones ad-hoc**: No existe una infraestructura fija en la red, es decir, todos los nodos son idénticos en funciones y capacidades. Un nodo, independientemente de que tenga o no parcialmente el bloque de información que se va distribuyendo, está abierto a recibir información de un vecino, o a transmitir la cantidad de información que posea a otros nodos interesados. **Los nodos son totalmente independientes**, no existe ningún tipo de jerarquía en la red.

- **Limitaciones hardware**: Posiblemente es el apartado más importante, ya que es el que más cambios provoca en el planteamiento inicial del protocolo. Una limitación es la de **memoria**. El bloque de información que se difunde será de 400 Kbytes. Dado que los paquetes que se enviarán los nodos tendrán una longitud de menos de 100 bytes, será necesario dividir el bloque en muchos fragmentos. Teniendo en cuenta que la memoria RAM de los dispositivos MICAz en los que se ejecuta el algoritmo es de sólo 4 Kbytes, es necesario que haya **un mecanismo de almacenamiento periódico** en la memoria flash de datos del mote (512 Kbytes). Estos accesos continuos a la memoria provocan retardos importantes a la hora de recibir fragmentos de información, que hacen necesaria la implementación de **mecanismos de control de flujo** (retardo entre accesos a la memoria o en el envío de fragmentos contiguos).

A continuación se presenta la lista de mensajes que se intercambiarán los nodos de la red, así como su formato (campos, longitud, etc...).

3.1. Formato de los mensajes

Existen tres tipos de mensajes que se intercambian los nodos:

- Mensajes de control: oferta y solicitud.
- Mensaje de datos

El **mensaje de oferta** es necesario para que los nodos conozcan la información que poseen sus vecinos. Se envía siempre **broadcast** y únicamente por los nodos que disponen de alguna parte de la información a difundir. Cuando es así, el MICAz lo envía periódicamente (cada TIEMPO_OFERTA) siempre que no esté recibiendo o enviando datos.

La estructura del mensaje de oferta es la siguiente:

frgsTotales (32 bytes)	**ultFrg** (32 bytes)

Figura 19: Formato del mensaje de oferta

- frgsTotales (**Fragmentos totales**): Este campo indica cuántos fragmentos tiene en total el mensaje que se está diseminando.

- ultFrg (**Último fragmento**): Indica cuál es el último fragmento del bloque de datos que tiene el dispositivo que está enviando la oferta. Dado que las transmisiones de información se producen de forma secuencial, éste valor indica implícitamente que el emisor de la oferta tiene todos los fragmentos anteriores.

Una vez que los nodos han recibido el mensaje de oferta, surge la necesidad de un mensaje que permita a los nodos solicitar fragmentos: el **mensaje de solicitud**.

Éste se envía siempre **unicast** como respuesta a un mensaje de oferta en el que el nodo esté interesado, es decir, si el identificador del último fragmento de la oferta es mayor que el último fragmento que tiene el nodo receptor.

El envío de la solicitud se produce siempre que haya pasado un **tiempo prudencial** respecto al envío de la solicitud anterior. Ésto es así para evitar que en un momento dado haya más de un emisor para un único receptor, situación que, como se verá más adelante, perjudica la capacidad de recepción del MICAz.

La estructura del mensaje solicitud es la más simple, ya que cuenta con único campo:

idFrg (32 bytes)

Figura 20: Formato del mensaje de solicitud

- idFragmento (**Identificador del fragmento**): Indica cuál es el fragmento en el que está interesado el solicitante.

Por último, el **mensaje de datos** contiene la información real que se tiene que difundir en la red. Este tipo de mensaje constituye un flujo de datos en la red, es decir, cuando un nodo acepta un mensaje de solicitud, comienza el envío **unicast** de todos los fragmentos de información de que disponga el emisor a partir del fragmento solicitado. Este proceso puede verse interrumpido si el receptor no confirma la recepción de los datos.

El formato del mensaje de datos es el siguiente:

id (32 bytes)	**Datos** (64 bytes)

Figura 21: Formato del mensaje de datos

- id (**Identificador del fragmento**): Contiene el identificador del fragmento que se envía.
- **Datos**: 64 bytes de información útil.

3.2. Especificación en SDL y descripción del protocolo

Inicialmente se declaran las estructuras y los tipos de mensaje que se intercambian los nodos. En la figura siguiente, cada nodo está representado por una instancia del bloque "MICAz" y se representa la conexión ad-hoc entre dos motas:

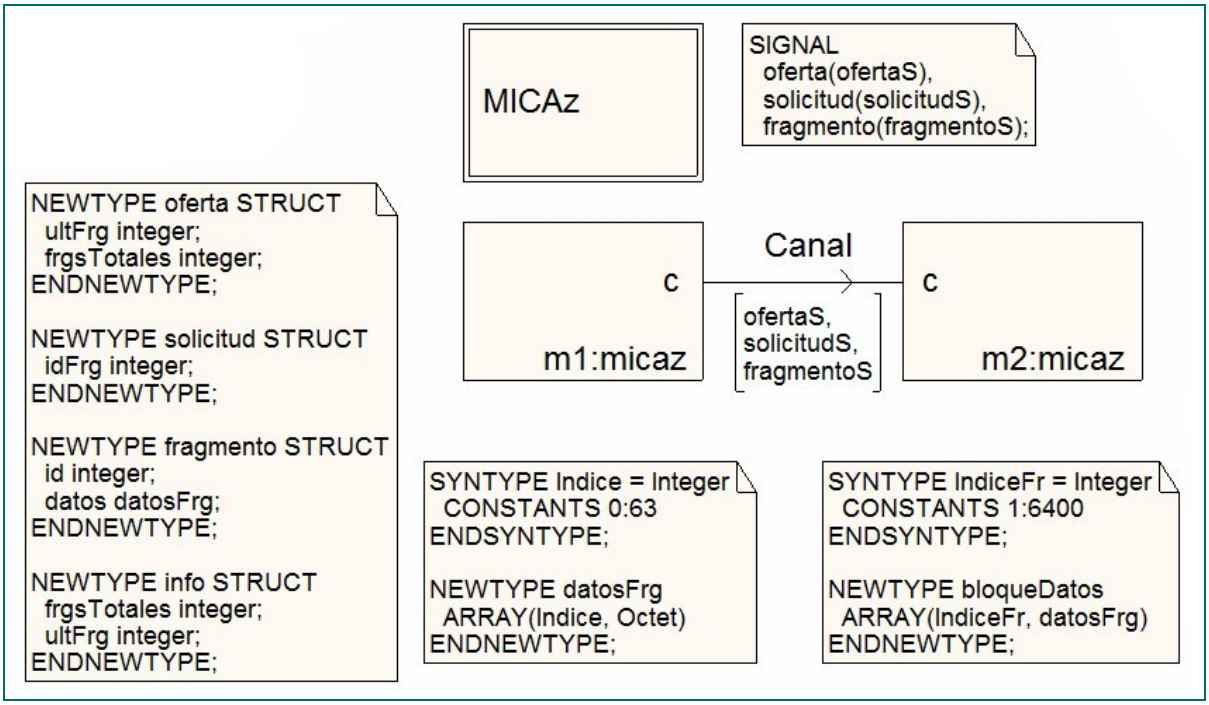


Figura 22: Declaración de estructuras en SDL

La siguiente figura muestra el principio del algoritmo que ejecuta cada uno de los nodos, así como las variables que mantienen durante la ejecución del mismo:

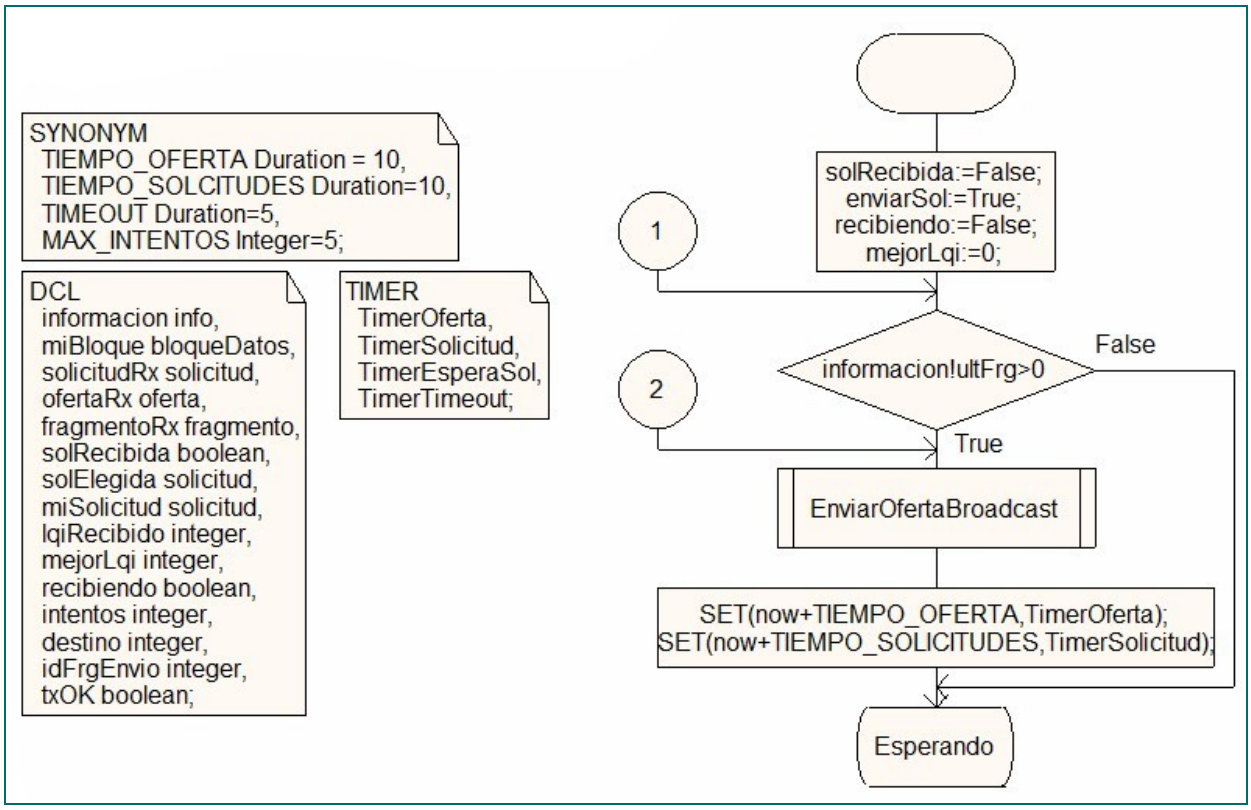


Figura 23: Inicio del algoritmo en SDL y declaración de variables locales

Al comenzar la ejecución del algoritmo, el nodo en cuestión inicializa algunas variables cuya función se irá viendo a medida que aparezcan en el flujograma. Cada nodo tiene una estructura local llamada *informacion*. Esta estructura tiene dos campos (figura 22, página 24): *frgsTotales* y *ultFrg*.

El primero indica el número total de fragmentos que tiene el bloque de información que se está diseminando. El segundo indica cuál es el último fragmento del bloque que posee el nodo. Sabiendo que los fragmentos se envían de forma secuencial, si ambos campos coincidieran (siendo distintos de cero) el nodo tendría la totalidad del archivo. Por eso, cuando el nodo se enciende, lo primero que hace es comprobar si tiene algún fragmento que ofrecer (el campo *ultFrg* de su estructura de información es mayor que cero), y si es así, envía una oferta broadcast.

Una vez enviada la oferta, inicializa los temporizadores de oferta y de solicitud con dos constantes respectivamente:

- *TIEMPO_OFERTA*: es el intervalo de tiempo mínimo entre el envío de ofertas consecutivas
- *TIEMPO_SOLICITUDES:* Es el tiempo que un nodo está "recolectando" solicitudes tras enviar una oferta para, posteriormente, enviar sus fragmentos al nodo emisor de la solicitud con mayor calidad en el enlace.

