Technical University of Madrid

Higher Technical School of
Telecommunications Engineering

FINAL THESIS REPORT

M.S. Degree in Telecommunication Engineering

Development of a multiple RF interfaced platform for Cognitive Wireless Sensor Networks

Agustín Tena García

September 2013

Documento formatted with TEXIS v.1.0+.

This document is prepared for duplex printing.

Final Thesis Report

Title:	Development of a multiple RF interfaced platform for Cognitive Wireless Sensor Networks
Author:	Agustín Tena García
Advisor:	Elena Romero Perales
Inspector:	Alvaro Araujo Pinto
Departament:	Electronic Engineering Department

Board of Examiners

President:	D. Álvaro de Guzman Fernández González
Member:	D. Alvaro Araujo Pinto
Secretary:	D. Pedro José Malagón Marzo
Assistant:	D. Miguel Ángel Sánchez García

SCORE:

Se certifica que Agustín Tena García ha completado con éxito su Trabajo Fin de Máster obteniendo así la titulación de Máster Oficial en Ingeniería de Sistemas Electrónicos de la ETSI Telecomunicación, Universidad Politécnica de Madrid, España.

Madrid, a de de

Development of a multiple RF interfaced platform for Cognitive Wireless Sensor Networks

AUTHOR: Agustín Tena García

ADVISOR: Elena Romero Perales

INSPECTOR: Alvaro Araujo Pinto

Electronic Engineering Department
Higher Technical School of Telecommunications Engineering
Technical University of Madrid

September 2013

Dedicatoria

Acknowledgements

Abstract

Nowadays Wireless Sensor Networks (WSNs) are subject to development constraints and difficulties such as the increasing RF spectrum saturation. This brings hindrances to Wireless Ad-Hoc Sensor Networks (WAHSNs) deployment, especially for critical and sensitive applications.

Cognitive Networks (CN), leaning on a cooperative communication model, represent a new paradigm aimed at improving wireless communications. Cognitive Wireless Sensor Networks (CWSNs) compound cognitive properties into common WSNs, developing new strategies to mitigate difficulties arising from the constraints these networks face regarding energy and resources.

It is important to investigate cognitive models to explore their benefit over our WAHSNs. However, few platforms allow their study due to their early research stage, and they still show scarce or specific features. Investigations take place mainly over simulators, which provide partial and incomplete results.

This paper presents a versatile platform that brings together cognitive properties into WSNs. It combines hardware and software modules as an entire instrument to investigate CWSNs. The hardware fits WSN requirements in terms of size, cost and energy. It allows communication over three different RF bands, becoming the only cognitive platform for WSNs with this capability. Besides, its modular and scalable design is widely adaptable to almost any WAHSN application.

KEY WORDS: *cognitive, wireless sensor networks, platform.*

Index

List of Figures

List of Tables

Chapter 1

Introduction

Frase
Fuente

Resumen: Resumen

Chapter 2

Cognitive Wireless Sensor Networks: State of Art

Frase
Fuente

Resumen: Resumen

Chapter 3

Review Study

Frase
Fuente

Resumen: Resumen

Chapter 4

Design Study

Frase
Fuente

Resumen: Resumen

Chapter 5

Software

No fracasé, sólo descubrí 999 maneras
de cómo no hacer una bombilla.

Thomas Alva Edison

Resumen: En este capítulo se describe el software desarrollado para el módulo de entrega remota de datos del sistema LIDO DCL. Concrétamente, este módulo se empleará en la implementación de dicho sistema sobre boyas autónomas en el mar, donde cada una actuará como un nodo dentro de una red ad hoc de datos. En primer lugar, se explica la funcionalidad de dicho módulo. A continuación, se realiza un análisis cualitativo de los tipos de protocolos de encaminamiento existentes para esta clase de redes, centrándose en el AODV y versiones basadas en él. Se presenta el AODV-LAB, una nueva implementación del AODV diseñada para este proyecto. Por último, se detalla el desarrollo en C/Linux del módulo de entrega remota de datos, se simula el mismo para validar las funcionalidades deseadas y se comentan los resultados obtenidos.

5.1. Módulo de entrega remota de datos

Como ya se vió en la Sección **??**, el sistema LIDO DCL está orientado al estudio y mitigación del impacto ambiental de la contaminación acústica en el medio marino. Se encarga de la aquisición de datos acústicos mediante hidrófonos para, a continuación, realizar el procesado de los mismos y, finalmente, en caso de detectar algún evento acústico definido inicialmente, envíar una alarma con dicha información a un nodo colector.

La primera parte del sistema, correspondiente a las etapas de adquisición, procesado de audio y generación de alarmas, ya ha sido desarrollada y probada en el LAB (*Laboratorio de Aplicaciones Bioacústicas*). La idea de este proyecto ha sido, por lo tanto, trabajar a partir ahí para desarrollar finalmente un prototipo del sistema completo capaz de implementarse sobre boyas autónomas. Concretamente, el software implementado se encarga de:

- Detectar la aparición de nuevas alarmas en el sistema.
- Descubrir rutas hacia el nodo colector empleando un protocolo de encaminamiento.

- Envío de las alarmas a través de dichas rutas.
- Verificar la correcta entrega de las mismas mediante los acuses de recibo.
- Indicar las coordenadas de la boya en caso de que el nodo colector lo solicite.

5.1.1. Envío, enrutado y acuse de recibo de alarmas

Una vez que el sistema detecta un determinado evento acústico genera la alarma correspondiente y la añade, como una nueva línea, en un fichero que contiene todas las alarmas. Este archivo es constantemente chequeado por el software implementado en este módulo, a la espera de detectar nuevas entradas. Cuando se detecta una nueva alarma se mira la tabla de rutas por si existe algún camino válido hacia el nodo colector y, si es así, se envía directamente. En caso contrario, se inicia el proceso de Descubrimiento de Ruta. Si éste finaliza satisfactoriamente existirá, por lo tanto, una ruta válida bidireccional entre ambos nodos, y se enviará la alarma. Si no encuentra un camino entre ellos, esperará un determinado tiempo, fijado en la configuración del sistema, y volverá a comenzar el Descubrimiento de Ruta.

Cuando se envía una alarma no se manda la siguiente hasta que no se recibe el correspondiente acuse de recibo de la primera. Si, tras un determinado tiempo también fijado en la configuración, no hay noticia del nodo colector se volverá a enviar la misma alarma nuevamente. Este proceso se repetirá hasta que se reciba el correspondiente acuse de recibo o, en su defecto, se agote el número de reintentos determinados en la configuración del sistema. En este último caso, se descartará dicha alarma, ya que tras los reintentos y esperas dejará de ser útil. Únicamente, en caso de existir nuevas alarmas esperando, se iniciará el proceso de Descubrimiento de Ruta de nuevo.

Al trabajar todos los nodos con un protocolo de encaminamiento de redes ad hoc, cada uno de ellos tiene capacidad de enrutamiento y, por lo tanto, reenviará las alarmas de otros nodos que pasen por él. Por último, destacar que el nodo colector genera un registro de todas las alarmas que recibe durante la ejecución. De esta manera, es posible disponer de dicha información para su estudio más adelante.

5.1.2. Localización de nodos

Cada nodo de la red irá implementado en una embarcación, estructura flotante o, la mayor parte de las veces, en boyas. En muchas aplicaciones estas no irán ancladas al fondo marino sino que se podrán desplegar y recoger desde pequeñas embarcaciones y, durante el periodo de operación, estarán flotando libres en el mar. Por esta razón, se ha previsto que cada nodo de la red incluya un sistema de GPS (*Global Positioning System*) que pueda obtener en cada momento sus coordenadas geográficas exactas.

El nodo colector tiene la posibilidad de enviar una solicitud de posición en modo broadcast que llegue a todos los nodos de la red para que cada uno responda mediante un mensaje con sus coordenadas y la fecha y hora exacta de la consulta. Estos mensajes, incialmente, se mandan únicamente bajo demanda pero, dependiendo de las aplicaciones finales que pueda tener el sistema, se podrá modificar para que se envíen periódicamente o según interese.

De esta manera, desde el nodo colector se puede realizar un registro de las posiciones de los demás nodos de la red. Esta información servirá de apoyo al estudio realizado con el registro de alarmas y, por supuesto, en muchos casos será vital para la recogida de las boyas al finalizar su utilización en el mar.

5.2. Protocolos de encaminamiento

Tal y como se refirió en la Sección **??**, las redes ad hoc ofrecen grandes posibilidades, sobre todo en aquellos medios donde no existe una infraestructura de red preexistente. El diseño de protocolos de encaminamiento para este tipo de redes es actualmente un gran desafío debido a su naturaleza cambiante: tamaño, densidad de tráfico, rutas entre nodos, etc.

Existen una serie de critérios importantes a tener en cuenta en la selección o implementación de un protocolo de encaminamiento para una red ad hoc (Domingo-Aladrén, 2005):

- Señalización mínima:
  La reducción del número de mensajes de control y del tamaño de estos ayuda a conservar la batería y el porcentaje de ciclo de trabajo disponible para transmitir, y a reducir la contienda de acceso al medio y, por lo tanto, a mejorar la comunicación entre los nodos.
- Tiempo de procesamiento mínimo:
  Cuanto más simples sean las operaciones computacionales realizadas por el protocolo menor será el tiempo de procesamiento necesario y, por tanto, las baterías tendrán mayor tiempo de vida.
- Mantenimiento en condiciones de topología dinámica:
  El protocolo debe ser capaz de detectar rupturas en los enlaces y detectar nuevas rutas alternativas lo antes posible.
- Modo de operación distribuido:
  Propiedad esencial de las redes ad hoc.
- Libre de bucles:
  Se debe evitar la existencia de paquetes perdidos circulando por la red, así como el envío de mensajes repetidos.

Existe una gran variedad de protocolos de encaminamiento para redes ad hoc. Con el objetivo de realizar la elección del más apropiado para su empleo en el módulo de entrega remota de datos del sistema LIDO DCL, en la siguiente subsección se realiza una posible clasificación de ellos atendiendo a su modo de creación de rutas bajo demanda o no.

5.2.1. Protocolos de encaminamiento proactivos, reactivos e híbridos

Una de las clasificaciones de protocolos de encaminamiento más extendidas es la siguiente (Royer y Toh, 1999):

- Protocolos proactivos, globales o basados en tablas
- Protocolos reactivos o bajo demanda
- Protocolos híbridos

A continuación, se prensentan las rasgos principales de cada una de las categorías, para al final de la subsección analizar cual de ellas se ajusta más a las características deseadas.

5.2.1.1. Protocolos de encaminamiento proactivo

Mantienen información de encaminamiento actualizada entre todos los nodos de la red. Esta información se guarda en las tablas de encaminamiento, que periódicamente se actualizan para adaptarse a los cambios en la topología de la red.

Los protocolos proactivos se diferencian unos de otros según los criterios y el procedimiento de actualización de la información, el número de tablas de encaminamiento y las entradas de dichas tablas. Los protocolos proactivos más destacados son: OLSR (*Optimized Link State Routing*), basado en el algoritmo de estado de enlace, y DSDV (*Destination Sequenced Distance Vector*), basado en el algoritmo de vector distancia.

5.2.1.2. Protocolos de encaminamiento reactivo

Se obtiene la información de encaminamiento entre dos nodos bajo demanda, es decir, únicamente cuando existe una información que debe ser transmitida entre ellos. De este modo recursos como la energía o el ancho de banda se utilizan de una manera más eficiente que en el caso anterior. El inconveniente está en que el retardo por descubrimiento de ruta es considerablemente mayor.

El descubrimiento de rutas se realiza mediante el intercambio de mensajes de solicitud de ruta, RREQ (*Route Request*), y respuesta de ruta, RREP (*Route Reply*).

A su vez, los protocolos de encaminamiento reactivos pueden subdividirse en dos grupos:

- Basados en la fuente:
  Los paquetes de datos transportan en su cabecera toda la ruta completa, de modo que los nodos intermedios deben consultar dicha cabecera para saber dónde deben reenviarlo. El protocolo más destacado de este grupo es el DSR (*Dynamic Source Routing*).

- Salto a salto:
  Los paquetes llevan en su cabecera únicamente la dirección de destino y la del próximo nodo en la ruta. Cada uno se encarga de ir actualizando dicha cabecera según sus tablas de encaminamiento. Es necesario que cada nodo vaya actualizando permanentemente su tabla de encaminamiento mediante el intercambio periódico de mensajes. El protocolo más destacado de este grupo es el AODV (*Ad hoc On-demand Distance Vector*).

5.2.1.3. Protocolos de encaminamiento híbridos

Básicamente, aprovechan las ventajas de los protocolos proactivos y reactivos. El procedimiento suele consistir en dividir la red en zonas, o bien árboles o clusters, para trabajar con diferentes tipos de nodos. De esta manera, cuando se buscan rutas entre nodos cercanos se utiliza ecaminamiento proactivo y, por el contrario, entre nodos lejanos encaminamiento bajo demanda. Es común encontrar protocolos que utilizan información auxiliar, como posiciones mediante un GPS, para complementar estos protocolos. El más destacado de este grupo es el ZRP (*Zone Routing Protocol*).

5.2.1.4. Elección de un protocolo de encaminamiento

En esta sección se realiza un análisis cualitativo de las diferentes categorías de protocolos de enrutamiento presentadas con el objetivo de seleccionar la más adecuada para su utilización en el sistema LIDO DCL. Para otras aplicaciones, incluso en el medio marino, el análisis podría tener conclusiones distintas (Kong et al., 2008).

Por sus características, los protocolos de encaminamiento proactivo son especialmente adecuados en aquellas redes en que es necesario que el procedimiento de descubrimiento de ruta no tenga una latencia excesiva. Además, estas redes deben disponer de los recursos suficientes, como energía y ancho de banda, para asumir su funcionamiento. En el caso de LIDO DCL, la entrega inmediata de las alarmas no es un factor especialmente crítico. Este sistema se utilizará para formar perímetros de seguridad entorno a fuentes de ruido antropogénico en el medio marino. Así, existirá un margen de tiempo suficiente para poder entregar las alarmas y detener la emisión de ruido antes de que éste tenga efectos nocivos en ciertas especies. Por otro lado, como se vió en las Secciones **??** y **??**, la energía y el ancho de banda si que son recursos limitados en el sistema. De esta manera se puede concluir que los protocolos de encaminamiento proactivo no son la opción más adecuada.

Por otra parte, los híbridos están especialmente pensados para redes extensas divididas en zonas. En ellos, se utilizan protocolos de encaminamiento proactivo para la comunicación entre nodos cercanos, y reactivo entre nodos lejanos. Inicialmente, el sistema LIDO DCL no está pensado para grandes poblaciones de nodos, por lo que la utilización de este tipo de protocolos tendría un resultado similar a la categoría anterior. Por ello, los protocolos de encaminamiento híbridos tampoco se consideran una opción apropiada.

Finalmente, están los protocolos reactivos entre los que diferenciamos dos tipos: basados en la fuente, como DSR, y salta a salto, como AODV. Anteriormente se comentó que la principal diferencia entre ellos es que el primer tipo transporta en la cabecera de sus paquetes la información de toda la ruta completa, mientras que el segundo únicamente lleva la dirección de destino y la del próximo nodo. Existen numerosos estudios comparando exclusivamente los protocolos de encaminamiento AODV y DSR (Das et al., 2000) (Domingo-Aladrén, 2005). Para evaluar que tipo es más apropiado, se destacan las siguientes características del sistema LIDO DCL:

- Limitación del ciclo de trabajo máximo en la banda 868 MHz: 10 %
- Tasa maxima de transmisión radio del módulo XBee-PRO 868: 24 kbps
- Generalmente no existe un gran flujo de alarmas.

Atendiendo a las dos primeras limitaciones el AODV supera al DSR, ya que tanto sus mensajes RREQ como los RREP utilizan una señalización menor. Además, de la tercera característica nombrada se extrae que es probable que entre el envío de diferentes alarmas la topología de la red vaya variando. En este caso, ambos protocolos deberán iniciar un nuevo proceso de búsqueda de ruta para cada alarma, siendo el del AODV el más óptimo de los dos. Cabe destacar que en el protocolo DSR cada nodo intermedio a lo largo de una ruta puede aprovechar la opción *promiscuous listening* y el hecho de que el encaminamiento esté basado en la fuente para aprender más rutas. Sin embargo, ya que en el sistema LIDO DCL existirá un nodo colector encargado de recoger todas las alarmas, las rutas se crearan hacia él y, por lo tanto, en este caso el protocolo AODV tendrá la misma ventaja.

En conclusión, los protocolos de encaminamiento reactivo salto a salto son la opción más adecuada para el módulo de entrega remota de datos. Dentro de este grupo, el más destacado, y del que se hará un estudio más en profundidad, es el AODV. Además, como veremos en las siguientes secciones, existen numerosos protocolos basados en él que proponen diferentes modificaciones para lograr una versión final más simple y óptima computacionalmente hablando.

Como se indicaba al inicio de esta subsección, el análisis realizado es cualitativo, ya que para un primer prototipo se considera que esto es suficiente. En versiones futuras, sería interesante y apropiado realizar pruebas con distintos protocolos de encaminamiento para obtener unos resultados cuantitativos y poder realizar una elección más precisa.

5.2.2. Protocolo AODV

5.2.2.1. Visión general

AODV (Perkins et al., 2003) (Perkins y Royer, 1999) es un protocolo de encaminamiento reactivo salto a salto. Por ello, las rutas de cada nodo a cada nodo de la red no están permanentemente actualizadas, sino que se descubren y mantienen únicamente cuando es necesario. Sus principales características son:

- Señalización de control baja. Dado su carácter reactivo.
- Señalización de procesamiento mínima. Mensajes sencillos que requieren poco cálculo.
- Ausencia de bucles. Existen mecanismos específicos para su prevención.
- Funciona sólo con enlaces bidireccionales.

Con el fín de evitar bucles, cada nodo dispone de un número de secuencia (también llamado número de secuencia de destino) que permite mantener sólo la información de encaminamiento más reciente. Este número aumenta en una unidad cada vez que se envía un mensaje de solicitud de ruta, RREQ. Cuando un nodo recibe un RREQ del que es destinatario final, antes de responder con un mensaje de respuesta de ruta, RREP, debe actualizar su número de secuencia $NumSec_D$ al valor máximo según la expresión:

$$NumSec_D = Max(NumSec_{D.actual}, RREQ.NumSec + 1) \tag{5.1}$$

donde $NumSec_D.actual$ es su número de secuencia actual y $RREQ.NumSec$ el número de secuencia del destino que se haya contenido en el RREQ.

En el caso de que un nodo reciba dos paquetes con el mismo identificador del nodo de origen pero diferentes números de secuencia, la información más reciente será la contenida en el paquete con el mayor de los dos. Si los números de secuencia son iguales se utilizará otra métrica para decidir, generalmente se obtará por la ruta con el menor número de saltos hacia el destino.

