Proyecto de Fin de Carrera

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

Escuela Técnica Superior de Ingeniería de Telecomunicación

Autor: Mario Torrecillas Rodríguez
Director: Javier Vales Alonso
20/09/2009

Autor	Mario Torrecillas Rodríguez
E-mail del autor	mariotr87@gmail.com
Director (es)	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 broadcast
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. Para conseguir ésto, los vehículos anunciarán sus partes disponibles, y de forma unicast los interesados solicitarán la parte deseada; posteriormente el receptor de la solicitud decidirá la parte a enviar y la distribuirá a todos los vehículos que estén a su alcance (broadcast).

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

Contenido

Índice de figuras

Índice de tablas

1.Introducción

1.1 Motivo del estudio

Actualmente las redes inalámbricas están en pleno auge, tanto en el terreno personal como el profesional, y ejemplo de ello son las diversas tecnologías de este tipo que han ido surgiendo a lo largo de los últimos años: redes *Bluetooth*, *Wi-Fi*, *WiMax* y, más recientemente, el caso que nos ocupa, *ZigBee*.

Probablemente las tecnologías inalámbricas que son el objetivo de los mayores grupos de investigación y desarrollo en estos momentos son las espontáneas, aquellas que no dependen de ningún tipo de infraestructura y por tanto sus nodos son capaces de comunicarse entre sí por sí solos.

Un ejemplo de este tipo de redes son las redes *VANET* o *Vehicular Ad-Hoc Network*, que como su propio nombre indica se trata de una red ad-hoc donde sus nodos se corresponden con vehículos; esta comunicación puede tener distintos fines, desde simple ocio, hasta el confort de los tripulantes del vehículo, por ejemplo, avisando de un accidente ocurrido a los vehículos vecinos, o transmitiendo un paquete de información meteorológica: las posibilidades son sumamente grandes.

El motivo de este trabajo es, por tanto, el interés generado alrededor de todo lo que concierne a estas redes y todas sus posibles aplicaciones, que podrían formar parte del día a día de las personas en un futuro próximo.

A continuación se definen algunos de los conceptos.

1.2 Definiciones

1.2.1 Redes Ad-Hoc

Las redes *ad-hoc* son aquellas que están formadas por una serie de dispositivos que se comunican entre sí sin ningún tipo de infraestructura que los controle (ver Figura 1). Una definición más formal de este concepto podría ser la de una red compuesta por terminales fijos y móviles o bien sólo móviles, que no dependan de una infraestructura preexistente, desplegándose de una forma espontánea en un entorno inalámbrico (1). Es importante mencionar el hecho de que una red *ad-hoc* no implica conexión sin cables, pudiendo existir éstos.

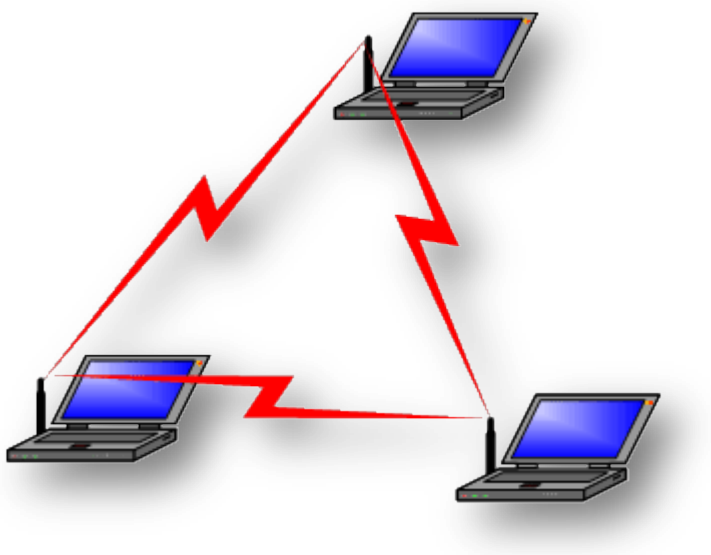

Figura 1: Ilustración de una red Ad-Hoc

Como se ha mencionado anteriormente, no es necesario tener una infraestructura (por ejemplo un *router*) o un administrador del sistema debido a que este tipo de redes es capaz de autoconfigurarse (2); esto quiere decir que pueden adaptar su funcionamiento a una posible topología cambiante.

En el supuesto escenario *ad-hoc* mostrado algunos de los nodos actuarán como *routers* permitiendo la conexión y comunicación entre dos dispositivos que no se encuentran en el radio de cobertura del otro (no es necesario que dos nodos estén conectados directamente para que puedan comunicarse entre sí).
Este fenómeno se denomina red inalámbrica *multisalto*, donde puede haber numerosos nodos cubriendo una zona de cobertura mucho mayor gracias a la interconexión entre ellos.

Este tipo de redes presenta topologías cambiantes a lo largo del tiempo (los nodos se mueven en el espacio sin seguir un patrón concreto.
El principal problema de las redes inalámbricas como ésta es la gran inestabilidad e imprecisión que ofrece el medio radio, que presenta distintos fenómenos físicos que dificultan la comunicación (como puede ser el *fading*, el *jitter*, la *atenuación* o el *efecto multicamino*) y por tanto obliga a establecer ciertas restricciones en la transmisión de información.

Las limitaciones del medio radio serán un fundamento determinante en el funcionamiento de las redes *ad-hoc* y también en de los protocolos que intervienen (comunicación, encaminamiento, ofrecer *QoS*, etc.), lo que hará comprender su comportamiento y estudio.

En cualquier caso habría que decir que se trata de un campo que actualmente se encuentra en plena investigación y desarrollo, debido a la multitud de posibles aplicaciones que ofrecería esta tecnología (áreas remotas, comunicación en rescates o

catástrofes, comunicaciones entre vehículos para prevenir de accidentes ocurridos…etc.) (1).

1.2.2 Redes VANET

Una *VANET* o *Vehicular Ad-Hoc Network*, como su propio nombre indica, se trata de una red *ad-hoc* donde sus nodos se corresponderían a vehículos (coches, camiones, autobuses …etc.); en este caso, cabría la posibilidad de que dichos nodos formaran la red en pleno movimiento (por ejemplo mientras se circula por una autopista), por tanto, nodos que se mueven de forma arbitraria y que se comunican entre ellos (*vehicle-to-vehicle*), pudiendo tener también un equipo fijo próximo que formara parte de la red y que también dotará a dicha red de una conexión hacia Internet por ejemplo (*vehicle-toroadside* o *vehicle-to-enviroment*) (3), al igual que hoy acceden nuestros terminales móviles a Internet a través de *GPRS* o *UMTS* (4) (ver la Figura 2).

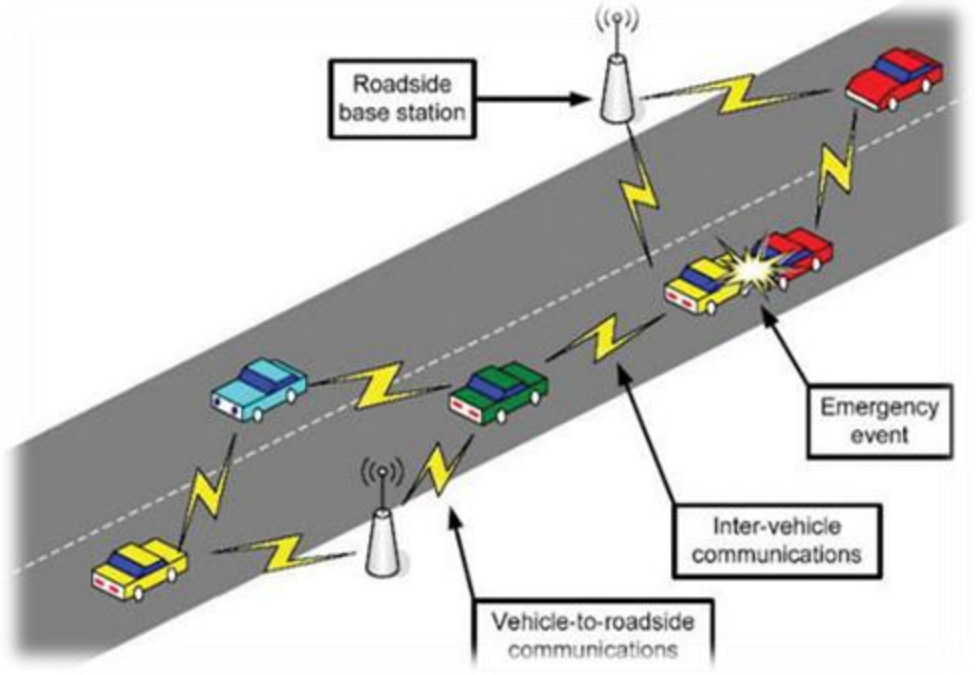

Figura 2: Ilustración de una VANET

En cuanto a nomenclatura, cabe decir que una *VANET* es un tipo de *MANET* (*Mobile Ad-Hoc Network*), es decir, una red *ad-hoc* móvil, por las razones que se han expuesto anteriormente; aunque cabe diferenciar que *MANET* describe sobre todo un campo de investigación académico, mientras que el término *VANET* está más enfocado a una aplicación en concreto de éstas (5).

Actualmente todo lo que envuelve el tema de las *VANETS* está en pleno desarrollo e investigación, de hecho, existen varios grupos de trabajo, tanto por parte de las universidades y los gobiernos, como de la industria, que investigan en este campo debido a la multitud de posibles aplicaciones que podría suponer su utilización; algunos de los consorcios son por ejemplo el VSC (USA), C2CCC (Europa), Internet ITS

(Japón), Sigmobile (USA) …etc (3) y el propio IEEE, que conjuntamente con el European Car-to-Car Communication Introducción a las VANETS 5 Consortium, está desarrollando el protocolo IEEE 802.11p, lo que se podría ver como una adaptación del propio IEEE 802.11, que actualmente conocemos, a estos escenarios; equivalente al estándar *DSRC o Dedicated Short Range Communication*, utilizado en Estados Unidos (4).

1.2.3 ZigBee

ZigBee es el nombre de la especificación de un conjunto de protocolos de alto nivel de comunicación inalámbrica para su utilización con radios digitales de bajo consumo, basada en el estándar IEEE 802.15.4 de redes inalámbricas de área personal (*wireless personal area network*, *WPAN*) (6). Su objetivo son las aplicaciones que requieren comunicaciones seguras con baja tasa de envío de datos y maximización de la vida útil de sus baterías.

La principal razón del éxito de este tipo de redes son diversas características que lo diferencian de otras tecnologías:

- Su bajo consumo.
- Su topología de red en malla.
- Su fácil integración (se pueden fabricar nodos con muy poca electrónica).

1.3 Objetivo del estudio

En este Proyecto de Fin de Carrera se abordará el problema de la comunicación *Ad-Hoc* entre este tipo de dispositivos móviles, desde el protocolo de comunicación hasta el programa a implementar en los dispositivos encargados de la transmisión de la información; dicha información será un conjunto de bytes que se tratará de propagar desde uno o varios vehículos portadores de los datos (*semilla(s)*) al resto de vehículos vecinos que formen la red.
Debido a que hablamos de redes móviles, dicha red es variable con el tiempo, sin una topología estable: los vehículos se desplazan, y por tanto unos salen del alcance de otros mientras nuevos entran; esto no debe suponer un problema para el protocolo, que debe estar preparado para aceptar estos cambios.
El objetivo de este proyecto es por tanto, en primer lugar, demostrar la posibilidad del uso de este tipo de redes para la transferencia de contenidos de un tamaño relativamente grande y, en segundo lugar, la viabilidad de implementación de un protocolo de este tipo para un posible uso real en un futuro próximo.

Para llevar a cabo este estudio se realizarán varios pasos de forma progresiva:

- **Documentación:** Familiarización con el sistema operativo *TinyOS* (versión 2) y aprendizaje del lenguaje *NesC* mediante tutoriales y documentación de la web de *TinyOs*.
- **Desarrollo del protocolo:** Creación del algoritmo que seguirán los vehículos para comunicarse entre sí (aceptar/rechazar información, anunciarla, etc.).

- **Implementación del protocolo:** Codificación del protocolo en *NesC* y programación de los dispositivos empleados para la comunicación (*MICAz*).
- **Pruebas:** Fase de pruebas para ver posibles fallos del programa así como su eficiencia en distintos escenarios.

1.4 Propuesta de solución

La solución que se propone en este estudio es la implementación de un protocolo de tipo *broadcast*, donde la información llega a todos los vehículos dentro del alcance del emisor al mismo tiempo, y éstos son los que deciden la información que les interesa y la que no.
Debido a que la longitud de la información es relativamente grande (pudiendo llegar a ocupar varios cientos de kilobytes) se dividirá la información en bloques o partes de tamaño fijo y éstos se transmitirán al medio en un orden no establecido, dependiendo de las necesidades de los vecinos del entorno; esto quiere decir que son los vecinos los que solicitan las partes necesarias (entre las anunciadas por el emisor) y, el portador de información o *semilla* envía aquella que ha sido más solicitada por dichos vecinos.
De la división del archivo en partes se obtienen dos ventajas claras: la primera de ellas, que disminuye el tiempo en el que dos vehículos deben estar en el rango de visibilidad, al sólo necesitar enviar un subconjunto de la información; la segunda, que se tiene la libertad de enviar dicha información en un orden no secuencial, lo que da la posibilidad de que, en un escenario donde ningún vehículo posee toda la información (no existe una *semilla*) todos terminen con ésta siempre y cuando entre todos puedan completarla.

1.5 Posibles problemas

Existen ciertas limitaciones (tanto a nivel software como a nivel hardware) y posibles impedimentos que podremos encontrar más adelante.
En primer lugar la implementación del protocolo así como ciertos aspectos de su diseño dependen de algunos factores hardware de los dispositivos que se van a emplear:

- Cantidad de memoria *RAM*
- Cantidad de memoria *FLASH*
- Velocidad del procesador
- Velocidad a la que los dispositivos son capaces de enviar/recibir varios paquetes de datos consecutivos.
- Tiempo que el dispositivo tarda en realizar operaciones de lectura/escritura sobre la memoria.

Por ejemplo, limitaciones en cuanto a la velocidad de transmisión/escritura/lectura (ya que los dispositivos deben almacenar la información a una velocidad superior a la que lee y envía el portador de información) y problemas derivados de la variabilidad de las redes: tiempo de visibilidad entre dos vehículos insuficiente para transmitir una parte,

además de la necesidad de elección de los valores adecuados para cada parámetro variable del protocolo.
Otra posible limitación que habrá que abordar es la cantidad de bytes que es posible enviar en un único paquete de información.
Por último, habrá que disminuir las colisiones entre los paquetes que envían los distintos nodos en un momento determinado para reducir en la medida de lo posible el tiempo total de transmisión del contenido.

1.6 Entorno de trabajo

Para el desarrollo del estudio se necesitan dispositivos móviles que se encarguen de la propagación de la información así como un sistema software que lo gestione y un lenguaje de programación que permita la implementación del protocolo; se explican cada uno de ellos en los siguientes subapartados.

1.6.1 MICAz

Los *MICAz* son los dispositivos que se colocarán en cada uno de los vehículos para que estos puedan comunicarse entre sí; se trata de pequeñas placas destinadas a redes de sensores para la comunicación a baja potencia, y siguen el estándar *IEEE/ZigBee 802.15.4* (7) (ver la Figura 3).

Figura 3: MICAz

Dichos dispositivos, además, permiten la conexión directa a placas de sensores, que permiten la medición de distintas variables del entorno, por ejemplo la temperatura, la luz, la humedad, así como otros parámetros; no son objetivo de este estudio, donde únicamente se pretende la transmisión de información, sin importar el contenido de ésta.

A continuación algunas de las características de estos dispositivos (8):

- Plataforma *wireless* para redes de sensores de baja potencia.
- Certificado *FCC*.
- Velocidad de transmisión de 250 Kbps.

- Duración de batería de varios años.
- Comunicaciones sin cables donde cada nodo puede actuar como *router*.

Debido a su reducido tamaño y consumo son ideales para este tipo de redes.