Tras iniciar los temporizadores, el nodo pasa al estado "Esperando".

En el estado “Esperando”, pueden ocurrir varias cosas:

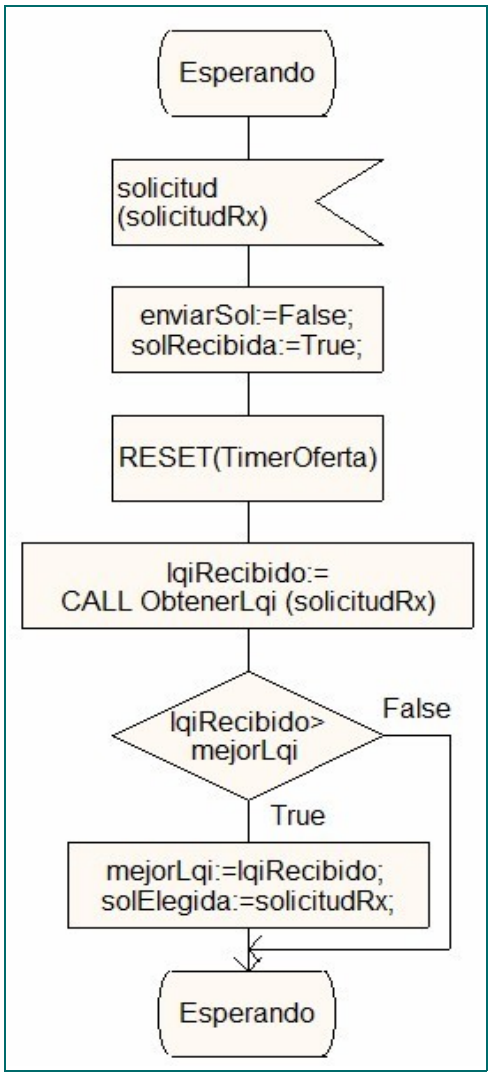


Figura 24: Se recibe una solicitud

1. Se recibe una solicitud:

Cuando se recibe una solicitud (*solicitudRx*) tras haber enviado una oferta, se cambia el valor de la variable *enviarSol*. Ésta se usa para impedir que se envíen solicitudes en momentos no deseados (mientras se está enviando, mientras se está recibiendo y si ha pasado poco tiempo desde la última solicitud realizada para evitar recibir datos de dos fuentes distintas).

La variable *solRecibida* se pone a *true* indicando que se ha recibido al menos una solicitud. Es necesaria porque cuando se acaba el tiempo para recibir solicitudes, hace falta saber si existe algún solicitante al que enviarle los fragmentos.

También se para el temporizador de oferta, ya que al haber al menos un solicitante, **se va a iniciar una conexión unicast** con otro nodo y no habrá más envíos mientras dure la misma.

Para cada solicitud recibida durante el intervalo *TIEMPO_SOLICITUDES* se comprueba la calidad del enlace con el nodo remoto y se compara con la mejor recibida hasta el momento. Si la calidad del enlace del mensaje recibido es mejor, el nodo emisor del mismo pasa a ser el mejor candidato para recibir los fragmentos del bloque de información. Y así sucesivamente hasta que acabe la recogida de solicitudes.

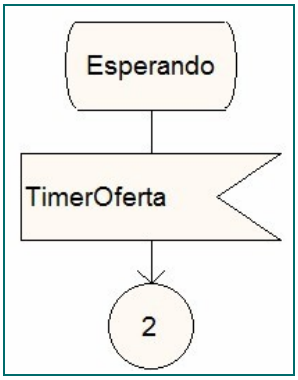


Figura 25: Finaliza TimerOferta

2. Finaliza el temporizador de oferta:

Si el temporizador de oferta no se ha parado y salta antes de que llegue alguna solicitud u otra oferta que interese al nodo, se vuelve a enviar la oferta broadcast, reseteando los temporizadores de oferta y de recolección de solicitudes (ver figura 23, página 25).

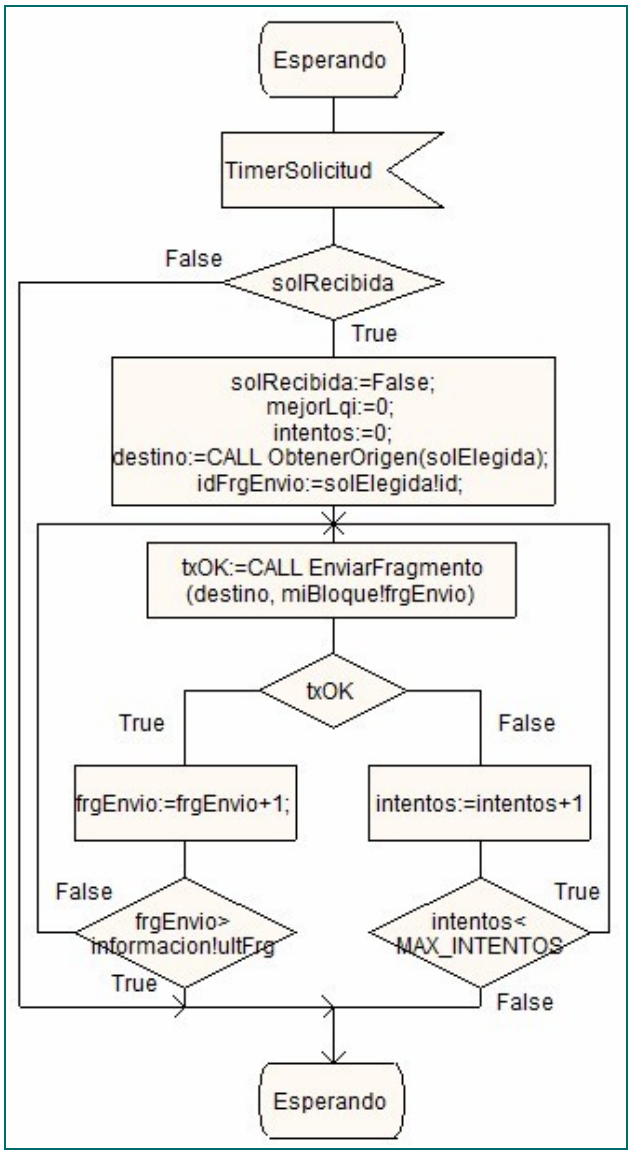


Figura 26: Finaliza el temporizador de solicitudes

3. Finaliza el temporizador para la recolección de solicitudes:

Cuando finaliza el temporizador para recoger solicitudes se comprueba inmediatamente el valor de la variable *solRecibida*. Si es *true* indica, como se ha visto en el punto anterior, que existe un nodo al que transmitirle la información. Tras comprobar su valor, se "resetean" algunas variables. La variable *intentos* mantiene el número de intentos de transmisión que lleva el fragmento que se está transmitiendo.

Antes de comenzar la transmisión, se extrae el origen de la mejor solicitud recibida, que es el destino de los fragmentos a enviar.

A continuación comienza el envío de una secuencia de fragmentos, empezando por el identificador que pide la solicitud elegida. La función encargada del envío del fragmento al medio devuelve *true* si el mensaje enviado ha sido reconocido a nivel MAC y *false* en caso contrario, por tanto, si se recibe un ACK que confirme el fragmento enviado, se incrementa en una unidad el identificador de fragmento a enviar y, mientras no se hayan enviado todos los fragmentos que tiene el emisor, se va repitiendo el proceso.

Cuando un mensaje no es confirmado por el receptor, se incrementa el número de intentos y se vuelve a enviar, repitiendo la operación hasta que se alcance el máximo de intentos, *MAX_INTENTOS*. Momento en el que finaliza la conexión entre los dos nodos.

Este mecanismo hace que la constante *MAX_INTENTOS* sea de gran importancia en entornos en los que haya muchas transmisiones paralelas, ya que como consecuencia habrá un número elevado de colisiones. Por ello, el valor de dicha constante determina la estabilidad de las parejas de nodos que se formen durante la ejecución del algoritmo, o sea, cuanto mayor sea su valor, mayor número de pérdidas aguantará el enlace entre ambos. No obstante, no se le puede dar un valor muy alto ya que el receptor podría no estar enviando los asentimientos por no estar ya en el radio de visión del emisor, con lo que éste tendría que dar por finalizado el envío.

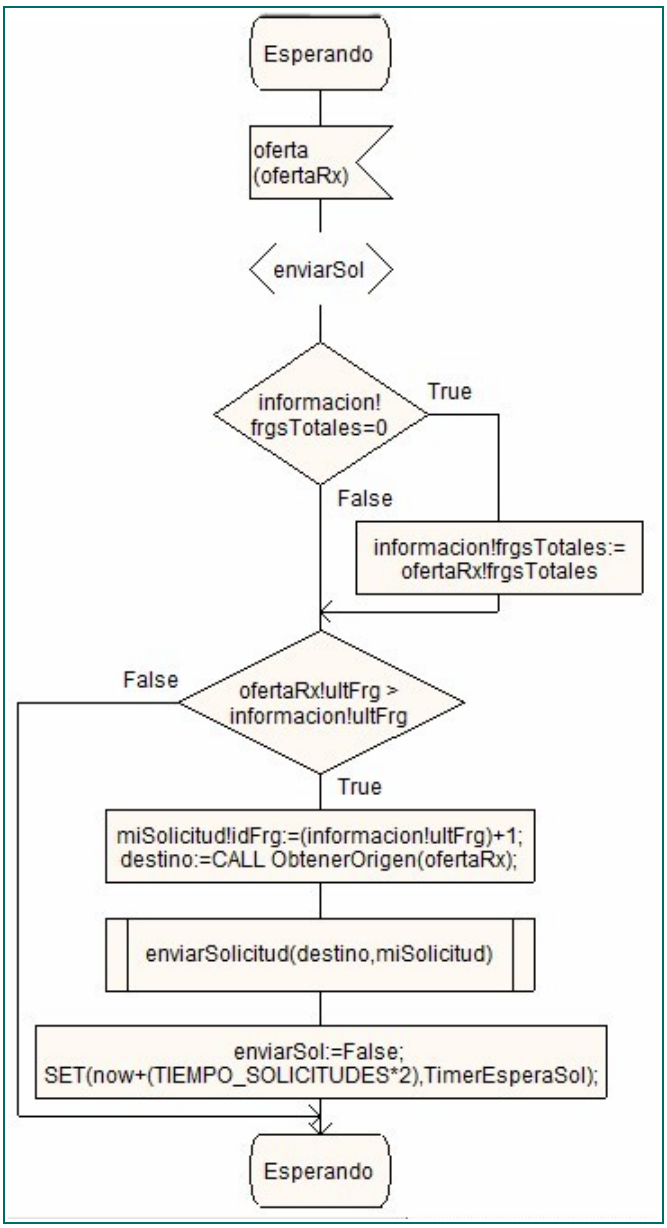


Figura 27: Se recibe una oferta

4. Se recibe una oferta:

Cuando se recibe una nueva oferta, antes de comprobar su contenido, se evalúa el valor de la variable *enviarSol*, ya que si ésta vale *false*, el nodo no podrá enviar solicitudes, con lo que no importará la información que lleve el mensaje recibido.

Si *enviarSol* es *true*, entonces **se evaluará el contenido de la oferta**. Primero se comprueba la información local del nodo para compararla con la recibida. Si el valor de la variable local *frgsTotales* es igual a 0, aprovechará el campo *frgsTotales* del anuncio recibido para actualizar su estructura local. Es decir, la oferta sirve, independientemente de que vaya a ser correspondida, para que los nodos inicialmente vacíos sepan el tamaño del fichero que se va distribuyendo por la red.

Después el nodo comprueba si está interesado en la oferta, es decir, si el emisor de la oferta tiene más fragmentos que él mismo. En caso afirmativo, compone un mensaje de solicitud pidiendo el fragmento inmediatamente posterior al último almacenado.