5.2.2.2. Declaración de aplicabilidad

El protocolo de encaminamiento AODV está especialmente diseñado para redes ad hoc dinámicas con poblaciones de entre decenas y miles de nodos móviles. AODV puede operar con tasas de movilidad bajas, medias y elevadas, así como con varios niveles de tráfico de datos. Está pensado para su utilización en redes donde todos los nodos pueden confiar en los demás, ya sea por el uso de claves preconfiguradas, o porque se sabe que no hay nodos intrusos maliciosos. AODV se ha diseñado para reducir la diseminación del tráfico de control, mediante la utilización del campo TTL (*Time To Live*, Tiempo de Vida), y eliminar la sobrecarga de cabeceras en el tráfico de datos, con el fin de mejorar la escalabilidad y el rendimiento.

5.2.2.3. Tabla de rutas y tabla de RREQ's

AODV utiliza tablas de rutas para almacenar la información de encaminamiento. Cada nodo de la red mantiene la suya propia, que tiene tantas entradas como destinos conoce. Cada entrada de la tabla cuenta con los siguientes campos:

- Dirección de destino
- Dirección del próximo salto
- Número de secuencia de destino
- Tiempo de vida (tiempo requerido para eliminar la ruta)

- Contador de saltos (número de saltos para alcanzar el destino)
- Otros indicadores de estado y rutas: validez del número de secuencia de destino, validez de ruta, si es o no reparable; y si lo es, si se está o no reparando.

Cada entrada de la tabla de rutas está asociada a un temporizador, de modo que, si una ruta no se utiliza durante un determinado periodo de tiempo expira y deja de ser válida. Sin embargo, cada vez que se emplea la información de la entrada o se recibe un mensaje de HELLO, el temporizador se actualiza.

Para cada entrada válida en la tabla de rutas de un determinado nodo, se mantiene una lista de precursores que contiene los nodos que utilizan dicha ruta como próximo salto en el camino hacia un nodo destino, es decir, los nodos vecinos que han recibido algún mensaje desde el nodo destino. Estos precursores recibirán notificaciones del nodo en caso de que se detecte algún problema en la ruta, como se explica más adelante en el proceso de Mantenimiento de Ruta.

Además, cada nodo mantiene un registro de los mensajes RREQ que recibe. Utiliza los campos RREQ ID y dirección de origen, que identifican inequívocamente cada RREQ para poder descartar los repetidos y optimizar el rendimiento general de la red.

5.2.2.4. Formato de mensajes

El protocolo de encaminamiento AODV define los siguientes tipos de mensajes:

- **Mensaje RREQ**

  El mensaje RREQ se utiliza para descubrir una ruta hacia un determinado nodo destino. En la Figura 5.1 se muestra el formato del mismo, que incluye los siguientes campos:

  - Tipo: indica el tipo de mensaje.
  - J: Join flag; utilizado cuando el nodo de origen quiere formar parte de un grupo multicast.
  - R: Repair flag; utilizado cuando un nodo se dispone a iniciar una reparación de dos partes de un árbol multicast que se han desconectado.
  - G: Gratuitous RREP flag; indica si es necesario enviar en modo unicast un RREP Gratuito al nodo indicado en el campo *dirección de destino*.
  - D: Destination only falg; indica que únicamente el nodo destino puede responder a este mensaje RREQ.
  - U: Unknow sequence number flag; indica que el campo *número de secuencia de destino* es desconocido.
  - Reservado: Bits reservados. Se envían con el valor 0, y son ignorados en la recepción.
  - Contador de saltos: indica el número de saltos desde el nodo de origen hasta el nodo que está procesando este mensaje RREQ.
  - RREQ ID: número de secuencia que junto con la dirección del nodo origen identifica inequívocamente el mensaje RREQ. Se incrementa cada vez que un nodo inicia el envío de uno de ellos.
  - Dirección de destino: identificador del nodo con el que se quiere realizar la comunicación.

- Número de secuencia de destino: el último número de secuencia recibido en el nodo origen de cualquier ruta hacia el nodo destino.
- Dirección de origen: identificador del nodo que inicia el descubrimiento de ruta.
- Número de secuencia de origen: número de secuencia actual del nodo origen.

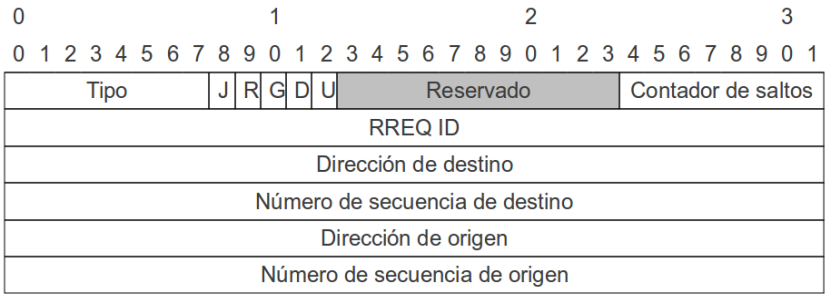


Figure 5.1: Formato del mensaje RREQ del protocolo AODV.

- **Mensaje RREP**

  El mensaje RREP es la contestación a un RREQ. Indica que existe una ruta hacia el nodo destino. En la Figura 5.2 se muestra el formato del mismo, que incluye los siguientes campos:

  - Tipo: indica el tipo de mensaje.
  - R: Repair flag; utilizado cuando un nodo se dispone a iniciar una reparación de dos partes de un árbol multicast que se han desconectado.
  - A: Acknowledgment required flag; se utiliza para comprobar si un enlace es unidireccional.
  - Reservado: Bits reservados. Enviados con el valor 0, y son ignorados en la recepción.
  - Longitud del prefijo: si no es cero, especifica que el siguiente salto indicado puede ser utilizado por nodos con el mismo prefijo, como si fuesen los destinatarios de la petición.
  - Contador de saltos: indica el número de saltos desde el nodo de origen hasta el nodo que está procesando este mensaje RREQ.
  - Dirección de destino: identificador del nodo con el que se quiere realizar la comunicación.
  - Número de secuencia de destino: el último número de secuencia recibido en el nodo origen de cualquier ruta hacia el nodo destino.
  - Dirección de origen: identificador del nodo que inicia el descubrimiento de ruta.
  - Tiempo de vida: tiempo en milisegundos durante el cual los nodos que han recibido el mensaje RREP deben considerar válida la ruta.

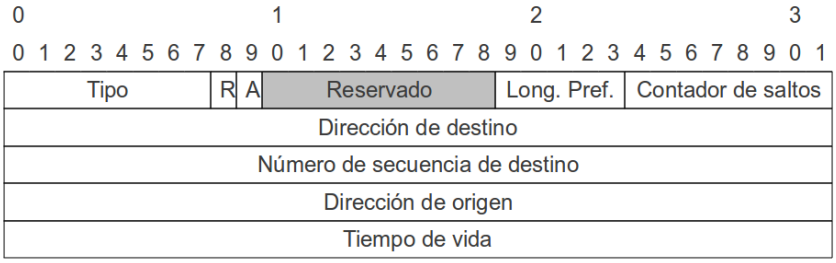


Figure 5.2: Formato del mensaje RREP del protocolo AODV.

- **Mensaje RERR (*Route Error*)**

  El mensaje RERR se utiliza si se rompe algún enlace provocando que uno o más nodos se vuelvan inalcanzables desde ciertos nodos vecinos. En la Figura 5.3 se muestra el formato del mismo, que incluye los siguientes campos:

  - Tipo: indica el tipo de mensaje.
  - N: No delete flag; se utiliza cuando se está realizando una reparación local *(local repair)* de un enlace roto y el nodo receptor del mensaje RERR no debe borrar ninguna ruta.
  - Reservado: Bits reservados. Enviados con el valor 0, y son ignorados en la recepción.
  - Contador de destinos: el número de destinos inalcanzables que se incluyen en el mensaje. El valor mínimo de este campo es uno.
  - Dirección de destino inalcanzable: identificador del nodo destino que actualmente es inalcanzable debido a la rotura del enlace.
  - Número de secuencia de destino inalcanzable: el número de secuencia del nodo destino inalcanzable indicado en el campo anterior que se encuentra en la entrada de la tabla de rutas.

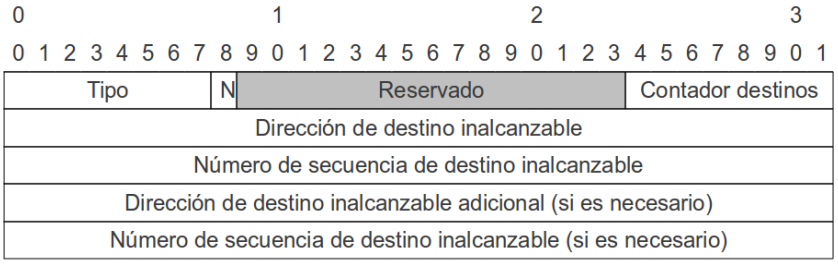


Figure 5.3: Formato del mensaje RERR del protocolo AODV.

5.2.2.5. Descubrimiento de Ruta

Cuando un nodo de la red necesita comunicarse con otro, en primer lugar, debe comprobar si en su tabla de rutas existe algun camino válido hacia el nodo destino. De ser así, el nodo podría comenzar a enviar su información sin necesidad de realizar ninguna otra operación. En el caso contrario, comenzará el proceso de Descubrimiento

de Ruta, que si concluye satisfactoriamente proporcionará la información necesaria al nodo origen para poder realizar la comunicación (Véase la Figura 5.4).

En el proceso pueden distinguirse dos fases: la formación del camino de vuelta y la formación del camino de ida. La primera de ellas establece todas las rutas posibles entre el origen y el destino, trazadas por el recorrido de los mensajes RREQ que se transmiten en modo difusión o *broadcast*. La segunda determina el itinerario que finalmente seguirán los paquetes desde un nodo al otro.

A continuación, se describen ambas fases. El material ilustrativo creado para ello está basado en el documento (Klein-Berndt, 2001) y en unas imágenes creadas por el profesor Nitin Vaidya incluidas en (Domingo-Aladrén, 2005).

- **Formación del camino de vuelta**

  Como representa la Figura 5.4b, el nodo origen, al no encontrar una ruta válida hacia el nodo destino en su tabla de rutas, genera un mensaje RREQ y lo envía en modo broadcast. Este mensaje podrá ser respondido con un RREP por el nodo destino o por cualquier nodo de la red que conozca una ruta actualizada hacia él.

  Cuando un nodo intermedio obtiene un RREQ debe comprobar si ya ha recibido anteriormente dicho mensaje. Para ello, mira los campos RREQ ID y dirección de origen y verifica si dicha información se encuentra en alguna entrada de su tabla de registro de RREQ's. Si es así, el nodo descarta el paquete, como ocurre por ejemplo en la Figura 5.4d donde el nodo C recibe los RREQ de los nodos G y H, pero no hace una retransmisión porque ya la hizo una vez anteriormente, concretamente en la Figura 5.4c. Si no ha recibido anteriormente dicho mensaje, el nodo registra la recepción del RREQ y crea una entrada en su tabla de rutas que contiene el camino hacía el nodo origen, es decir, el de vuelta (Véase la Figura 5.4c). Esta ruta tiene un tiempo de vida determinado tras el que es eliminada. El camino de vuelta tiene su utilidad cuando el nodo recibe más tarde un RREP que debe ser entregado al nodo origen a través del mismo.

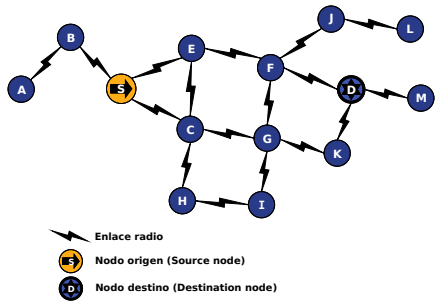
  Para poder responder a un mensaje RREQ, un nodo debe ser el destinatario final del mismo o verificar que:

  - Exista en la tabla de rutas una entrada con un camino hacia el destino que no haya expirado.
  - El número de secuencia de destino de dicha entrada en la tabla de rutas sea mayor o igual que el número de secuencia de destino del mensaje RREQ, es decir:

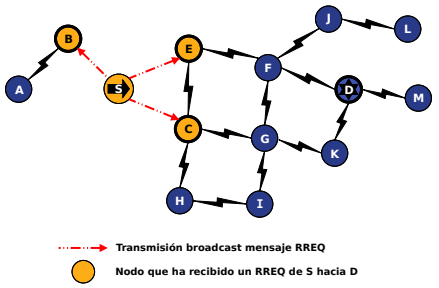
    $$NumSec_{tabla} \geq NumSec_{RREQ} \tag{5.2}$$

  Si esto se cumple, el nodo incrementa el contador de número de saltos del RREQ y genera el correspondiente mensaje RREP. Si no, el nodo incrementa el contador del número de saltos del RREQ y reenvía dicho mensaje en modo broadcast para continuar con el Descubrimiento de Ruta.

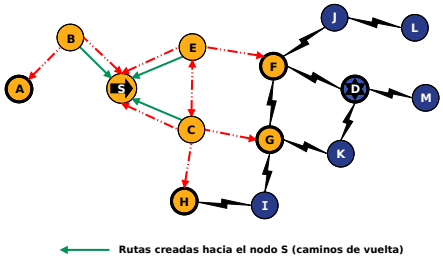
  Para evitar un consumo excesivo de recursos, el protocolo cuenta con un mecanismo conocido como búsqueda expansiva en anillo (*expanding ring search*). Éste consiste en inicialmente asignar a los mensajes RREQ un TTL pequeño, tras el cual el mensaje se descarta. Si en un determinado tiempo no se recibe un RREP, este valor se va incrementando prograsivamente en los siguientes reintentos de Descubrimiento de Ruta, hasta alcanzar el número máximo de ellos.



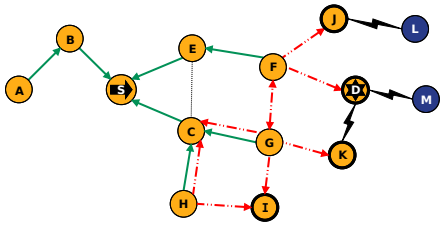
(a) Escenario inicial propuesto.



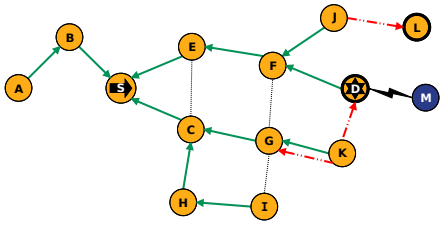
(b) Inicio del Descubrimiento de Ruta



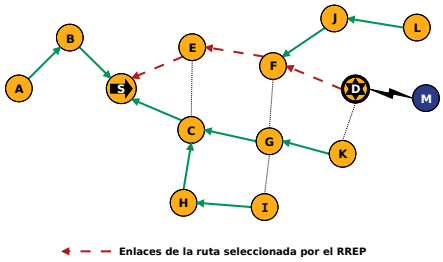
(c) Propagación de los RREQ por la red (1).



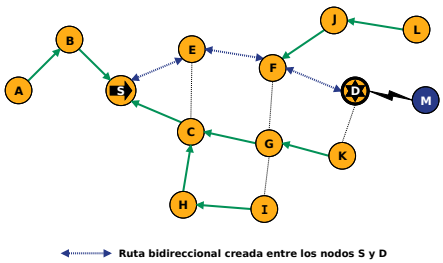
(d) Propagación de los RREQ por la red (2).



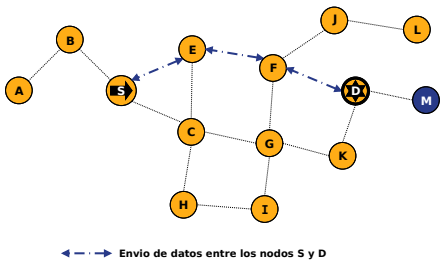
(e) Propagación de los RREQ por la red (3).



(f) Ruta seleccionada para el envío unicast de mensajes RREP.



(g) Ruta bidireccional establecida entre los nodos S y D.



(h) Envío de datos entre los nodos S y D.

Figure 5.4: Proceso de Descubrimiento de Ruta en una red ad hoc que utiliza el protocolo de encaminamiento AODV para enviar datos desde un nodo origen S hasta un nodo destino D.

- **Formación del camino de ida**

  Dependiendo de si es el nodo destino el que responde el RREQ, o es un nodo intermedio, se procede de una u otra manera. Concretamente en el primer caso el nodo destino:

  - Introduce su número de secuencia en el paquete de acuerdo con la expresión 5.1.
  - Sitúa el contador de saltos a cero.

  Si es un nodo intermedio el que inicia el envío del RREP, procede de la siguiente manera:

  - Introduce el número de secuencia del nodo destino en el paquete. Este valor se encuentra en la correspondiente entrada de su tabla de rutas.
  - Sitúa el contador de saltos al mismo valor que el número de los mismos que tiene en su ruta hacia el nodo destino.

  A continuación, en ambos casos se asigna un valor determinado al temporizador de tiempo de vida y, por último, se envía en modo unicast el mensaje según el camino de vuelta formado en la primera fase del proceso de Descubrimiento de Ruta.

  Cuando un nodo intermedio recibe un mensaje RREP incrementa el contador de saltos del paquete en una unidad y guarda en su tabla de rutas la información del camino de ida hacia el nodo destino, utilizando como próximo salto el nodo del cual recibió el RREP. Finalmente, reenvía este paquete al siguiente nodo en dirección a la fuente.

  Podría producirse la situación de que un nodo reciba un RREP por parte de más de un nodo vecino. En tal caso, reenviaría el primero de ellos, y un segundo únicamente en caso de que el número de secuencia de destino contenido en el paquete fuese mayor, o el valor del contador de saltos menor, actualizando en tal caso su tabla de rutas. Este mismo criterio es utilizado por el nodo origen para establecer el camino de ida al recibir varios paquetes RREP. Una vez establecido este camino, es posible comenzar a utilizarlo para el envío bidireccional de paquetes de datos (Véase la Figura 5.4h).

  En la Figura 5.4f vemos como el nodo destino tiene dos opciones, a través del nodo F o del K, para establecer el camino de vuelta. De ellas, opta por la que tiene el menor número de saltos, pero esto no siempre implica que se trate de la trayectoria más óptima de toda la red. En el protocolo de encaminamiento AODV los nodos únicamente aceptan y procesan un RREQ, rechazando aquellos paquetes, con el mismo RREQ ID y dirección de destino, que se reciben posteriormente. De este modo, algunas trayectorias más óptimas que la ruta final podrían no descubrirse al utilizar algún nodo común.

5.2.2.6. Mantenimiento de Ruta

El protocolo de encaminamiento AODV está especialmente pensado para operar en el medio radio y con nodos móviles. Esto hace común la rotura de enlaces entre ellos. Además, como ya se ha visto, las rutas que se descubren tienen un tiempo de vida durante el que pueden ser utilizadas y que al terminar las inhabilita. Esto permite

que no sea necesario establecer un camino cada vez que se quiere mandar un mensaje de datos pero, a su vez, hace necesaria la existencia de algún proceso encargado de la monitorización del estado de los enlaces y de su gestión en caso de rotura. Este proceso se conoce como Mantenimiento de Ruta.