1.6.2 TinyOS (versión 2.x)

TinyOS es un sistema operativo de código abierto diseñado especialmente para las redes de sensores inalámbricas de recursos hardware muy limitados (9). Es un sistema basado en componentes, lo que quiere decir que cada *componente* (por ejemplo, el sensor de luz, el altavoz, o el sistema de leds) está gestionado por una librería distinta ya diseñada, lo que ahorra notablemente trabajo al programador, que sólo tiene que diseñar sus propios componentes y conectarlos al resto para obtener el programa.
Otra de sus principales características es que está conducido por eventos, esto es, el sistema se queda parado, a la espera de que algo ocurra: se reciba un mensaje, se haya completado la lectura de un sensor, o ha saltado un temporizador; todos estos *eventos* inician una *señal* cuyo comportamiento decide el programador en cada una de las aplicaciones.
Este sistema está escrito completamente en el lenguaje de programación *NesC*, que se explica a continuación.

1.6.3 NesC (version 1.1.3)

NesC es una extensión del lenguaje de programación *C*, diseñado para el modelo de ejecución basado en componentes del sistema operativo anteriormente mencionado, *TinyOS* (10).

Algunos de los conceptos básicos que rigen este lenguaje de programación son los siguientes:

- Separación de la construcción y la composición: los programas están construidos por *componentes*, que son conectados entre sí para formar programas completos. Los componentes tienen concurrencia interna en forma de tareas. Los hilos de control pueden interactuar con un componente a través de sus *interfaces*. Dichos hilos son iniciados en una tarea o en una interrupción hardware.
- La especificación del comportamiento de un componente se realiza mediante un conjunto de interfaces. Las interfaces pueden ser *proporcionadas* o *usadas* por otros componentes: las *interfaces proporcionadas* representan la funcionalidad que el componente proporciona a un usuario, mientras que las *interfaces usadas* representan la funcionalidad que el componente necesita para llevar a cabo su trabajo.
- Las interfaces son bidireccionales: especifican un conjunto de funciones a ser implementadas por el proveedor de la interfaz (*comandos*) y un conjunto a ser implementado por el usuario de la interfaz (*eventos*). Esto permite a una interfaz representar la interacción entre componentes. Esta es la base de su funcionamiento: los *comandos* en *TinyOS* (por ejemplo, enviar un paquete) no son bloqueantes, ya que su finalización es señalada a través de una *señal* (en

este caso, *sendDone* o envío completado). Especificando interfaces, un componente no puede llamar al comando enviar paquete a menos que dicho componente implemente del *evento sendDone*.

- Los componentes se enlazan de forma estática a otros mediante sus interfaces, lo que incrementa la eficiencia en tiempo de ejecución y permite un mejor análisis de los programas.

1.6.4 Lenguaje SDL y TTCN Suite 4.3

SDL (*Specification and Description Language*) es un lenguaje de especificación orientado a la especificación y descripción del comportamiento de los sistemas distribuidos (11).

Es definido por la *ITU-T* (recomendación Z.100) y, aunque originalmente fue concebido para los sistemas de telecomunicación, actualmente sus áreas de aplicación abarcan procesos de control y aplicaciones en tiempo real de forma general.

SDL proporciona tanto un entorno gráfico de programación (*SDL/*GR, *Graphic Representation*) centrado en los sistemas de telecomunicación como un entorno textual (*SDL/PR*, *Phrase representation*), siendo ambas equivalentes de forma semántica (un programa desarrollado en *SDL* de forma gráfica tiene su equivalencia directa en el entorno textual, y viceversa).

Un sistema es especificado como un conjunto de máquinas abstractas interconectadas, que son una extensión de las máquinas de estado finito (*FSM, Finite State Machines*).

Un sistema en *SDL* está construido a partir de cinco componentes: *estructura*, *comunicación*, *comportamiento*, *datos* y *herencia*. El *comportamiento* de dichos componentes son explicados por la división del sistema en una serie de *niveles de jerarquía*. La comunicación entre los distintos *niveles de jerarquía* se lleva a cabo a través de las *puertas* y los *canales*.

En este proyecto, para la especificación del protocolo en *SDL*, se empleará la suite *SDL and TTCN* en su versión 4.3; se trata de un programa que ofrece las herramientas necesarias para la elaboración del código (en este caso, se empleará su contexto gráfico) así como para su compilación y posterior verificación (12). Las fases por las que se pasará serán las siguientes:

- **Implementación** del código.
- **Compilación** de la aplicación.
- **Validación** del sistema: consiste en verificar que no existen mensajes sin consumir o casos no contemplados que puedan llevar lugar a bloqueos y otros comportamientos indeseados.
- **Simulación** del sistema: se probará el sistema para comprobar su buen funcionamiento antes de la implementación final en nesC.

Debido a las características de este lenguaje de programación características es una de las mejores alternativas a la hora de especificar el protocolo a utilizar en este estudio.

1.6.5 Entorno de programación

El proyecto se divide en dos partes claramente diferenciadas: la especificación del protocolo, y la implementación de dicho protocolo en un lenguaje de programación válido para los dispositivos móviles encargados de la comunicación.

Para cada una de estas partes se hará uso de dos sistemas distintos:

- **Microsoft Windows XP Service Pack 3**, necesario para el uso de la Suite *SDL and TTCN*, donde se realizará por completo la especificación del protocolo.
- **XubunTOS**, una distribución de *GNU/Linux* basada en la conocida *Xubuntu*, con todas las librerías, compiladores y aplicaciones necesarias para la programación en *NesC* bajo el sistema *TinyOS* (13).

1.7 Enumeración del resto de apartados

En el próximo apartado, el número dos, se resumirán los objetivos de otros estudios relacionados con este tipo de redes, así como las conclusiones a las que se llegaron con ellos.
En el apartado número tres se describirá el protocolo propuesto para el estudio; se verá la especificación en *SDL* así como los tipos de mensajes que se emplearán y se verá de forma resumida la implementación del programa que controlará los dispositivos móviles.
En el apartado número cuatro se verán las pruebas realizadas con el protocolo ya descrito; se establecerán varios escenarios, y se probará cada uno de ellos variando distintos parámetros, se mostrarán gráficos ilustrativos y se explicará con detalle el transcurso de dichas pruebas así como sus resultados.
Por último, en el quinto apartado se mostrarán y comentarán los resultados obtenidos, sacando conclusiones a partir de cada una de las pruebas realizadas y proponiendo otros cambios interesantes que podrían afectar el comportamiento de este tipo de redes en estudios futuros.

2.Trabajos relacionados

Son diversos los distintos trabajos relacionados con este tipo de redes que han sido y siguen siendo desarrollados en estos momentos por distintos grupos de desarrollo e investigación; se mencionarán algunos interesantes.

2.1 Estudio de las WSN

Las *Wireless Sensors Networks* (o Redes de Sensores Inalámbricas) han sido estudiadas y desarrolladas por numerosos grupos de ingenieros debido a la enorme utilidad de éstas; se trata de dispositivos muy pequeños, de muy bajo consumo y coste que intercambian pequeños paquetes de información entre ellos relativa a distintos parámetros del entorno sin necesidad de ningún tipo de infraestructura.

En el año 2003 el *MIT* identificó las *WSN* como una de las 10 tecnologías emergentes que cambiarían el mundo (14). Entonces las redes de sensores inalámbricas 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 desarrollos en campos como el medio ambiente (cambio climático), seguridad (vigilancia) y *tracking* (15), a la vez que evidenciaban un conjunto de desafíos muy apetecibles para la comunidad investigadora (16).

De forma general, los nodos mencionados anteriormente se encargan de tomar muestras de los distintos parámetros del entorno (el propio dispositivo se encarga de la conversión A/D) y se transmite a través de una puerta de enlace (ver Figura 4).

Figura 4: MIB600 Ethernet Gateway

Finalmente la información llega a una estación base denominada *stargate* (ver Figura 5) que se encarga de almacenarla durante un tiempo determinado para posteriormente enviarla a un servidor de gran capacidad que se ocupará de realizar un análisis de toda la información recibida y tratarla debidamente.

Figura 5: Plataforma Stargate de Crossbow

Son muchos los usos de este tipo de redes en el entorno industrial, la mayoría de ellos siguen sin ser explotados y otros muchos no han sido descubiertos por el momento; las posibilidades son enormes, y a continuación se muestran los más importantes (17):

- **Monitorización del entorno:** Se trata de observar (en tiempo real o no) las variaciones de los distintos parámetros de un entorno concreto; es especialmente útil, por ejemplo, en la agricultura. Son necesarios un gran número de nodos sincronizados, midiendo y transmitiendo periódicamente. La topología física es relativamente estable ya que los nodos no se mueven si no que están situados en posiciones fijas. En función de los resultados que devuelven los dispositivos se puede proceder a efectuar unas acciones u otras sobre el entorno, lo que ahorra mucho en tiempo y recursos.

- **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, si no que se encuentran en un estado de muy bajo consumo hasta que detecten alguna anomalía. En ese momento será cuando transmitan la información correspondiente. La principal característica de estos entornos es un aprovechamiento importante de la energía. Existen requisitos de tiempo real; la latencia de las comunicaciones juega un papel importante debido a que las acciones que desatan la transmisión suelen tener una importancia crítica y necesitan de intervención inmediata. 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. Es posible, por ejemplo, tener la necesidad de monitorizar un espacio concreto al mismo tiempo que se lleva cierto control sobre la seguridad de las instalaciones.

- **Redes vehiculares:** Más relacionado con el proyecto que nos ocupa, se trata de la transmisión de información entre vehículos en movimiento. Dicha transmisión puede ser con fines de seguridad (informar de un accidente en la vía), confort (recepción de información meteorológica) u ocio (descarga de un podcast). En este caso se tratará de ampliar las posibilidades de estos dispositivos con la intención de transmitir grandes bloques de información. La topología de este tipo de redes es completamente inestable y puede cambiar en cuestión de segundos, por lo que precisan de una adaptación continua por parte de los nodos.

En resumen, el uso de este tipo de redes no es uno concreto, si no que sigue en proceso de estudio y aumentando sus posibilidades día a día, lo que promete una implantación futura en campos donde actualmente no están en uso. Una posible ilustración de una red estática de este tipo podría ser la siguiente (ver Figura 6).

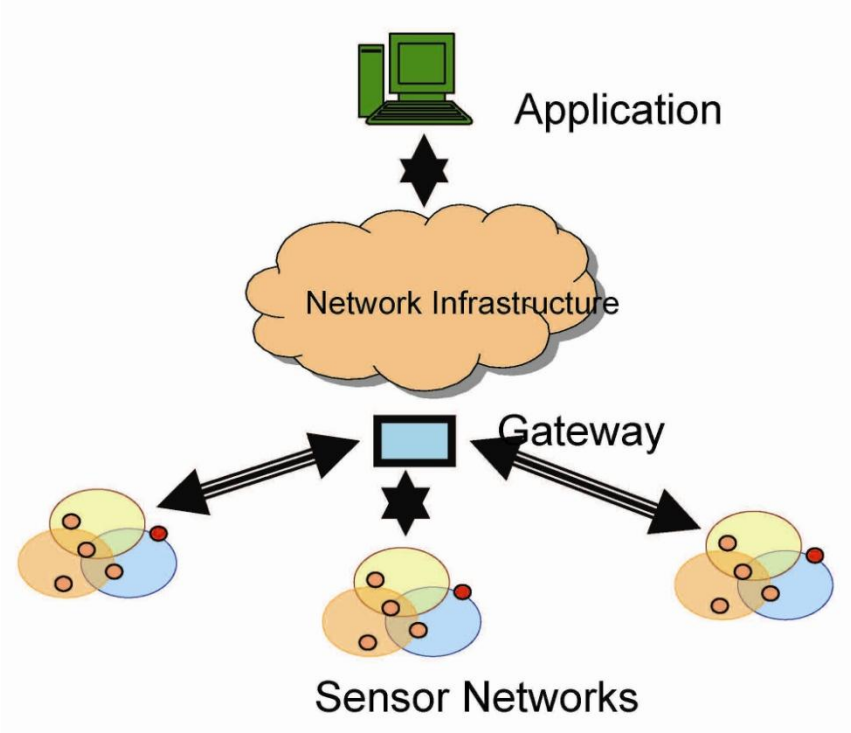

Figura 6: Ilustración de una red inalámbrica de sensores

2.2 Protocolos desarrollados por TinyOS

Los distintos grupos de desarrollo de *TinyOS* han elaborado también una serie de protocolos como forma de estudio y demostración de lo que se puede llegar a hacer con este tipo de redes.

2.2.1 Diseminación de bloques de información

Diseminación es un servicio que consiste en compartir una variable manteniendo su consistencia entre una serie de nodos en una red (todos los nodos de la red mantienen una copia de esta variable). El servicio de diseminación le dice a los nodos de la red cuándo cambia el valor de la variable, e intercambia paquetes para mantener la consistencia de dicha variable (18). Aunque en cualquier momento dos nodos pueden tener valores distintos para la variable en cuestión, a lo largo del tiempo todos los nodos de la red convergerán hacia un mismo valor.

La consistencia permanence incluso ante desconexiones temporales o una pérdida grande de paquetes. A diferencia de los protocolos de inundación de paquetes, que terminan y no consiguen la consistencia, el servicio de diseminación asegura que la red alcanzará un consenso acerca del valor.

Dependiendo del tamaño del dato, los protocolos de diseminación pueden diferir bastante; para la diseminación eficiente de varias decenas de kilobytes de un binario se requiere un protocolo diferente que el necesario para mantener una constante de a penas un par de bytes. No obstante, tienen varias similitudes.

Si separamos un protocolo de diseminación en dos partes (tráfico de control y tráfico de datos) se puede observar que, mientras que los protocolos de tráfico de datos son completamente dependientes del tamaño del dato, el tráfico de control tiende a ser muy similar en ambos casos.

Ser capaz de diseminar pequeños valores en una red es muy útil en el entorno de las aplicaciones de redes de sensores; permite al administrador inyectar pequeños programas, comandos y constantes de configuración a todos los nodos.

2.2.2 Deluge: el protocolo para reprogramar toda una red de nodos

Los mecanismos de diseminación se usan con frecuencia con el objetivo de reconfigurar o reprogramar una red. El protocolo *Deluge* (18) 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 la red.

El crecimiento de las WSN 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.

La programación de red implica varios problemas:

- Las imágenes de programa ocupan mucho más que los datos que difunden otros protocolos de diseminación (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.
- El protocolo de diseminación debe tolerar, además, una densidad nodal que puede sufrir variaciones del orden de cientos de motes.
- 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.
- 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 a desconexiones temporales, fallos, etc.
- 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 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 minizar el número de colisiones. Esta solución tiene como principal inconveniente el aumento de la latencia.

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.
- 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.

2.2.3 Protocolo para la recolección de datos

El mecanismo de recolección de TinyOS (20) proporciona un servicio de entrega de paquetes en una red *multisalto* al nodo raíz de una topología en árbol. Puede haber más de un nodo raíz en una red. En ese caso, el algoritmo se encarga de que al menos una 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.

Como se ha explicado en apartados anteriores, 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 arriba" 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, que constituyen una parte de los problemas con los que cuentan todos los protocolos de red para WSNs:

- **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 circulando por la red, consumiendo ancho de banda.

- **Estimación del enlace:** evaluación de la calidad del enlace entre vecinos (nodos

con visión directa).

- **Auto-interferencias:** prevenir que el reenvío de los paquetes introduzca interferencias en los mensajes propios de un nodo.

2.2.4 CTP (Collection Tree Protocol)

CTP es un algoritmo de recolección de datos en topologías en árbol (21). Algunos nodos de la red se anuncian como raíces del árbol, de forma que el 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).

El protocolo CTP asume que la capa de enlace proporciona cuatro cosas:

- Proporciona una dirección de broadcast local.
- Proporciona asentimientos síncronos para paquetes unicast.
- Proporciona un campo de datos para dar soporte a protocolos de capas superiores.
- Tiene 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.

2.2.5 LEEP (Link Estimation Exchange Protocol)

LEEP 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...).

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

2.2.6 El protocolo TYMO

TYMO es la implementación del protocolo *DYMO* en TinyOS (23). *DYMO (Dynamic MANET On-Demand*) es un protocolo de encaminamiento punto a punto para redes *Ad-Hoc* móviles (*MANETs: Mobile Ad-hoc Networks*). Éste protocolo no almacena explícitamente la topología de la red, en su lugar, un nodo calcula una ruta unicast 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. 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. Además, como la información de encaminamiento que se almacena es mínima, *DYMO* se puede aplicar a dispositivos con poca memoria, como motes.

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.