Inmediatamente después de enviar la solicitud, la variable *esperaSol* se cambia a *false* y se inicia un temporizador, *TimerEsperaSol*, para prevenir que haya dos solicitudes como respuesta a dos ofertas disitintas, provocando que el nodo pueda pasar a recibir fragmentos de dos emisores al mismo tiempo, colapasándolo.

El tiempo que transcurre como mínimo entre dos solicitudes consecutivas será igual al doble del tiempo que el nodo emisor está recogiendo solicitudes. Recordemos que todos los nodos tienen el mismo programa y, por tanto, todos tienen definida la constante *TIEMPO_SOLICITUDES*. Esperando este tiempo, el nodo solicitante se asegura de que la mota que ha realizado la oferta ya ha tomado una decisión acerca de quién recibirá la información, así que si el nodo en cuestión no ha sido elegido, podrá volver a solicitar información ante nuevas ofertas.

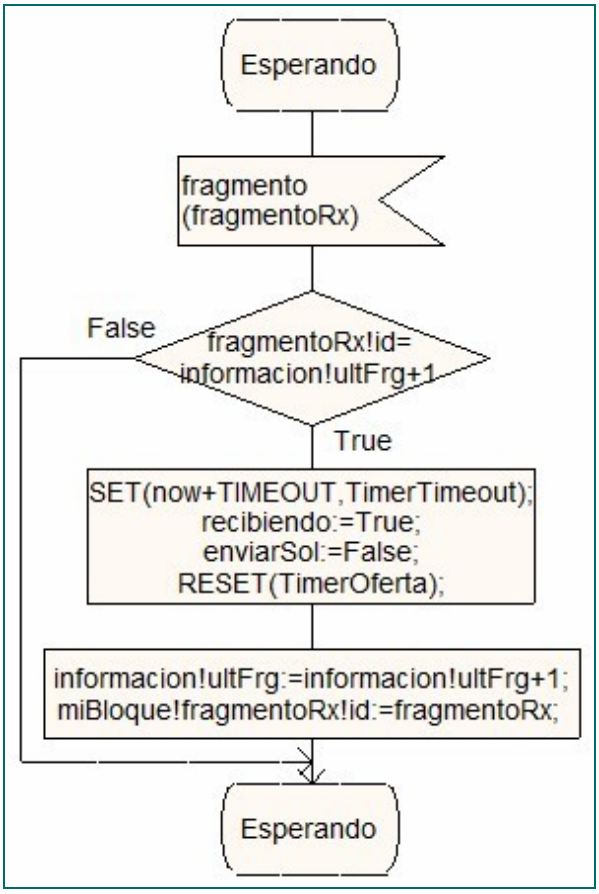


Figura 28: Se recibe un fragmento

5. Se recibe un fragmento:

Al recibir un fragmento, se comprueba si se trata de uno "esperado", es decir, si su identificador es igual al último fragmento del receptor más uno. Si es así, el nodo entra en el estado ***recibiendo*** y deja de enviar solicitudes.

En ese momento se inicia el temporizador *TimerTimeout* con la constante *TIMEOUT*, que indica el tiempo que esperará un nodo que esté recibiendo fragmentos antes de salir del estado "recibiendo".

Si un nodo que está recibiendo un flujo de fragmentos deja de recibirlos durante un tiempo TIMEOUT, considerará acabada la transmisión, bien porque hay recibido correctamente todo el archivo, o bien porque quede fuera del alcance del emisor.

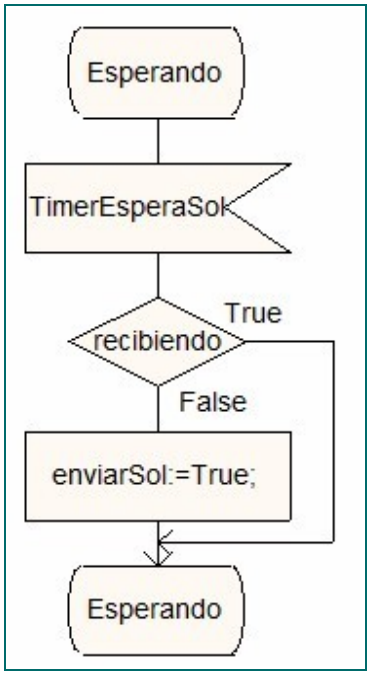


Figura 29: Finaliza TimerEsperaSol

6. Finaliza el temporizador para poder enviar solicitudes:

Cuando pasa el tiempo *TIMEOUT*, el nodo ya está fuera del intervalo de tiempo prudencial en el que no puede contestar a las ofertas para evitar recepciones simultáneas, por lo que la variable *enviarSol* se volverá a poner a *true* siempre que el nodo no haya sido elegido en la última solicitud que realizó (estado *recibiendo*).

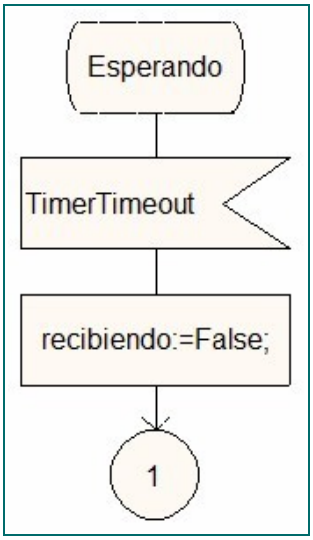


Figura 30: Finaliza TimerTimeout

7. Finaliza el temporizador *TimerTimeout*:

Por último, cuando pasa un tiempo *TIMEOUT* sin recibir fragmentos, salta el temporizador *TimerTimeout* y el nodo sale del estado *recibiendo*.

Cuando ésto ocurre, se vuelve al punto 1 (ver figura 23, página 25), es decir, dado que al menos se ha recibido un fragmento, se comprueba que el último fragmento del mote sea mayor que cero y si es así, los fragmentos se han guardado correctamente y puede comenzar a anunciar su información.

3.3. Particularidades y limitaciones de TinyOS y NesC

TinyOS proporciona una serie de interfaces y componentes que permiten controlar los recursos software y hardware del MICAz. A continuación se muestra una lista con las interfaces que usa la implementación del protocolo en NesC, con una breve descripción de su funcionalidad.

3.3.1. Lista de interfaces utilizadas

- ***Boot***
  Esta interfaz se usa en todos los programas de TinyOS 2.x. La proporciona el componente *MainC*. La ejecución del programa comienza cuando se señaliza el evento *booted* de dicha interfaz.

- ***Timer***
  Esta interfaz se ha instanciado hasta 6 veces para implementar los temporizadores del programa. Los cuatro explicados anteriormente: *TimerOferta, TimerSolicitud, TimerTimeout* y *TimerEsperaSol*. Y dos más: *TimerEnvio* (para insertar un retardo entre fragmentos consecutivos) y *TimerError* (para controlar la frecuencia de parpadeo de los leds cuando ocurre un error).

- ***AMSend***
  La interfaz *AMSend* controla el envío de paquetes al medio radio. Dado que existen tres tipos de paquetes, se ha instanciado tres veces.

- ***Receive***
  Análoga a la interfaz *AMSend*. Proporciona los eventos de recepción de paquetes del medio radio, ofreciendo un puntero a su campo de datos, que permite tratarlos debidamente.

- ***BlockWrite***
  Proporciona comandos para la escritura de grandes bloques de información en la memoria flash de datos del MICAz: escritura (*write()*), borrado (*erase()*) y sincronización (*sync()*). Se ha usado para ir almacenando los fragmentos recibidos.

- ***BlockRead***
  Análoga a *BlockWrite* pero con comandos para la lectura de la memoria.

- ***Mount***
  Se encarga de inicializar el volumen de la memoria flash de datos destinado a almacenar de forma permanente pequeños datos de configuración. La información relativa a este volumen se da en el apartado siguiente (Distribución de la memoria del MICAz, página Error: No se encuentra la fuente de referencia).

- ***ConfigStorage***
  Proporciona comandos para la lectura y escritura de datos del volumen de configuración de la memoria.

- ***Leds***
  Proporciona los comandos necesarios para controlar el encendido/apagado de los leds del MICAz.

- ***CC2420Packet***
  Es una interfaz muy importante en este protocolo, ya que proporciona el comando necesario para obtener la calidad del enlace a partir de un paquete recibido: *getLqi (message_t)*. Se usa exclusivamente durante el proceso de recolección de solicitudes (ver figura 24, página 26).

- ***SplitControl***
  Esta interfaz permite arrancar o detener los módulos destinados a las comunicaciones. Se encarga de inicializar el medio radio, de forma que no se pueden recibir o enviar paquetes antes de ello.

- ***AMPacket***
  También es una interfaz muy relevante para el protocolo propuesto, ya que se basa en conexiones unicast. Permite obtener la dirección origen de un paquete recibido. Se usa cuando un nodo recibe una oferta, para enviar la solicitud al emisor de la misma, y también en la elección de la mejor solicitud, para enviar los fragmentos a la dirección del solicitante.

- ***Packet***
  La interfaz *Packet* se usa para obtener algún campo de un mensaje. En el protocolo se ha usado para obtener el campo de datos de un paquete. El comando *command void *getPayload(message_t *msg, uint8_t len)* devuelve un puntero al campo de datos del mensaje *msg*, permitiendo leerlos y modificarlos.

- ***PacketAcknowledgements***
  Una de las interfaces más características del protocolo, dada su naturaleza unicast. Como se

ha explicado en apartados anteriores (ver figura 26, página 27), se implementa un mecanismo de asentimientos (ACKs) a nivel MAC que garantiza la fiabilidad del algoritmo en la entrega de paquetes, sin que exista desorden.

La interfaz *PacketAcknowledgements* proporciona dos comandos que acompañan la transmisión de cada fragmento en el emisor:

- *command error_t* ***requestAck****(message_t *msg)*
  Este comando indica al componente de radio que, cuando envía un paquete, se debe confirmar con un ACK síncrono.

- *command bool* ***wasAcked****(message_t *msg)*
  Como su nombre indica, el comando *wasAcked* señala, mediante una variable booleana si el mensaje *msg* que se le pasa como parámetro ha sido asentido por el receptor o no.

3.3.2. Distribución de la memoria del MICAz

Toda la información del protocolo se almacena en la memoria flash del MICAz, de 512 Kbytes de capacidad. TinyOS permite crear volúmenes de esta memoria, de forma que se puedan acceder y referenciar independientemente, equivalentes a las particiones de un ordenador. Dichas particiones deben ser declaradas en un fichero XML que se encuentre en el mismo directorio que los ficheros del componente que hará uso de la memoria.

Para este protocolo se han declarado **dos volúmenes**:

- El primer volumen tiene un tamaño de 2 Kbytes. Su uso es **almacenar permanentemente información sobre el estado del nodo**. En esta partición se almacena la estructura de información local que posee cada mote, con los valores del número total de fragmentos del bloque de información y con el identificador del último fragmento que fue almacenado.

  Gracias a esta partición, la información del nodo permanece invariable al apagarlo y encenderlo. Cuando el MICAz se enciende, lo primero que hace es "montar" (interfaz *Mount*) esa partición y leer su contenido (interfaz *ConfigStorage*). Para saber que la información que contiene la estructura es veraz, se incluye un tercer campo llamado *version*, con un entero definido al azar al compilar el protocolo. Cuando el MICAz lee el campo, comprueba que se corresponda con el definido al compilar. Si es así, la información de la memoria flash es correcta, y la copiará en la RAM para la ejecución del programa.