El protocolo AODV utiliza mensajes HELLO, que contienen la dirección del nodo, su número de secuencia actual y el tiempo de vida de los enlaces, y que cada nodo envía periódicamente en modo broadcast a sus vecinos para comprobar si existe conectividad entre ellos. De este modo, cada nodo puede actualizar la información referente a sus vecinos en su tabla de rutas. Si durante un determinado intervalo de tiempo un nodo deja de recibir estos mensajes de alguno de sus vecinos, elimina la entrada asociada a dicho vecino de su tabla de rutas. La utilización de mensajes HELLO no es necesaria si existe algún otro mecanismo que realice su tarea, como puede ser retroalimentación procedente de la capa de enlace de datos, LLN (*Link Layer Notification*).

Cuando un nodo detecta un fallo en un enlace, envía un mensaje de error de ruta, RERR (*Route Error*), hacia el nodo fuente para informar de los destinos que ya no pueden alcanzarse. Si existen varios nodos precursores (entre el nodo origen y el nodo que detectó la rotura) que estaban utilizando el enlace, el mensaje RERR se envia en modo broadcast, en caso contrario se envia en modo unicast.

Al recibir un paquete de RERR, un nodo verifica que efectivamente la rotura afecta a alguna de sus entradas en la tabla de rutas, la invalida y por último reenvía el mensaje hacia la fuente. Cuando es el nodo origen el que recibe el RERR, éste puede optar por iniciar un nuevo proceso de Descubrimiento de Ruta si lo considera necesario.

AODV también cuenta con algunas opciones de optimización, como la posibilidad de realizar reparaciones locales de los enlaces. Cuando un nodo detecta la rotura de un enlace, en vez de mandar inmediatamente un mensaje RERR a la fuente, envía un RREQ hacia el nodo destino con el número de secuencia de destino incrementado en una unidad. Mientras espera el RREP va almacenando los paquetes de datos. Si tras un determinado tiempo no se recibe respuesta, la reparación local no habrá tenido éxito y será necesario avisar a la fuente con un mensaje RERR.

Otra característica adicional del protocolo AODV consiste en enviar mensajes RREP gratuitos desde un nodo intermedio hacia el nodo destino para informarle de que el propio nodo intermedio ha respondido un RREQ. Esto permite que el nodo destino conozca una ruta hacia el nodo origen por si la comunicación fuese bidireccional. De otro modo, el nodo destino no tendría noción de que existe tal ruta hacia él y, por lo tanto, no podría responder a la fuente si tuviera que hacerlo.

5.2.3. Protocolos basados en el AODV

El protocolo de encaminamiento AODV es uno de los más estudiados para su utilización en redes ad hoc móviles, también conocidas como MANET (*Mobile Ad hoc Network*). Esto ha dado lugar a la aparición de numerosos protocolos que proponen diferentes modificaciones sobre el original, según la aplicación final que se les quiera dar. En la presente subsección se realiza una revisión de los principales protocolos, orientados a redes de sensores, que están basados en el AODV (Gomez et al., 2006).

5.2.3.1. AODVjr

AODVjr (Chakeres y Klein-Berndt, 2002) (Klein-Berndt y Chakeres, 2002) es una de las primeras versiones simplificadas del AODV. Los autores muestran a través de

varias simulaciones unos resultados muy próximos a los obtenidos con el protocolo AODV completo. Al mismo tiempo, reducen significativamente la complejidad de implementación de éste al eliminar varias características definidas en su especificación original. En primer lugar, dejan de utilizarse los números de secuencia. El protocolo AODVjr evita la formación de bucles definiendo que únicamente el nodo destino puede generar los mensajes RREP. Esto también elimina la necesidad del envió de mensajes RREP gratuitos desde nodos intermedios. Además, este protocolo ya no emplea el número de saltos de las rutas como métrica. En su lugar establece que la ruta relacionada con el primer mensaje RREP que recibe el nodo origen es la elegida. Por último, se eliminan los mensajes HELLO, RERR y la lista de precursores, y el mantenimiento de ruta pasa a realizarse de la siguiente manera. Si la comunicación es unidireccional, el nodo destino manda mensajes de conexión (*connect messages*) al nodo origen. Si es bidireccional, no se utilizan mensajes adicionales; un nodo origen detecta la rotura de una ruta cuando deja de recibir mensajes desde el nodo destino.

5.2.3.2. AODVbis

AODVbis es una revisión de la especificación original del AODV que aclara algunos aspectos de funcionalidad y deja varias características como opcionales. La utilización de mensajes RERR ya no es obligatoria. En caso de utilizarse, estos se envían en modo broadcast localmente, de modo que se evita tener que mantener una lista de precursores para cada entrada de la tabla de rutas. Cada nodo que reciba un RERR debe añadir los destinos inalcanzables que provienen de procesar dicho mensaje a los nuevos mensajes RERR que sean generados. El contador de saltos también deja de ser la métrica obligatoria para las rutas, y por defecto se procede como en el caso del AODVjr. También se eliminan las reparaciones locales y se añade la característica acumulación de rutas (*path accumulation*), que permite a un determinado nodo adquirir información de enrutamiento extra de una lista con el camino recorrido incluida en los mensajes RREQ y RREP que pasan por él.

5.2.3.3. LoWPAN-AODV

LoWPAN-AODV es una propuesta para adaptar AODV a entornos LoWPAN (*Low-Rate Wireless Personal Area Network*), como es el caso de las redes de sensores. AODVbis es la especificación por defecto del LoWPAN-AODV, salvo por algunos aspectos que conserva de la filosofía del AODVjr. Los únicos mensajes de control definidos son RREQ y RREP. Sólo el nodo destino puede responder un RREQ con un RREP. La conectividad local se mantiene por el mecanismo LLN, según establece el estándar IEEE 802.15.4. En este caso, el contador de saltos sí que es la métrica utilizada para determinar las mejores rutas.

5.2.3.4. LOAD

LOAD, al igual que AODVjr y LoWPAN-AODV, sólo permite que el nodo destino genere los mensajes RREP, evitando así el uso de números de secuencia. Permite las reparaciones locales, y en caso de que estas no finalicen satisfactoriamente, el nodo que detectó el fallo del enlace debe generar y enviar en modo unicast un mensaje RERR al nodo origen. La métrica de encaminamiento utilizada en LOAD es el coste acumulado del enlace entre el origen y el destino, usando el indicador de calidad de enlace (*Link*

Quality Indicator (LQI)) de la capa física del IEEE 802.15.4 como entrada para el cálculo. Finalmente, como en el LoWPAN-AODV, se utiliza el mecanismo LLN para evaluar la conectividad de las rutas.

5.2.3.5. TinyAODV

TinyAODV es una implementación minimalista del AODV para dispositivos que utilizan el sistema operativo TinyOS. En este protocolo, si un paquete de datos debe enviarse y no existe una ruta válida hacia el destino, se inicia el proceso de Descubrimiento de Ruta, pero el paquete que requirió la ruta es descartado. De este modo, el siguiente paquete de datos será el primero en utilizar la ruta descubierta. El número de reintentos de Descubrimiento de Ruta se puede modificar, pero su valor por defecto es tres.

De igual modo que algunos de los protocolos anteriores, sólo el nodo destino puede generar los mensajes RREP. El mecanismo LLN puede utilizarse para detectar fallos en los enlaces, aunque por defecto no está habilitado. Por ello, TinyAODV está especialmente pensado para redes con topologías estáticas, donde no se esperan apenas fallos en los enlaces. Si un paquete de datos no puede entregarse por un fallo en un enlace, éste será descartado. A continuación, se generará un mensaje RERR que será enviado en modo broadcast localmente, como hacía el AODVbis. Por último, la reparación local no está soportada, y el contador de saltos es la métrica utilizada para determinar las mejores rutas.

5.2.3.6. NST-AODV

Not So Tiny - AODV (NST-AODV) es una implementación de un protocolo de encaminamiento en nesC para plataformas TinyOS. Fué diseñado con el objetivo de añadir varias prestaciones adicionales a las básicas del TinyAODV con el objetivo de lograr un mejor rendimiento, especialmente en redes dinámicas.

Sus características principales son:

- El mecanismo LLN está activado por defecto, ya que se asume que el protocolo operará en redes con topología dinámica.
- El paquete de datos que dispara un proceso de Descubrimiento de Ruta no es descartado, y en cuanto la ruta está disponible es enviado.
- Tras una transmisión fallida a nivel de enlace, desde la capa de red se pueden realizar hasta dos reintentos adicionales.
- Si un paquete no puede ser enviado tras los tres intentos de la capa de red, será almacenado en un buffer y enviado si se encuentra una nueva ruta. Esta operación puede realizarla tanto el nodo origen como un nodo intermedio que repare localmente la ruta.
- Se han añadido dos colas FIFO (*First In, First Out*): una para guardar los paquetes recibidos mientras se realiza un Descubrimiento de Ruta, y otra para los paquetes a enviar. Ambas colas pueden dimensionarse según interese en su configuración.
- Un nodo intermedio puede generar un mensaje RREP si conoce una ruta válida al destino solicitado en el RREQ.

5.2.4. Definición del protocolo de encaminamiento AODV-LAB

Como se vió en la Sección **??**, actualmente no existen en el mercado módulos de comunicación de gran alcance en la banda de 868 MHz que incorporen en su firmware el soporte para una topología de malla dinámica o ad hoc. Por ello, surgió la necesidad de incorporar esta funcionalidad a los módulos de comunicación seleccionados. En primer lugar, se realizó una búsqueda de protocolos de encaminamiento reactivos ya desarrollados que pudiesen ser válidos para el proyecto, pero únicamente se encontraron implementaciones orientadas a su utilización en: Internet, el simulador de redes ns-2 y el sistema operativo TinyOS. Por ello, finalmente se decidió desarrollar en C/Linux un protocolo de encaminamiento propio, llamado AODV-LAB, para poder utilizarlo con el módulo de entrega remota de datos, que en su versión final irá implementado en un PC embebido con el sistema operativo GNU/Linux.

A continuación, a partir del análisis de los protocolos presentados en la Subsección 5.2.3, se definen las características principales del AODV-LAB:

- Para evitar bucles y la utilización de mensajes RREP gratuitos, únicamente el nodo destino puede generar mensajes RREP. Además, se mantienen los números de secuencia como en el AODV. Esto pérmite diferenciar los mensajes RREQ y RREP más recientes. De este modo, en el AODV-LAB cuando un nodo recibe uno de estos mensajes, en primer lugar da prioridad a la información de encaminamiento más actual y, únicamente en caso de que el número de secuencia del mensaje coincida con el de la tabla de rutas, utilizará el contador de saltos para optar por el camino con menor número de ellos. Como mejora en un futuro, se contempla añadir sistemas de evaluación de la calidad de los enlaces como criterio para la selección de rutas que, especialmente en el medio marino, puede suponer importantes mejoras.

- Se eliminan los mensajes HELLO, RERR, y la lista de precursores, reduciendo la complejidad de implementación, la utilización de recursos del sistema y manteniendo unas características de funcionamiento similares, como se explicó anteriormente en el caso de AODVjr.

- Como mecanismo para el mantenimiento de la conectividad se utiliza el envío de mensajes de acuse de recibo de datos, llamados ACK (*Acknowledgement*). Si tras un periodo de tiempo definido un nodo que ha enviado un paquete de datos no recibe su correspondiente ACK, volverá a enviar el paquete tantas veces como se indique en el número de reintentos. Una vez agotados se dará la ruta por inválida y se volverá a iniciar el proceso de Descubrimiento de Ruta. El empleo de un número de reintentos por encima de la capa de enlace se utiliza para evitar que por pérdidas de conexión temporales y esporádicas, o por pequeños periodos con peores condiciones para la transmisión radio, se inicien nuevos Descubrimientos de Ruta innecesarios, con el gasto de recursos que esto supone.

- No se contempla la utilización del mecanismo LLN ya que éste depende del hardware utilizado, aunque en un futuro podría suponer una mejora considerable del sistema gracias a sus numerosas ventajas: detección más rápida de roturas de enlace, utilización más eficiente del ancho de banda y menor consumo de energía.

- Cuando una ruta se da como inválida, el paquete que agotó sus reintentos es descartado. Esto es así ya que en ese momento habrá transcurrido demasiado

tiempo y dicha alarma ya no será útil, por lo que enviarla únicamente supondría un gasto de recursos.

- Existe una cola FIFO (*First In, First Out*) para los mensajes de entrada, de modo que los mensajes serán procesados en el orden en que se recibieron.
- No se realiza reparación local de rutas. La aplicación princial de este sistema será crear perímetros de seguridad alrededor de fuentes de contaminación acústica. Estos perímetros no requerirán a priori rutas de más de 3 saltos entre un nodo origen y uno destino, por lo que la reparación local en este caso no tiene mucho sentido, ya que este mecanismo está especialmente pensado para grandes redes.

5.2.5. Comparación entre los distintos protocolos de encaminamiento

En la Tabla 5.1 se muestra un resumen de todos los protocolos de encaminamiento basados en el AODV que se han presentado anteriormente.

Nombre del protocolo	Mecanismo de mantenimiento de conectividad	Mensaje RERR	Lista precursores	Reparación local	Sólo el nodo destino genera RREP	Métrica de encaminamiento	Especificación y/o estado de implementación
AODV	Mensajes HELLO, LLN, etc.	Si	Si	Si (puede)	No	Contador de saltos	Implementado para múltiples plataformas
AODVjr	Mensajes de de conexión (tráfico unidir.)	No	No	No	Si	Primer RREP recibido	Implementado para el simulador ns-2
AODVbis	Mensajes HELLO, LLN, etc.	Si (debe)	No	No	No	Primer RREP recibido	Proyecto Internet expirado
LoWPAN-AODV	LLN	No	No	No	Si	Contador de saltos	Proyecto Internet expirado
LOAD	LLN	Si (puede)	No	Si (puede)	Si (debe)	Basado en LQI	Proyecto Internet
TinyAODV	LLN (deshabilitado por defecto)	Si	No	No	Si	Contador de saltos	Implementado para TinyOS
NST-AODV	LLN	Si	No	Si	No	Contador de saltos	Implementado para TinyOS
AODV-LAB	Mensajes ACK y timeouts	No	No	No	Si	Contador de saltos	Implementado en C/Linux

Table 5.1: Comparación de protocolos de encaminamiento basados en el AODV

5.3. Implementación en C/Linux

5.3.1. Introducción y justificación

El sistema LIDO DCL, en su primera aproximación, introducirá muy poco tráfico en la red. Las alarmas correspondientes a las distintas vocalizaciones de cetáceos no son muy frecuentes ya que la presencia de ellos no es contínua y, además, el sistema no enviará una por sonido detectado sino que tratará de identificar la presencia de individuos en una determinada zona y avisar de esta periódicamente mientras continúen allí. Esto ha permitido, en la realización de este proyecto, centrarse directamente en la implementación de un prototipo sin la necesidad previa de: llevar a cabo un estudio

exahustivo en un simulador de redes del protocolo de encaminamiento, realizar comparaciones con otros ya existentes y, finalmente, buscar la mejor solución. El objetivo es dar una primera aproximación al problema y demostrar su viabilidad en condiciones reales.

Se destaca la importancia de realizar una implementación de un protocolo de encaminamiento (Chakeres y Belding-Royer, 2004). La simulación es una gran herramienta para el desarrollo de los mismos, pero esta no garantiza que un protocolo funcione en la práctica. Los simuladores realizan suposiciones y utilizan modelos simplificados que pueden no reflejar del todo el funcionamiento real de la red. Por otro lado, en un simulador el código se encuentra contenido en una única unidad lógica, que está cláramente definida. Sin embargo, una implementación requiere la utilización de un sistema con muchos componentes, algunos incluso con poca documentación o sin ella. Esto obliga al desarrollador a comprender no solo el protocolo de encaminamiento, sino también todos los componentes del sistema y sus interacciones.

Como ya se comentó en la Subsección 5.2.4, para este proyecto en primer lugar se trató de encontrar alguna implementación ya desarrollada para utilizarla al completo o como punto de partida. De esta búsqueda resultaron principalmente dos grupos que, como se explica a continuación, no se ajustan del todo a lo requerido:

- Implementaciones orientadas a Internet (Chakeres y Belding-Royer, 2004) (Kawadia et al., 2003). Son muy complejas. Van integradas en el kernel de Linux, utilizando para su aplicación muchas señales e indicadores que para este proyecto no son necesarias y lo complican considerablemente.

- Implementaciones para el sistema operativo TinyOS (Gomez et al., 2006) (Wang et al., 2006). TinyOS es un sistema operativo de código abierto diseñado para dispositivos inalámbricos de bajo consumo como los utilizados en redes de sensores. El sistema LIDO DCL posee algunas características de las redes de sensores pero, al trabajar con audio y tener que procesarlo en tiempo real, requiere de un sistema operativo más potente, lo que a su vez supone un mayor consumo energético. Concretamente operará sobre GNU/Linux.

Por último, tal y como se puede ver en la Subsección **??**, los módulos XBee poseen un modo de operación transparente en el que se comportan como una conexión serie. Esto nos permite crear tramas propias, enviarlas al módulo como si se hiciera hacia el puerto serie y recibirlas del mismo modo.

Por todo ello, como muestra la Figura 5.5, finalmente se ha optado por realizar una implementación del sistema como un programa en C/Linux donde el módulo de entrega remota de datos utiliza el protocolo de encaminamiento AODV-LAB para encontrar rutas hacia el nodo colector y poder mandar sus alarmas y recibir los correspondientes ACKs. A continuación se detallará la organización del código, las tramas que utiliza y el flujo del programa.

5.3.2. Organización del código

En la Figura 5.6 se muestra como está organizado el código implementado para este proyecto. Dentro de cada fichero se han indicado los archivos con extensión `.c` y las funciones que contiene cada uno. Para darle una mayor claridad al diagrama se han omitido los archivos con extensión `.h` que contienen los prototipos de las funciones y

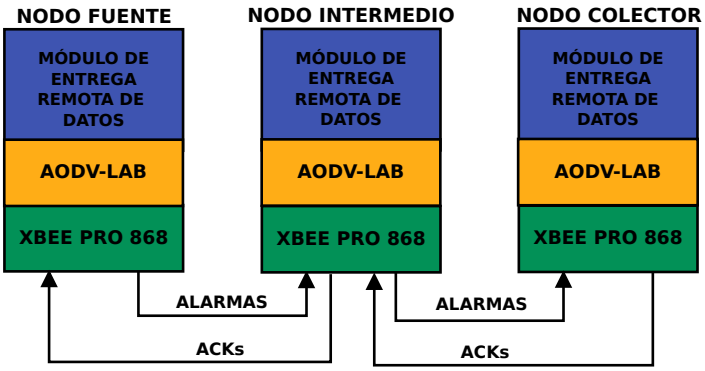


Figure 5.5: Arquitectura de operación del módulo de entrega remota de datos.

definiciones de variables. Las cabeceras de las funciones más importantes se pueden consultar en el Apéndice C.