2.3 CitySense, una red de sensores a escala urbana

CitySense es una red de sensores a escala urbana que está siendo desarrollada por investigadores de la Universidad de Harvard y *BBN Technologies* (24). *CitySense* consiste en 100 sensores inalámbricos a lo largo de una ciudad colocados, por ejemplo, en farolas (ver Figura 7) y edificios públicos o privados, cuyo principal objetivo en este momento es Cambridge, MA.

Figura 7: Nodo CitySense en Cambridge

Cada nodo consiste en un dispositivo *802.11a/b/g* y varios sensores para monitorizar las condiciones meteorológicas y distintos agentes contaminantes (ver Figura 8).

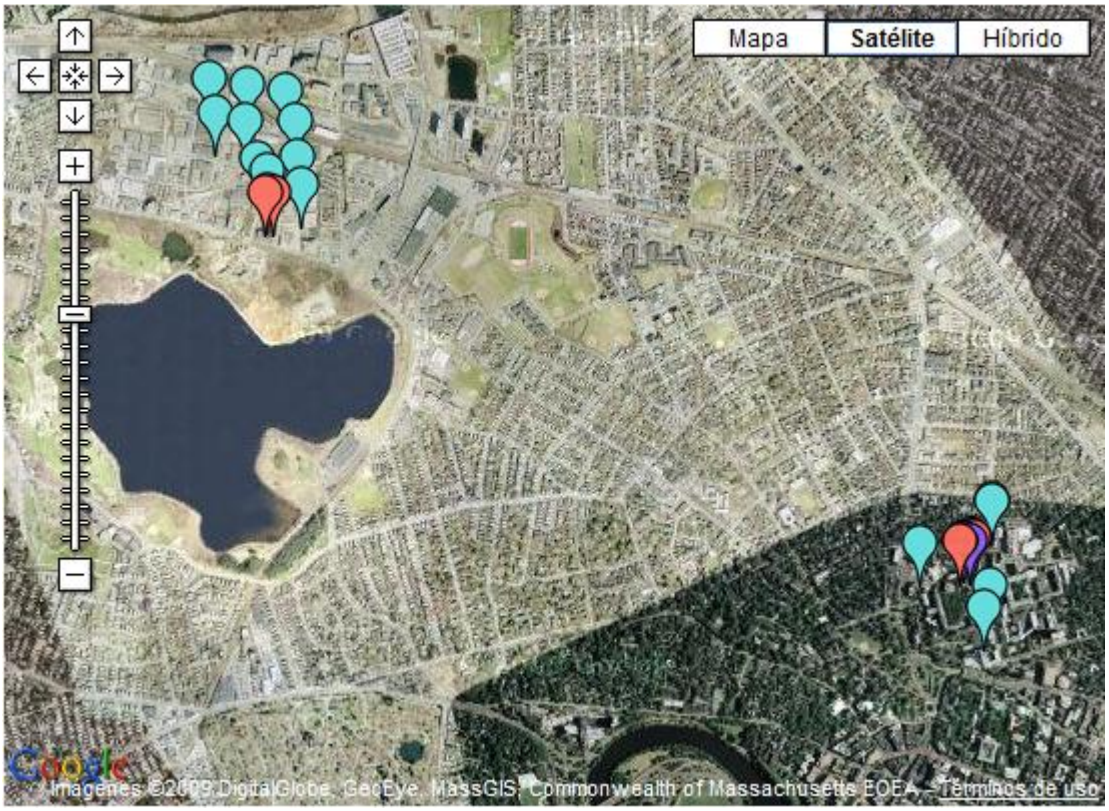

Figura 8: Situación de algunos de los nodos de CitySense

Principalmente *CitySense* pretende ser un banco de pruebas abierto que los distintos investigadores de todo el mundo puede usar para evaluar las redes de sensores inalámbricas en un entorno grande como puede ser una ciudad.

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

De forma paralela a este proyecto se ha estado llevando a cabo otro de idénticos objetivos pero con características e implementación bien distintas debido a que, a diferencia del actual, éste sigue una estrategia unicast para conseguir el envío del contenido (25).

El objetivo del proyecto mencionado es, de forma análoga, 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 difunde se divide en fragmentos que se envían siempre de forma secuencial, sin que haya desorden entre éstos.

Existen tres tipos de mensajes a emplear en la comunicación entre nodos:

- Mensajes de control: *OFERTA* y *SOLICITUD*.
- Mensaje de *DATOS*

El mensaje de oferta es enviado 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. Los estructura del mensaje de oferta es la siguiente (ver Figura 9):

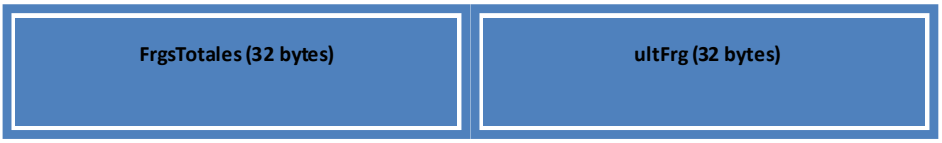

Figura 9: Estructura del mensaje de tipo OFERTA para un protocolo unicast

- **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, este valor indica implícitamente que el emisor de la oferta tiene todos los fragmentos anteriores.

El mensaje de solicitud 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 perjudica la capacidad de recepción del MICAz. La solicitud cuenta con único campo (ver Figura 10):

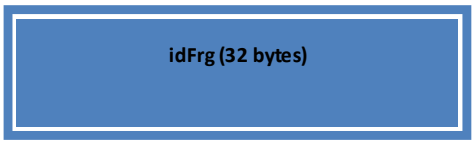

Figura 10: Estructura del mensaje de tipo SOLICITUD para un protocolo unicast

- **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 ha de difundir en la red (ver Figura 11). 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.

Figura 11: Estructura del mensaje de tipo DATOS para un protocolo unicast

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

Mediante estos tipos de mensaje se consigue una transmisión completamente unicast del contenido; un nodo sólo puede transmitir a otro en un momento dado. De forma análoga, un nodo sólo puede recibir de otro nado en un momento dado.

No obstante, las principales bazas de esta estrategia es la de prescindir de anuncios entre cada una de las partes (lo que debería reducir el tiempo de transmisión punto a punto) así como la eliminación de un tiempo de espera de solicitudes entre cada uno de los paquetes.

La intención de la elaboración de dos protocolos para el mismo fin es el de comparar las virtudes y defectos de cada una de las estrategias, para llegar a conclusiones más claras acerca de cómo llevar a cabo este tipo de transmisiones móviles en un entorno hostil como es el medio radio.

3.Descripción del protocolo

3.1 Tipos de mensaje

Como se ha dicho anteriormente, la especificación del protocolo se llevará a cabo mediante el lenguaje de especificación SDL, que permite una descripción de su comportamiento y también las comprobaciones y simulaciones necesarias para saber que su implementación real funciona sobre el papel.

En primer lugar, se hará un estudio de los tipos de mensaje necesarios entre los dispositivos para llevar a cabo la transmisión completa del contenido. 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; como se verá más adelante, en este caso se propone un tamaño de P_s=6400 bytes, ya que es necesario transmitir un volumen relativamente grande de información.

Otro de los problemas lógicos a resolver es el de que cada vehículo anuncie en un momento determinado que tiene una información que transmitir, en este caso, que tiene ciertas partes de una información para transmitir. Para resolver esto se ha optado por utilizar un tipo de mensaje, *ANUNCIO*, que se enviará periódicamente al medio para informar de las partes que un nodo está dispuesto a transmitir en un momento dado. La estructura de este tipo de mensaje sería como sigue (ver Figura 12).

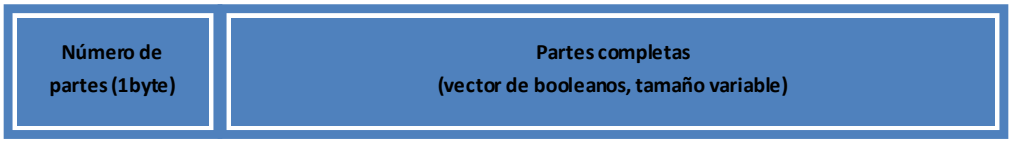

Figura 12: Estructura del mensaje de tipo ANUNCIO

Como se puede observar, se ha prescindido de un valor que identifique unívocamente al contenido; esto es así porque no es un parámetro interesante en este estudio, ya que se pretende el diseño de un protocolo que transmita la información sin importar la

finalidad de ésta. En una implementación real será necesario la inclusión de un campo del tipo identificador a la hora de anunciar un contenido.

El primer byte del mensaje, *Número de partes*, como su propio nombre indica, almacena el número de partes (y por tanto, el tamaño de la información completa, ya que cada parte tiene la misma longitud, y ésta se conoce con anterioridad) de que se compone la información; este valor permite a un nodo receptor conocer la longitud del vector de bytes que viene a continuación. Es importante recalcar que este tipo de mensaje se enviará de forma broadcast, es decir, a todos los nodos que entren dentro del rango de visibilidad del emisor en un momento dado; cada uno de los nodos receptores del anuncio, por su parte, analizarán el anuncio y decidirán si necesitan alguna de las partes.

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 sería la siguiente (ver Figura 13).

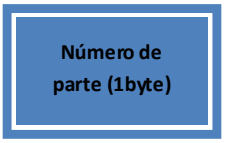

Figura 13: Estructura del mensaje de tipo SOLICITUD

Donde, una vez más, se prescinde de un campo que identifique al contenido.

No es necesario más que un campo de un byte que identifique el número de parte deseada del contenido; sin embargo, 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 (ver Figura 14).

Figura 14: Estructura del mensaje de tipo PARTE

Donde el primer byte del mensaje identifica la parte enviada y, a continuación, el bloque de información. Este tipo de mensaje se enviará de forma broadcast al medio, siguiendo la propuesta de este estudio.

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.

De esta forma se tienen suficientes medios para la comunicación broadcast entre los nodos, de forma completamente autónoma, donde los receptores solicitan la parte deseada siguiendo un criterio y los emisores envían una parte cualquiera, sin haber compromiso en ningún momento y sin poner en riesgo la transmisión del contenido al estar dividido en bloques.

3.2 Especificación del protocolo en SDL

La especificación en SDL del protocolo propuesto sería la siguiente (ver Figura 15).

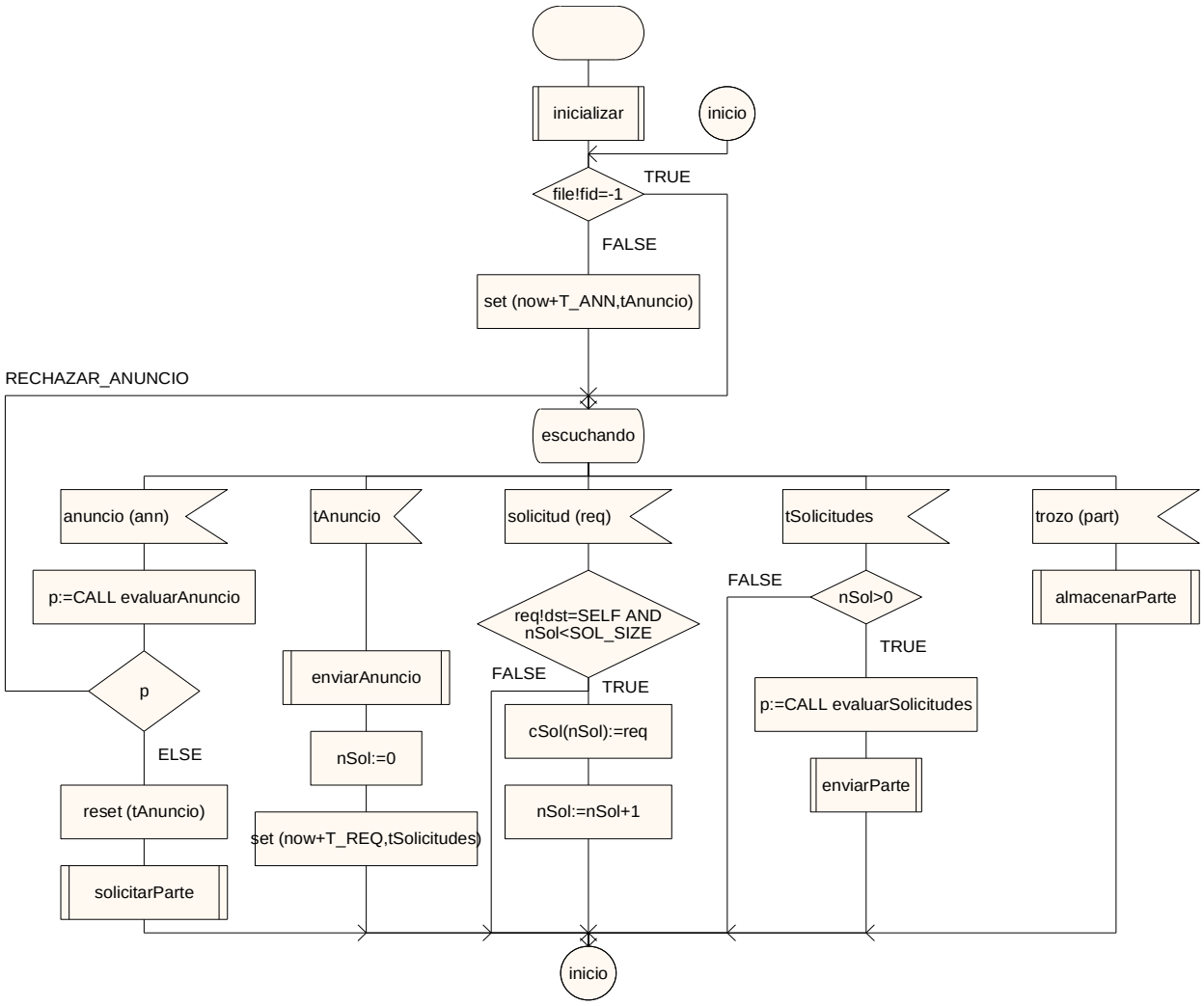

Figura 15: Proceso principal de un nodo en lenguaje SDL

Algunas de las características importantes de la especificación son las siguientes:

- Los mensajes de anuncio se envían, siempre que se tenga como mínimo una parte, de forma periódica al medio con un período *TANN*.
- Los nodos están dispuestos en todo momento a recibir cualquier tipo de mensaje, así como cualquiera de sus temporizadores (el que se encarga de separar los anuncios, *TANN*, y el que se encarga de esperar las solicitudes tras un anuncio, *TREQ*).
- El hecho de obtener una parte convierte al receptor en un nuevo portador de información, esto es, un nodo comenzará a enviar anuncios en el momento en que tenga una única parte.

3.3 Explicación del protocolo

Para entrar más en profundidad en el funcionamiento del protocolo, supondremos un escenario como el que sigue (ver

Figura 16).

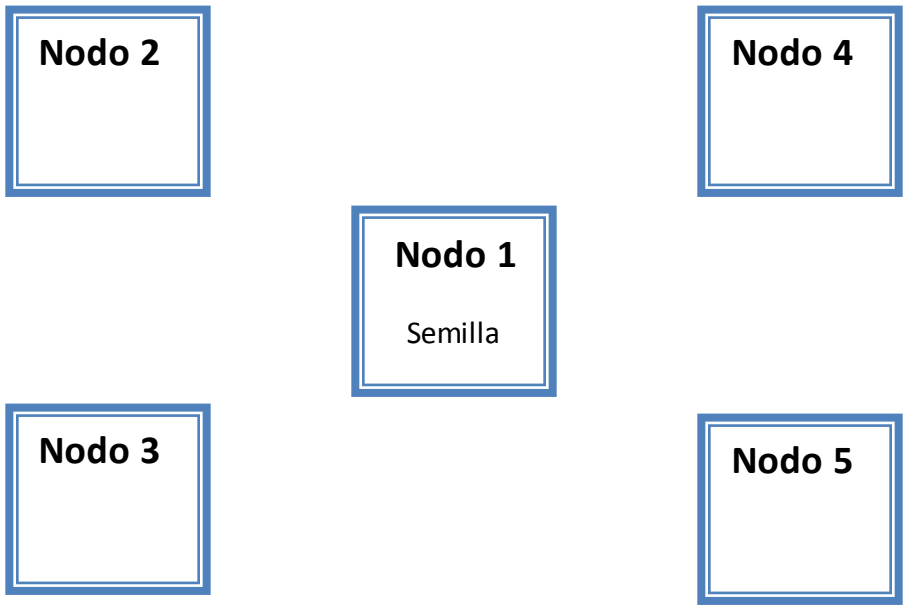

Figura 16: Escenario de ilustración inicial

Donde el primero de ellos, el Nodo 1, tiene por completo el contenido, y el resto, no tienen nada.