- El segundo volumen tiene un tamaño de 400 Kbytes. Almacenará un total de 6400 fragmentos, ya que cada fragmento tiene un tamaño de 64 bytes. En él se irán **guardando los fragmentos** secuencialmente a medida que se vayan recibiendo. Y se irán leyendo a medida que haya que enviarlos. En cualquier caso, es el campo *ultFrg* del primer volumen el que actúa como índice para acceder al fragmento correspondiente en el segundo volumen, es decir, el segundo volumen no almacena ningún tipo de información de control sobre la distribución de los fragmentos. Únicamente guarda el contenido de los fragmentos, sin ningún tipo de división entre ellos, o sea, es una serie de bytes que el propio programa se encargará de indexar en función del fragmento a leer.

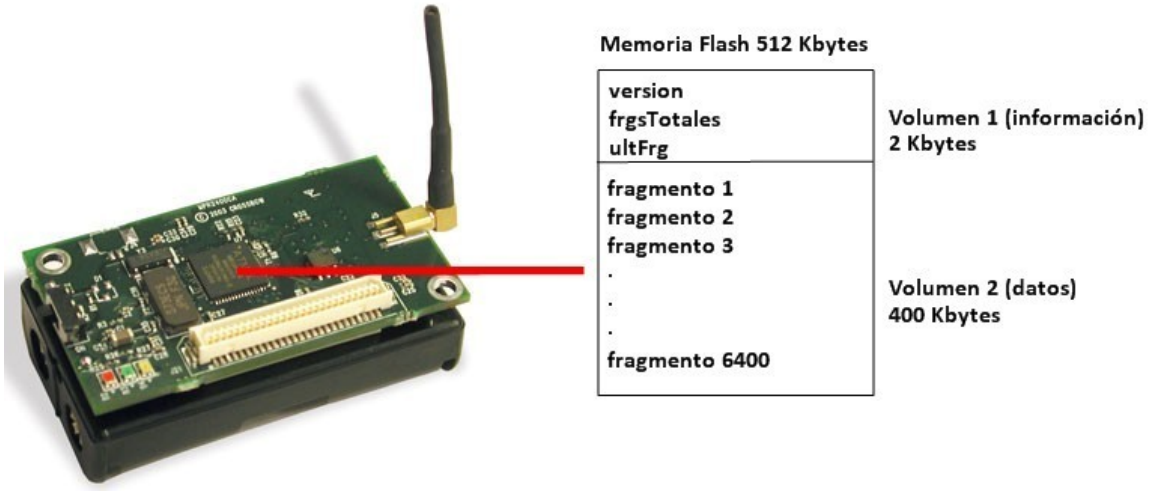


Figura 31: División de la memoria en el MICAz

Las **particiones** de la memoria **pueden ser de distintos tipos** según la información que se vaya a almacenar en ellas. El primer volumen está destinado a guardar información de configuración y por ello se tratará con unas interfaces distintas a las del segundo volumen: *Mount* y *ConfigStorage* (ver Lista de interfaces, página 30). El volumen de datos está destinado a almacenar grandes cantidades de información, por lo que se accederá mediante las interfaces *BlockWrite* y *BlockRead*.

Acceder al volumen de configuración (especialmente escribir) es bastante más lento que acceder al volumen de datos (decenas de milisegundos más). Por ello hay que restringir la velocidad con la que el emisor envía fragmentos, ya que en cada paquete recibido, el destinatario tiene que acceder a la memoria flash para guardarlo.

En versiones iniciales del protocolo, no había ningún tipo de control de flujo en este sentido, por tanto, cuando un nodo recibía fragmentos a máxima velocidad (aproximadamente un milisegundo) tenía problemas al acceder a la memoria. Tras distintas pruebas, se concluyó que el MICAz era capaz de acceder al volumen de datos de la memoria para guardar el fragmento únicamente si existía un **retardo mínimo de 8 milisegundos entre fragmentos consecutivos**.

Si además se quería actualizar la estructura de información en tiempo real en la memoria flash (volumen de configuración) el retardo mínimo debía de ser superior a 75 milisegundos, haciendo inviable este mecanismo.

A la vista de estas limitaciones se optó por retardar los fragmentos 8 milisegundos, para permitir al MICAz guardarlos en memoria flash en tiempo real (ver figura 26, página 27).

En cuanto a la parte de configuración, la estructura local del nodo se va actualizando, es decir, va incrementando el identificador del último fragmento recibido, en tiempo real en la memoria RAM cada vez que se almacena el propio fragmento en el segundo volumen de la memoria flash. Sin embargo, **sólo se actualizará en el volumen de configuración de la flash cuando haya acabado la transmisión** (ver figura 30, página 30).

4. Pruebas

Durante la fase de pruebas fue necesario programar muchas veces y con distintos programas los MICAz. En esta fase del desarrollo del proyecto fue necesario hacer uso del programador MIB520 para instalar el programa en las motas, así como de los leds del MICAz para saber cómo se comporta y la interfaz de comunicación PC-Mota.

4.1. El programador MIB520

Para cargar poder programar los dispositivos se usó el *gateway* MIB520, que cuenta con un conector estándar de 51 pines que **proporciona una interfaz USB entre el ordenador y el MICAz**.

Figura 32: MIB520 (a la derecha con un MICAz montado)

El MIB520, además de programar las motas, acoplado con un MICAz puede actuar también como estación base , es decir, como una interfaz inalámbrica entre el ordenador y la red de nodos, ya que **ofrece dos puertos lógicamente separados** para dichas tareas, permitiendo inyectar parámetros en la red, reprogramar los nodos, etc. También, mediante la alimentación de la interfaz USB es capaz de alimentar al dispositivo sin necesidad de fuentes externas.

4.2. Depuración del código

Las pruebas comenzaron con una versión inicial del código del protocolo. La variedad de pruebas realizadas han hecho que el programa vaya sufriendo cambios hasta alcanzar la versión estable que se detalla en el tercer apartado.

Para poder depurar el código y ver los errores existentes fue necesario usar **dos interfaces**:

- Los tres **leds** (0: rojo, 1: verde y 2: amarillo) del MICAz.
- La interfaz de **comunicación con el ordenador** que proporciona el MIB520.

4.2.1. Interfaz Leds

Los leds constituyen la interfaz con el usuario para saber qué está haciendo el nodo en un momento dado, dependiendo de la frecuencia de parpadeo y la combinación de colores que haya.

El control de los leds pasa por el uso de una interfaz que proporciona TinyOS, llamada *Leds.* El componente que proporciona esa interfaz usado en las pruebas fue *LedsC*.

La interfaz *Leds* tiene **tres comandos** principales para cada uno de los tres leds:

- *command void ledXOff()*: **apaga** el led X.
- *command void ledXOn()*: **enciende** el led X.
- *command void ledXToggle()*: **alterna** el led X (si está apagado, lo enciende, y viciversa).

También es posible mostrar un número natural en binario (entre 1 y 7) mediante el comando *command void set(uint8_t val)*, pero ese y algún otro comando de la interfaz no se ha utilizado en las pruebas.

Para poder controlar los errores que tuvieran lugar durante la ejecución del programa se le incorporó una **función** que se encarga de detener el algoritmo para **mostrar el error** ocurrido a través de los leds.

Esta función tiene como único parámetro de entrada un entero que representa al error en cuestión. Cuando es llamada, se encarga de detener el componente de la radio, para evitar que se sigan recibiendo paquetes, y los temporizadores que haya en marcha. A continuación muestra en los tres leds una combinación de colores que parpadean simultáneamente con una frecuencia constante de 250 milisegundos.

Existen **cuatro errores** que detienen la ejecución del protocolo:

1. **Error de lectura**: Parpadean los leds amarillo, verde y rojo. Este error se produce cuando no es posible montar el volumen de configuración al inicio del programa o bien cuando uno de los parámetros con los que se intenta acceder a cualquiera de las dos particiones no es correcto (por ejemplo, una dirección fuera del rango permitido).

2. **Error de escritura**: Parpadean los leds amarillo y verde. Este error tiene lugar de forma análoga al anterior, es decir, con problemas en los accesos a la memoria con funciones de escritura. En las primeras pruebas fue un error frecuente, antes de implementar el mecanismo de control de flujo explicado al final del apartado anterior.

3. **Error de ACK**: Parpadean los leds amarillo y rojo. Este error se muestra cuando no es posible solicitar el reconocimiento a nivel MAC de algún fragmento enviado, una condición imprescindible en las transmisiones unicast. Es un error muy poco frecuente y no supuso problemas en ningún momento de la fase de pruebas.

4. **Error de envío**: Parpadea el led amarillo. El error de envío ocurre cuando se llama a la función correspondiente con algún argumento incorrecto, como, por ejemplo la dirección

de destino de un paquete, o cuando el componente de radio está ocupado realizando otro envío.

El control de estos cuatro errores es necesario para garantizar la fiabilidad del programa y la correcta ejecución del algoritmo. Sin embargo, los únicos errores que han estado presentes durante las comprobaciones del protocolo (sobre todo en las fases iniciales) son los dos primeros: lectura y escritura.

No obstante, aunque no exista ninguno de los errores anteriores, la interfaz de los leds ha hecho mucha falta durante la totalidad de las pruebas realizadas, ya que indican qué está haciendo el MICAz: cuándo se está haciendo una oferta, cuando se están enviando o recibiendo fragmentos, etc.

Independientemente de los errores, durante el correcto funcionamiento del programa, el significado de los leds es el siguiente:

- Led amarillo: Se alterna cada vez que se envía una oferta al medio.
- Led verde: Se alterna cada vez que se envía un fragmento.
- Led rojo: Se alterna cada vez que se recibe un fragmento.

Con este código de luces, la secuencia habitual que ejecutan los leds en la conexión entre dos nodos es como sigue: se enciende el led amarillo en el nodo que tiene información, y por tanto, hace la oferta (pueden ser los dos). Si al mote receptor de la oferta le interesa, enviará una solicitud (el envío de solicitudes no se muestra en los leds). Inmediatamente, parpadeará a una frecuencia de 8 milisegundos el led verde del nodo que tenga más información, pues comenzará a enviarle fragmentos al segundo, cuyo led rojo parpadeará al compás del verde del emisor. Cuando la información que tienen ambos se ponga en común, quedarán parpadeando los leds amarillos con un periodo de 1 segundo en ambos MICAz, mientras hacen ofertas a la espera de nuevos solicitantes.

4.2.2. Comunicación con el ordenador: interfaz MICAz - PC

La interfaz de comunicación con el puerto serie que proporciona el MIB520 a través de la interfaz USB también ha estado muy presente en las primeras pruebas. Los componentes de TinyOS que permiten hacer uso de esta interfaz (análogos a los que controlan el acceso al medio radio) son:

- ***SerialActiveMessageC***: proporciona la interfaz *SplitControl* para inicializar el puerto.
- ***SerialAMSenderC***: proporciona la interfaz *AMSend* para el envío de paquetes al PC.

Funciona como la interfaz que permite enviar paquetes al medio radio, pero en su lugar, las envía al puerto de comunicaciones con el ordenador y, ejecutando la herramienta Java que proporciona TinyOS llamada *Listen*, es posible ver la estructura del paquete enviado por el nodo en la pantalla del ordenador. Para arrancar la herramienta se ejecuta el comando siguiente:

```
$ java net.tinyos.tools.Listen -comm serial@/dev/ttyUSB0:micaz
```

El comando indica que la aplicación debe leer los paquetes recibidos por el puerto USB */dev/ttyUBS0*, y que la plataforma usada será el MICAz.

El formato típico de un paquete de datos es el siguiente:

dest addr	link source addr	msg len	groupID	handlerID	source addr	counter
ff ff	00 00	04	22	06	00 02	00 0B

Figura 33: Formato de un paquete en TinyOS

- Destination adress (2 bytes)
- Link Source Address (2 bytes)
- Message length (1 byte)
- Group ID (1 byte)
- Active Message handler type (1 byte)
- Payload (variable)

Los campos que aparecen en verde en la figura 33 constituyen el campo de datos del mensaje. Nótese que el MICAz envía los datos en formato *big-endian*, es decir, comenzando por el byte más significativo.