5.3.3. Tramas

En el sistema únicamente se definen seis tipos de tramas. En la Tabla 5.2 se indican sus nombres, identificadores y una breve descripción de cada una. Posteriormente, se detallan todas ellas indicando, en cada caso, los campos de la trama y una especificación de los mismos.

Nombre de la trama	Identificador	Descripción
RREQ	Q	Mensaje de solicitud de ruta
RREP	P	Respuesta a solicitud de ruta
DATA	D	Mensaje de datos de alarma
ACK	A	Acuse de recibo de datos
PREQ	W	Mensaje de solicitud de posición (Where are you?)
PREP	C	Mensaje de posición (Coordinates)

Table 5.2: Tramas definidas en el sistema

En el programa en C/Linux se utilizan corchetes para delimitar cada trama. De esta manera se evita procesar mensajes incompletos. Además, los campos van separados por el caracter "|". Esto permite no asignarles un tamaño determinado. Así, por ejemplo, la dimensión de una trama que contenga el campo *número de secuencia* irá aumentando según aumente el número de dígitos de éste, y no utilizará desde el principio un tamaño definido. Esto ayuda a reducir la cantidad de información transmitida. En este proyecto no se ha trabajado la compresión de las tramas pero, como futuro trabajo, supondría una mejora para el sistema. Todos los campos se han definido de manera parametrizable dotando al código de una mayor versatilidad. Por último, indicar que el orden de

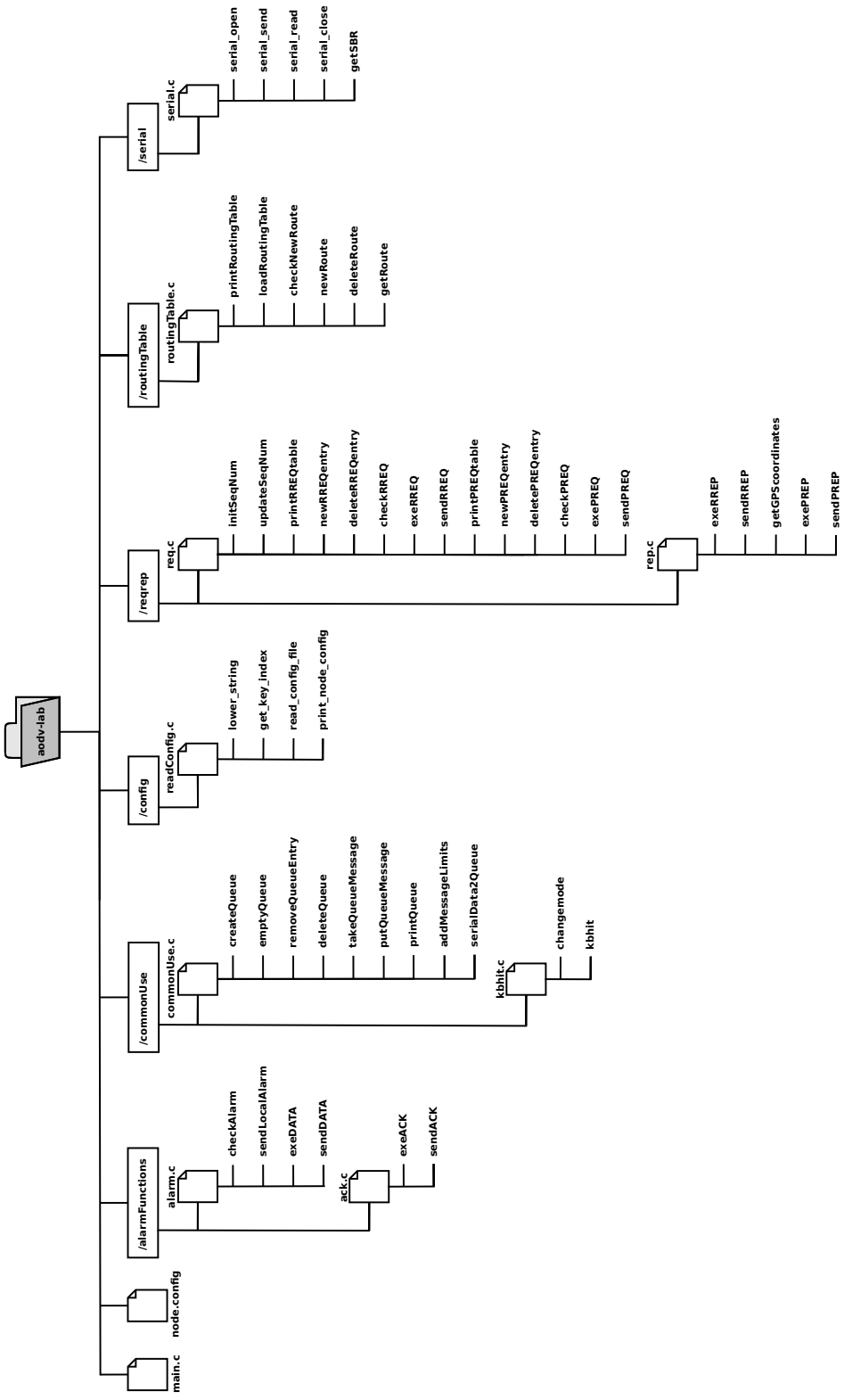


Figure 5.6: Organización del código implementado.

los campos en las tramas está determinado por la necesidad de la información que contienen en el procesado de los mismos por el sistema.

5.3.3.1. Trama RREQ

Se describe en la Tabla 5.3.

Campos de la trama RREQ	Identificador	Definición	Descripción
Next Node ID	NNID	`char nnid[NID_BYTES];`	Indica el destinatario inmediato de la trama, es decir, el próximo nodo en la ruta hacia el destinatario final. El valor 0 se corresponde con el modo Broadcast.
Frame ID	frameID	`char frameID[FID_BYTES];`	Indica el tipo de trama.
Source Node ID	SNID	`char snid[NID_BYTES];`	Indica el nodo que inició la búsqueda de ruta hacia el nodo destino indicado.
Sequence Number	seqN	`unsigned long int seqN;`	Número de secuencia. Se utiliza para evitar bucles e identificar la información más reciente.
Destination Node ID	DNID	`char dnid[NID_BYTES];`	Indica el nodo destino por el que se inició la búsqueda. En esta aplicación coincidirá con el nodo colector.
Previous Node ID	PNID	`char pnid[NID_BYTES];`	Indica el predecesor inmediato en la ruta, es decir, al nodo del que hemos recibido la trama. El nodo fuente se pone a sí mismo como predecesor.
Hop Counter	HOPS	`int hops;`	Indica el número de saltos en la ruta de la trama en el momento de su recepción, por lo tanto, se incrementa justo antes de enviarse. Este campo permite controlar la dispersión de los mensajes.

Table 5.3: Campos de la trama RREQ del protocolo AODV-LAB

Para una mejor comprensión de la trama y sus campos, a continuación se muestra un ejemplo comentado.

```
[NNID|frameID|SNID|   seqN   |DNID|PNID|hops]
[ 0  |   Q   | S  |1304427351| D  | S  | 1  ]
```

El campo `NNID` a cero indica que esta trama se ha enviado en modo broadcast. Por ello, ningún nodo que la reciba la descartará a priori. Únicamente lo harán si debe ser así tras su procesado. El campo `frameID` con el caracter `Q` nos informa que se trata de un mensaje RREQ. Esta solicitud de ruta la ha iniciado el `SNID`, es decir, el nodo `S`. Su número de secuencia, `seqN`, es `1304427351`. El campo `DNID` se corresponde con el nodo con el que se quiere comunicar `SNID`, en este caso el `D`. `PNID` es el identificador del nodo inmediato del que se recibe esta trama y su valor aquí coincide con `SNID`. Esto quiere decir que este mensaje se recibe directamente desde el nodo que lo genera, sin pasar por ningún nodo repetidor, aunque podría perfectamente ser así. Los nodos contenidos en el campo `PNID` de los mensajes recibidos son vecinos. Por último, `hops` contiene el número de saltos que ha dado el mensaje previamente a su recepción. En el ejemplo, su valor es el mínimo posible, es decir, uno. Esto indica que el mensaje se ha recibido directamente del nodo que lo generó.

5.3.3.2. Trama RREP

Se describe en la Tabla 5.4.

Campos de la trama RREP	Identificador	Definición	Descripción
Next Node ID	NNID	`char nnid[NID_BYTES];`	Indica el destinatario inmediato de la trama, es decir, el próximo nodo en la ruta hacia el destinatario final.
Frame ID	frameID	`char frameID[FID_BYTES];`	Indica el tipo de trama.
Previous Node ID	PNID	`char pnid[NID_BYTES];`	Indica el predecesor inmediato en la ruta, es decir, al nodo del que hemos recibido la trama. El nodo fuente se pone a sí mismo como predecesor.
Source Node ID	SNID	`char snid[NID_BYTES];`	Indica el nodo que inició la respuesta de solicitud de ruta. Se corresponde con el campo DNID del mensaje RREQ al que responde. En esta aplicación coincidirá con el nodo colector.
Sequence Number	seqN	`unsigned long int seqN;`	Número de secuencia. Se utiliza para evitar bucles e identificar la información más reciente.
Hop Counter	HOPS	`int hops;`	Indica el número de saltos en la ruta de la trama en el momento de su recepción, por lo tanto, se incrementa justo antes de enviarse. Este campo permite controlar la dispersión de los mensajes.
Destination Node ID	DNID	`char dnid[NID_BYTES];`	Indica el nodo destino de la respuesta de solicitud de ruta. Se corresponde con el campo SNID del mensaje RREQ al que responde.

Table 5.4: Campos de la trama RREP del protocolo AODV-LAB

Para una mejor comprensión de la trama y sus campos, a continuación se muestra un ejemplo comentado.

```
[NNID|frameID|PNID|SNID|   seqN   |hops|DNID]
[ S  |   P    | A  | D  |1304433774| 2  | S  ]
```

Al haberse formado el camino de vuelta con un mensaje RREQ, su respuesta se realiza de modo unicast, por ello, el campo `NNID` en los mensajes RREP lleva el identificador de un nodo concreto, en este caso el `S`. El `frameID` es la letra `P`, correspondiente a este tipo de mensajes. El campo `PNID` señala que el nodo `A` es el nodo anterior inmediato en la ruta, es decir, del que se recibe la trama. El nodo `D`, indicado en `SNID`, se corresponde con el nodo que generó el RREP, por consiguiente, coincide con el campo `DNID` del mensaje RREQ al que responde. En este caso, el número de secuencia, `seqN`, es `1304433774`. El número de saltos, `hops`, es dos. Concretamente del nodo `D` al `A`, y éste que se lo envía al `S`. Por último, el campo `DNID`, que indica el destinatario del RREP y se corresponde con el `SNID` del mensaje RREQ, es el nodo `S`. Finalmente se deduce que cuando los campos `NNID` y `DNID` coinciden el mensaje ha llegado a su destinatario final, de manera que, se habrá creado la correspondiente ruta bidireccional entre dos nodos.

5.3.3.3. Trama DATA

Se describe en la Tabla 5.5.

Campos de la trama DATA	Identificador	Definición	Descripción
Next Node ID	NNID	`char nnid[NID_BYTES];`	Indica el destinatario inmediato de la trama, es decir, el próximo nodo en la ruta hacia el destinatario final.
Frame ID	frameID	`char frameID[FID_BYTES];`	Indica el tipo de trama.
Destination Node ID	DNID	`char dnid[NID_BYTES];`	Indica el nodo destino de la trama DATA. En esta aplicación coincidirá siempre con el nodo colector.
Source Node ID	SNID	`char snid[NID_BYTES];`	Indica el nodo que manda los datos. En esta aplicación serán siempre alarmas.
Alarm ID	alarmID	`char alarmID[11];`	Junto con el campo SNID identifica inequívocamente cada alarma. Cada nodo va incrementando este valor según genera nuevas alarmas.
Timestamp	timeStamp	`char timeStamp[11];`	Timestamp (fecha y hora) en que se generó la alarma.
Alarm type	alarmType	`char alarmType[ATYPE_BYTES];`	Indica el tipo de alarma generada en función del evento acústico registrado.
Latitude	latitude	`char latitude[MAXALEN];`	Latitud geográfica del nodo en el momento de generar la alarma. Se obtiene de un GPS.
Longitude	longitude	`char longitude[MAXALEN];`	Longitud geográfica del nodo en el momento de generar la alarma. Se obtiene de un GPS.
Percentage of confidence	perConf	`char perConf[MAXALEN];`	Porcentaje de confianza asociado a la alarma generada.

Table 5.5: Campos de la trama DATA del protocolo AODV-LAB

Para una mejor comprensión de la trama y sus campos, a continuación se muestra un ejemplo comentado.

```
[NNID|frameID|DNID|SNID|alarmID| timeStamp |alarmType|latitude|longitude|perConf]
[ A  |   D    | D  | S  |   1   |1304433727|    W    | 41.2061|  1.7300 |   87   ]
```

La trama se envía al nodo `A`. Es de tipo DATA ya que su `frameID` es la letra `D`. El mensaje va del nodo `S` al `D` como indican respectivamente los campos `SNID` y `DNID`. El identificador de alarma, `alarmID` es uno, lo que señala que es la primera alarma que genera el nodo `S`. El campo `timeStamp`, con valor `1304433727`, nos permite conocer la fecha y hora en que se generó la alarma. Concretamente, en el sistema operativo GNU/Linux se utiliza el timestamp UNIX, que es el número de segundos desde 00:00:00 UTC del 1 de enero de 1970 hasta el momento en que se crea el timestamp. De esta manera, utilizando un conversor, se puede obtener que esta alarma se generó el 3 de mayo de 2011 a las 15:42:07 (GMT+1). Como se indica en la Tabla 5.5, el campo `alarmType` determina el tipo de alarma en función del evento acústico registrado. En este proyecto no se ha definido un diccionario de alarmas, así que, se ha utilizado la letra `W` únicamente como ejemplo. En el momento de generar la alarma el nodo se encontraba en las coordenadas geográficas `41.2061 N - 1.73 E`. Los valores negativos corresponderán a la latitud sur (S) y longitud oeste (W). Por último, el porcentaje de confianza de la alarma generada es del 87 %, como indica el campo `perConf`.

5.3.3.4. Trama ACK

Se describe en la Tabla 5.6.

Campos de la trama ACK	Identificador	Definición	Descripción
Next Node ID	NNID	`char nnid[NID_BYTES];`	Indica el destinatario inmediato de la trama, es decir, el próximo nodo en la ruta hacia el destinatario final.
Frame ID	frameID	`char frameID[FID_BYTES];`	Indica el tipo de trama.
Destination Node ID	DNID	`char dnid[NID_BYTES];`	Indica el nodo destino de la trama ACK. Se corresponderá con el campo SNID de la alarma a la que responde.
Alarm ID	alarmID	`char alarmID[11];`	Identificador de alarma. Cuando un nodo recibe un ACK y él es el DNID de la trama, con el campo alarmID puede saber que alarma de las que envió se entregó con éxito.

Table 5.6: Campos de la trama ACK del protocolo AODV-LAB

Para una mejor comprensión de la trama y sus campos, a continuación se muestra un ejemplo comentado.

```
[NNID|frameID|DNID|alarmID]
[ S  |   A   | S  |   1   ]
```

La trama se envía al nodo `S`, que además es el destinatario final de la misma, como indican los campos `NNID` y `DNID`. Es una trama ACK ya que su `frameID` es la letra `A`. Concretamente este acuse de recibo le indica al nodo `S` que su primera alarma generada y enviada (`alarmID = 1`) se ha recibido con éxito.

5.3.3.5. Trama PREQ

Se describe en la Tabla 5.7.

Campos de la trama PREQ	Identificador	Definición	Descripción
Next Node ID	NNID	`char nnid[NID_BYTES];`	Indica el destinatario inmediato de la trama, es decir, el próximo nodo en la ruta hacia el destinatario final. El valor 0 se corresponde con el modo Broadcast.
Frame ID	frameID	`char frameID[FID_BYTES];`	Indica el tipo de trama.
Source Node ID	SNID	`char snid[NID_BYTES];`	Indica el nodo que inició la solicitud de posición. En esta aplicación coincidirá con el nodo colector.
Sequence Number	seqN	`unsigned long int seqN;`	Número de secuencia. Se utiliza para evitar bucles e identificar la información más reciente. La trama PREQ descubre rutas de vuelta hacia el nodo que la envió, por eso incluye este campo.
Previous Node ID	PNID	`char pnid[NID_BYTES];`	Indica el predecesor inmediato en la ruta, es decir, el nodo del que hemos recibido la trama. El nodo fuente se pone a sí mismo como predecesor.
Hop Counter	HOPS	`int hops;`	Indica el número de saltos en la ruta de la trama en el momento de su recepción, por lo tanto, se incrementa justo antes de enviarse. En los mensajes PREQ no es un factor para controlar la dispersión ya que una solicitud de posición debe llegar a todos los nodos de la red.

Table 5.7: Campos de la trama PREQ del protocolo AODV-LAB

Para una mejor comprensión de la trama y sus campos, a continuación se muestra un ejemplo comentado.

```
[NNID|frameID|SNID|   seqN   |PNID|hops]
[ 0  |   W   | D  |1304516600| D  | 1  ]
```

El nodo `D` envía un mensaje PREQ en modo broadcast. El número de secuencia en el mensaje es `1304516600`. En este caso, el nodo predecesor, `PNID`, se corresponde con el `SNID`, por lo tanto, la trama capturada únicamente ha dado un salto, como se puede observar en el campo `hops` del ejemplo.

5.3.3.6. Trama PREP

Se describe en la Tabla 5.8.

Campos de la trama PREP	Identificador	Definición	Descripción
Next Node ID	NNID	`char nnid[NID_BYTES];`	Indica el destinatario inmediato de la trama, es decir, el próximo nodo en la ruta hacia el destinatario final.
Frame ID	frameID	`char frameID[FID_BYTES];`	Indica el tipo de trama.
Destination Node ID	DNID	`char dnid[NID_BYTES];`	Indica el nodo destino de la respuesta de solicitud de posición. Se corresponde con el campo SNID del mensaje PREQ al que responde, que en esta aplicación siempre será el nodo colector.
Timestamp	timeStamp	`char timeStamp[11];`	Timestamp (fecha y hora) en que se genera el PREP.
Source Node ID	SNID	`char snid[NID_BYTES];`	Indica el nodo que inició la respuesta de solicitud de posición, es decir el que está indicando sus coordenadas geográficas en esta misma trama.
Latitude	latitude	`char latitude[MAXALEN];`	Latitud geográfica del nodo en el momento de generar el PREP. Se obtiene de un GPS.
Longitude	longitude	`char longitude[MAXALEN];`	Longitud geográfica del nodo en el momento de generar el PREP. Se obtiene de un GPS.