A continuación una representación de los distintos mensajes que se enviarán durante una fase de la transmisión:

- La semilla envía un anuncio a todos sus vecinos y queda a la espera de la recepción de solicitudes. En dicho anuncio presenta al resto de nodos las partes que tiene en ese momento (ver Figura 17).

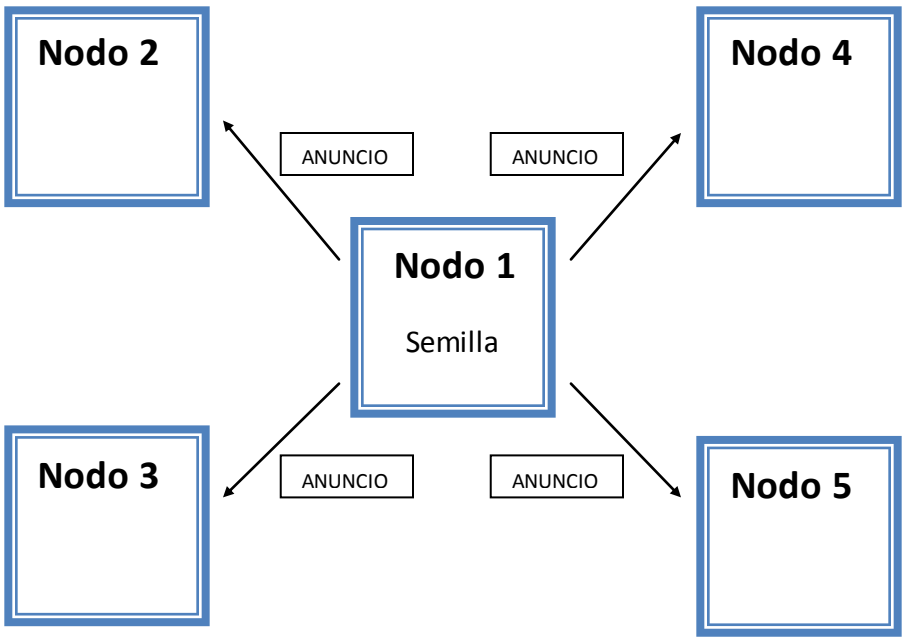

Figura 17: Escenario de ilustración, fase 1

- Los nodos receptores analizan el anuncio y solicitan una parte a la semilla siguiendo un determinado criterio (ver Figura 18).

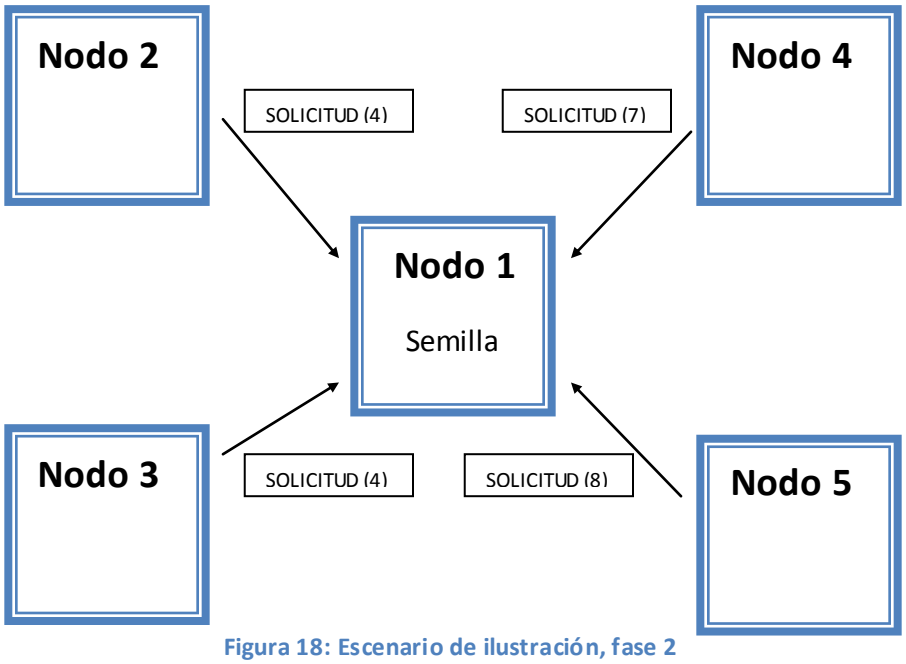

Figura 18: Escenario de ilustración, fase 2

- La semilla analiza las solicitudes recibidas durante el tiempo de espera y selecciona la más repetida, para enviarla posteriormente vía broadcast.
  Los receptores, de forma individual, decidirán si almacenan la parte o la rechazan, dependiendo de si ya la tienen almacenada en memoria (ver Figura 19).

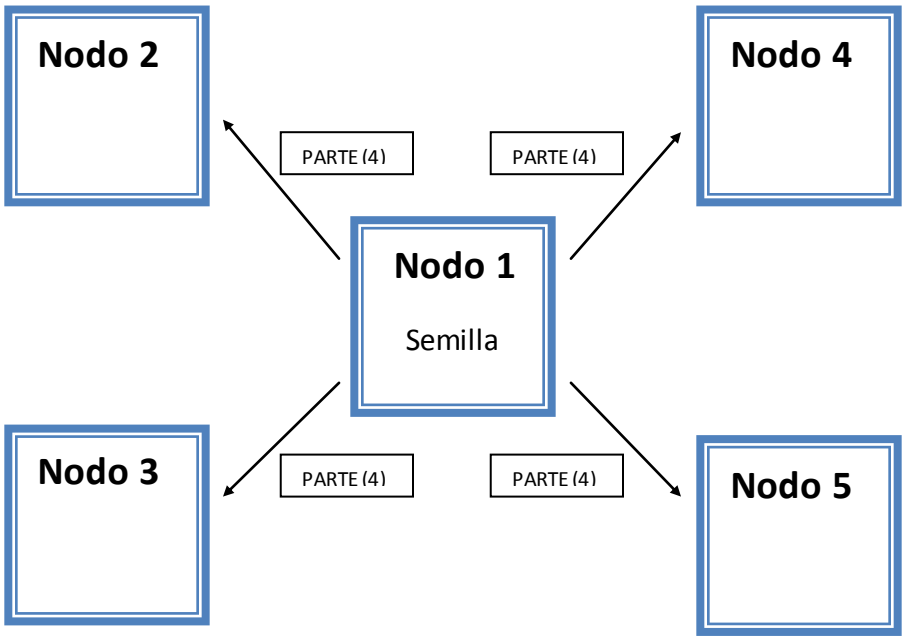

Figura 19: Escenario de ilustración, fase 3

Este proceso se repetirá una y otra vez hasta que todos los nodos del escenario tengan completamente el contenido que tenía semilla al comienzo de la transmisión.

Según lo que se dijo anteriormente, y basándonos en el comportamiento que se acaba de ver, en el momento en que un nodo reciba una parte de la semilla, éste pasará a ser, en cierto modo, una semilla más; esto produciría comportamientos indeseables en el protocolo, debido a que tras la primera fase de la transmisión los cinco nodos comenzarían a enviar anuncios al mismo tiempo, aumentando el número de colisiones de forma innecesaria (ya que no es necesario que los nodos receptores anuncien su contenido parcial mientras siga habiendo una semilla en el escenario).

Para evitar este comportamiento se puede hacer un ajuste en el temporizador de envío de anuncio de un nodo que acaba de enviar una parte. La solución que se propone es la de dividir *TANN* entre un factor; en este caso, se hará una división entre cincuenta.

De esta forma, si el período de anuncio es de 1000 milisegundos, un nodo que termina la transmisión de un fichero ofrecerá un anuncio 20 milisegundos más tarde, impidiendo de esa forma que el resto de nodos lo haga al mismo tiempo.

Bajo este supuesto, los nodos receptores sólo enviarán sus anuncios cuando:

- Tengan el contenido completo.
- La semilla salga del escenario; esto es, cuando hayan pasado TANN milisegundos desde la última recepción de una parte y no hayan recibido un nuevo anuncio de la semilla.

Se obtiene así el comportamiento ideal en un escenario estático con una semilla y uno o varios receptores: la semilla enviará por completo el contenido y, una vez hecho esto, los cinco nodos enviarán anuncios de forma periódica buscando nuevos vecinos en el entorno a los que transmitir la información.

Otro problema surge de la no existencia de compromiso entre un receptor que envía una solicitud y un emisor que recibe dicha solicitud: ni el receptor se compromete a esperar su transmisión, ni el emisor se compromete a transmitirlo, ya que éste podría recibir un anuncio de un tercer nodo con información valiosa para él y, de la misma forma, enviaría su solicitud. Para explicar este hecho, se tiene el siguiente escenario (ver Figura 20).

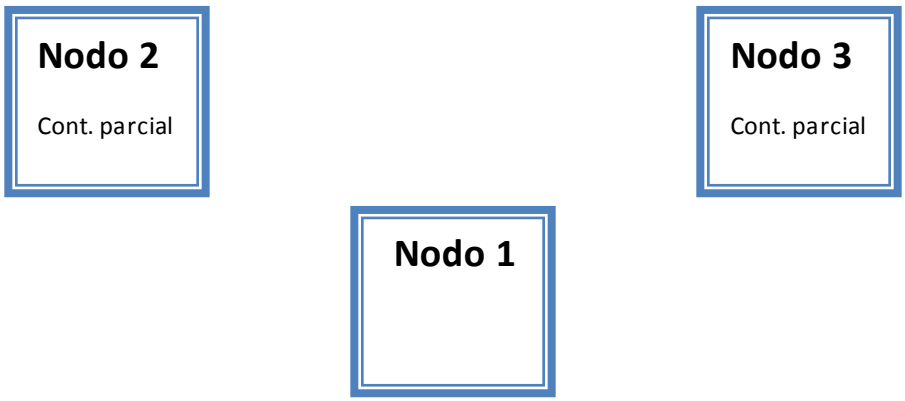

Figura 20: Escenario de ilustración 2 inicial

Donde el primero de ellos, Nodo 1, no tiene nada, Nodo 2 tiene la primera mitad del contenido, y Nodo 3, la segunda mitad.

- Nodo 2 y Nodo 3 envían sus respectivos anuncios vía broadcast (ver Figura 21).

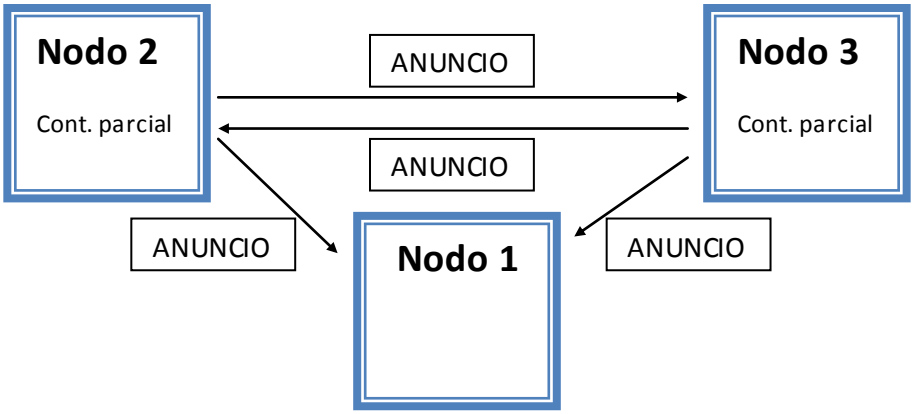

Figura 21: Escenario de ilustración 2, fase 1

- Nodo 2 envía su solicitud a Nodo 3 y viceversa; Nodo 1, por su parte, envía una solicitud a cada uno de los otros dos (ver Figura 22).

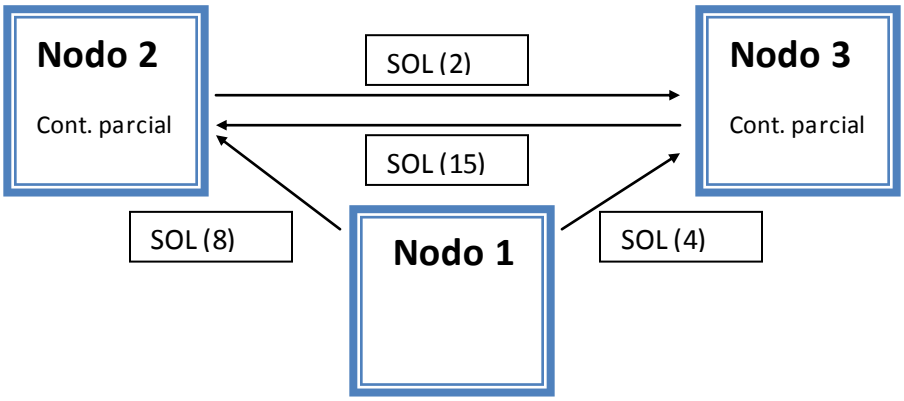

Figura 22: Escenario de ilustración 2, fase 2

Como se puede observar, el comportamiento en este caso es impredecible y dependerá, entre otras cosas, de la distancia entre cada uno de los nodos y otros factores del entorno. Podrían pasar varias cosas:

- Nodo 2 es el primero en enviar su parte (que seleccionará aleatoriamente entre la 15 y la 8, ya que no se repite ninguna). Nodo 1 y Nodo 3 la reciben al mismo tiempo. Nótese que Nodo 3, en este caso, decide no enviar ninguna a pesar de haber recibido dos solicitudes.
- Nodo 3 es el primero en enviar su parte (que seleccionará aleatoriamente entre la 2 y la 4, ya que no se repite ninguna). De forma análoga, Nodo 1 y Nodo 2 las reciben, y Nodo 2 prescinde de enviar alguna de las partes que se le solicitaron.

- Nodo 2 y Nodo 3 comienzan a enviar en un instante de tiempo parecido. En este caso, Nodo 1 podrá aceptar una, y sólo una de ellas; sin embargo, en el medio se están transmitiendo dos partes, una de ellas sin un receptor.

De forma parecida se podrían pensar otros escenarios problemáticos, por ejemplo, un nodo receptor y 4 semillas; éstos enviarán sus anuncios, y el receptor enviará una solicitud a cada una de las semillas (las cuales no envían ninguna solicitud, al tener el contenido completo). Una de las 4 partes que acabarán transmitiéndose en el medio será aceptada por el receptor; las otras tres, serán descartadas.

En estos ejemplos se producirán colisiones, una y otra vez, multiplicando el tiempo necesario para enviar la parte completa al receptor.

La solución que se propone en este estudio es la de establecer un compromiso entre solicitud enviada/solicitud recibida; esto es, una vez que un nodo envía una solicitud, se compromete a esperar un tiempo determinado, *TPART*, durante el cual estará pendiente únicamente de que el nodo al que envió la solicitud le responda con una parte. De forma análoga, un nodo emisor que recibe una solicitud, se compromete a enviar una parte, sin posibilidad de aceptar por tanto otras partes o anuncios que le lleguen durante el período de espera de solicitudes. Se resume este hecho en dos puntos:

- Un receptor entra en el estado **RECIBIR** una vez que envía una solicitud, y puede salir de este estado de dos formas posibles.
  - Cuando completa la recepción de la parte.
  - Cuando ha transcurrido un tiempo TPART sin recepción de la parte.

- Un emisor entra en el estado **ENVIAR** una vez que recibe una solicitud, y ÚNICAMENTE puede salir de este estado una vez que ha completado la transmisión de la parte.

De esta forma, bajo el escenario del ejemplo anterior (un nodo receptor y cuatro semillas), el receptor únicamente enviará una solicitud a una de las semillas (en este caso, a la semilla que la envió antes), y esperará durante un tiempo *TPART* su parte; una vez recibida, sale del estado **RECIBIR** y vuelve a estar abierto a la recepción de nuevos anuncios.

3.4 Descripción del código fuente

Como se ha comentado en apartados anteriores, la implementación del código fuente se ha escrito por completo en lenguaje *NesC*: un lenguaje orientado a *eventos* donde cada uno de los *componentes* del sistema se comunica a través de sus *interfaces*.