El uso de esta herramienta permitió medir, por ejemplo, la latencia en los procesos de escritura y lectura en la memoria flash, así como el tiempo en el envío y recepción de paquetes de radio. Se establecieron marcas temporales en distintos puntos del programa, se calculaba la diferencia entre ellas y el resultado se enviaba al ordenador. También fue muy útil a la hora de establecer contadores, por ejemplo, para ver si el número total de fragmentos enviados era el correcto.

4.3. Programas auxiliares

Para las distintas pruebas, todos los MICAz cuentan con un programa idéntico. Lo único en lo que difieren es en el contenido de sus memorias.

Para poder establecer un nodo como semilla (con la totalidad del bloque de información) es necesario modificar los dos volúmenes de su memoria flash (ver figura 31, página 33). Para acceder a la memoria flash del dispositivo son necesarios algunos comandos de las interfaces vistas en apartados anteriores. Ejecutar esos comandos requiere un programa independiente del programa del protocolo que se encarga de modificar tanto los campos de información del primer volumen como de rellenar todo el segundo. Por ello adaptar cada MICAz antes de comenzar una prueba implica programarlo dos veces: la primera con el programa correspondiente para escribir o borrar la memoria y la segunda con el propio protocolo una vez actualizada la memoria.

Los tres programas auxiliares principales que se desarrollaron para acceder a la memoria del MICAz fueron los siguientes:

4.3.1. Programa para escribir la memoria

El programa para escribir la memoria flash usa las siguientes interfaces:

- *Boot*
- *Leds*
- *Mount*
- *ConfigStorage*
- *BlockWrite*

La funcionalidad de las mismas se conoce de apartados anteriores. Este programa lo único que hace cuando arranca es montar el volumen de configuración y modificar los datos del mismo para que el nodo se comporte como una semilla cuando se le instale, posteriormente, el protocolo. Es decir, dado que el bloque de información tiene 6400 fragmentos, los campos *frgsTotales* y *ultFrg* pasarán a tener ese valor. Una vez actualizada la información del primer volumen, se llena el segundo volumen con el valor que se quiera dar al contenido de los fragmentos.

Para optimizar la velocidad en las pruebas, se desarrolló una versión de este programa que únicamente accedía al volumen de configuración, ya que un nodo sabe que tiene información fijándose sólo en su último fragmento. El contenido de los fragmentos que se envíen posteriormente no tiene mayor relevancia en las pruebas.

4.3.2. Programa para borrar la memoria

El programa para borrar la memoria flash es simplemente una modificación del programa usado para escribirla. Las interfaces que usa son exactamente las mismas. Lo único que lo diferencia es que el valor que pone en los campos *frgsTotales* y *ultFrg* es 0, indicando que el nodo no tiene nada de información. Además, el segundo volumen también se rellena con valores nulos.

De la misma forma que en el programa para escribir la memoria, existe una versión que modifica únicamente el primer volumen para ahorrar tiempo.

4.3.3. Programa para leer la memoria

Este programa es un poco más complicado que los anteriores, porque además de acceder a la memoria, hace uso de la interfaz de comunicación con el PC para mostrar su contenido.Las interfaces que usa son:

- *Boot*
- *Leds*
- *Mount*
- *ConfigStorage*
- *BlockRead*
- *SplitControl*
- *AMSend*
- *Packet*

El programa monta el volumen de configuración, copia su contenido a un paquete, y lo envía al puerto serie. Después hace lo mismo con todos y cada uno de los fragmentos del volumen de datos. Este programa se usa siempre junto con la aplicación Java *Listen* mencionada anteriormente, para poder visualizar los paquetes en pantalla.

En su primera versión, el programa tenía una constante booleana definida en su cabecera para mostrar únicamente el contenido del primer volumen o el del segundo, según el valor de la constante.

En una segunda versión se introdujo un retardo de tres segundos entre el envío del paquete con la información del primer volumen y el primer paquete de datos, para poder ver el contenido de ambos volúmenes sin tener que modificar la cabecera y reprogramar el nodo. Así daba tiempo a ver el primero, que es de información, antes de saturar la pantalla con el contenido del volumen de datos.

Cuando, tras muchas pruebas se comprobó que la escritura en el segundo volumen no presentaba ningún problema, se optó por mostrar únicamente la información de configuración.

Además de estos tres programas, se desarrollaron otras aplicaciones muy básicas con el objetivo de comprobar funcionalidades concretas de TinyOS. Tres de ellas se implementaron para probar puntos clave del algoritmo: la obtención de la calidad del enlace entre emisor y receptor, la obtención de la dirección origen de un paquete y el funcionamiento del mecanismo de ACKs.

Tras realizar multitud de pruebas iniciales con hasta tres MICAz ejecutando el algoritmo simultáneamente, se llegó a una versión estable del protocolo. Entonces se plantearon varios escenarios con el fin de medir en profundidad su eficiencia.

4.4. Descripción de los escenarios propuestos

En los escenarios de prueba propuestos hay **dos parámetros** que cambian:

1. **La topología**

   La posición de los nodos en la prueba es muy relevante, ya que habrá distintas calidades en de enlace, y se producirá una dispersión de la información de una forma distinta según cada configuración. Aunque el protocolo está pensado para soportar las incorporaciones y desconexiones de dispositivos propias de redes con nodos móviles, en cada uno de los escenarios estudiados se ha mantenido fija la posición de cada mota. Además ha sido necesario forzar qué nodos son vecinos, es decir, qué nodos tienen visión directa entre sí, ya que resulta complicado alejarlos hasta conseguir la topología deseada.

2. **El número de “semillas”**

   Este concepto ya se ha comentado anteriormente. Un nodo será una “semilla” cuando tenga la totalidad del bloque que se disemina. El número de semillas disminuye el tiempo de diseminación de forma exponencial, ya que cuantas más haya, más transmisiones paralelas habrá, y por tanto más nodos podrán recibir la información al mismo tiempo.

En todos los escenarios, independientemente del número inicial de semillas y de la posición de los nodos, el **parámetro a medir** es el mismo: **el tiempo que pasa desde que se inicia la prueba hasta que todos los nodos, sin excepción, tiene la totalidad del bloque de información**. Las mediciones se realizaron con ayuda de un cronómetro y a mano, ya que no es necesaria una precisión mayor que la de segundos. En ningún caso se inicia una prueba con nodos que tengan contenidos parciales, o sea, inicialmente, un nodo puede tener todo el fichero a distribuir o nada de él.

Las mediciones se llevaron a cabo hasta **cinco veces en cada escenario** y siempre de la misma forma: el cronómetro se arranca en el momento en que se enciende el último MICAz, que es la semilla para evitar que haya preferencia entre nodos vacíos. Inicialmente se encienden los nodos que no tienen información, ya que no habrá ningún tráfico en la red hasta que entre una semilla que comience a anunciar su información. Cuando hay más de una semilla, se encienden, igualmente, al mismo tiempo.

Cuando se arranca la semilla, comienza un proceso de intercambio de ofertas, solicitudes y fragmentos que hace que la información se propague por toda la red. En el momento en el que todos los nodos de la red estén haciendo ofertas (el led amarillo parpadea con un segundo de periodo) y no haya ninguna transmisión, todos los nodos habrán recibido el fichero completo. En ese momento se detiene el cronómetro y se registra el resultado obtenido.

En el protocolo existe una serie de **constantes** definidas en tiempo de compilación cuyo valor puede afectar a los resultados obtenidos en las pruebas:

- ***TIEMPO_OFERTA***: Es el tiempo mínimo que espera un nodo entre el envío de ofertas consecutivas. Su valor se fija en **un segundo**.

- ***TIEMPO_SOLICITUDES***: Es el tiempo que un nodo está escuchando solicitudes tras enviar una oferta. Su valor se fija en **un segundo**.

- ***VELOCIDAD_ENVIO***: Es el retardo que introduce un nodo entre el envío de fragmentos consecutivos. Su valor está fijado en **8 milisegundos** (el mínimo posible para que la transmisión sea viable).

- ***TIMEOUT***: Es el tiempo que espera un nodo que ha dejado de recibir fragmentos antes de considerar finalizada la transmisión. Fijado en **300 milisegundos**.

Es posible llevar a cabo pruebas con distintas combinaciones de estos valores, pero teniendo en cuenta que el valor mayor es de un segundo, las variaciones de los resultados obtenidos serían mínimas. Por eso y para no tener que realizar tantas pruebas (que implican dos programaciones por nodo) se optó por fijarlas en los valores indicados.

A continuación se describe con detalle cada uno de los escenarios propuestos.

4.4.1. Primer escenario: 2 nodos

Es el escenario más **simple**. En él, un nodo tiene todo el fichero y el otro no tiene ningún fragmento. Las pruebas se han llevado acabo dejando una separación de unos 5 centímetros entre los nodos. Este escenario ha sido el que ha estado presente durante la mayor parte de la fase de pruebas, ya que el primer hito durante la codificación del protocolo es conseguir que haya un intercambio de información entre dos nodos.

Una vez conseguida la comunicación entre parejas, incorporar más nodos no añade una complejidad excesiva.

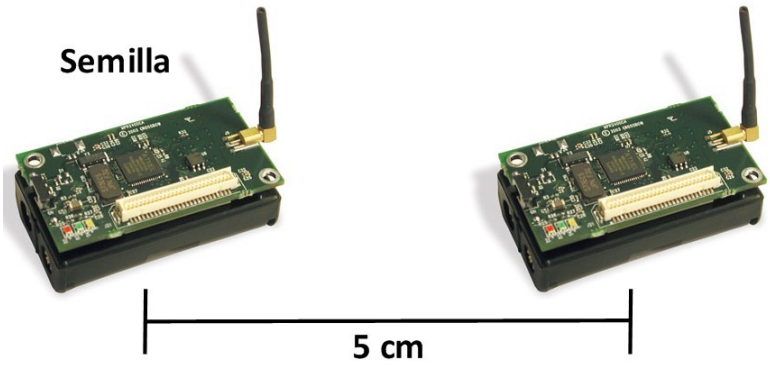


Figura 34: Escenario 1: dos nodos

4.4.1.1. Resultados

Los resultados de las 5 pruebas que se han llevado acabo con el protocolo totalmente acabado son los siguientes:

Número de prueba	Resultado (minutos)
1	02:09
2	02:09
3	02:09
4	02:09
5	02:09

Figura 35: Resultados de las pruebas del primer escenario

Media (minutos)	Varianza
02:09	0

Figura 36: Media y varianza de los resultados del primer escenario

Nótese que los resultados se mantienen constantes en todas las pruebas. Esto ocurre porque éste es un escenario totalmente **libre de colisiones**, ya que no existen transmisiones paralelas que puedan provocar que la información se distribuya de formas distintas, provocando variaciones en los resultados.

4.4.2. Segundo escenario: 4 nodos

Este escenario añade la complejidad de que **existen transferencias paralelas** de información. Los cuatro nodos se distribuyen formando una **matriz de 2x2** de forma que un nodo sólo puede "ver" al nodo situado en su misma fila o en su misma columna. Es decir, en ningún momento puede haber una conexión diagonal en la matriz. De nuevo sólo hay una semilla inicialmente y la distancia entre vecinos es la misma que en el apartado anterior (5 centímetros aproximadamente).