Table 5.8: Campos de la trama PREP del protocolo AODV-LAB

Para una mejor comprensión de la trama y sus campos, a continuación se muestra un ejemplo comentado.

```
[NNID|frameID|DNID| timeStamp |SNID|latitude|longitude]
[ D  |   C   | D  |1304516621| A  | 41.2061|  1.7300 ]
```

Esta trama PREP, con `frameID C`, va dirigida al nodo D, que además fué el que inició la solicitud de posiciones con un mensaje PREQ. El timestamp que contiene la fecha y hora en la que el nodo `A` consulta su posición al GPS es `1304516621`. Por último, los campos `SNID`, `latitude` y `longitude` indican que en el momento en que se generó el PREP el nodo `A` se encontraba en las coordenadas geográficas `41.2061 N - 1.73 E`.

5.3.4. Estructura y flujo de ejecución del programa

En la Figura 5.7 se muestra la estructura del programa principal, donde se pueden diferenciar claramente dos etapas, una primera que realiza una serie de operaciones de manera secuencial y, la segunda que lo hace en bucle indefinidamente hasta que se detiene la ejecución del programa.

En la primera parte se realizan las siguientes operaciones:

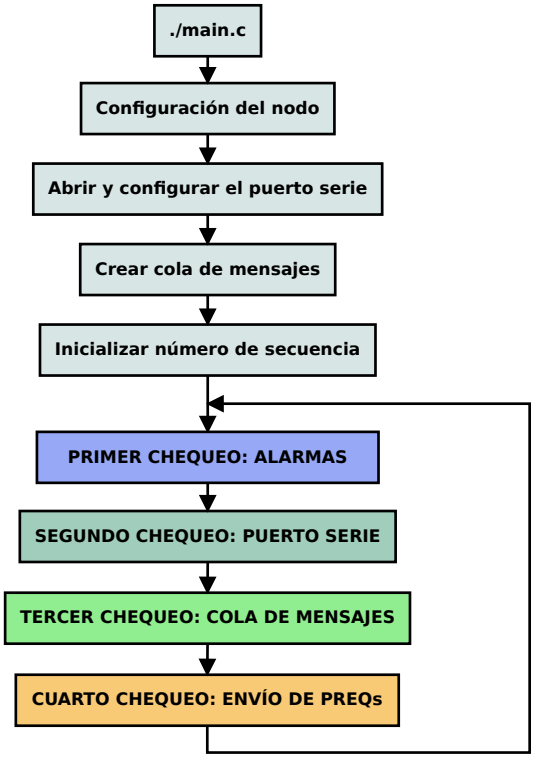


Figure 5.7: Estructura y flujo de ejecución del programa.

- **Configuración del nodo**. Lectura del fichero de configuración `node.conf` donde se definen los siguientes parámetros:
  - `SERIAL_NAME`. Nombre del puerto serie en el sistema. En GNU/Linux serán de la siguiente manera: `/dev/ttyUSB0`.
  - `SERIAL_BAUD_RATE`. Velocidad del puerto serie en bits por segundo (bps). En el fichero `node.conf` se indican los valores posibles.
  - `NODE_ID`. Identificador del propio nodo.
  - `SINK_NODE_ID`. Identificador del nodo colector.
  - `MAX_NUM_HOPS`. Número máximo de saltos en una ruta.
  - `LIFETIME_RTENTRY`. Tiempo de vida de las entradas de la tabla de rutas. Tras este periodo las rutas dejan de ser válidas.
  - `ALARM_RETRIES`. Número de reintentos de envío de una alarma en caso de que no se entregue con éxito.
  - `ALARM_TIMEOUT`. Tiempo de espera tras el envío de una alarma para recibir su correspondiente acuse de recibo (ACK). Si en ese periodo no llega, se procederá al reenvío de la alarma.
  - `RREQ_TIMEOUT`. Tiempo de espera tras el envío de una solicitud de ruta (RREQ) para recibir su correspondiente respuesta (RREP). Si en ese periodo no llega, se procederá al reenvío del RREQ.

- **Abrir y configurar el puerto serie**, utilizando los valores leídos anteriormente en el fichero `node.conf`.
- **Crear cola de mensajes**. Ésta seguirá la disciplina FIFO. En ella se guardaran los mensajes que se reciban por el puerto serie, y se tomarán posteriormente para ser procesados.
- **Inicializar número de secuencia**. El número de secuencia se inicializa siempre que se ejecuta el programa con la función `time()`. De esta manera, aunque un nodo se resetee, al reiniciarse no causará ningún problema en la red, ya que los números de secuencia que utilizará serán mayores que los empleados antes del reseteo.

Estas operaciones son imprescindibles para para que el nodo pueda operar en la red correctamente. En caso de que alguna de ellas falle, se detendrá la ejecución del programa automáticamente, indicando por pantalla el motivo de error para facilitar su corrección.

En la segunda parte del programa se realizan en bucle cuatro procesos distintos. A continuación, se detalla cada uno de ellos.

5.3.4.1. Primer chequeo: Alarmas

El proceso de chequeo de alarmas se encarga de gestionar: la recepción de acuses de recibo, el envío y reenvío alarmas y, en caso de ser necesario, el inicio del descubrimiento de ruta hacia el nodo colector. En la Figura 5.8 se muestra el diagrama de flujo de este proceso. En él se puede observar que operaciones se van realizando según las condiciones actuales del sistema como: estado de espera de acuse de recibo, contador de reeintentos de envío de alarmas, caminos válidos disponibles en la tabla de rutas, etc.

5.3.4.2. Segundo chequeo: Puerto serie

En segundo lugar, se realiza el chequeo del puerto serie. Como se observa en la Figura 5.9, inicialmente se lee el puerto durante `SERIAL_TIMEOUT`, definido en el programa. Si no hay nada de información se finaliza el proceso. En caso contrario, se buscan tramas completas entre los datos recibidos y, si se encuentra alguna, se coloca en la cola de mensajes. Si, a pesar de haber recibido información por el puerto serie, no se detecta ninguna trama completa esta se descarta y se finaliza el chequeo.

5.3.4.3. Tercer chequeo: Cola de mensajes

En esta parte del programa principal se leen todos los mensajes recibidos y almacenados en la cola de mensajes hasta ese momento. Esto es así por dos razones:

- Cuando se procesa un mensaje se actualiza la información del propio sistema, del estado de la red, etc. Por ello, cada vez que se llega a esta parte del programa principal, interesa procesar todos los mensajes actualizando así al máximo dicha información, que será utilizada en otros chequeos, como el de alarmas. Esto evita el envío de mensajes innecesarios, cosa que podría ocurrir si al pasar por el chequeo de la cola de mensajes solo se procesase un mensaje por pasada.

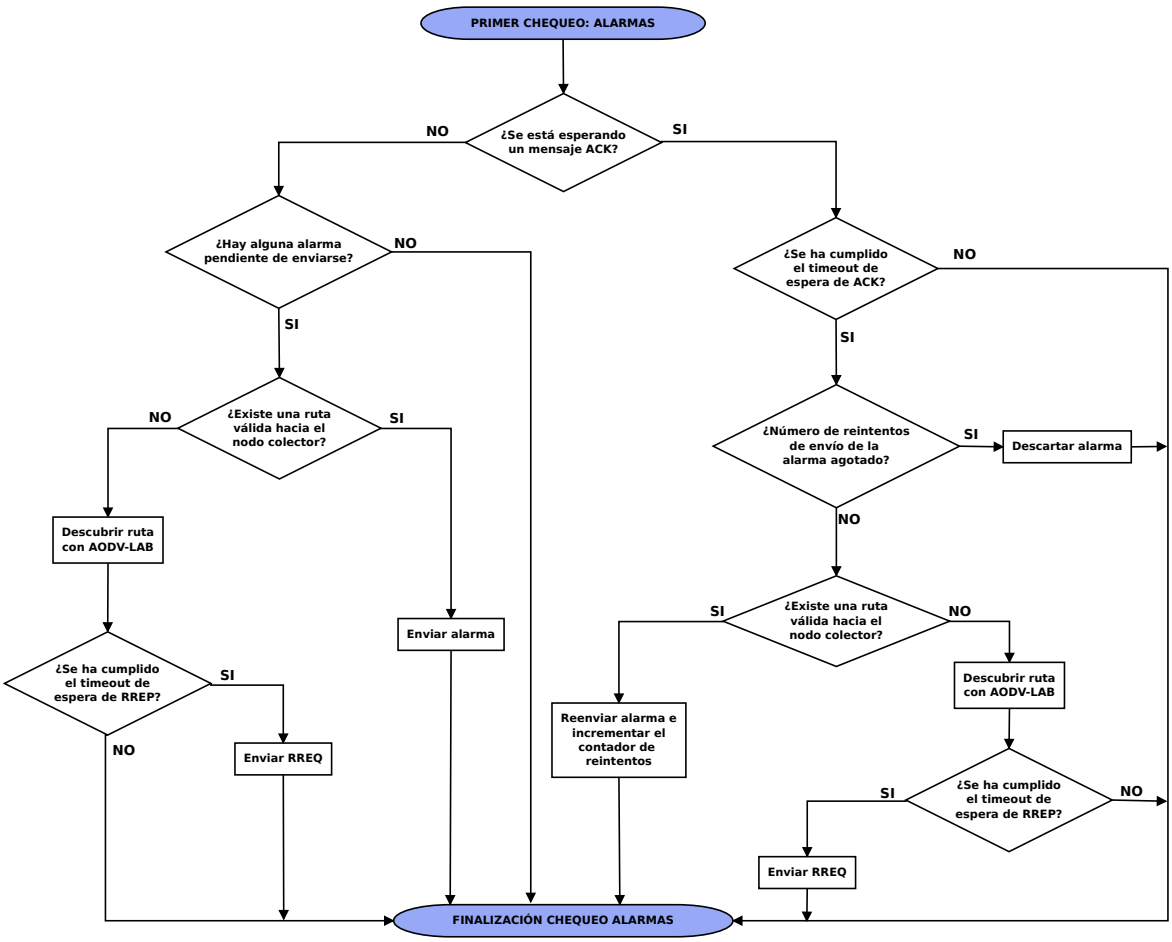


Figure 5.8: Diagrama de flujo de la parte de chequeo de alarmas.

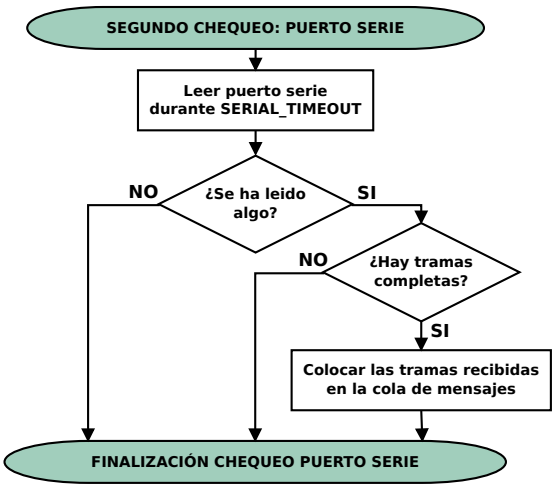


Figure 5.9: Diagrama de flujo de la parte de chequeo del puerto serie.

- Cada vez que se finaliza esta acción la cola de mensajes queda vacía, evitando problemas de almacenamiento de datos.

Como se observa en la Figura 5.10, cuando se toma un mensaje de la cola, el nodo se encarga de comprobar si es el destinatario del mismo, o si éste fue enviado en modo broadcast. Si se da uno de estos dos casos, existe una función encargada del procesamiento de cada cada tipo de mensaje. Así, dependiendo de su campo `frameID`, el mensaje es enviado a la función correspondiente para su procesado. Al terminar esta operación, o tras haber descartado un mensaje, se vuelve a comprobar si hay más en la cola y, únicamente cuando esta está vacía, se termina el proceso.

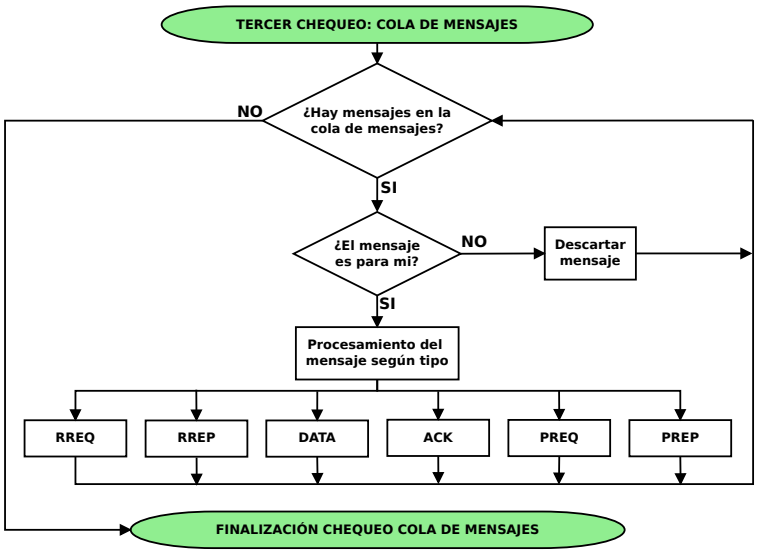


Figure 5.10: Diagrama de flujo de la parte de chequeo de la cola de mensajes.

5.3.4.4. Cuarto chequeo: Envío de PREQ

Como se comentó en la Subsección 5.1.2, el nodo colector tiene la posibilidad de enviar mensajes de solicitud de posición en modo broadcast para conocer la localización exacta del resto de nodos de la red. De la gestión de esta operación se encarga el proceso de chequeo de envío de mensajes PREQ. Como se muestra en la Figura 5.11, en esta parte del programa el nodo comprueba si el mismo es el colector o no. En caso de serlo, la señal para enviar un mensaje PREQ (*Position Request*) en modo broadcast es la pulsación de la letra w o W, que proviene de la sentencia *Where are you?*. Si esta tecla se ha pulsado, al llegar a este punto el nodo enviará un PREQ y finalizará este cuarto chequeo, volviendo nuevamente al primero, como se mostraba en la Figura 5.7. Incluir esta parte en el programa permite emplear un código génerico para todos los nodos, independientemente de ser o no colector, utilizando el archivo de configuración para definir estas características.

5.4. Validación y resultados

Para verificar el correcto funcionamiento del software implementado se ha realizado una simulación del sistema completo. Es importante destacar que un sistema de estas características en funcionamiento puede dar lugar a un gran número de situaciones distintas, que vendrán determinadas por: el flujo de alarmas, el número de nodos de la red,

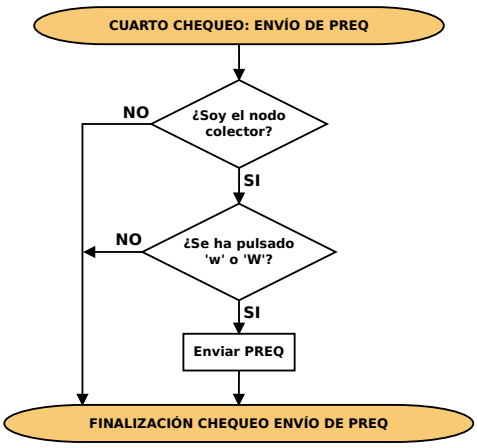
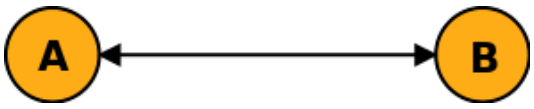


Figure 5.11: Diagrama de flujo de la parte de envío de mensajes PREQ.

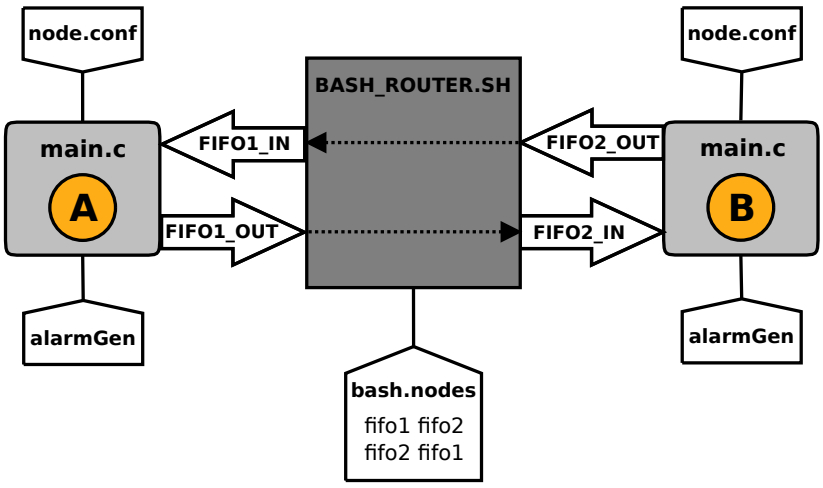
los parámetros de configuración u otras circunstancias. Contemplar todos estos casos en una simulación es una tarea larga y compleja que, para la implementación de una primera versión o prototipo, no tiene lugar. Por ello, esta sección únicamente pretende constatar las principales funcionalidades deseadas, tanto del protocolo de encaminamiento AODV-LAB, como del módulo de entrega remota de datos. Concretamente se han simulado cuatro escenarios, cada uno de ellos con una distribución distinta, en número y posición, de nodos. A continuación, se detalla el procedimiento seguido para la simulación, así como las características de cada escenario y los resultados obtenidos.

5.4.1. Método de simulación

Emular el funcionamiento real del sistema completo desarrollado implica, por un lado, simular los nodos de la red operando independientemente y, por otro, los enlaces entre ellos. Las Figuras 5.12 y 5.13 representan un esquema global del método utilizado.



(a) Representación de un enlace bidireccional entre dos nodos.



(b) Esquema de simulación de un enlace bidireccional entre dos nodos.

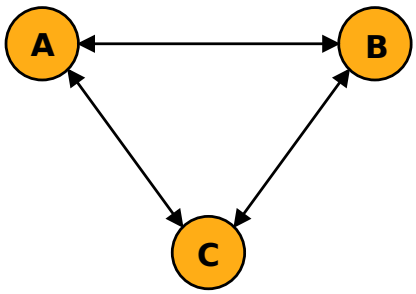
Figure 5.12: Simulación de un enlace bidireccional entre dos nodos.