3.4.1 La interfaz FILE

La primera necesidad que surge a la hora de escribir el código es la de, en cierto modo, separar la funcionalidad del protocolo de las operaciones de bajo nivel que conciernen a la escritura, lectura y control de la memoria del sistema.
En este caso, se empleará la memoria FLASH del MICAz para el almacenamiento de la información, cuyo tamaño es de 512 Kilobytes.
Las librerías que nos ofrece TinyOS incluyen interfaces (y componentes que proporcionan dichas interfaces) con las funciones necesarias para esta gestión, y son, entre otras, las siguientes:

- **BlockRead:** nos ofrece los comandos y eventos necesarios para las operaciones de lectura de grandes volúmenes de datos.
- **BlockWrite:** nos ofrece los comandos y eventos necesarios para las operaciones de escritura de grandes volúmenes de datos.
- **Mount:** interfaz que nos permite inicializar la utilización de un sistema de ficheros de pequeños volúmenes de datos (todos aquellos que tienen que ver con la configuración del dispositivo, y no con el contenido de información).
- **ConfigStorage:** nos ofrece los comandos y eventos necesarios para las operaciones de lectura y escritura de pequeños volúmenes de datos.

Existen otras interfaces que TinyOS nos ofrece para la gestión de memoria, pero son éstas las que se emplearán en este trabajo.
Las dos últimas, *Mount* y *ConfigStorage*, se emplearán para el almacenamiento de información de control:

- **Número de partes del archivo:** que será un valor nulo en caso de no poseer información de dicho archivo. En el instante inicial, sólo la(s) semilla(s) conocen este valor. Este valor se envía al medio al anunciar un contenido.
- **Número de partes restantes:** este valor es usado por los nodos que tienen total o parcialmente el contenido, y representa el número de partes que quedan por obtener. El principal objetivo de este valor es el de aumentar la velocidad de análisis de anuncios al conocer de antemano las partes almacenadas sin necesidad de analizar el vector.
- **Partes obtenidas:** representadas por un vector de booleanos de tamaño *NÚMERO DE PARTES* con un valor TRUE en aquellas posiciones cuya parte se tiene y un valor FALSE en aquellas cuya parte no se tiene almacenada. Este vector es el mismo que se envía al medio al anunciar un contenido.

Además de estos tres campos, se añade uno más, *VERSION*, cuyo objetivo es el de distinguir cuándo un nodo ha sido inicializado antes y cuándo no: si su contenido no coincide con el esperado por la aplicación, ésta inicializa toda la información con unos valores por defecto. Si coincide, se procede a la extracción de los valores a la memoria RAM para su posterior uso.
Respecto a las dos primeras interfaces, *BlockRead* y *BlockWrite*, se emplearán para el almacenamiento y lectura del contenido a transmitir.
Con estas herramientas se ha codificado una interfaz, denominada *FILE*, y un componente que la proporciona, denominado *FILEC*, que permite al programa

principal abstraerse de la gestión de memoria; esto es, no tendrá que llamar directamente a las funciones de lectura/escritura y controlar los parámetros del fichero, encargándose de ese trabajo dicho componente.

La interfaz *FILE* proporciona los siguientes comandos:

- **Void init():** Inicializa los parámetros del fichero: comprueba si existe o no, las partes completadas, las que faltan por completar, etc. Toda esa información la extrae de la FLASH, y la carga en la RAM. En caso de no existir dicha información la inicializará con unos valores por defecto.
- **Uint8_t parts():** Devuelve las partes que tiene el fichero almacenado total o parcialmente.
- **Uint8_t remaining():** Devuelve las partes que faltan por completar en el fichero.
- **Bool vector (uint8_t n):** Devuelve TRUE si la parte *N* del fichero está completa, FALSE en caso contrario.
- **Bool exists():** Devuelve TRUE si el fichero está parcial o totalmente completado, FALSE en caso de no tener ninguna parte completa.
- **Void setParts (uint8_t n):** Establece el número de partes que tiene el fichero y almacena dicha información en la FLASH para su obtención en futuros reinicios.
- **Void get (prt *p):** Obtiene 64 bytes de una parte de la memoria FLASH. Se señalizará el evento *getDone* cuando la información esté disponible en la RAM.
- **Void store (prt *p):** Almacena 64 bytes de una parte en la memoria FLASH. Se señalizará el evento *storeDone* cuando la información esté guardada en la FLASH.
- **Void completePart** (uint8_t part): Actualiza el vector de partes así como el número de partes pendientes, y lo almacena en la FLASH. Esta función ha de ser llamada únicamente cuando se ha completado la parte en su totalidad.

Además de los comandos, exige la implementación por parte del componente usuario de esta interfaz de los siguientes eventos:

- **Void initDone():** Se señaliza cando se ha completado la inicialización del archivo con éxito.
- **Void getDone():** Se señaliza cuando la información ya está disponible tras una lectura de la FLASH.
- **Void storeDone():** Se señaliza cuando la información sea terminado de almacenar en la FLASH.
- **Void error (uint8_t error):** Se señaliza cuando ha ocurrido un error grave en alguna parte de la ejecución del programa. Indica que el programa debe gestionar el error de la forma adecuada.

Mediante el uso de esta interfaz, el programa principal se centrará principalmente en las tareas de envío y recepción de mensajes al medio, así como de análisis de los distintos tipos de mensajes y su actuación en consecuencia, dejando al componente *FILEC* encargarse de las tareas de la memoria.

3.4.2 El programa principal

Sin entrar en profundidad en la codificación, se explica a continuación las interfaces que utiliza el programa principal:

- **Boot:** Obliga al usuario a implementar el evento *booted*; su objetivo es el de indicar a TinyOS dónde comienza el programa.
- **Leds:** Permite al usuario la utilización de los leds. En este caso será la forma de conocer en tiempo de ejecución lo que hace la aplicación.
- **Random:** Proporciona los comandos necesarios para la generación de números pseudo-aleatorios.
- **Init:** Interfaz que ejecuta la inicialización de la semilla para la generación de números aleatorios.
- **File:** Como se ha dicho anteriormente, abstrae al programa de la gestión de memoria.
- **SplitControl:** Interfaz para controlar el estado de conexión del dispositivo.
- **Timer:** Permite el uso de temporizadores; en este caso se usarán, por ejemplo, para establecer el período de envío de anuncio, *TANN*, entre otros.
- **Packet y AMPacket:** Permiten la lectura/escritura de los paquetes que se envían al medio radio, así como la extracción de distintos parámetros, por ejemplo, la dirección origen de un paquete recibido.
- **AMSend:** Se encarga del envío de paquetes al medio.
- **Receive:** Se encarga de la recepción de los distintos tipos de paquete.

Una vez se entra en la codificación, se observa un problema debido a las limitaciones del propio sistema: el tamaño máximo de bytes en un paquete de datos. TinyOS, por defecto, permite un tamaño máximo de 29 bytes. Para aumentar este valor, siguiendo la especificación del sistema, se ha de incluir la siguiente directiva en el Makefile de la aplicación:
CFLAGS +=-DTOSH_DATA_LENGTH=TAMAÑO_MÁXIMO

No obstante, *TAMAÑO_MÁXIMO*, debido a limitaciones de hardware, no puede ascender a más de, aproximadamente, 120 bytes.
Esta limitación impide el planteamiento inicial, que como se ha explicado, consiste en dividir el contenido en partes de un tamaño fijo, en nuestro caso, 6400 bytes.
Una posible solución a este problema podría ser disminuir notablemente dicho tamaño hasta un máximo de 120 bytes; no obstante, una cifra tan pequeña exige un número de partes demasiado elevado, incrementando el vector de partes obtenidas hasta un tamaño excesivo como para transmitirlo al medio en un solo paquete. Además, exigiría que los campos de mensaje relativos al número de partes extendieran su tamaño a un valor superior al byte, aumentando el volumen de datos de control y haciendo ineficiente el sistema.

La solución propuesta para el problema, es la de enviar cada parte del archivo en varios trozos, que se denominarán *subpartes*, al medio.
Es importante mencionar el hecho de que, aún teniendo que enviar la parte en varios paquetes, esta transmisión y, por tanto, la recepción de cada conjunto de subpartes que conforman una parte, es una operación completamente atómica; esto implica los siguientes puntos:

- Un nodo emisor ha de enviar todas las subpartes de una parte dada de forma continuada, y no sale del estado **ENVIAR** hasta cumplir dicha transmisión. Esto implica que, una vez recibe una solicitud, no sale de dicho estado hasta transmitir el último paquete correspondiente a la parte enviada.
- Un receptor ha de esperar a la recepción completa de todas las subpartes de una parte dada para dar por concluida la transmisión; la falta de una de ellas se supone consecuencia de la pérdida de conexión con respecto al emisor y ha de descartar lo almacenado hasta el momento, volviendo a estar dispuesto a recibir nuevos anuncios.

El tamaño de la subparte es otro de los parámetros modificables para la aplicación; en este caso, se empleará un tamaño de 64 bytes; la razón, son los posibles problemas derivados de la utilización de la memoria a la hora de escribir/leer cifras que no son potencia de dos. Una cifra mayor, como 128, está fuera del límite hardware que estos dispositivos imponen a la hora de transmitir paquetes.
De esta forma podemos concluir que, con un tamaño de parte de 6400 bytes, y un tamaño de subparte de 64, cada parte se compondrá de 100 subpartes; es decir, un nodo emisor enviará de forma consecutiva 100 paquetes para dar por concluida la transmisión de una parte.
Esta solución implica, por lo tanto, añadir cierta información de control en los tipos de mensaje; no obstante, y debido a que se trata de una operación atómica y transparente para el usuario (un nodo NO puede solicitar la transmisión de una subparte, únicamente puede solicitar una parte) sólo habrá cambios en los mensajes del tipo PARTE, quedando como sigue (ver Figura 23).

Figura 23: Estructura definitiva del tipo de mensaje PARTE

Donde se puede observar que ahora el campo de información es de 64 bytes y, además, se ha añadido un campo *Número de subparte*, que indica al receptor la subparte recibida.

Por su parte, el receptor, deberá comprobar dicho campo con cada paquete recibido: si el campo *Número de subparte* no es igual a la subparte esperada, esto es, se ha perdido una subparte, saldrá del estado **RECIBIR** y volverá a responder a los anuncios.

El emisor, ante una situación de pérdida, y debido a que el protocolo se basa en una alternativa broadcast, terminará de enviar las 100 subpartes y saldrá del estado enviar. Es importante recalcar el hecho de que el emisor NO sabe cuándo un nodo concreto ha recibido o no la información, puesto que no hay asentimientos.

A continuación, una descripción de cada una de las funciones que componen la implementación del protocolo:

- **sendAnnouncement:** Se encarga de la transmisión de un anuncio. Es llamada únicamente cuando salta el temporizador *TANN*. La función llama a los comandos de la interfaz *FILE* para obtener información del contenido y posteriormente envía un paquete del tipo *ANUNCIO* al medio radio.
- **requestPart:** Se encarga de solicitar una parte a un nodo concreto. Es llamada tras analizar un anuncio y tomar la decisión de solicitar una parte. Envía al medio un paquete del tipo *SOLICITUD*.
- **sendPart:** Se encarga de iniciar la transmisión de una parte. La función solicita a la interfaz *FILE* la lectura de la primera subparte y la aplicación queda a la espera de completar la lectura.
- **checkAnnouncements:** Esta función es la que se encarga de analizar un anuncio recibido y, por lo tanto, es llamada tras la señalización del evento correspondiente a la recepción de un anuncio. En este caso particular, se ha optado por una estrategia aleatoria: el algoritmo se encarga de almacenar en un vector todas aquellas partes que (a) no tiene y (b) tienen un valor true en el vector de partes del anuncio que está siendo analizado, quedando fuera de dicho vector todas aquellas que no cumplen las condiciones (a) y (b). Una vez se tienen las partes, se selecciona una de ellas de forma aleatoria; la ventaja de esto es clara: con una selección aleatoria el contenido se dispersa más rápidamente a través de todos los nodos, mientras que con una selección secuencial se tiende a inundar el medio de nodos con las mismas partes. Esto permite que, en una situación en la que ningún nodo tiene el fichero completo, todos terminen con él, si la unión de las partes de cada uno forman el contenido completo.
- **checkRequests:** Esta función analiza todas las solicitudes recibidas durante el tiempo de recepción de solicitudes (*TREQ*) y selecciona una parte a enviar; es llamada, por tanto, cuando termina el temporizador *TREQ*. En este caso se ha optado por seleccionar aquella más veces solicitada por los nodos vecinos; en caso de haber igualdad, se enviará la primera que se recibió. Se hace así debido a la suposición de que la calidad de conexión con el solicitante es mayor, y por esta razón respondió antes al anuncio. De esta forma aumenta ligeramente la probabilidad de que la parte llegue correctamente a su destino.

Además, el código está principalmente compuesto por los eventos que señalizan cada una de las interfaces que utiliza la aplicación, y son los siguientes:

- **Boot.booted():** Es señalizado cuando el programa comienza. La aplicación únicamente dedica este evento a tareas de inicialización: es inicializada la semilla para la generación de números aleatorios, y también el fichero; se

comprueba el campo *VERSION* y se extrae la información de control, además de darle un valor en caso de no existir.

- **File.initDone():** Es señalizado cuando se ha completado la inicialización del fichero, y comienza la inicialización de la pila de protocolos necesarios para la transmisión y recepción de información a través del medio radio.
- **AMControl.startDone():** Es señalizado cuando la pila de protocolos del dispositivo está funcionando. En la implementación se inicializan algunas variables para el posterior almacenamiento de los paquetes que se recibirán.
- **AMControl.stopDone():** Es señalizado cuando ha dejado de funcionar la pila de protocolos; se empleará en la fase de pruebas para el control de errores.
- **File.error():** Es señalizado cuando ha ocurrido un error grave con la gestión de memoria. Se parará el funcionamiento de la pila de protocolos debido a que el programa no puede continuar.
- **TAnn.fired():** Se ha cumplido un período del temporizador de anuncios. Se enviará un anuncio a los nodos vecinos.
- **TReq.fired():** Ha finalizado el tiempo de recepción de solicitudes; se procederá al análisis de éstas para el posterior envío de una parte al medio.
- **TPart.fired():** Se ha cumplido el tiempo de recepción de parte; esto es, el nodo emisor no ha respondido a tiempo enviando la subparte correspondiente, y el nodo receptor cancelará la recepción.
- **SendAnn.SendDone():** Es señalizado cuando se ha completado la transmisión de un anuncio. Se utilizará para el control del sistema, encendiendo/apagando el led ROJO para indicar su correcto envío.
- **SendReq.SendDone():** Es señalizado cuando se ha completado la transmisión de una solicitud. Se utilizará para el control del sistema, encendiendo/apagando el led VERDE para indicar su correcto envío.
- **SendPart.SendDone():** Es señalizado cuando se ha completado la transmisión de una subparte. Se utilizará, de forma análoga, para el control del sistema, encendiendo/apagando el led AMARILLO para indicar su correcto envío. Además, esta función solicitará a la interfaz *FILE* la lectura de la siguiente subparte en caso de haberla; en caso contrario, se sale del estado **ENVIAR**.
- **File.getDone():** Es señalizado cuando se ha completado la lectura de una subparte. Se procede al envío de dicha subparte.
- **File.storeDone():** Es señalizado cuando se ha completado la escritura de una subparte. Se verifica si es la última; si lo es, se actualiza la información de control y se sale del estado **RECIBIR**. En caso contrario la aplicación queda a la espera de la próxima subparte.
- **RecAnn.receive():** Es señalizado cuando se recibe un anuncio. Se procede a su análisis en caso de no estar en ningún estado de envío/recepción. En caso de solicitar una parte, activa el temporizador *TPART* que asegura el compromiso de recepción durante un tiempo determinado.

- **RecReq.receive():** Es señalizado cuando se recibe una solicitud. Se almacena en la cola de solicitudes y se espera a recibir otras; además, en caso de ser la primera, se entra en el estado **ENVIAR**.
- **RecPart.receive():** Es señalizado cuando se recibe una subparte. Se hacen las comprobaciones pertinentes: si pertenece al nodo con el que se mantiene el compromiso, si es la parte esperada, y si la subparte coincide con la siguiente a recibir. En caso de no cumplirse la última condición se supone la pérdida de una de ellas y, por tanto, se sale del estado **RECIBIR**.

Una vez estudiado y probado el código, surge un problema más: la adecuación de la velocidad entre los dispositivos emisores y receptores.