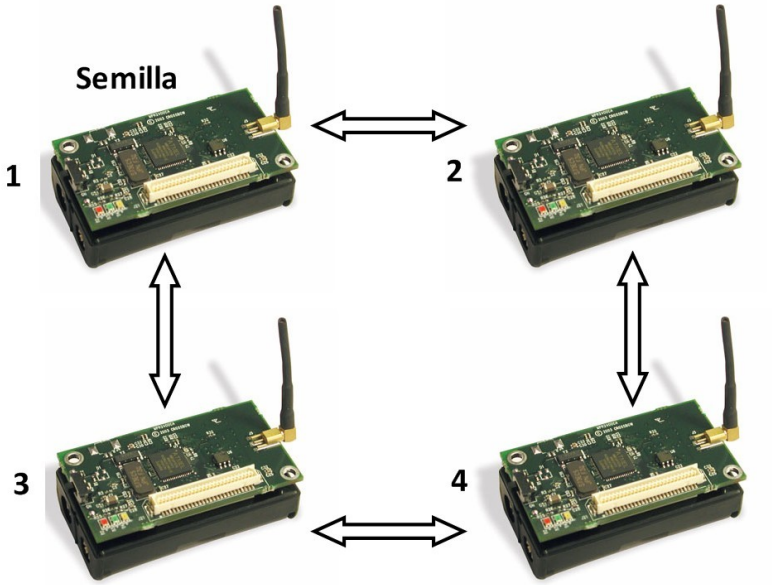


Figura 37: Escenario 2: cuatro nodos con visibilidad parcial

La figura 37 muestra la configuración del escenario y la asignación de direcciones que tiene los nodos. No puede haber comunicación entre los nodos 1 y 4 ni entre los nodos 2 y 3.

Evitar de forma real la visibilidad entre dichos nodos resulta muy complicado, es decir, no es viable buscar una distancia tal que el nodo 1 no vea al 4 pero que tenga una visibilidad total con los nodos 2 y 3. Por ello se optó por forzar esta topología mediante el uso de una función que, basándose en la dirección local del nodo y en la dirección origen de un mensaje recibido, un nodo

es “consciente” de la posición que ocupa en la matriz y de cuáles son sus vecinos. En realidad un MICAz sí que recibe, en todo momento, los mensajes broadcast de cualquier nodo de la prueba, sea o no vecino, pero si, observando su dirección se da cuenta de que no es vecino, simplemente ignora el paquete recibido.

4.4.2.1. Resultados

Los resultados obtenidos en las pruebas para el segundo escenario fueron los siguientes:

Número de prueba	Resultado (minutos)
1	04:22
2	04:21
3	04:21
4	04:22
5	04:21

Figura 38: Resultados de las pruebas del segundo escenario

Media (minutos)	Varianza
04:21,4	0,3

Figura 39: Media y varianza de los resultados del segundo escenario

Teniendo en cuenta los resultados obtenidos en las pruebas del primer escenario, en el que sólo había dos MICAz, resulta lógico esperar que los tiempos obtenidos en esta prueba fueran el doble de los de la primera, es decir, el tiempo de dos transmisiones: 2:08 + 2:08 = 4:16.

Inicialmente se arrancan los nodos 2, 3 y 4, que están vacíos y no generan ningún tráfico en la red. Cuando se enciende el nodo 1, que es la semilla, es cuando comienza la medición. Mediante el sistema de vecinos, los nodos 2 y 3 escucharán la oferta que haga la semilla y enviarán una solicitud a la misma. Ésta elegirá uno de los dos solicitantes para enviarle su información basándose, como ya se ha visto, en la calidad del enlace. Si los dos tienen el mismo valor de calidad, algo más que probable teniendo en cuenta la cercanía y equidistancia con los dos emisores de la solicitud, simplemente enviará los fragmentos al primero que haya enviado su mensaje.

En ese momento se inicia un intercambio de información idéntico al que se observa en el primer escenario: durante 2 minutos y 9 segundos el nodo elegido está recibiendo fragmentos del primer MICAz. Cuando haya pasado ese tiempo, habrá dos nodos con la totalidad de la información (los nodos 1 y 2 o los nodos 1 y 3). En esta situación el nodo 1 enviará una oferta que será recibida por los nodos 2 y 3 una vez más, pero en este caso sólo obtendrá respuesta del nodo que no haya sido elegido en la primera transmisión, por tanto comenzará a enviarle los fragmentos a éste.

Por su parte, el nodo que se haya convertido en semilla en la primera conexión hará una oferta que escucharán los nodos 1 y 4, y, evidentemente sólo tendrá respuesta de éste último, por lo que se iniciará una tercera **transmisión de información paralela** a la segunda. Si los nodos estuvieran lo suficientemente distanciados, ambas transmisiones se llevarían a cabo en 2 minutos y 9 segundos, de forma que en 4 minutos y 16 segundos debería haber acabado la prueba. El motivo de que la duración esté por encima de 4:20 minutos es que **ambas semillas están interfiriendo entre sí**, inundando el medio radio con fragmentos y generando colisiones que, aunque no provocan la finalización de las conexiones, sí **hacen que el tiempo sea algo mayor que el teórico esperado**.

4.4.3. Tercer escenario: 16 nodos

Se trata del escenario **más hostil** para las conexiones ya que, en un momento dado pueden darse hasta 8 transmisiones de información simultáneas que interfieren, en este caso muy notablemente, entre sí, llegando a provocar desconexiones temporales y **redistribuciones en las parejas** emisor-receptor.

De nuevos los 16 nodos se colocan formando una **matriz**, esta vez de 4x4 y con la misma distancia entre dispositivos contiguos en la misma fila y en la misma columna (5 centímetros aproximadamente).

El planteamiento inicial de esta prueba presentaba distintas combinaciones en la topología y en el número de semillas.

- **Topología 1: Visibilidad total entre nodos** (todos ven a todos), con tres variantes:
  - 1 semilla y 15 nodos vacíos
  - 2 semillas y 14 nodos vacíos
  - 4 semillas y 12 nodos vacíos

- **Topología 2: Visibilidad parcial**, es decir, entre vecinos, con el mismo concepto que en el escenario 2 (sólo existe comunicación directa entre nodos contiguos dentro de una misma fila o de una misma columna). También con tres variantes:
  - 1 semilla y 15 nodos vacíos
  - 2 semillas y 14 nodos vacíos
  - 4 semillas y 12 nodos vacíos

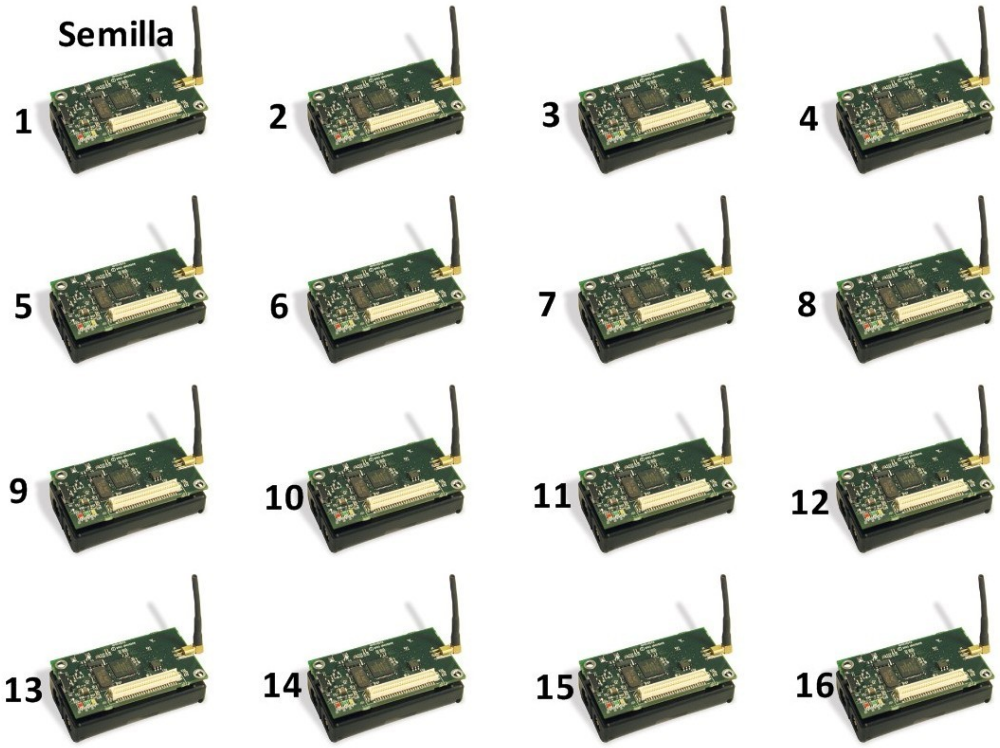
Sin embargo, pruebas preliminares en la segunda topología mostraron un comportamiento demasiado caótico en los nodos, que se veían continuamente interferidos por otras transmisiones de nodos supuestamente “lejanos”, es decir, las transmisiones que provocaban interferencias provenían de conexiones unicast que, en un caso real, sin necesidad de forzar los vecinos, no afectarían en ningún momento a otras transmisiones. Recordemos que los vecinos están simulados y que, en realidad son conscientes de los mensajes que hay en el medio, pero los ignoran.

A la vista de estos resultados, y aunque finalmente todos los nodos podrían llegar a convertirse en semillas, se optó por **realizar las pruebas únicamente para la primera topología**, en la cual pueden estar presentes las colisiones de la misma forma que en la segunda, pero el

resultado obtenido sería el de un caso real. Las mediciones obtenidas en la topología número 2 no habrían tenido ninguna utilidad, ya que el sistema de fijación de vecinos no representa la realidad.

4.4.3.1. Primera variante: 1 semilla

En la primera variante del tercer escenario se colocan 16 MICAz formando una matriz cuadrada de 4x4 de forma que cada uno de los nodos pueda recibir los mensajes de los 15 restantes, con una sóla semilla:



4.4.3.1.1. Resultados

Los resultados que se obtuvieron en la primera variante del tercer escenario fueron los siguientes:

Número de prueba	Resultado (minutos)
1	11:35
2	11:48
3	11:43
4	11:45
5	11:52

Figura 41: Resultados de las pruebas del tercer escenario (con una semilla)

Media (media)	Varianza
11:44,6	40,3

Figura 42: Media y varianza de los resultados del tercer escenario (con una semilla)

Inicialmente se encienden los 15 MICAz vacíos (del 2 al 16). A continuación se enciende el nodo con la dirección número uno, que es la semilla. Inmediatamente lanza su oferta al medio, que le llega al resto de nodos de la red. Como todos están vacíos, responderán a la oferta de la semilla al mismo tiempo.

Como ya se conoce, la semilla comparará todas las solicitudes recibidas durante un tiempo *TIEMPO_SOLICITUDES* (igual a un segundo). Cuando finalice ese tiempo, la semilla habrá seleccionado a uno de los 15 MICAz como destinatario de la información. Si los nodos hubieran estado bastante más distanciados entre sí, como podría ocurrir en una situación real, sería lógico pensar que la semilla escogería a uno de sus nodos más próximos como receptor de sus datos (nodos 2, 5 o 6), ya que tendrían una mejor calidad de enlace y las solicitudes llegarían las primeras.

En las pruebas del laboratorio, el nodo inicial elegido por la semilla para la transmisión ha sido más o menos aleatorio, teniendo en cuenta que todos se encuentran muy próximos entre sí, y que sólo difieren en la distancia a la semilla en varios centímetros.

Después de elegir su receptor, la semilla comienza a transmitirle fragmentos a uno de los nodos vacíos. La conexión se mantiene igual que ocurría en el primer escenario, sin ningún tipo de colisión o pérdida de paquetes, por lo cual, pasados 2 minutos y 9 segundos habrá dos semillas en el escenario.

En el minuto 2:09 comienzan dos transmisiones paralelas exactamente como ocurría en el segundo escenario con cuatro nodos. Los dos nodos que son semilla eligen a otros dos entre los solicitantes. En este aspecto la transmisión de la semilla número uno podría comenzar hasta dos segundos antes que la nueva semilla. Esto ocurre porque, cuando la primera semilla finaliza su transmisión a la segunda, inmediatamente lanza una oferta al medio, que será respondida por los 14 nodos vacíos que quedan. Mientras ésto ocurre, la nueva semilla debe esperar un tiempo prudencia *TIMEOUT* para asegurarse de que no va a recibir más fragmentos. Cuando ese tiempo pasa, es cuando, como ya se ha descrito, actualiza el valor de sus datos de configuración en el primer volumen de la memoria flash.