Cada nodo del sistema en la simulación consta principalmente de tres partes:

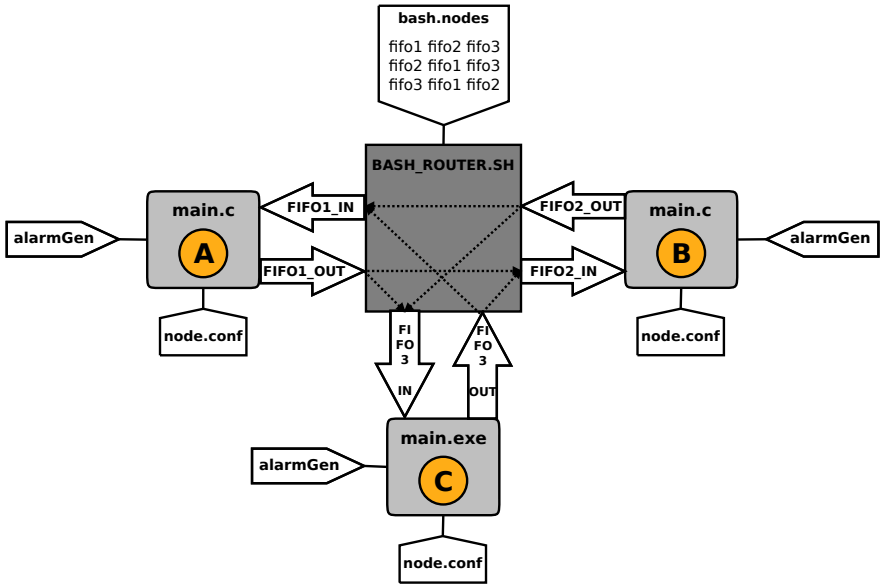
- Fichero de configuración (`node.config`). Contiene los parámetros de configuración del nodo, por ejemplo: identificador del propio nodo, identificador del nodo colector, número máximo posible de saltos en una ruta, etc. (Véase la Subsección 5.3.4).
- Generador aleatorio de alarmas (`alarmGen`). Cada cierto intervalo de tiempo, de duración variable entre dos márgenes definidos, se forma una alarma y se incluye en el fichero `alarm.dat`, que si no existe previamente se crea. Cada alarma lleva su correspondiente identificador que la diferencia inequívocamente de las demás.
- Programa principal (`main`). Consiste en el módulo de entrega remota de datos descrito en la Sección 5.1, que a su vez utiliza el protocolo de encaminamiento AODV-LAB, descrito en la Subsección 5.2.4.

Durante la simulación, los distintos nodos existentes se ejecutan en paralelo en la misma máquina, en este caso bajo el sistema operativo GNU/Linux Ubuntu 10.10 64 bits.

Por otro lado, se debe definir el modo en que los nodos se comunicarán entre sí. En la realidad, dos nodos podrán comunicarse cuando se encuentren respectivamente uno dentro del área de cobertura del otro. En la simulación, esto se simplifica y se definen directamente los enlaces entre los nodos. Esto tiene lugar en el fichero `bash.nodes`, donde en la primera columna aparecen listados todos los nodos de la simulación y a la derecha de cada uno la lista de nodos con los que puede comunicarse. Por ejemplo,



(a) Representación de enlaces bidireccionales entre tres nodos.



(b) Esquema de simulación de enlaces bidireccionales entre tres nodos.

Figure 5.13: Simulación de enlaces bidireccionales entre tres nodos.

el siguiente fragmento nos indican que el nodo 1 puede comunicarse con el 2, éste a su vez con el nodo 1 y 3 y, por último, el nodo 3 únicamente con el nodo 2.

Ejemplo fichero `bash.nodes`:

```
fifo1 fifo2
fifo2 fifo1 fifo3
fifo3 fifo2
```

Se emplea la notación `fifoX`, donde `X` es el número correspondiente al nodo, ya que la información se escribe y lee en ficheros tipo FIFO. Concretamente se utilizan ficheros `fifoX_out` donde los nodos escriben la información que transmiten y `fifoX_in` donde leen la que reciben. Para crear estos ficheros desde consola basta con introducir el comando: `$ mkfifo fifoX_out`. Para que los nodos trabajen en modo simulación, es decir utilizando los ficheros FIFO, en vez de en modo de operación, que emplea el puerto serie, debe ponerse la variable global `SIMULATION` en `serial.c` a uno.

Por último, se ha implementado un script en bash, `bash_router.sh`, que se encarga de leer periódica y secuencialmente los ficheros `fifoX_out` y en caso de encontrar algún mensaje escrito en alguno de ellos lo copia en los ficheros `fifoX_in` correspondientes

según esté indicado en `bash.nodes`.

Este método de simulación se comporta como si se trabajase en un medio ideal, sin interferencias ni ruido. Para emular una rotura de enlace se procederá deteniendo la ejecución del `main` de algún nodo. Esto se tratará con mayor detalle en las próximas subsecciones.

Además, se ha programado el script `bash_router.sh` para que imprima por pantalla los mensajes que se escriben en cada fichero `fifoX_out`. De esta manera, y apollándose en el formato de las tramas descrito en la Subsección 5.3.3, se comprenderá mejor el modo de operación de la red. También se utilizarán los ficheros `alarmReg.dat` que genera el nodo colector, `nodo D` en todos los ejemplos, para registrar las alarmas que recibe. Así, se tendrá constancia del orden de llegada de estas, o de si llegan repetidas. Se procederá de igual manera con el fichero `positionReg.dat` cuando sea necesario. Por último, en algunos casos, cuando esto clarifique la explicación de lo ocurrido durante la simulación, se utilizarán fragmentos de la información que van reportando los nodos durante su operación en la red.

5.4.2. Escenario 1

El primer escenario a simular es el caso más simple posible. Como se observa en la Figura 5.14, se trata de un enlace entre dos nodos, donde el `nodo S` sería un nodo generador de alarmas y el `nodo D` un nodo colector.

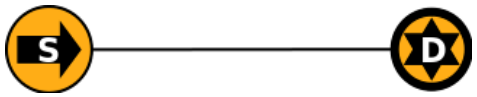


Figure 5.14: Escenario 1 de simulación:
Enlace bidireccional entre dos nodos.

A continuación, se detallan las diferentes situaciones simuladas sobre este escenario y se comentan los resultados obtenidos. Destacar que `fifo2_out` es lo que transmite el `nodo S` y `fifo1_out` lo del `nodo D`.

1. **Descubrimiento de nodos vecinos, envío de alarmas y acuses de recibo.** El `nodo S` tiene dos alarmas pendientes de enviar.

```
1  // El nodo S envía en modo broadcast un RREQ buscando una
2  // ruta hacia el nodo D
3  fifo2_out: [0|Q|S|1304421715|D|S|1]                        // RREQ
4  // Cuando el nodo D recibe el RREQ, guarda la ruta hacia
5  // el nodo S y responde en modo unicast con un RREP
6  fifo1_out: [S|P|D|D|1304421716|1|S]                        // RREP
7  // Tras recibir el nodo S el RREP, conoce una ruta hacia
8  // el nodo D y envía la primera alarma
9  fifo2_out: [D|D|D|S|1|1304421690|W|41.2061|1.7300|87]   // ALARM1
10 // Cuando el nodo D recibe la alarma envía un acuse de
11 // recibo hacia el nodo S
12 fifo1_out: [S|A|S|1]                                       // ACK1
13 // Sólo cuando el nodo S recibe el acuse de recibo de la
14 // alarma anterior, procede a enviar la siguiente
15 fifo2_out: [D|D|D|S|2|1304421694|W|41.2061|1.7300|87]   // ALARM2
16 // Al recibir el nodo D la alarma, envía un acuse de
```

```
// recibo hacia el nodo S
fifo1_out: [S|A|S|2]                                          // ACK2
// El nodo S recibe el acuse de recibo y como no tiene más
// alarmas pendientes, por el momento no envía nada más.
```

Fichero `alarmReg.dat` generado por el `nodo D`:

alarm ID	timeStamp	alarmType	latitude	longitude	perConf
1	1304421690	W	41.2061	1.7300	87
2	1304421694	W	41.2061	1.7300	87

Destacar que los campos `alarmType`, `latitude`, `longitude` y `perConf` de las alarmas que genera el `alarmGen` no varían, únicamente varían el `alarmID` y el `timeStamp`.

2. **Falla la comunicación temporalmente entre los dos nodos, pero ninguno se resetea**. El `nodo S` tiene dos alarmas pendientes de enviar.

```
// El nodo S envía en modo broadcast un RREQ buscando una
fifo2_out: [0|Q|S|1304423364|D|S|1]                           // RREQ
fifo1_out: [S|P|D|D|1304423365|1|S]                           // RREP
fifo2_out: [D|D|D|S|1|1304423354|W|41.2061|1.7300|87]         // ALARM1
// Tras ALARM_TIMEOUT sin recibir el correspondiente ACK y,
// como ALARM_RETRIES es 2, el nodo S vuelve a intentar
// enviar su alarma
fifo2_out: [D|D|D|S|1|1304423354|W|41.2061|1.7300|87]         // ALARM1
// Esta vez llega correctamente al nodo D y éste responde
// con el correspondiente acuse de recibo
fifo1_out: [S|A|S|1]                                          // ACK1
fifo2_out: [D|D|D|S|2|1304423359|W|41.2061|1.7300|87]         // ALARM2
fifo1_out: [S|A|S|2]                                          // ACK2
```

Fichero `alarmReg.dat` generado por el `nodo D`:

alarm ID	timeStamp	alarmType	latitude	longitude	perConf
1	1304423354	W	41.2061	1.7300	87
1	1304423354	W	41.2061	1.7300	87
2	1304423359	W	41.2061	1.7300	87

Al contemplar el fichero de registro de alarmas se ve que la primera está repetida. Esto indica que la alarma llegó perfectamente al `nodo D` y que la comunicación falló exáctamente en el envío del acuse de recibo. Por ello, el `nodo S` al no recibirlo desconoce que su alarma ha llegado correctamente y, tras comprobar que en su configuración se ha fijado el número de reintentos a dos, la vuelve a enviar.

3. **Agotar los reintentos con la última alarma del** `alarm.dat`. El `nodo S` tiene dos alarmas pendientes de enviar.

```
fifo2_out: [0|Q|S|1304424109|D|S|1]                          // RREQ
fifo1_out: [S|P|D|D|1304424110|1|S]                          // RREP
fifo2_out: [D|D|D|S|1|1304424100|W|41.2061|1.7300|87]        // ALARM1
fifo1_out: [S|A|S|1]                                         // ACK1
fifo2_out: [D|D|D|S|2|1304424104|W|41.2061|1.7300|87]        // ALARM2
// Tras ALARM_TIMEOUT, primer reintento
fifo2_out: [D|D|D|S|2|1304424104|W|41.2061|1.7300|87]        // ALARM2
// Tras ALARM_TIMEOUT segundo y último reintento
fifo2_out: [D|D|D|S|2|1304424104|W|41.2061|1.7300|87]        // ALARM2
// Descarta la alarma y, como no hay nuevas alarmas pendientes,
// no realiza ninguna operación más
```

Fichero `alarmReg.dat` generado por el `nodo D`:

alarm ID	timeStamp	alarmType	latitude	longitude	perConf
1	1304424100	W	41.2061	1.7300	87

Tras enviar una alarma correctamente, al tratar de enviar la segúnda y última pendiente en el fichero `alarm.dat`, no recibe respuesta del `nodo D`. Efectúa el número de reintentos definido por `ALARM_RETRIES` y una vez se agotan, al ser la última pendiente, la descarta y no realiza más operaciones. En el fichero `alarmReg.dat` generado por el `nodo D` únicamente aparece la primera alarma. Esto índica que la comunicación falló antes de que este nodo recibiese la segunda y probablemente no se volvió a recuperar. Esto podría corresponderse con una situación en la realidad en la que el nodo colector sale del área de cobertura de un nodo que le estaba enviando alarmas. Al proceder como se ha descrito se evita que el nodo continúe enviando mensajes sin sentido y por lo tanto malgastnado recursos. La alarma se descarta porque tras dos reintentos habrá transcurrido tres veces el tiempo definido en `ALARM_TIMEOUT`, por lo cual, la información de dicha alarma será anticuada.

4. **Agotar los reintentos con una alarma y tener otras posteriores pendientes en el** `alarm.dat`. El `nodo S` tiene tres alarmas pendientes de enviar.

```
fifo2_out: [0|Q|S|1304424294|D|S|1]                          // RREQ
fifo1_out: [S|P|D|D|1304424295|1|S]                          // RREP
fifo2_out: [D|D|D|S|1|1304424257|W|41.2061|1.7300|87]        // ALARM1
fifo1_out: [S|A|S|1]                                         // ACK1
fifo2_out: [D|D|D|S|2|1304424260|W|41.2061|1.7300|87]        // ALARM2
// Tras ALARM_TIMEOUT, primer reintento
fifo2_out: [D|D|D|S|2|1304424260|W|41.2061|1.7300|87]        // ALARM2
// Tras ALARM_TIMEOUT segundo y último reintento
fifo2_out: [D|D|D|S|2|1304424260|W|41.2061|1.7300|87]        // ALARM2
// Descarta la alarma 2 y como todavía tiene una tercera
// pendiente inicia un Descubrimiento de Ruta para tratar
// de encontrar un camino válido al nodo D, ya que el que
// tenía no le ha permitido entregar la alarma anterior.
fifo2_out: [0|Q|S|1304424297|D|S|1]                          // RREQ
fifo1_out: [S|P|D|D|1304424347|1|S]                          // RREP
// Descubre una ruta, que puede ser la misma de antes
// o no, y envía la siguiente alarma pendiente
fifo2_out: [D|D|D|S|3|1304424267|W|41.2061|1.7300|87]        // ALARM3
fifo1_out: [S|A|S|3]                                         // ACK3
// El nodo S no tiene más alarmas que entregar
```

Fichero `alarmReg.dat` generado por el `nodo D`:

alarm ID	timeStamp	alarmType	latitude	longitude	perConf
1	1304424257	W	41.2061	1.7300	87
3	1304424267	W	41.2061	1.7300	87

Esta situación es similar a la anterior, pero esta vez, después de la alarma descartada, hay otra pendiente de ser enviada. Por ello, el `nodo S` reinicia el proceso de Descubrimiento de Ruta. Si éste concluye con éxito, como es el caso, continúa mandando las alarmas por la nueva ruta, que podría ser la misma que la que falló. En el fichero `alarmReg.dat` se puede ver que únicamente han quedado registradas la primera y última alarma.

5. **Tratar periódicamente de descubrir una ruta hacia el nodo colector cuando hay alarmas pendientes de ser enviadas**. El `nodo S` tiene dos alarmas pendientes de enviar.

```
fifo2_out: [0|Q|S|1304424779|D|S|1]                          // RREQ1
// Tras RREQ_TIMEOUT se vuelve a enviar un RREQ
fifo2_out: [0|Q|S|1304424780|D|S|1]                          // RREQ2
// Tras RREQ_TIMEOUT se vuelve a enviar un RREQ
fifo2_out: [0|Q|S|1304424781|D|S|1]                          // RREQ3
// Tras RREQ_TIMEOUT se vuelve a enviar un RREQ
fifo2_out: [0|Q|S|1304424782|D|S|1]                          // RREQ4
// Tras RREQ_TIMEOUT se vuelve a enviar un RREQ
fifo2_out: [0|Q|S|1304424783|D|S|1]                          // RREQ5
fifo1_out: [S|P|D|D|1304424811|1|S]                          // RREP
fifo2_out: [D|D|D|S|1|1304424753|W|41.2061|1.7300|87]        // ALARM1
fifo1_out: [S|A|S|1]                                         // ACK1
fifo2_out: [D|D|D|S|2|1304424756|W|41.2061|1.7300|87]        // ALARM2
fifo1_out: [S|A|S|2]                                         // ACK2
```

Fichero `alarmReg.dat` generado por el `nodo D`:

alarm ID	timeStamp	alarmType	latitude	longitude	perConf
1	1304424753	W	41.2061	1.7300	87
2	1304424756	W	41.2061	1.7300	87

El Descubrimiento de Ruta, al contrario que el envío de una alarma, no tiene un número de reintentos definido. Mientras haya alarmas pendientes de ser enviadas, el sistema tratará de descubrir una ruta hacia el nodo colector periódicamente cada `RREQ_TIMEOUT`. Una vez que encuentre una ruta, enviará las alarmas que tenga acumuladas en su fichero `alarm.dat`.

6. **Importancia de la inicialización del número de secuencia con la función `time()`**. El `nodo S` tiene dos alarmas pendientes de enviar.

```
1  fifo2_out: [0|Q|S|1304425663|D|S|1]                          // RREQ
2  fifo1_out: [S|P|D|D|1304425664|1|S]                          // RREP
3  fifo2_out: [D|D|D|S|1|1304425637|W|41.2061|1.7300|87]        // ALARM1
4  fifo1_out: [S|A|S|1]                                         // ACK1
5  // El nodo S se resetea y por lo tanto pierde toda
6  // la información que tenía: sus tablas, el identificador
7  // de la última alarma que envió, etc. Tras un periodo
8  // de tiempo se reinicia y como tiene alarmas en el
9  // alarm.dat inicia el Descubrimiento de Ruta.
10 fifo2_out: [0|Q|S|1304425677|D|S|1]                          // RREQ
11 fifo1_out: [S|P|D|D|1304425678|1|S]                          // RREP
12 // Una vez tiene una ruta hacia el nodo D, vuelve a
13 // enviar la primera alarma, ya que en el reseteo perdió
14 // la información de cual fué la última que envió.
15 fifo2_out: [D|D|D|S|1|1304425637|W|41.2061|1.7300|87]        // ALARM1
16 fifo1_out: [S|A|S|1]                                         // ACK1
17 fifo2_out: [D|D|D|S|2|1304425643|W|41.2061|1.7300|87]        // ALARM2
18 fifo1_out: [S|A|S|2]                                         // ACK2
```

Fichero `alarmReg.dat` generado por el `nodo D`:

alarm ID	timeStamp	alarmType	latitude	longitude	perConf
1	1304425637	W	41.2061	1.7300	87
1	1304425637	W	41.2061	1.7300	87
2	1304425643	W	41.2061	1.7300	87

Cuando un nodo se resetea pierde su número de secuencia y la información que guardaba en sus tablas de: RREQ, encaminamiento y PREQ. Por ello, al reiniciarse vuelve a lanzar el proceso de Descubrimiento de Ruta, como se observa en la línea 10. Si sus nodos vecinos no se han reseteado, para que estos al recibir el mensaje RREQ no lo descarten, su número de secuencia debe ser superior al guardado anteriormente en sus tablas de RREQ, que en este caso se correspondería con el del RREQ de la línea 1, es decir, `1304425663`. Tal como se veía en la Subsección 5.3.4, al utilizar la función `time()` para inicializar el número de secuencia, en vez de hacerlo con cero, por ejemplo, obtendremos un número mayor que el primero, en este caso `1304425677`. La diferencia entre ambos valores, que

es 14, nos indica el número de segundos transcurrido entre las dos inicializaciones. Este número siempre será mayor que el número de mensajes enviados en ese intervalo, por lo tanto, procediendo de esta manera la red soporta sin problemas el reseteo de alguno de sus nodos.

Como vemos en el fichero `alarmReg.dat`, la primera alarma se recibió dos veces, ya que en el reseteo además se perdió el identificador de la última alarma enviada del fichero `alarm.dat`. Este efecto podría ser muy negativo si hubiese muchas alarmas ya enviadas y hubiese que volver a enviarlas pero, por la aplicación que tendrá el servicio de alarmas del sistema LIDO DCL, será muy poco común que se produzca una gran acumulación de estas en un nodo.