Como se podía prever en un principio, un nodo emisor que ha de leer y enviar información funciona de una forma mucho más rápida que un nodo receptor, ya que éste ha de actualizar la información en la memoria FLASH con cada paquete recibido (una operación muy costosa para el dispositivo). Este hecho obliga a establecer un período para el envío de las subpartes.

Las diversas pruebas llevadas a cabo ponen de manifiesto que un período de 8 milisegundos entre paquetes es suficiente como para que el receptor pueda almacenar la información y quedarse a la espera del siguiente bloque de información. Para conseguir esto, se ha introducido un nuevo temporizador en el emisor, *TWAIT*, que iniciará la lectura de la siguiente subparte al completar la transmisión de la actual, en lugar de realizar la nueva transmisión inmediatamente después.

4.Pruebas

4.1 Elementos necesarios

A la hora de probar el funcionamiento del protocolo se hacen necesarios una serie de elementos; el primero de ellos alguna forma de comunicar el dispositivo con el exterior para saber qué está haciendo en cada momento. Son dos los métodos empleados para esta tarea:

- Conexión del MICAz al puerto serie de un PC: Mediante la aplicación *SerialForwarder* proporcionada por TinyOS se pueden leer los mensajes que el MICAz recibe y envía, lo que permite saber que la información es correcta.
- Un sistema de encendido/apagado de leds que describa el funcionamiento de la aplicación. En este caso, bajo un funcionamiento correcto, cada uno de ellos representa el envío de cada uno de los tres tipos de mensaje; de esta forma se puede saber qué envían a cada momento. Además, se ha capturado cada uno de los posibles errores que pueda haber en tiempo de ejecución (problema de lectura/escritura, problema a la hora de inicializar la pila de protocolos, o errores inesperados a la hora de enviar mensajes al medio) y se ha establecido un código de error para cada uno de ellos; la aplicación hará parpadear a una frecuencia determinada cada uno de los leds siguiendo dicho código, de forma que sea sencillo interpretar cada uno de los distintos fallos del sistema. La siguiente tabla recoge el código de cada uno de los errores (ver Tabla 1).

Naranja	Verde	Rojo	Significado
0	0	1	Error de escritura de información
0	1	0	Error de lectura de información
0	1	1	Error de sincronización de FLASH
1	0	0	Error del bloque de configuración
1	0	1	Error de envío
1	1	1	Error de inicialización del sistema

Tabla 1: Tipos de errores posibles

Donde un 1 representa que el led está activo durante el parpadeo y un 0 que está apagado.

Además para el proceso de pruebas son necesarios mecanismos de escritura, lectura y borrado de la memoria FLASH del dispositivo, y para ello se han desarrollado tres aplicaciones distintas:

- **escribirFlash:** La primera de ellas se encarga de inicializar los datos de la memoria FLASH para aquellos dispositivos portadores de información; modifica el vector de partes para que todos sus valores sean TRUE y escribe el volumen de datos de la memoria FLASH con unos valores por defecto que representan el contenido.
- **leerFlash:** Se encarga de la lectura de la memoria; para ello se ha de conectar el dispositivo al puerto serie de un PC y ejecutarlo en el MICAz. La aplicación muestra por pantalla cada uno de los bytes de información; es especialmente importante el vector de bytes, ya que describe cada una de las partes que el nodo tiene almacenadas.
- **borrarFlash:** Elimina toda información del dispositivo, convirtiéndolo en un nodo receptor dentro de un escenario.

Con todos estos elementos se pasará a describir los distintos escenarios de prueba.

4.2 Descripción de los escenarios

Para la realización de las pruebas se han buscado distintos escenarios, con distintas topologías y distinto número de semillas/receptores; en total, son 7 los escenarios a estudiar, y se enumeran a continuación.

4.2.1 Escenario número 1

El primero de los escenarios a estudiar es el más simple de todos. En él hay dos dispositivos: uno de ellos será el portador de información (semilla) y el otro el receptor.
El objetivo de la primera prueba es verificar el correcto funcionamiento del protocolo, que servirá como base para los próximos (ver Figura 24).

Figura 24: Ilustración del escenario número 1

4.2.2 Escenario número 2

En el segundo escenario se tienen 4 dispositivos: uno de ellos portador de la información (semilla) y el resto serán receptores; sin embargo, no todos serán vecinos (la configuración se muestra a continuación).

El principal objetivo de la prueba es la de analizar el comportamiento del protocolo bajo condiciones de carga baja, con pocos nodos, pero donde no se obtiene un comportamiento completamente broadcast al no verse todos ellos. Esta configuración podría representar fácilmente una situación real como, por ejemplo, una carretera poco transitada con cierta separación entre los vehículos en un momento dado (ver Figura 25).

Figura 25: Ilustración del escenario número 2

4.2.3 Escenario número 3

Se tienen 16 dispositivos, vecinos entre todos ellos, donde uno de ellos es una semilla y el resto son nodos receptores.

El análisis de esta prueba es interesante, ya que representa el comportamiento de una vía relativamente masificada donde todos los vehículos están separados por una corta distancia; en posteriores escenarios, además, se estudiará esta misma configuración con más de una semilla (ver Figura 26).

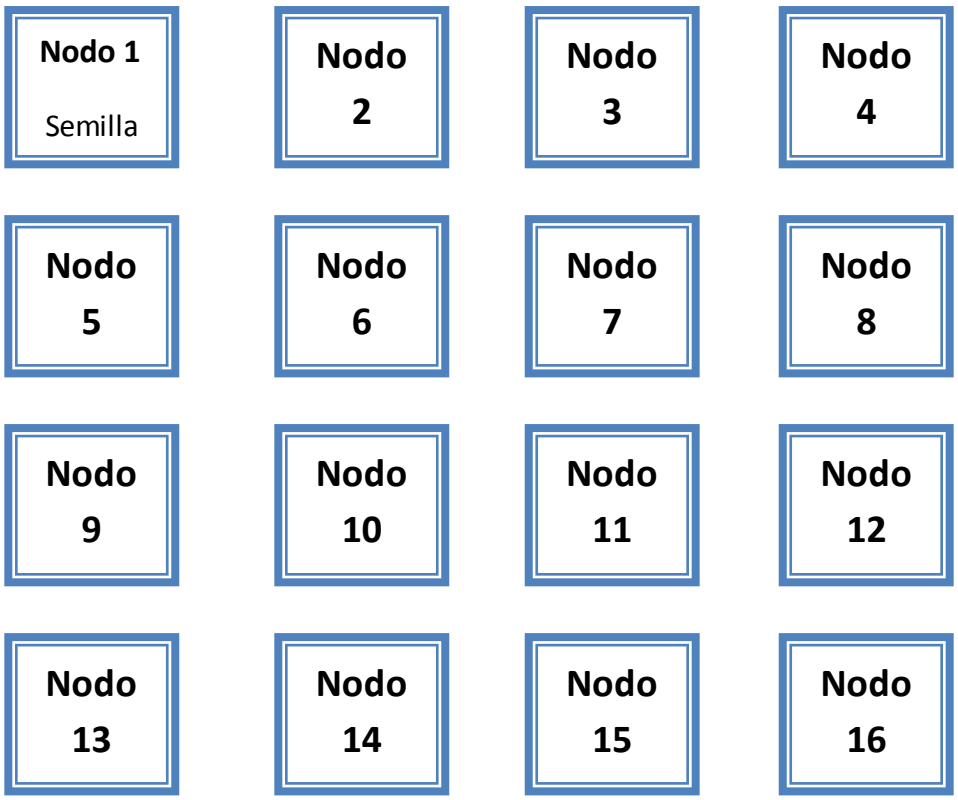

Figura 26: Ilustración del escenario número 3

4.2.4 Escenario número 4

Se tienen 16 dispositivos, vecinos entre todos ellos, donde dos de ellos son semillas y el resto son nodos receptores.

Esta prueba es parecida a la anterior; se intenta comparar el comportamiento con más de una semilla, ya que es previsible que este hecho produzca un incremento del tiempo necesario para la transmisión, debido al mayor número de mensajes intercambiados (ver Figura 27).

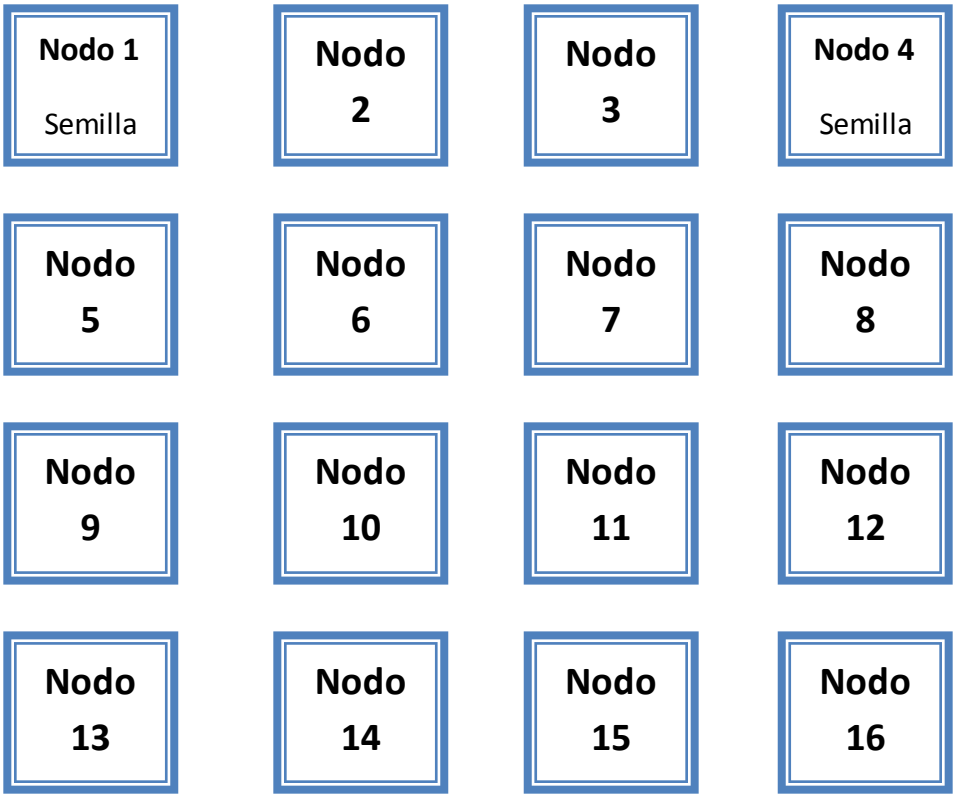

Figura 27: Ilustración del escenario número 4

4.2.5 Escenario número 5

Se tienen 16 dispositivos, vecinos entre todos ellos, donde cuatro de ellos son semillas y el resto son nodos receptores.

Del mismo modo que en la prueba anterior, se incrementa el número de semillas en el escenario para analizar el comportamiento del protocolo ante una alta carga de mensajes en el medio (ver Figura 28).

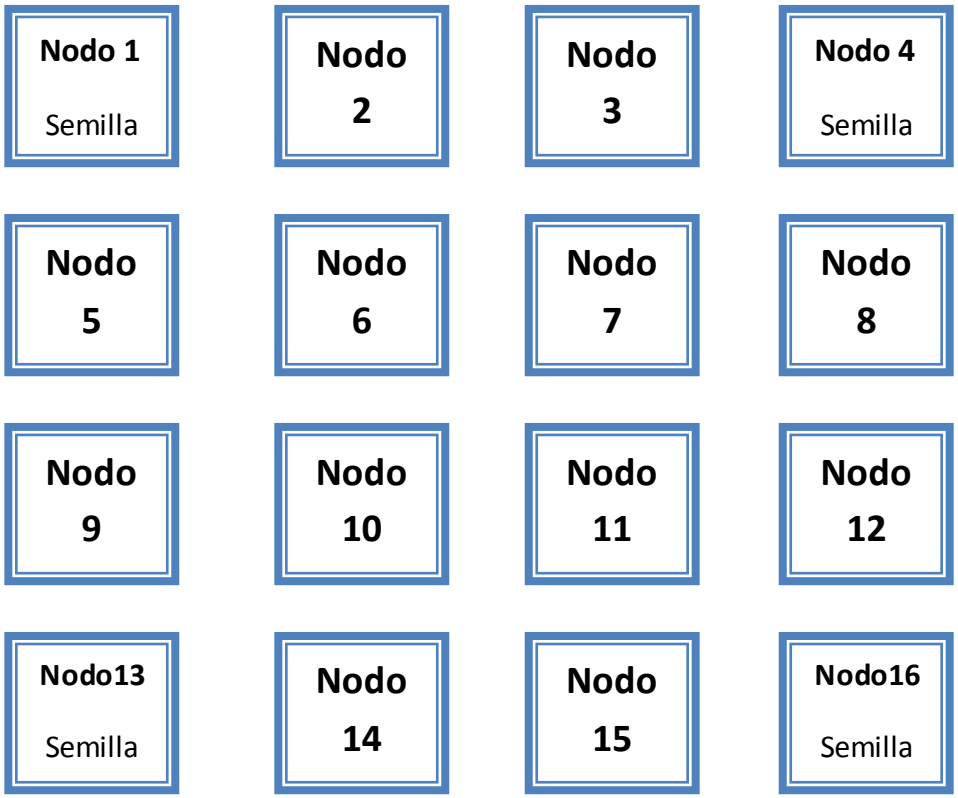

Figura 28: Ilustración del escenario número 5

4.2.6 Escenario número 6

Se tienen 16 dispositivos, donde cada uno de ellos es vecino del inmediatamente anterior y del inmediatamente posterior, con una única semilla en uno de los extremos.

Con esta prueba se pretende simular el comportamiento en línea del protocolo, similar al de una vía larga donde los vehículos están muy dispersos y separados unos de otros (ver Figura 29).

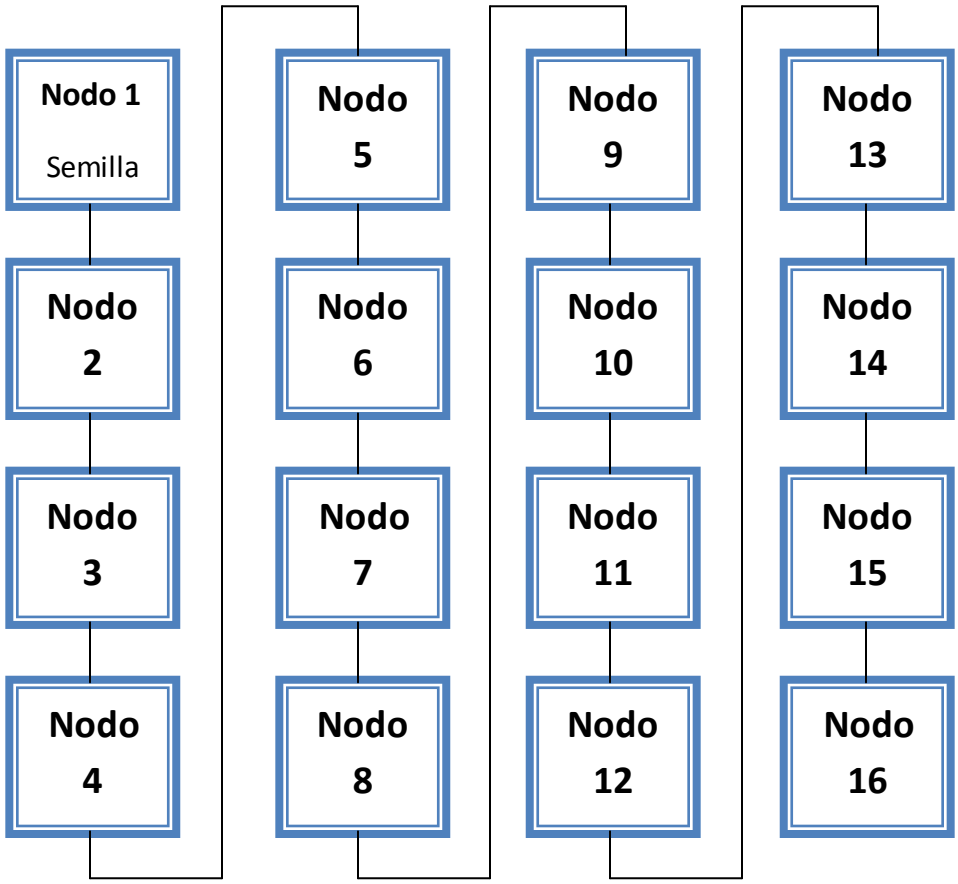

Figura 29: Ilustración del escenario número 6

4.2.7 Escenario número 7

En este último escenario se tienen 16 dispositivos, donde todos son vecinos entre ellos, y donde cada uno de ellos es un poseedor parcial del contenido.