Tras finalizar la escritura, la nueva semilla lanzará una nueva oferta al medio, pero ésta no será respondida por ningún nodo inmediatamente, ya que todos acaban de responder a la oferta del MICAz número 1, por lo que tienen que esperar rigurosamente el tiempo establecido para conseguir la conexión entre no más de dos nodos. Éste tiempo está establecido en las pruebas con un valor de dos segundos (2***TIEMPO_SOLICITUDES*).

La segunda transmisión de información se iniciará como mucho dos segundos más tarde que la primera. De nuevo, igual que ocurría en la segunda fase del segundo escenario, habrá dos conexiones paralelas, que provocarán algunas colisiones entre sí, pero que se mantendrán durante

todo el envío.

En un intervalo de entre 4:20 minutos y 4:25 habrá un escenario nuevo: cuatro nodos serán semillas, y 12 estarán vacíos. Al igual que ocurre con dos semillas, se inician en cuestión de segundos cuatro nuevas conexiones con nodos vacíos. Las cuatro conexiones que hay simultáneamente inundando el medio a partir de este momento provocan problemas de pérdidas e interferencias mucho más notables que en el caso anterior. Por este motivo, las 4 parejas que se forman no son muy estables, pudiendo, en un momento dado, haber tantas colisiones que uno de los receptores deje de recibir información durante más de su *TIMEOUT* (300 milisegundos) y/o que se pierdan *MAX_INTENTOS* paquetes consecutivos provocando que receptor y/o emisor den por finalizada la conexión, y quedándose el receptor con un contenido parcial del archivo.

En este momento, los dos nodos que acaban de "romper" su conexión inyectan dos nuevas ofertas, y puede haber cinco conexiones simultáneas pese a no haber cinco semillas (1 de los transmisores tiene un contenido parcial). A partir de ese punto se inicia un proceso totalmente impredecible de conexiones y desconexiones, alternando los nodos su papel de emisor-receptor continuamente.

A pesar de todas las interferencias que se produzcan, la información se distribuye por la red poco a poco pero con éxito, en intercambios parciales, hasta que los 16 nodos tienen todo el bloque de datos.

4.4.3.2. Segunda variante: 2 semillas

La segunda variante del tercer escenario parte de dos semillas inicialmente, en lugar de una. La segunda semilla se sitúa en esta ocasión en el nodo con la dirección 4, como muestra la figura 43.

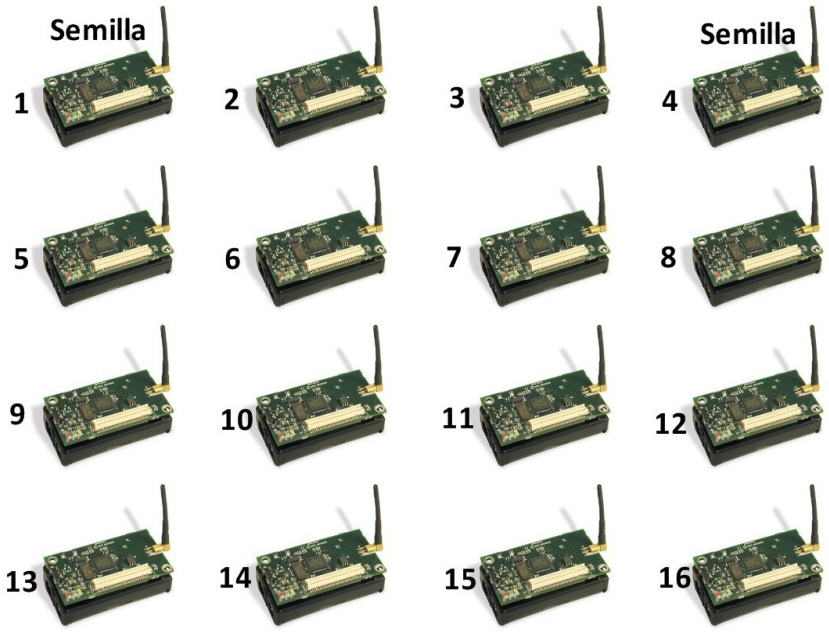


Figura 43: Topología del escenario 3 (con dos semillas)

Esta prueba es muy parecida a la de una semilla, ya que lo único que cambia es que no existe la primera conexión necesaria para pasar a tener dos semillas en lugar de una.

Por ello se puede esperar que los resultados varíen en un rango con la misma amplitud que las mediciones para una semilla, pero 2:09 minutos por debajo, que es el tiempo que hacía falta para tener dos semillas en la primera variante del escenario.

4.4.3.2.1. Resultados

Las siguientes tablas muestran los resultados obtenidos:

Número de prueba	Resultado (minutos)
1	09:52
2	09:28
3	09:43
4	09:36
5	09:33

Figura 44: Resultados del tercer escenario (con dos semillas)

Media (minutos)	Varianza
09:38,4	87,3

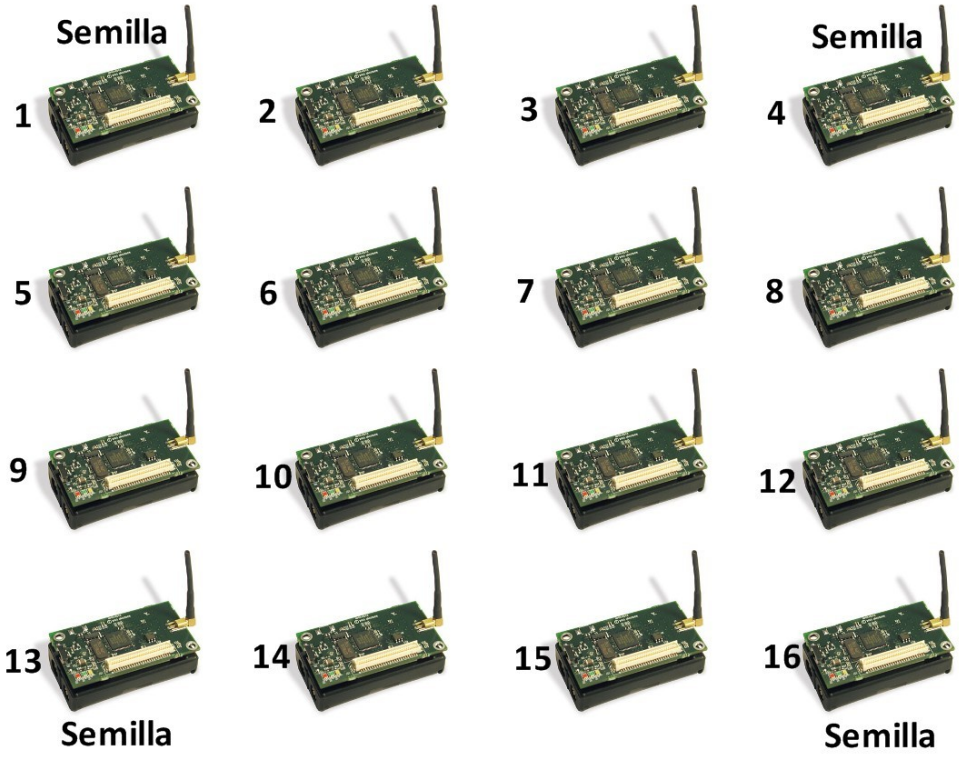
Figura 45: Media y varianza de los resultados del tercer escenario (con dos semillas)

El procedimiento en esta prueba es el mismo que en las anteriores. Se encienden primero los 14 nodos vacíos y posteriormente se encienden las dos semillas a la vez. A partir de ahí, la situación es la misma que en el escenario anterior: dos transmisiones paralelas sin incidentes y una posterior diseminación impredecible de datos.

4.4.3.3. Tercera variante: 4 semillas

Al igual que ocurre en las pruebas anteriores, esta topología viene a ser un subconjunto de las dos primeras variantes, ya que en ambas llega un momento en el que existen cuatro semillas y 12 nodos vacíos, que es el comienzo de este escenario.

En esta ocasión las semillas están situadas en los vértices de la matriz de nodos, es decir, en los nodos 1, 4, 13 y 16.



De nuevo la situación es la misma que la que se daba en las pruebas anteriores. Es posible que cada una de los cuatro nodos semilla se asocien con un nodo y consigan enviar el archivo completamente, llegando a una situación donde la mitad de los MICAz tienen todo el archivo.

Sin embargo, la situación que más se ha observado en las pruebas ha sido la distribución parcial del archivo por la red, con las consiguientes reasociaciones entre nodos. A pesar de ello, los resultados obtenidos se acercan a los valores teóricos que cabía esperar, es decir, el tiempo de la prueba con dos semillas, pero quitando el intervalo de tiempo que se ahorra en que esas dos pasen a duplicarse, que es la situación inicial de la tercera variante del tercer escenario.

4.4.3.3.1. Resultados

Las siguientes tablas muestran los valores obtenidos en las mediciones de la tercera variante del tercer escenario:

Número de prueba	Resultado (minutos)
1	07:13
2	07:28
3	07:21
4	07:22
5	07:19

Figura 47: Resultados de las pruebas del tercer escenario (con cuatro semillas)

Media (minutos)	Varianza
07:20,6	29,3

Figura 48: Media y varianza de los resultados del tercer escenario (con cuatro semillas)

4.4.4. Cuarto escenario: 16 nodos en línea

La topología del último escenario observado en las pruebas está formada de nuevo por 16 nodos, pero dispuestos de una forma distinta a las anteriores: **en línea**.

Para esta prueba también **se han forzado los vecinos**, de forma que un MICAz de la fila sólo "ve" al nodo que tiene inmediatamente delante o detrás de él. Para ello se han programado los dispositivos con direcciones de memoria consecutivas según su posición en la línea y se ha usado una función equivalente a la que se vio en el segundo escenario. Así, un nodo sabe si un mensaje recibido es o no de un vecino observando la dirección origen del mismo, de forma que si la dirección es igual a la local del nodo ± 1, se tratará de un nodo vecino.

Las pruebas se han realizado con **una única semilla** en todas las ocasiones, situada en el primer nodo de la fila, para poder observar cuánto tarda la propagación lineal de los datos, sin que haya saltos ni bifurcaciones de la información en la red.

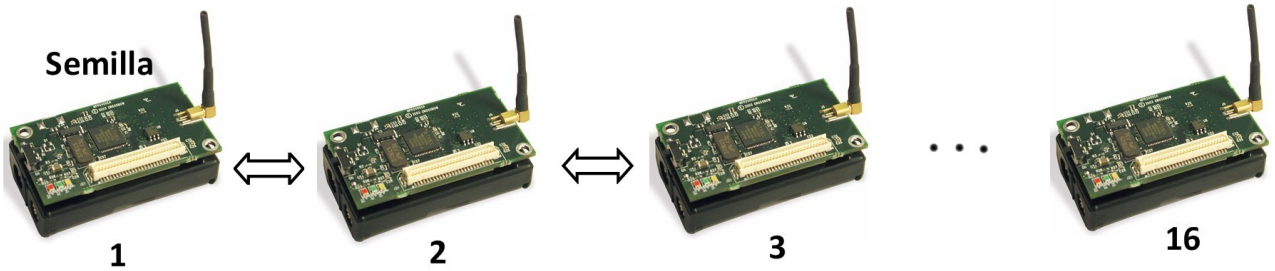


Figura 49: Topología del escenario 4

Las flechas de la figura 49 muestran la visibilidad entre nodos. El número uno sólo se puede comunicar con el número 2, éste puede con el 1 y con el 3, el 3 puede con el 2 y con el 4, y así sucesivamente.

Con las restricciones presentes en este escenario, la información se va a distribuir mediante **conexiones independientes y separadas en el tiempo**, es decir, el primer nodo hará una oferta que sólo será escuchada por el nodo 2, que le solicitará el primer fragmento. Cuando pasen 2:09 minutos, el segundo nodo pasará a ser semilla, entonces lanzará una oferta respondida únicamente por el nodo tres, que pasará a ser semilla 2:09 minutos más tarde, etc.