7. **Caducidad de las rutas guardadas en la tabla de encaminamiento**. El `nodo S` tiene inicialmente una alarma pendiente de enviar y, tras un periodo de tiempo superior a `LIFETIME_RTENTRY`, aparecen dos nuevas alarmas que entregar.

```
fifo2_out: [0|Q|S|1304427351|D|S|1]                          // RREQ
fifo1_out: [S|P|D|D|1304427352|1|S]                          // RREP
fifo2_out: [D|D|D|S|1|1304427340|W|41.2061|1.7300|87]        // ALARM1
fifo1_out: [S|A|S|1]                                         // ACK1
// Tras el envío de la primera alarma transcurre un
// tiempo superior a LIFETIME_RTENTRY, y la ruta al
// nodo colector caduca. Al aparecer nuevas alarmas
// es necesario iniciar de nuevo el proceso de
// Descubrimiento de Ruta.
fifo2_out: [0|Q|S|1304427352|D|S|1]                          // RREQ
fifo1_out: [S|P|D|D|1304427353|1|S]                          // RREP
fifo2_out: [D|D|D|S|2|1304427389|W|41.2061|1.7300|87]        // ALARM2
fifo1_out: [S|A|S|2]                                         // ACK2
fifo2_out: [D|D|D|S|3|1304427393|W|41.2061|1.7300|87]        // ALARM3
fifo1_out: [S|A|S|3]                                         // ACK3
```

Fichero `alarmReg.dat` generado por el `nodo D`:

alarm ID	timeStamp	alarmType	latitude	longitude	perConf
1	1304427340	W	41.2061	1.7300	87
2	1304427389	W	41.2061	1.7300	87
3	1304427393	W	41.2061	1.7300	87

Si tras haber utilizado una ruta descubierta transcurre un tiempo sin utilizarla superior a `LIFETIME_RTENTRY`, esta caduca y deja de ser válida. Por ello, si en algún momento se requiere enviar algo al mismo destinatario será necesario volver a realizar el proceso de Descubrimiento de Ruta. Lo ideal será ajustar el valor de `LIFETIME_RTENTRY` en función de la frecuencia de cambio de posición de los nodos en la red. Si hay mucha movilidad interesa que sea un valor pequeño, mientras que si hay poca, o es prácticamente estática, se pueden fijar valores más altos.

8. **Reseteo del `nodo D` y recepción de alarmas**. El `nodo S` tiene tres alarmas pendientes de enviar.

```
fifo2_out: [0|Q|S|1304430565|D|S|1]                          // RREQ
fifo1_out: [S|P|D|D|1304430566|1|S]                          // RREP
```

```
fifo2_out: [D|D|D|S|1|1304430528|W|41.2061|1.7300|87]   // ALARM1
fifo1_out: [S|A|S|1]                                    // ACK1
// En este punto se resetea el nodo D, perdiendo la
// información de su tabla de encaminamiento entre
// entre otras. El nodo S continúa enviando sus alarmas
fifo2_out: [D|D|D|S|2|1304430534|W|41.2061|1.7300|87]   // ALARM2
// Al reiniciarse, el nodo D recibe la segunda alarma,
// pero su tabla de encaminamiento está vacía y para
// poder mandar su acuse de recibo debe encontrar una
// ruta hacia el nodo S, por ello esta vez es el nodo D
// el que inicial el proceso de Descubrimiento de Ruta
fifo1_out: [0|Q|D|1304430573|S|D|1]                     // RREQ
fifo2_out: [D|P|S|S|1304430574|1|D]                     // RREP
// Mientras que se completa el Descubrimiento de Ruta
// transcurre ALARM_TIMEOUT y el nodo S vuelve a
// enviar su segunda alarma
fifo2_out: [D|D|D|S|2|1304430534|W|41.2061|1.7300|87]   // ALARM2
// Esta vez el nodo D ya tiene una ruta válida en su
// tabla de encaminamiento y puede mandar su ACK
// correspondiente al nodo S
fifo1_out: [S|A|S|2]                                    // ACK2
fifo2_out: [D|D|D|S|3|1304430539|W|41.2061|1.7300|87]   // ALARM3
fifo1_out: [S|A|S|3]                                    // ACK3
```

Fichero `alarmReg.dat` generado por el `nodo D`:

alarm ID	timeStamp	alarmType	latitude	longitude	perConf
1	1304430528	W	41.2061	1.7300	87
2	1304430534	W	41.2061	1.7300	87
2	1304430534	W	41.2061	1.7300	87
3	1304430539	W	41.2061	1.7300	87

Como se comentaba en el punto 6, cuando un nodo se resetea pierde, entre otra información, su tabla de encaminamiento. En este caso, esto le ocurre al `nodo D`. Sucede que tras reiniciarse, recibe una alarma del `nodo S`, y debe arrancar el proceso de Descubrimiento de Ruta para poder contestarle con un ACK. Cuando esta operación termina, envía el correspondiente acuse de recibo y todo sigue funcionando con normalidad. Si el tiempo que tarda en descubrirse la nueva ruta es superior a `ALARM_TIMEOUT`, como ocurre en este caso, el `nodo S` volverá a enviar su alarma, y por eso vemos la número dos repetida en el fichero `alarmReg.dat`.

5.4.3. Escenario 2

El segundo escenario a simular consta de tres nodos alineados, dos de ellos generadores de alarmas, `nodo S` y `nodo A`, y el otro colector, `nodo D`. Su configuración se puede apreciar en la Figura 5.15. Ya que el `nodo S` no tiene conexión directa con el `nodo D`, cuando estos requieran comunicarse dependerán del `nodo A`, que actuará de puente entre ambos.

A continuación, se detallan las diferentes situaciones simuladas sobre este escenario y se comentan los resultados obtenidos. Destacar que `fifo3_out` es lo transmitido por el `nodo S`, `fifo2_out` lo del `nodo A` y `fifo1_out` lo del `nodo D`.

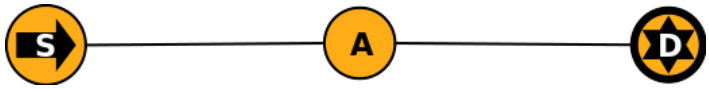


Figure 5.15: Escenario 2 de simulación:
Enlace bidireccional entre tres nodos alineados.

1. **Descubrimiento de nodos no vecinos, envio de alarmas y acuses de recibo. Multisalto**. El `nodo S` tiene dos alarmas pendientes de enviar y el `nodo A` ninguna.

```
// El nodo S envía en modo broadcast un RREQ buscando una
// ruta hacia el nodo D
fifo3_out: [0|Q|S|1304433773|D|S|1]                          // RREQ
// El nodo A recibe el RREQ del nodo S, guarda una ruta
// hacia él en su tabla de encaminamiento y vuelve a
// enviar en modo broadcast el mensaje RREQ con sus
// campos correspondientes actualizados
fifo2_out: [0|Q|S|1304433773|D|A|2]                          // RREQ
// Cuando el nodo D recibe el RREQ, guarda la ruta hacia
// el nodo S a través del A, y responde con un RREP unicast
fifo1_out: [A|P|D|D|1304433774|1|S]                          // RREP
// El nodo A recibe el RREP del nodo D, guarda una ruta
// hacia él en su tabla de encaminamiento y vuelve a
// enviar en modo unicast el mensaje RREP con sus
// campos correspondientes actualizados al nodo S
fifo2_out: [S|P|A|D|1304433774|2|S]                          // RREP
// Tras recibir el nodo S el RREP, conoce una ruta hacia
// el nodo D y envía la primera alarma
fifo3_out: [A|D|D|S|1|1304433727|W|41.2061|1.7300|87]    // ALARM1
// El nodo A recibe la alarma 1 del nodo S y la reenvía
// según su tabla de encaminamiento al nodo D
fifo2_out: [D|D|D|S|1|1304433727|W|41.2061|1.7300|87]    // ALARM1
// Cuando el nodo D recibe la alarma envía un acuse de
// recibo hacia el nodo S, a través del A
fifo1_out: [A|A|S|1]                                         // ACK1
// El nodo A recibe el ACK1 del nodo D y lo reenvía
// según su tabla de encaminamiento al nodo S
fifo2_out: [S|A|S|1]                                         // ACK1
// Sólo cuando el nodo S recibe el acuse de recibo de la
// alarma anterior, procede a enviar la siguiente
fifo3_out: [A|D|D|S|2|1304433732|W|41.2061|1.7300|87]    // ALARM2
fifo2_out: [D|D|D|S|2|1304433732|W|41.2061|1.7300|87]    // ALARM2
fifo1_out: [A|A|S|2]                                         // ACK2
fifo2_out: [S|A|S|2]                                         // ACK2
// El nodo S recibe el acuse de recibo y como no tiene más
// alarmas pendientes, por el momento no envía nada más.
```

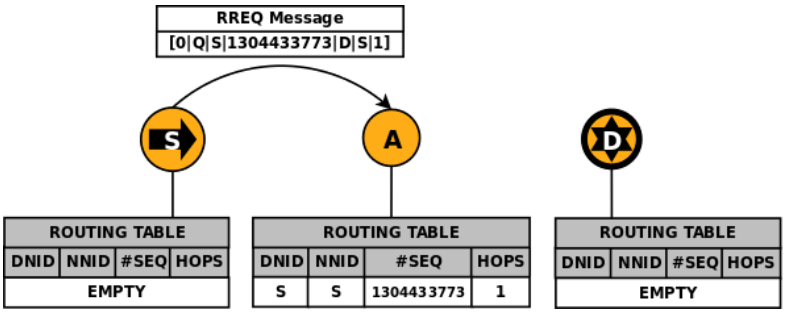
   Fichero `alarmReg.dat` generado por el `nodo D`:

El sistema es capaz de descubrir rutas hacia nodos no vecinos y, de igual manera, hacerles llegar alarmas y acuses de recibo. Esta característica se conoce con el nombre de multisalto. El parámetro `MAX_NUM_HOPS` del fichero de configuración

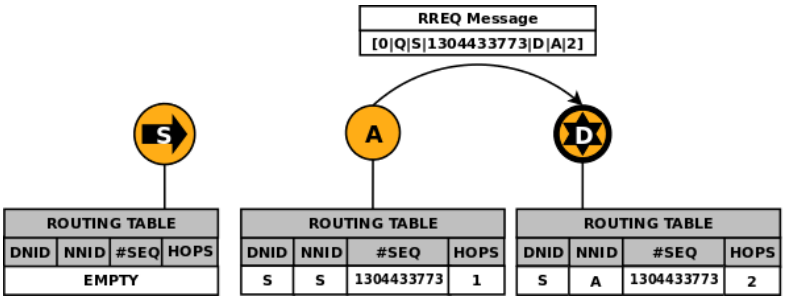
alarm ID	timeStamp	alarmType	latitude	longitude	perConf
1	1304433727	W	41.2061	1.7300	87
2	1304433732	W	41.2061	1.7300	87

`node.conf` determinará el número de saltos que puede haber entre dos nodos que se quieren comunicar.

En las Figuras 5.16 y 5.17, se muestra gráficamente la formación de los caminos de vuelta e ida indicando, en cada caso, el contenido de la tabla de encaminamiento de cada nodo.



(a) El nodo A recibe el RREQ que el nodo S envía en modo broadcast.

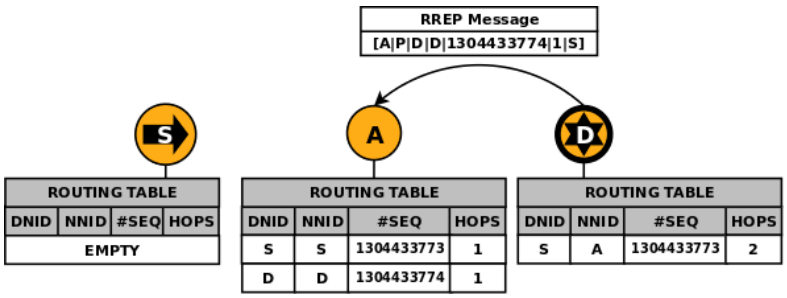


(b) El nodo D recibe el RREQ que el nodo A reenvía en modo broadcast.

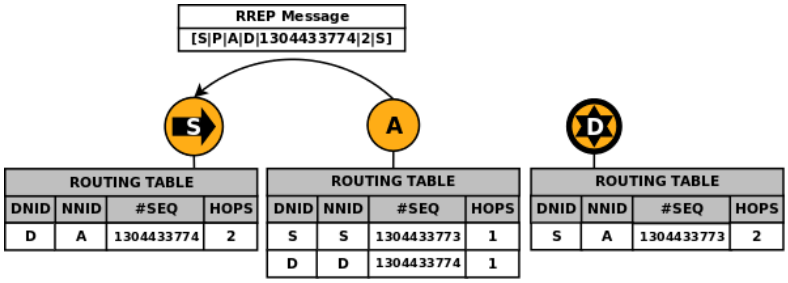
Figure 5.16: Formación del camino de vuelta del nodo D al S.

2. **Descarte de paquetes no dirigidos al propio nodo**.
Por clarificar la Figura 5.16b se ha representado el mensaje RREQ que recibe el `nodo D` del `nodo A`, pero en realidad, al retransmitirse ese mensaje con una antena omnidireccional también le llega al `nodo S`, que fué el que inició el Descubrimiento de Ruta. Este nodo, en primer lugar, no descarta el mensaje ya que el campo `NNID` indica modo broadcast y, por lo tanto, debe continuar procesándolo. Es más adelante, al comprobar que el campo `SNID` del mensaje corresponde con él mismo, cuando reconoce que ese mensaje fué generado por él y lo descarta. Por pantalla, el `nodo S` devuelve los siguientes mensajes:

```
RREQ message sent (24 bytes): [0|Q|S|1304433773|D|S|1]
Timeout!
Serial data received (25 bytes): [0|Q|S|1304433773|D|A|2]
RREQ message received!
This message is not for me.
Serial data received (25 bytes): [S|P|A|D|1304433774|2|S]
```



(a) El nodo A recibe el RREP que le envía el nodo D.



(b) El nodo S recibe el RREP que le reenvía el nodo A.

Figure 5.17: Formación del camino de ida del nodos S al D.

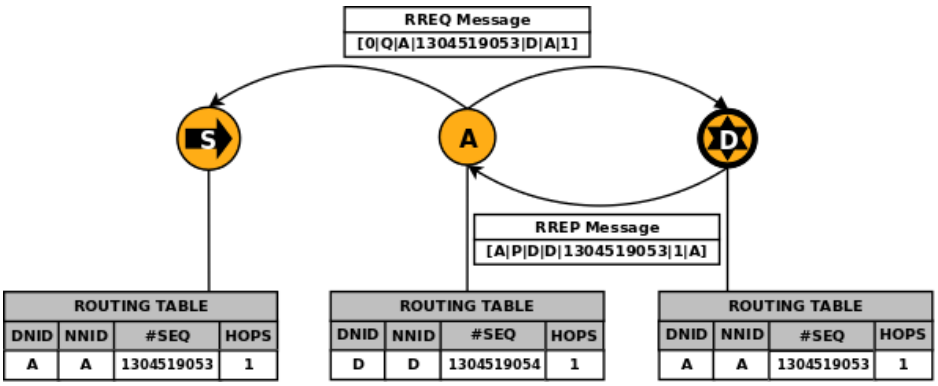
```
RREP message received!
```

Cuando un nodo recibe un mensaje que no se ha transmitido en broadcast y él no es el `NNID`, lo descarta de inmediato. Esto sucede por ejemplo cuando el `nodo S` envía su alarma hacia el `nodo D` y recibe la retransmisión que hace de esta el `nodo A`. Como las alarmas se envian de modo unicast, el `nodo S` descarta de inmediato ese mensaje y continua esperando su correspondiente ACK. Esta es la salida por pantalla del `nodo S`:

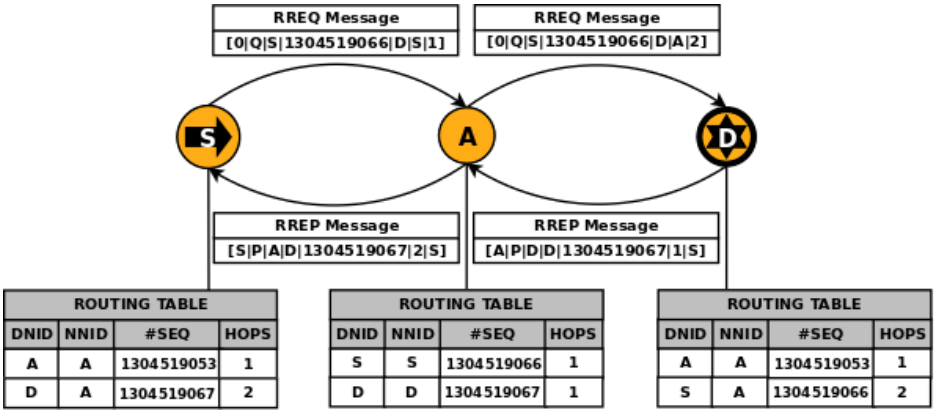
```
Alarm message sent (42 bytes): [A|D|D|S|1|1304433727|W|41.2061|1.7300|87]
Timeout!
Serial data received (43 bytes): [D|D|D|S|1|1304433727|W|41.2061|1.7300|87]
This message is not for me.
Serial data received (10 bytes): [S|A|S|1]
ACK message received!
```

3. **Actualización de tablas de encaminamiento**.

Inicialmente el `nodo A` dispone de varias alarmas que debe hacer llegar al `nodo D`. Por ello, realiza un Descubrimiento de Ruta hacia dicho nodo, como se puede observar en la Figura 5.18a. En ella, se muestran además las tablas de encaminamiento de todos los nodos al finalizar el proceso. A continuación, el `nodo S` empieza a generar alarmas y, como no tiene una ruta hacia el nodo colector, debe realizar otro Descubrimiento de Ruta, que se representa en la Figura 5.18b. Se puede observar como en las tablas de los nodos `S` y `D` se incorpora una entrada con la nueva ruta descubierta. La tabla del `nodo A` actualiza la entrada que ya tenía por una ruta más actual, es decir, con un número de secuencia mayor, y



(a) El nodo A realiza un Descubrimiento de Ruta hacia el nodo D.



(b) El nodo S realiza un Descubrimiento de Ruta hacia el nodo D.

Figure 5.18: Actualización de tablas de encaminamiento.

añade una nueva ruta hacia el `nodo S`. De esta manera, quedan todas actualizadas a la nueva situación en la que ambos nodos pueden enviar sus alarmas al nodo colector.

4. **Envío de alarmas desde dos nodos**. En la simulación, el `nodo A` envía siete alarmas y el `nodo S` cinco. En este caso se han modificado los campos `alarmType` de las alarmas poniendo en su lugar el identificador del nodo que las envía. De esta manera, al concluir la simulación, en el fichero `alarmReg.dat` del `nodo D` se puede observar en que orden llegaron y si se entregaron todas, como en este caso.

   Fichero `alarmReg.dat` generado por el `nodo D`:

alarm ID	timeStamp	alarmType	latitude	longitude	perConf
1	1304441541	A	41.2061	1.7300	87
2	1304441548	A	41.2061	1.7300	87
1	1304441543	S	41.2061	1.7300	87
3	1304441554	A	41.2061	1.7300	87
2	1304441549	S	41.2061	1.7300	87
4	1304441561	A	41.2061	1.7300	87
5	1304441566	A	41.2061	1.7300	87
3	1304441552	S	41.2061	1.7300	87
6	1304441570	A	41.2061	1.7300	87
7	1304441577	A	41.2061	1.7300	87
4	1304441555	S	41.2061	1.7300	87
5	1304441559	S	41.2061	1.7300	87

   Como se puede ver, todas las alarmas llegaron al nodo colector correctamente. Las del `nodo A` se entregaron antes a pesar de ser más, ya que la ruta que siguieron no era multisalto.