Con esta prueba se pretende mostrar el comportamiento que el protocolo consigue en situaciones donde no hay un portador de la información completa, si no que cada uno de ellos posee una parte de ésta.

Es especialmente interesante para el estudio ya que, ante la imposibilidad de obtener la información al completo en una vía donde la red cambia continuamente, este mecanismo permite la rápida dispersión de la información en el medio (ver Figura 30).

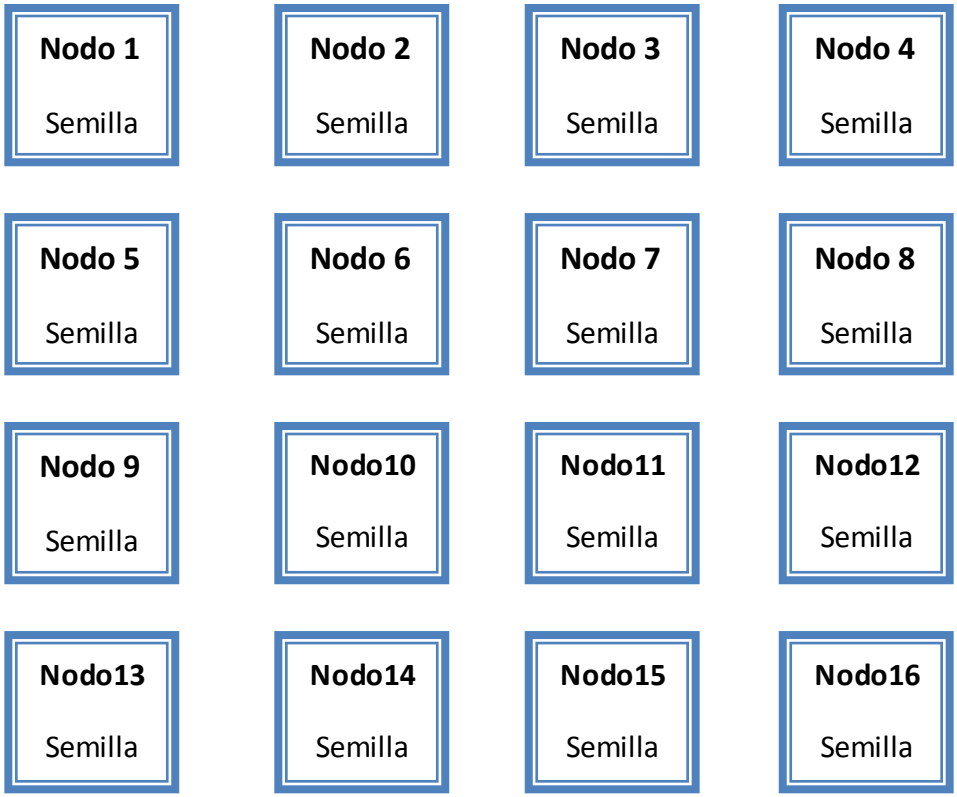

Figura 30: Ilustración del escenario número 7

4.3 Objetivo de las pruebas

El objetivo de cada una de las pruebas es, en primer lugar, verificar el correcto funcionamiento de la implementación del protocolo, y en segundo lugar, medir la eficiencia del mismo.
Para medir su eficiencia se ha cronometrado el número de segundos que transcurren desde el instante en que los dispositivos son iniciados hasta que todos ellos han obtenido por completo el contenido.

4.4 Realización de las pruebas

Para la realización de las pruebas se han establecido los períodos de cada uno de los temporizadores, así como el tamaño del contenido.

También se han forzado los vecinos de cada uno de los nodos modificando el código fuente de la aplicación principal, para cada uno de los escenarios.

4.4.1 Parámetros para las mediciones

- **Período de anuncio:** Corresponde al tiempo que transcurre entre dos anuncios; se ha establecido un tiempo de un segundo. Es importante recalcar que este tiempo disminuye considerablemente para el primer anuncio enviado tras finalizar la transmisión de una parte; en esta implementación el período es dividido entre 50 para que un portador de información la envíe por completo sin hacer pausas extremadamente largas entre cada una de las partes del contenido. La variación de este parámetro no influye en exceso en el resultado de las pruebas, y se ha de escoger dependiendo de la cantidad de vehículos que hay en el entorno en un momento dado.

- **Tiempo de espera de solicitudes:** Corresponde al tiempo que permanece esperando el portador de información tras enviar un anuncio para recibir las solicitudes de los nodos vecinos. Se ha seleccionado un tiempo de 75 milisegundos para este parámetro, suficiente como para que los nodos vecinos reciban el anuncio, lo almacenen y envíen su correspondiente solicitud. Es un parámetro muy importante, del cual depende la velocidad a la que la transmisión completa del contenido puede llevarse a cabo: debe ser lo suficientemente largo como para que los vecinos tengan tiempo de responder, sin llegar a serlo tanto como para que la transmisión se retarde en exceso.

- **Tiempo máximo para el inicio de la transmisión:** Corresponde al tiempo que transcurre entre que un nodo envía una solicitud y éste recibe el primer paquete de una parte; si dicho temporizador llega a su fin, el nodo abandona el intento de recepción. Se ha seleccionado un tiempo de 500 milisegundos. En las pruebas no influye este temporizador, ya que las condiciones son lo suficientemente buenas como para que no haya grandes pérdidas; no obstante es un parámetro interesante para una implementación real, ya que los nodos receptores no deberían esperar más de lo necesario, perdiendo la oportunidad de recibir información por parte de otros nodos emisores. Es importante tener en cuenta que este tiempo depende directamente del tiempo de espera de solicitudes por parte del emisor ya que, suponiendo una transmisión de duración nula, el nodo receptor tendrá que esperar dicho tiempo.

- **Tiempo de adaptación envío/escritura:** Corresponde al tiempo que transcurre entre el envío de dos paquetes por parte del nodo emisor; como se ha mencionado, el dispositivo es más lento escribiendo que leyendo, por lo que se ha de adaptar a la velocidad correspondiente. Es un parámetro muy importante cuyo incremento dentro del orden de los milisegundos puede incrementar considerablemente el tiempo final de transmisión del contenido. Tras numerosas pruebas se ha seleccionado un tiempo de 8 milisegundos, justo lo necesario como para que el receptor pueda escribir la información tras recibir cada paquete.

- **Número máximo de solicitudes recibidas:** Corresponde al tamaño de la cola, en bytes, que almacena las solicitudes recibidas por un emisor durante el tiempo de recepción de solicitudes. No influye directamente en las pruebas; no obstante es un parámetro a tener en cuenta en la implementación. Se ha seleccionado un tamaño de 15, suficiente como para almacenar las solicitudes de todos los nodos vecinos durante las pruebas.

- **Tamaño del contenido:** Corresponde al tamaño total de la información a transmitir. Se ha seleccionado un tamaño de 400 Kilobytes lo que, dividido entre el tamaño fijado para cada parte (6400 bytes) hace un total de 64 partes. Obviamente, del tamaño del contenido depende directamente el tiempo total de transmisión; se ha seleccionado un tamaño cercano al máximo posible de estos dispositivos, cuya memoria FLASH es de 512 Kilobytes.

4.4.2 Forzando los vecinos

Debido a la dificultad a la hora de establecer distancias reales entre los dispositivos para que éstos entren dentro del alcance o no del resto, se ha optado por una modificación del código de forma que se simula dicha distancia.

Se ha realizado una "tabla de vecinos" para cada uno de los nodos, de forma que el comportamiento sea distinto para los mensajes procedentes de nodos que no pertenezcan a dicha tabla.

- Se ignoran los anuncios de los nodos no-vecinos; de esta forma se evitará el envío de solicitudes a dichos nodos.
- Se ignoran las partes procedentes de los nodos no-vecinos; si no fuera así, los receptores almacenarían la parte recibida independientemente de que hubiera contestado al anuncio anteriormente o no.

Para evitar un código fuente distinto para cada uno de los nodos se ha implementado una función (dependiente de la prueba), *esVecino*, que devuelve TRUE en caso de que el nodo pasado como parámetro sea vecino del nodo programado (que se puede obtener gracias a la macro *TOS_NODE_ID*) y FALSE en caso contrario.

Se emplearán tres implementaciones distintas para llevar a cabo las pruebas:

- Para los escenarios 1, 3, 4, 5 y 7 se mantiene el código original, no se ignora ningún tipo de mensaje ni se establece la tabla de vecinos debido a que todos los nodos del escenario se ven.

- Para el escenario 2, se ignorarán los anuncios y las partes procedentes de los nodos que no cumplan una las siguientes condiciones:
  - Su dirección es dos unidades mayor.
  - Su dirección es dos unidades menor.
  - Es múltiplo de dos, y su dirección es una unidad menor.
  - No es múltiplo de dos, y su dirección es una unidad mayor.

- Para el escenario 6, se ignorarán los anuncios y las partes procedentes de los nodos que no cumplan una de las siguientes condiciones:
  - Su dirección es una unidad mayor.
  - Su dirección es una unidad menor.

Para cada uno de los escenarios se han realizado los siguientes pasos:

- Se han programado los nodos semilla con la aplicación *escribirFlash*, y posteriormente se ha ejecutado.
- Se han programado los nodos receptores con la aplicación *borrarFlash* y posteriormente se ha ejecutado.
- Se han programado todos los nodos con la implementación del protocolo correspondiente a cada uno de los distintos escenarios.
- Se ha medido el tiempo transcurrido entre el inicio de los dispositivos y el momento en el que todos han obtenido por completo el contenido; esta medida se ha hecho cinco veces por cada uno de los escenarios, para la obtención de la media y la varianza.
- Se han programado todos los nodos con la aplicación *leerFlash* y posteriormente se ha ejecutado, para extraer el contenido de la memoria FLASH y verificar la correcta recepción de la información.

Es interesante mencionar la modificación aplicada a la aplicación *escribirFlash* en el caso del escenario número 7; debido a que no existe un portador de la información en dicho escenario, era necesario modificar el vector de partes dependiendo de la dirección de cada uno de los nodos. Para evitar una aplicación distinta para cada uno de ellos, se ha establecido que las partes de cada nodo están en el rango comprendido entre **[**(TOS_NODE_ID - 1) * 4 **,** (TOS_NODE_ID * 4) – 1**]**, de forma que el primero de ellos posee las partes [0…3], el segundo [4…7], y así consecutivamente hasta las 64 partes que forman el contenido.

4.5 Resultados

4.5.1 Escenario número 1

	Tiempo (min)	Media	Varianza
Prueba 1	2:08.391		
Prueba 2	2:08.61		
Prueba 3	2:08.201	2:08,329	0,0347697
Prueba 4	2:08.13		
Prueba 5	2:08.312		

Tabla 2: Resultados del escenario número 1

Como se puede observar los tiempos son prácticamente los mismos para cada una de las cinco simulaciones (ver Tabla 2).

En esta prueba, con dos MICAz y una semilla, se puede observar el comportamiento del protocolo ante una transmisión punto a punto, del mismo modo que una transmisión unicast; es previsible que, en este caso, una versión unicast del protocolo daría mejores resultados, debido a que queda eliminada la necesidad de enviar un anuncio entre cada una de las partes, así como la espera de recepción de solicitudes por parte de la semilla (a pesar de que dicha diferencia no debería ser muy alta).

Podría considerarse una situación de este tipo como ideal, ya que el medio está ocupado completamente por el nodo semilla, libre de colisiones, y la transmisión del contenido se realiza de principio a fin sin paradas.

En este caso, además, no afecta cuál de los vehículos se incorpore antes a la vía: si es la semilla, ésta permanecerá enviando anuncios hasta la recepción de una solicitud; si es el receptor, éste quedará a la espera de la recepción de un anuncio.

4.5.2 Escenario número 2

	Tiempo (min)	Media	Varianza
Prueba 1	4:15.201		
Prueba 2	4:21.861		
Prueba 3	4:22.123	4:18.166	12,4467427
Prueba 4	4:15.232		
Prueba 5	4:16.412		

Tabla 3: Resultados del escenario número 2

En este segundo escenario se observan variaciones en el número de segundos que transcurren hasta que los cuatro nodos completan el contenido (ver Tabla 3).

En la prueba se ha supuesto que los tres nodos receptores se encuentran en la vía y es el nodo semilla el que se incorpora posteriormente; bajo este supuesto el nodo semilla comienza la transmisión ofertando un anuncio a sus dos nodos vecinos, y éstos responden solicitando una de las partes. La semilla, posteriormente, envía una de ellas al medio y ambos nodos almacenan la parte. A partir de este punto se repite el proceso hasta que ambos nodos obtienen por completo la información.

El cuarto nodo no llegó a recibir los anuncios de la semilla debido a que ésta no es su vecino; por tanto, comienza la recepción de la información por su parte en el instante en que el segundo y tercer nodo terminan de recibir el contenido.

Se ha observado en las pruebas que el cuarto nodo no recibe la información siempre del mismo; a veces lo recibe del segundo, y a veces del tercero. Esto depende únicamente de cuál de ellos envíe en primer lugar el anuncio y, suponiendo colisión, dependerá entonces del algoritmo de *backoff* implementado en el nivel de enlace de la pila de protocolos. No obstante, nuevamente el último nodo recibe cada una de las partes del contenido de forma continuada y sin paradas, por lo que, aproximadamente, el tiempo de transmisión total para todos los nodos del escenario es el doble que en el escenario anterior, al haber en este caso dos transmisiones.

En este caso afectaría el hecho de que el segundo o tercer nodo se incorpore más tarde a la vía, ya que dicho nodo perdería alguna de las partes transmitidas al medio y tendría que volver a solicitarlas posteriormente, retrasando ligeramente el tiempo total de transmisión.

4.5.3 Escenario número 3

	Tiempo (min)	Media	Varianza
Prueba 1	2:08.635		
Prueba 2	2:08.234		
Prueba 3	2:08.67	2:08.485	0,0826978
Prueba 4	2:08.12		
Prueba 5	2:08.764		

Tabla 4: Resultados del escenario número 3

Nuevamente, al igual que en el primer escenario, el tiempo es de dos minutos y ocho segundos y a penas varía entre cada una de las pruebas (ver Tabla 4).

De la misma forma que en el resto de los escenarios, se ha supuesto que los nodos receptores se encontraban en la vía en el momento en que la semilla se incorpora.

En esta prueba se pueden ver claramente las ventajas de un protocolo broadcast frente a uno unicast en una situación en la que hay una gran cantidad de vehículos: basta con que uno de ellos tenga la información para que el resto pueda obtenerla al mismo tiempo, independientemente de cuántos haya.

Se puede observar que el comportamiento del protocolo será el mismo para uno, dos o más nodos receptores, ya que éstos envían sus respectivas solicitudes al mismo tiempo y reciben cada una de las partes de forma simultánea.

4.5.4 Escenario número 4

	Tiempo (min)	Media	Varianza
Prueba 1	2:09.090		
Prueba 2	2:09.560		
Prueba 3	2:09.980	2:09.440	0,1342573
Prueba 4	2:09.457		
Prueba 5	2:09.111		

Tabla 5: Resultados del escenario número 4

Se pretende con esta prueba observar el comportamiento del escenario anterior cuando aumenta el número de semillas.

En este caso el tiempo total necesario para la transmisión ha aumentado ligeramente, aunque la diferencia es casi inapreciable (ver Tabla 5).

Es debido a la existencia de dos nodos emisores: ahora, dependiendo del orden en que se incorporen a la vía los distintos nodos receptores, éstos solicitarán el contenido a una de las dos, por lo que probablemente se llegará a una situación en la que ambas semillas estarán transmitiendo partes al mismo tiempo y esto producirá colisiones que retardarán el tiempo de transmisión.

4.5.5 Escenario número 5

	Tiempo (min)	Media	Varianza
Prueba 1	2:11.963		
Prueba 2	2:11.231		
Prueba 3	2:11.61	2:11.547	0,0898283
Prueba 4	2:11.65		
Prueba 5	2:11.279		

Tabla 6: Resultados del escenario número 5

Nuevamente se desea probar el comportamiento del sistema incrementando el número de semillas.