De esa forma se irá propagando la información hasta llegar al último nodo, momento en el que finalizará la prueba.

4.4.4.1. Resultados

Éstos son los resultados obtenidos en las cinco pruebas realizadas en el cuarto escenario:

Número de prueba	Resultado (minutos)
1	32:12
2	32:12
3	32:12
4	32:13
5	32:11

Figura 50: Resultados de las pruebas del cuarto escenario

Media (minutos)	Varianza
32:12	0,5

Figura 51: Media y varianza de los resultados del cuarto escenario

Las tabla de resultados de la figura 50 muestra unos valores de tiempo totalmente acordes con los valores teóricos que se podrían esperar.

Teniendo en cuenta que, en un entorno libre de colisiones (como es el caso), un MICAz tarda exactamente 2 minutos y 9 segundos en transmitir los 400 Kbytes del fichero, y que hay 15 transmisiones no simultáneas, el resultado esperado debería rondar los 32 minutos y 15 segundos.

Los resultados empíricos son un par de segundos más rápidos que los teóricos, pero se trata de variaciones insignificantes que pueden deberse a factores como la velocidad de procesamiento del MICAz, o al hecho de que los dispositivos no tengan que ejecutar la secuencia inicial de arranque (montar el volumen de configuración, leerlo, etc.).

5. Conclusiones

Las redes de sensores constituyen una tecnología relativamente nueva pero con un gran potencial. La posibilidad de desarrollar aplicaciones distribuidas con un coste tan reducido y con características como su alta autonomía y su posible utilización en una abanico tan amplio de sectores de investigación hace que sea una de las tecnologías del momento.

Sin embargo, la **falta de estándares** en las WSNs imposibilitan la **interoperabilidad** entre redes, la integración entre componentes, etc. **No existe una plataforma hardware común** en las redes inalámbricas de sensores y tampoco una **estandarización definitiva del sistema operativo** de sus nodo. Estas diferencias entre implementaciones y la inexistencia de acuerdos globales hacen que las WSNs sigan siendo hoy en día una tecnología en desarrollo.

En España todavía son pocas las empresas que se atreven con esta tecnología y las universidades que la investigan (Universidad Politécnica de Cartagena, Universidad de Castilla La Mancha, Universidad Politécnica de Cataluña, Universidad de Valencia y la Universidad de Málaga, principalmente).

En el caso particular de las redes vehiculares es necesario salir fuera de España, a países como Alemania, para encontrar proyectos sólidos en este campo, por ejemplo *NOW* (*Network On Wheels*).

Alcanzar un acuerdo entre fabricantes, tanto de vehículos como de los dispositivos que incorporaran, haría posible la implantación de un protocolo de diseminación, como el que aquí se estudia, en una red vehicular.

5.1. Limitaciones del protocolo de diseminación en una red vehicular y posibles mejoras

Uno de los aspectos clave en el éxito de la implantación de un protocolo de estas características pasa por el **tamaño del bloque de información** que se disemina.

A la vista de los resultados obtenidos en los distintos escenarios parece inviable llevar a cabo el intercambio de información de una forma eficaz, porque en condiciones óptimas para la transmisión, el tiempo mínimo es de 2:09 minutos, un valor demasiado alto si se tiene en cuenta que los coches están continuamente en movimiento y a distancias del orden de decenas de metros.

Sin embargo, en una red vehicular, la información a intercambiar no tiene que ser necesariamente de cientos de kilobytes. Si existiera una serie de **mensajes predefinido**s para la información durante la circulación, de forma que todos los vehículos conocieran dichos mensajes (accidentes, calzada en obras, limitación de velocidad, información meteorológica, etc.) sería posible intercambiar esa información en unos cientos de bytes, con no más de dos o tres paquetes de datos, simplemente indicando el código del mensaje. Este intercambio se realizaría de forma

casi instantánea, en cuestión de segundos, y permitiría a los vehículos conocer datos muy precisos sobre el estado de la calzada, adelantándose a posibles situaciones adversas.

Esa es una de las **ventajas** del protocolo propuesto, que **se puede adaptar para volúmenes muy bajos de datos**, obteniendo unos resultados muy positivos en cuanto al tiempo de distribución. Sobre todo en espacios urbanos, en los que el tiempo de visión entre emisor y receptor sería bastante alto.

No obstante, si el bloque de información que se quisiera difundir fuera tan grande como el tratado en las pruebas, se podría considerar un cambio de hardware.

Como ya se ha visto, intercambiar bloques de información del orden de cientos de kilobytes resulta muy difícil en redes tan dinámicas como las VANETs con la tecnología usada. El MICAz, y en concreto el estándar ZigBee, están pensados para asegurar el intercambio de pequeños fragmentos de información con tasas de transmisión más o menos bajas. En una red vehicular es necesario que la transmisión se produzca de la manera más rápida posible, ya que sirve para alertar a los conductores sobre peligros cercanos.

Por ello el tiempo que se tarda en intercambiar grandes bloques de información con la tecnología empleada hace imposible la implantación de este protocolo en una red vehicular. La mejor opción para poder llevarla a cabo pasa por usar bloques de información mucho menores, conteniendo únicamente información como el tipo de situación que se da en la carretera y algún parámetro adicional (kilómetros hasta el lugar en cuestión, máxima velocidad permitida, etc.).

Otro de los aspectos a tratar sería una mejora en el **control de la diseminación**. El protocolo desarrollado está pensado para distribuir la información a toda costa, independientemente de la información que se transmita. En el caso real de una red vehicular podría plantearse enviar la información (o mejor dicho, solicitarla) **sólo si es útil** para el vehículo receptor. Por ejemplo, podría ser útil transmitir la información sólo para los vehículos de un sentido de la circulación. En este aspecto se podría dotar a los nodos con **placas de sensores** como los de la figura 2 (página 5).

Las placas de sensores incluyen acelerómetros y características más sofisticadas como sistema de posicionamiento GPS, de forma que un coche puede obtener la velocidad y la dirección en la que se mueve, pudiendo incorporar dicha información al mensaje de oferta, de forma que los nodos receptores podrían solicitar el mensaje sólo si el emisor va en una dirección concreta, o descartar la oferta si la velocidad es excesiva como para que se complete el envío.

Como ya es conocido, otra de las características del protocolo planteado es el **envío secuencial de los fragmentos** de información. Esta opción tiene la ventaja de simplificar mucho el planteamiento y la codificación e implantación del protocolo. Sin embargo, tiene como principal inconveniente que los receptores podrían quedar con **contenidos parciales** si la semilla abandonara su campo de visión. Es decir, mientras exista una semilla, los nodos podrían recibir partes secuenciales del fichero. Sin embargo, en el momento en que la semilla abandone el escenario, podría distribuirse la información entre nodos con contenidos parciales, pero no sería posible que alcanzaran la totalidad del fichero, simplemente todos tendrían el mismo contenido que aquel que más tiempo hubiera estado conectado con la semilla.

Una posible mejora en este aspecto sería la de enviar los fragmentos siguiendo un criterio distinto. Así, si la semilla abandonara el escenario, el resto de nodos quedarían con contenidos parciales, pero distintos, pudiendo llegar a poner en común el fichero completo. Implementar este mecanismo implicaría que los nodos manejasen e intercambiasen información sobre qué partes tienen o no, es decir, no bastaría con indicar, como ocurre ahora, cuál es el último fragmento recibido, sino que sería necesario un **control individualizado de cada fragmento** recibido, algo bastante complicado si se tiene en cuenta que, para un bloque de 400 Kbytes, hay 6400 fragmentos.

Por último, el protocolo desarrollado **no incorpora identificadores del bloque de información** que se disemina porque sólo existe uno. En la situación real de una red vehicular habría **distintos bloques** (o tipos) de datos a transmitir, haciendo necesaria la **incorporación de un campo adicional** en los tres tipos de mensaje intercambiados. Este campo identificaría el bloque de información que se ofrece, que se solicita o al que pertenece el fragmento que se envía.

En resumen, el protocolo desarrollado ofrece un servicio de diseminación de información eficaz y, aunque no en todos los escenarios, eficiente, y su aplicación en redes vehiculares pasa por un tamaño reducido de la información a distribuir, pudiendo optimizarse con algún mecanismo adicional como la incorporación de sensores a los nodos de la red.

REFERENCIAS

[1] Wikipedia. *Wireless Sensor Network*. http://en.wikipedia.org/wiki/Wireless_sensor_network

[2] Wikipedia. *Sensor node*. http://en.wikipedia.org/wiki/Sensor_node

[3] Wikipedia. *Ad-Hoc network*. http://en.wikipedia.org/wiki/Ad-hoc_network

[4] Revista Technolgy Review. *10 emerging technologies that will change the world*. 2003.

[5] Escolar, S., Carretero, J., García, F., Isaila, F., Fernández, J.. *Acabando con los desarrollos ad-hoc en Wireless Sensor Networks*. Albacete: 2006. Universidad Carlos III de Madrid.

[6] Escolar, M. Soledad. *Wireless Sensor Networks: Estado del arte e investigación*. 2006. Universidad Carlos III de Madrid.

[7] Wikipedia. *Vehicular Ad-hoc Network*. http://en.wikipedia.org/wiki/VANET

[8] Wikipedia. *Mobile Ad-hoc Network*. http://en.wikipedia.org/wiki/MANET

[9] Página oficial de Crossbow. http://www.xbow.com

[10] Crossbow Technologies. *MICAz Datasheet*.

[11] Fernández, R., Ordieres, J., Martínez, F.J., González, A., Alba, F., Lostado, R., Pernía, A.V.. *Redes inalámbricas de sensores: teoría y aplicación práctica*. Logroño: 2009. Universidad de La Rioja.

[12] Página oficial de TinyOS. http://www.tinyos.net

[13] Universidad de Berkeley. *NesC: A programming language for deeply networked systems*. 2004.

[14] Página web de XubunTOS. http://toilers.mines.edu/Public/XubunTOS

[15] Levis, P., Tolle, G.. *TEP 118: Disemination of small values*.

[16] Levis, P., Patel, N., Culler, D., Shenker, S.. *Trickle: A Self-Regulating Algorithm for Code Maintenance and Propagation in Wireless Sensor Networks*. 2004.

[17] Lin, K., Levis, P.. *Data discovery and dissemination with DIP.* 2008.

[18] Hui, J., Culler, D.. *The dynamic behavior of a data dissemintion protocol for network programming at scale*. Universidad de Berkeley.

[19] Ni, S.-Y. , Tseng, Y.-C., Chen, Y.-S., Sheu, J.-P.. *The broadcast storm problem in a mobile ad hoc network*. Universidad de Taiwan.

[20] Fonseca, R., Gnawali, O., Jamieson, K., Levis, P.. *TEP 119: Collection*. 2006. Universidad de Berkeley.

[21] Fonseca, R., Gnawali, O., Jamieson, K., Kim, S., Levis, P., Woo, A.. *TEP 123: The collection tree protocol*. 2006-2007. Universidad de Berkeley.

[22] Gnawali, Omprakash. *TEP 124: The link estimation exchange protocol*. Universidad de Berkeley.

[23] *TYMO Protocol*. http://docs.tinyos.net/index.php/Tymo

[24] Página web de *Wath-Over*. http://www.watchover-eu.org

[25] Página web de *WILLWARN*. http://www.prevent-ip.org

[26] Página web de *CitySense*. http://www.citysense.net

[27] Torrecillas Rodríguez, Mario. *Protocolo para la diseminación de información en una red vehicular mediante una estrategia broadcast*. 2009. Universidad Politécnica de Cartagena.

Agradecimientos

Este trabajo ha sido realizado gracias a la subvención concedida por el Ministerio de Educación y Ciencia con el proyecto DEP2006-56158-C03—03/EQUI.