De igual forma, se vuelve a observar un incremento en el tiempo total de transmisión, aunque no de forma significativa (ver Tabla 6).

Se puede extraer de las últimas pruebas que el hecho de que haya más semillas no afecta positivamente al resultado, al contrario de lo que podría pasar en un protocolo unicast, donde el hecho de haber más semillas implica que habrá más receptores en un instante determinado; en este caso el número de receptores es independiente del número de semillas.

No obstante, es interesante mencionar el hecho de que estos resultados dependerán directamente del orden en que los dispositivos sean conectados; en caso de activar los nodos receptores en primer lugar, éstos se conectarán a la semilla que envíe el primer anuncio, por lo que el resultado no se verá afectado con respecto al primer escenario (las otras tres semillas permanecerán enviando anuncios que no obtendrán respuesta).

4.5.6 Escenario número 6

	Tiempo (min)	Media	Varianza
Prueba 1	30:11:165		
Prueba 2	30:41:496		
Prueba 3	30:51:124	30:35,410	232,984189
Prueba 4	30:41:98		
Prueba 5	30:31:289		

Tabla 7: Resultados del escenario número 6

Se ha estudiado el comportamiento del sistema en un escenario en el que todos los dispositivos se encuentran en línea, uno delante del otro, y donde en uno de los extremos se ha situado la semilla.

Se puede observar que en este caso los tiempos son extremadamente elevados en comparación con el resto de pruebas, debido a que se merman por completo las capacidades broadcast de la implementación: en esta ocasión sólo hay una transmisión simultánea, y por tanto un nodo no puede comenzar a transmitir su información hasta que el anterior se la ha transmitido a dicho nodo (ver Tabla 7).

Viendo las pruebas que conciernen al primer escenario se podría pensar que el tiempo necesario para completar la transmisión en este último es de 128*15 segundos (aproximadamente unos dos minutos) debido a que en este caso tenemos una transmisión unicast multiplicada por 15 nodos receptores; sin embargo, el tiempo se reduce ligeramente.

Este hecho es así debido a que, en ocasiones, un nodo puede comenzar a transmitir su contenido parcial al siguiente (sin haber recibido por completo la información) lo que produce un pequeño reparto del trabajo entre los siguientes nodos.

4.5.7 Escenario número 7

	Tiempo (min)	Media	Varianza
Prueba 1	4:40.12	4:37.353	130,495138
Prueba 2	4:24.225		
Prueba 3	4:44.123		
Prueba 4	4:51.183		
Prueba 5	4:27.112		

Tabla 8: Resultados del escenario número 7

Por último, en este escenario se pretende estudiar el comportamiento del protocolo cuando no hay semilla, si no que el contenido se reparte entre todos los nodos de la red.

En este caso, todos los nodos tienen algo que ofrecer, y por lo tanto enviarán anuncios al mismo tiempo hasta recibir una solicitud del resto.

Como se puede observar las variaciones en cuanto al número de segundos necesarios para completar la transmisión varía notablemente en cada una de las cinco pruebas que se han realizado con 16 nodos vecinos (ver Tabla 8). La razón es que, al ser todos portadores de información, han de ponerse de acuerdo a la hora de comenzar el envío de una parte, y esto produce colisiones entre los anuncios y tiempos de espera variables; no obstante, debido a que todos están interesados en el contenido del resto, no se observan transmisiones de partes simultáneas: un dispositivo termina de enviar todo su contenido, dando paso al siguiente, hasta que todos obtienen la información al completo.

El comportamiento del protocolo dado este escenario es justamente el esperado: no es necesario que haya un portador de la información completa para que la transmisión pueda llevarse a cabo correctamente de forma completa.

5.Conclusiones

5.1 Resultados de la implementación

La más inmediata conclusión a la que se llega tras la finalización de las distintas pruebas es que la transmisión de grandes bloques de información en un entorno móvil como lo es una vía de tráfico es perfectamente posible; en este caso se ha podido probar mediante una estrategia broadcast.

Como se ha observado a lo largo de los distintos escenarios esta estrategia es especialmente efectiva cuando se trata de una vía con una gran cantidad de vehículos. Si se comparan los escenarios 1 y 3, se observa que el tiempo de transmisión del contenido completo a los vehículos vecinos es independiente del número de éstos; sin embargo, se puede ver en el escenario número 2 que el tiempo total aumenta considerablemente si los vehículos están dispersos en la vía sin haber una visión directa entre todos ellos. Este hecho es debido a que las capacidades broadcast del protocolo quedan mermadas al no existir una conexión directa entre todos los vehículos del escenario (la información ha de pasar por un nodo concreto para que ésta alcance a los más lejanos). Este efecto se vería incrementado en situaciones con mayor separación entre los vehículos.

En los escenarios 4 y 5 se ha podido observar el efecto que tiene sobre la transmisión el hecho de existir un número mayor de semillas: éstas interfieren entre ellas al ser vecinas, lo que produce retardos (aunque mínimos) en el tiempo para completar la transmisión del contenido. No obstante, y como se ha mencionado anteriormente, sólo influirá si las semillas ofertan sus partes al mismo tiempo o si los vehículos receptores se incorporan a la vía en un orden no arbitrario, ya que en caso contrario todos los nodos solicitarán una parte a la primera semilla que envíe el anuncio y el mecanismo de compromiso entre transmisor y receptor impedirá que soliciten partes al resto de semillas.

En cuanto al escenario número 6 se ha comprobado que el comportamiento en línea (y por tanto sin varios receptores al mismo tiempo en ningún momento) es el punto débil de un protocolo de este tipo, donde el tiempo para completar la transmisión es proporcional al número de nodos que formen el escenario.

Por último, en el escenario número 7 se ha verificado el comportamiento del protocolo cuando no hay un nodo portador de la información completa, si no que ésta se distribuye homogéneamente entre todos los nodos. La aplicación funcionaría correctamente aunque esta distribución no fuera homogénea: puede haber nodos sin ninguna parte, otros con el contenido casi completo, así como otros que tengan la

mitad. No es importante, porque en cualquier caso la información tiende a completarse en todos y cada uno de ellos.

Cabe decir que hay ciertas situaciones que no se han podido representar en este trabajo debido a la imposibilidad de recrear una situación real mediante el método de forzado de los vecinos; un claro ejemplo sería el del escenario número dos extendido hasta 16 dispositivos, cuya ilustración sería la siguiente (ver Figura 31).

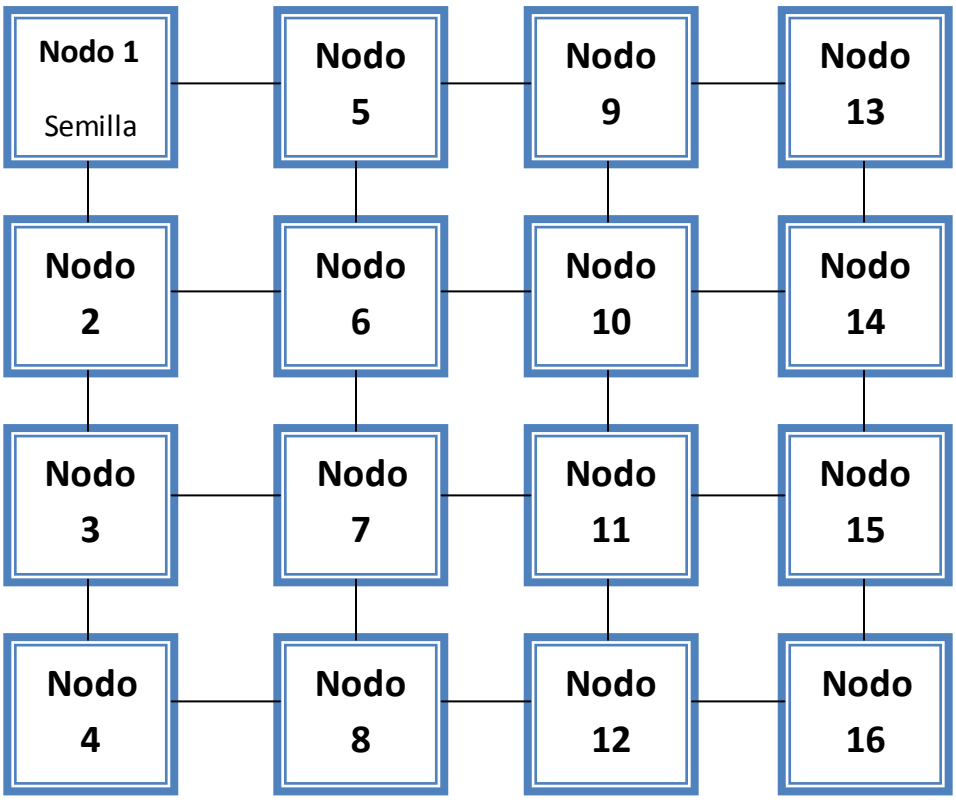

Figura 31: Escenario número 2 extendido a 16 nodos

Como se puede predecir, Nodo 1 enviaría la información a sus nodos vecinos y, una vez completada, tanto Nodo 5 como Nodo 2 harían lo mismo.

El problema que surge es que, mientras que en una prueba real en una vía de tráfico Nodo 2 y Nodo 5 no se ven en absoluto, en este caso, a pesar del forzado de vecinos (se ignoran el uno al otro) siguen interfiriéndose entre ellos: sus paquetes de datos, así como sus anuncios y solicitudes colisionarían de forma aleatoria con los del otro, y por tanto el resultado de la prueba no representaría el mismo en una implementación real.

Este hecho se hace más acusado conforme avanza la transmisión, llegando a haber hasta cuatro transmisiones "simultáneas".

En resumen, se ha alcanzado el objetivo esperado (que era el de realizar transmisiones de grandes bloques de información utilizando dispositivos específicos para redes de

sensores inalámbricas) de una forma eficiente y llegando a resultados válidos y viables en un posible uso real.

5.2 Posibles mejoras en estudios futuros

Las posibles mejoras aplicables a la implementación surgen de la evolución de los propios dispositivos empleados o de la variación de los distintos parámetros modificables del protocolo, que pueden ayudar a reducir los tiempos de transmisión en cada uno de los escenarios.

Una mejora interesante sería la de eliminar los tiempos de parada existentes entre la transmisión de dos paquetes consecutivos de una parte; como se ha visto en apartados anteriores existe un tiempo de parada de 8 milisegundos para dar tiempo a los receptores a almacenar la información en la memoria FLASH.

Esta necesidad podría eliminarse, por ejemplo, con una memoria RAM mayor; los dispositivos MICAz empleados en este trabajo poseen una memoria RAM de 4 Kilobytes, la cual se mantiene ocupada en buena parte por las distintas variables del programa, dejando un espacio insuficiente como para almacenar en ella toda una parte (que se ha seleccionado, por los motivos expuestos anteriormente, de 6400 bytes).

Una memoria más grande (posiblemente el doble) sería suficiente como para almacenar la información recibida en la memoria RAM, y volcar este contenido a la FLASH únicamente en el momento en que se ha completado la recepción de las 100 subpartes que componen cada parte. Esto da ciertas ventajas:

- Deja de existir la necesidad de esperar entre dos subpartes, ya que el receptor es capaz de almacenarlas a una velocidad mucho mayor.
- Simplifica el código considerablemente, eliminando la necesidad de controlar las escrituras de cada subparte (y por tanto, gestionar los distintos errores derivados de este control).

Eliminando dicha necesidad la transmisión de la información en el escenario número 1 (transmisión punto a punto) se reduce en 8*99 milisegundos, aproximadamente unos 48 segundos menos (o lo que es lo mismo, una reducción del 37,5 %). Esta reducción será mayor cuantas más transmisiones totales se realicen en el escenario.

Otra mejora, esta vez por parte del sistema operativo TinyOS, sería la de poder gestionar los mensajes recibidos a una velocidad mayor o, en su defecto, la de poder gestionar varios simultáneamente.

El evento *receiveDone* explicado anteriormente, al ser de tipo *síncrono*, su señalización no interrumpe cualquier otra tarea que estuviera haciendo el dispositivo en el momento de recibir un mensaje.

Esto produce pequeñas situaciones indeseables como por ejemplo, la de que un nodo receptor esté ocupado atendiendo el anuncio de un nodo en el escenario cuando, en ese mismo momento, un nodo emisor está transmitiendo la siguiente parte que esperaba; dicha parte se perderá ya que el nodo receptor no sabrá de su existencia al estar ocupado preparando el rechazo del anuncio anterior. Esta situación podría evitarse si la recepción de un mensaje interrumpiese la tarea que se estaba llevando a cabo.

Finalmente, mencionar el hecho de que es posible alterar los parámetros de la implementación para conseguir tiempos de transmisión óptimos dependientes de la vía en la que uno se encuentre. Algunos ejemplos interesantes:

- Variar el período de anuncio *TANN* dependiendo del número de vehículos que respondieron anuncios anteriores (por ejemplo, si no hay vehículos en la vía, no es necesario enviarlos cada segundo).
- Variar el tiempo de espera de recepción *TPART*, dependiendo de la calidad de la conexión con el nodo emisor en un momento dado.
- De forma análoga al punto anterior, se podría modificar el tiempo de espera de solicitudes *TREQ*, reduciendo en ciertas situaciones el tiempo total de transmisión.

Bibliografía

1. **Santos Leiva, Raul.** *Simulación de VANETS (Vehicular Ad-Hoc Networks).* Castelldefels : s.n., 2007.

2. **Aladrén, Domingo.** *Diferenciación de servicios y mejora de la supervivencia en redes ad-hoc conectadas a redes fijas.* Castelldefels : s.n., 2005.

3. **VANET 2006, The Third ACM International Sponsored by ACM SIGMOBILE.** *Workshop on Vehicular Ad Hoc Networks.* Los Angeles : s.n., 2006.

4. **Eichler, S., Schroth, C., Eberspächer, J.** *Car-to-Car Communication.* Technische Universität München, München : s.n., 2006.

5. Vehicular Ad-Hoc Network (VANET). [En línea] http://en.wikipedia.org/wiki/VANET.

6. ZigBee. [En línea] http://es.wikipedia.org/wiki/ZigBee.

7. Página web oficial de Crossbow. [En línea] http://www.xbow.com.

8. **Crossbow.** *MICAz Datasheet.*

9. Página web oficial de TinyOS. [En línea] http://www.tinyos.net/.

10. Página web oficial de NesC. [En línea] http://nescc.sourceforge.net/.

11. Specification and Description Language. [En línea] http://en.wikipedia.org/wiki/Specification_and_Description_Language.

12. Página web oficial de Telelogic. [En línea] http://www.telelogic.com/.

13. Wiki de XubunTOS. [En línea] http://toilers.mines.edu/Public/XubunTOS.

14. Web de Technology Review (MIT). [En línea] http://www.technologyreview.com/es/.

15. **Hill, Jason.** *System Architecture for Wireless Sensor Networks.* University of California : s.n., 2003.

16. **D. Estrin, R. Govindan, J. Heidemann, S. Kumar.** *Next Century Challenges: Scalable Coordination in Sensor Networks.* MOBICOM : s.n., 1999.

17. Wireless Sensors Networks. [En línea] http://en.wikipedia.org/wiki/Wireless_sensor_network.

18. **Net2 Working Group (Philip Levis, Gilman Tolle).** *TEP 118: Dissemination of Small Values.*

19. **Hamann, Alexander.** *Ad-Hoc Networks.* 2004.

20. **Net2 Working Group (Rodrigo Fonseca, Omprakash Gnawali, Kyle Jamieson, Philip Levis).** *TEP 119: Collection.* 2006.

21. **Network Working Group (Rodrigo Fonseca, Omprakash Gnawali, Kyle Jamieson, Sukun Kim, Philip Levis, Alec Woo).** *TEP 123: The Collection Tree Protocol (CTP).* 2006-2007.

22. **Network Protocol Working Group (Omprakash Gnawali).** *TEP 124: The Link Estimation Exchange Protocol (LEEP).* 2006-2007.

23. Wiki de TinyOS. [En línea] http://docs.tinyos.net/index.php/Tymo.

24. Página web oficial de CitySense. [En línea] http://www.citysense.net/.

25. **Pastor González, Luis.** *Protocolo para la diseminación de información en una red vehicular mediante una estrategia unicast.* Universidad Politécnica de Cartagena : s.n., 2009.

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.