5. **Envío de alarmas desde dos nodos con pérdida temporal de la comunicación**. El `nodo A` entrega cinco alarmas y el `nodo S` seis. Durante el proceso se pierde temporalmente la comunicación entre los nodos generadores.

   Fichero `alarmReg.dat` generado por el `nodo D`:

alarm ID	timeStamp	alarmType	latitude	longitude	perConf
1	1304511821	A	41.2061	1.7300	87
2	1304511827	A	41.2061	1.7300	87
3	1304511830	A	41.2061	1.7300	87
1	1304511825	S	41.2061	1.7300	87
4	1304511821	A	41.2061	1.7300	87
1	1304511825	S	41.2061	1.7300	87
5	1304511856	A	41.2061	1.7300	87
2	1304511830	S	41.2061	1.7300	87
3	1304511837	S	41.2061	1.7300	87
4	1304511843	S	41.2061	1.7300	87
5	1304511850	S	41.2061	1.7300	87
6	1304511856	S	41.2061	1.7300	87

   Llegan todas las alarmas correctamente al nodo colector. Lo único destacable es que la alarma 1 del `nodo S` llega repetida ya que es entregada correctamente la primera vez pero; al perderse temporalmente la comunicación entre el `nodo A` y el `S`, éste no recibe el ACK correspondiente, por lo que tras `ALARM_TIMEOUT` la reenvía.

6. **Solicitud de posición de los nodos**. Durante el envío de los mensajes PREQ y PREP no se están enviando alarmas.

```
// El nodo D realiza una solicitud de posición al
// resto de nodos de la red
fifo1_out: [0|W|D|1304516600|D|1]
// El nodo A recibe la solicitud, la responde con
// mensaje PREP y reenvía esa solicitud nuevamente
// para propagarla por la red
fifo2_out: [D|C|D|1304516621|A|+90.1234|-127.4321]
fifo2_out: [0|W|D|1304516600|A|2]
// El nodo S recibe la solicitud que reenvió el nodo
// A, la responde con un mensaje PREP y la reenvía
// nuevamente para propagarla por la red
fifo3_out: [A|C|D|1304516622|S|+90.1234|-127.4321]
fifo3_out: [0|W|D|1304516600|S|3]
// Cuando el nodo A recibe el PREP del nodo S, lo
// envía según su tabla de encaminamiento hacia el
// nodo D
fifo2_out: [D|C|D|1304516622|S|+90.1234|-127.4321]
```

Fichero `positionReg.dat` generado por el `nodo D`:

timeStamp	NID	latitude	longitude
1304516621	A	+90.1234	-127.4321
1304516622	S	+90.1234	-127.4321

Observando los timestamp correspondientes a los nodos `A` y `S`, vemos que la respuesta de éste último se generó un segundo después que la del `nodo A`. Por lo tanto, en un caso real esa diferencia de tiempo será igual o mayor a ese valor.

7. **Envío conjunto de alarmas, PREQ y PREP**. El `nodo A` envía cuatro alarmas, el `S` otras cuatro y el `nodo D` realiza dos solicitudes de posición.

   Fichero `alarmReg.dat` generado por el `nodo D`:

alarm ID	timeStamp	alarmType	latitude	longitude	perConf
1	1304516437	S	41.2061	1.7300	87
1	1304516442	A	41.2061	1.7300	87
2	1304516449	A	41.2061	1.7300	87
2	1304516443	S	41.2061	1.7300	87
3	1304516453	A	41.2061	1.7300	87
4	1304516456	A	41.2061	1.7300	87
3	1304516449	S	41.2061	1.7300	87
4	1304516454	S	41.2061	1.7300	87

   Fichero `positionReg.dat` generado por el `nodo D`:

timeStamp	NID	latitude	longitude
1304516448	A	+90.1234	-127.4321
1304516452	S	+90.1234	-127.4321
1304516466	A	+90.1234	-127.4321
1304516469	S	+90.1234	-127.4321

   Tanto el envío de alarmas como las solicitudes de posición se realizan satisfactoriamente. En este caso, al haber más tráfico en la red, la diferencia de tiempo entre los timestamp de los mensajes PREP es mayor que en el caso anterior.

5.4.4. Escenario 3

El tercer escenario a simular consta de cinco nodos. Como se observa en la Figura 5.19, están configurados de modo que entre el `nodo S` y el `nodo D` existen dos posibles caminos. Uno de ellos es idéntico al del escenario dos, y el otro depende de los nodos `B` y `C`, por consiguiente, cuenta con tres saltos, en este caso el máximo permitido para una ruta. De esta manera, se rompe la dependencia del `nodo A` que tenían los nodos `S` y `D` para comunicarse, ya que en caso de fallo existiría una ruta alternativa.

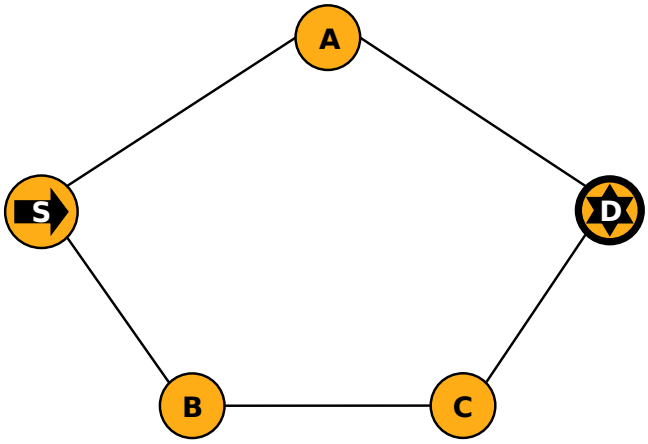


Figure 5.19: Escenario 3 de simulación: Varias rutas distintas entre dos nodos.

A continuación, se detallan las diferentes situaciones simuladas sobre este escenario y se comentan los resultados obtenidos.

1. **Descubrimiento de la ruta más corta entre los nodos `S` y `D`.**
   Partiendo de que todos los nodos tienen sus tablas de encaminamiento vacías, al realizar el `S` un Descubrimiento de Ruta hacia el `D`, el resultado es el que se observa en la Figura 5.20. El `nodo D`, en condiciones ideales, recibirá un RREQ reenviado por el `A` y otro por el `C`. Independientemente de cuál de los dos le llegue primero, la ruta elegida finalmente será la que cuente con el menor número de saltos, en este caso a través del `nodo A`. Esto es así ya que los dos mensajes RREQ llevarán el mismo número de secuencia, por lo cual, pertenecerán al mismo Descubrimiento de Ruta. En caso de que tuvieran un número de secuencia distinto se daría prioridad a la ruta más actual, aunque tuviese un mayor número de saltos. En el caso simulado se extrae que el RREQ llegó primero a través del `nodo A`. Por eso, los `nodos B` y `C` únicamente tienen rutas hacia el `nodo S`, y no hacia el `D`, ya que estas se habrían formado al propagar un mensaje RREP.

2. **Redescubrimiento de ruta. Primera prueba**.
   Inicialmente los cinco nodos de la red están activos. El `nodo S` tiene cuatro alarmas pendientes de entregar. Por ello, realiza un Descubrimiento de Ruta que, como se explico en el punto anterior, al finalizar deja su tabla de encaminamiento como en la Figura 5.21a. Una vez dispone de una ruta válida comienza a enviar sus alarmas. Tras las dos primeras, el `nodo A` falla. El `S` agota los reintentos de envío de la alarma 3 y, al no tener respuesta, la descarta. Como todavía tiene una cuarta alarma que entregar, relanza un Descubrimiento de Ruta obteniendo

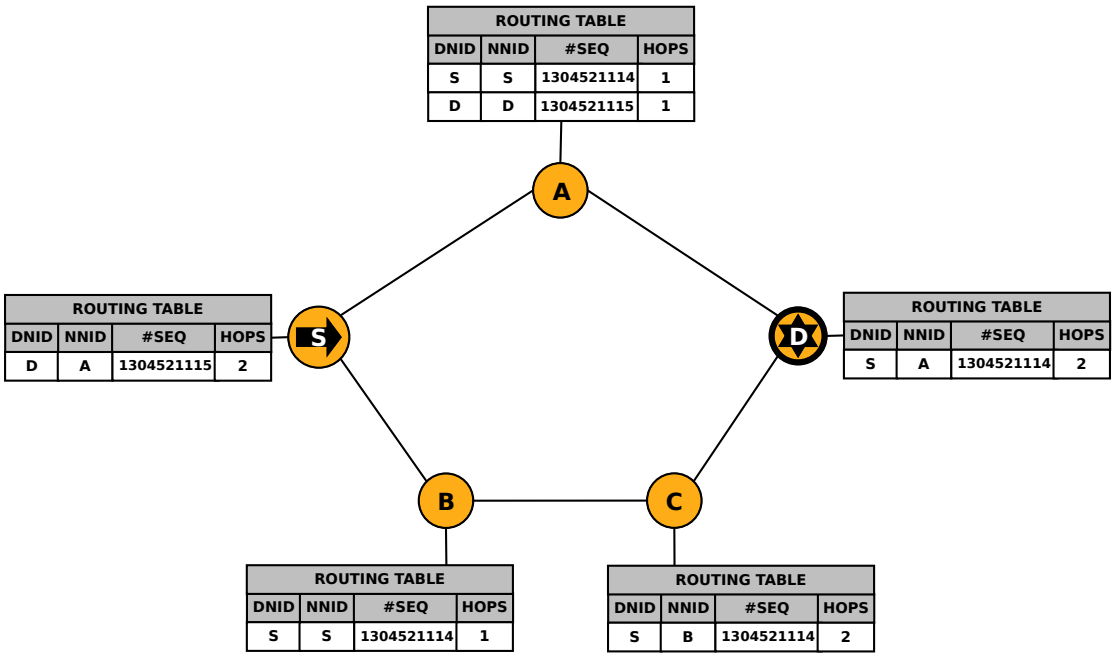


Figure 5.20: Descubrimiento de la ruta más corta entre los nodos S y D.

una nueva ruta a través de los `nodos B` y `C`, como se observa en la Figura 5.21b. A través de ella, logra enviar su última alarma.

ROUTING TABLE			
DNID	NNID	#SEQ	HOPS
D	A	1304523718	2

(a) Tabla de encaminamiento del nodo S tras el primer Descubrimiento de Ruta.

ROUTING TABLE			
DNID	NNID	#SEQ	HOPS
D	B	1304523719	3

(b) Tabla de encaminamiento del nodo S tras el segundo Descubrimiento de Ruta.

Figure 5.21: Evolución de las tablas de encaminamiento del nodo S.

Fichero `alarmReg.dat` generado por el `nodo D`:

alarm ID	timeStamp	alarmType	latitude	longitude	perConf
1	1304523038	S	41.2061	1.7300	87
2	1304523044	S	41.2061	1.7300	87
4	1304523052	S	41.2061	1.7300	87

Como se puede ver, la caída por un periodo prolongado de tiempo del `nodo A` supuso la pérdida de la tercera alarma.

3. **Redescubrimiento de ruta. Segunda prueba**.
Esta prueba es una continuación de la anterior. Inicialmente el `nodo S` tiene ocho alarmas que entregar. Después del primer Descubrimiento de Ruta (ver Figura 5.22a) se envían las dos primeras alarmas. El `nodo A` deja de operar en la red, y trás perderse la alarma 3, el `S` vuelve a buscar un camino para enviar sus alarmas (ver Figura 5.22b). En este momento, el `nodo A` vuelve a estar activo, pero mientras que la nueva ruta no caduque o de problemas no se cambiaría, aunque esté operativa una más corta. Tras enviar la alarma 4, el `nodo C` deja de operar. Con la quinta alarma ocurre lo mismo que con la tercera y a continuación se realiza un nuevo Descubrimiento de Ruta (ver Figura 5.22c). Por este último camino se logran entregar las últimas tres alarmas.

ROUTING TABLE			
DNID	**NNID**	**#SEQ**	**HOPS**
D	**A**	**1304527513**	**2**

(a) Tabla de encaminamiento del nodo S tras el primer Descubrimiento de Ruta.

ROUTING TABLE			
DNID	**NNID**	**#SEQ**	**HOPS**
D	**B**	**1304527514**	**3**

(b) Tabla de encaminamiento del nodo S tras el segundo Descubrimiento de Ruta.

ROUTING TABLE			
DNID	**NNID**	**#SEQ**	**HOPS**
D	**A**	**1304527515**	**2**

(c) Tabla de encaminamiento del nodo S tras el tercer Descubrimiento de Ruta.

Figure 5.22: Evolución de las tablas de encaminamiento del nodo S.

Fichero `alarmReg.dat` generado por el `nodo D`:

alarm ID	timeStamp	alarmType	latitude	longitude	perConf
1	1304524407	S	41.2061	1.7300	87
2	1304524411	S	41.2061	1.7300	87
4	1304524418	S	41.2061	1.7300	87
6	1304524428	S	41.2061	1.7300	87
7	1304524434	S	41.2061	1.7300	87
8	1304524441	S	41.2061	1.7300	87

Tras esta prueba y la anterior, se puede concluir que ante caídas prolongadas de nodos que forman parte de una ruta activa se perderá siempre una alarma.

4. **Envío de alarmas desde cuatro nodos. Influencia del parámetro `ALARM_TIMEOUT` en el tráfico**.
Cada nodo de la red tiene dos alarmas para entregar al nodo colector. Tras los Descubrimientos de Ruta, la tabla de encaminamiento del `nodo D` se muestra en la Figura 5.23.

ROUTING TABLE			
DNID	**NNID**	**#SEQ**	**HOPS**
C	**C**	**1304528418**	**1**
A	**A**	**1304528420**	**1**
S	**A**	**1304528417**	**2**
B	**C**	**1304528420**	**2**

Figure 5.23: Tabla de encaminamiento del nodo D.

En primer lugar, el parámetro de configuración `ALARM_TIMEOUT` se fija a 8 segundos y el fichero `alarmReg.dat` generado tras la simulación por el `nodo D` es:

alarm ID	timeStamp	alarmType	latitude	longitude	perConf
1	1304528134	C	41.2061	1.7300	87
1	1304528134	A	41.2061	1.7300	87
2	1304528137	C	41.2061	1.7300	87
2	1304528137	A	41.2061	1.7300	87
1	1304528134	S	41.2061	1.7300	87
1	1304528134	B	41.2061	1.7300	87
1	1304528134	S	41.2061	1.7300	87
2	1304528137	S	41.2061	1.7300	87
2	1304528137	B	41.2061	1.7300	87

En este caso, la primera alarma del `nodo S` se envía y recibe dos veces. Esto se debe a que la primera vez su ACK tarda en llegar más de `ALARM_TIMEOUT` por el tráfico de la red, y el nodo reintenta el envío. Esto solo ocurre con la primera ya que es el momento en el que más tráfico hay en la red.

A continuación, se repite la prueba pero esta vez con el parámetro `ALARM_TIMEOUT` a 12 segundos. El fichero `alarmReg.dat` generado por el `nodo D` es el siguiente:

alarm ID	timeStamp	alarmType	latitude	longitude	perConf
1	1304529073	C	41.2061	1.7300	87
1	1304529073	A	41.2061	1.7300	87
2	1304529080	C	41.2061	1.7300	87
2	1304529080	A	41.2061	1.7300	87
1	1304529073	S	41.2061	1.7300	87
1	1304529073	B	41.2061	1.7300	87
2	1304529080	S	41.2061	1.7300	87
2	1304529080	B	41.2061	1.7300	87

En este segundo caso, ninguna alarma llega repetida. Se observa que las alarmas de los `nodos A` y `C` llegan antes que las de `S` y `B`. Esto se debe a que los primeros únicamente tiene un salto de distancia con el nodo colector, mientras que los otros tienen dos. Por último, resaltar que el parámetro `ALARM_TIMEOUT` deberá ajustarse correctamente según la aplicación para optimizar la utilización de recursos de la red.

5. **Envío de alarmas desde tres nodos en línea**.
Realizando una pequeña modificación del fichero `bash.nodes` se configura la red de modo que el `nodo A` no puede comunicar con ningún otro y el resto se mantienen como antes. De este modo, se logra tener tres nodos generadores de alarmas en línea que envian a un nodo colector. Como se puede observar en la Figura 5.24, en la tabla de encaminamiento del `nodo D` se vé que la distancia a los `nodos C`, `B` y `S` es de uno, dos y tres saltos, respectivamente. Cada uno de estos nodos tiene dos alarmas para enviar.

ROUTING TABLE			
DNID	**NNID**	**#SEQ**	**HOPS**
C	**C**	**1304529688**	**1**
B	**C**	**1304529688**	**2**
S	**C**	**1304529688**	**3**

Figure 5.24: Tabla de encaminamiento del nodo D.

Igual que en el caso anterior, inicialmente se realiza la simulación con el parámetro `ALARM_TIMEOUT` a 8 segundos. El fichero `alarmReg.dat` generado por el `nodo D` es el siguiente:

alarm ID	timeStamp	alarmType	latitude	longitude	perConf
1	1304529073	C	41.2061	1.7300	87
2	1304529080	C	41.2061	1.7300	87
1	1304529073	B	41.2061	1.7300	87
1	1304529414	S	41.2061	1.7300	87
2	1304529080	B	41.2061	1.7300	87
1	1304529414	S	41.2061	1.7300	87
2	1304529420	S	41.2061	1.7300	87

Nuevamente, y por los mismos motivos, observamos que la primera alarma del `nodo S` se vuelve a entregar duplicada.

Al repetir la simulación modificando `ALARM_TIMEOUT` a 12 segundos, los resultados son:

alarm ID	timeStamp	alarmType	latitude	longitude	perConf
1	1304530117	C	41.2061	1.7300	87
2	1304530120	C	41.2061	1.7300	87
1	1304530117	B	41.2061	1.7300	87
1	1304530117	S	41.2061	1.7300	87
2	1304530120	B	41.2061	1.7300	87
2	1304530120	S	41.2061	1.7300	87

Se concluye que en redes con muchos nodos, donde puede haber varias alarmas que se entreguen a la vez, el parámetro `ALARM_TIMEOUT` deberá ser seleccionado a conciencia para evitar reenvíos innecesarios y aumentar de esa manera el tráfico de la misma.

5.4.5. Escenario 4

El cuarto, y último, escenario a simular consta de cinco nodos. Como se observa en la Figura 5.25, se representa una red dinámica en la que inicialmente el `nodo D` únicamente puede comunicarse con el `C`. A continuación, tras un desplazamiento, esta conexión se perderá y el `nodo D` establecerá una nueva con el `B`. Este hecho reducirá en un salto la distancia entre `S` y `D`. Para simular esta situación, se han implementado los ficheros `bash1.nodes` y `bash2.nodes`, correspondientes relativamente a las situaciones descritas.

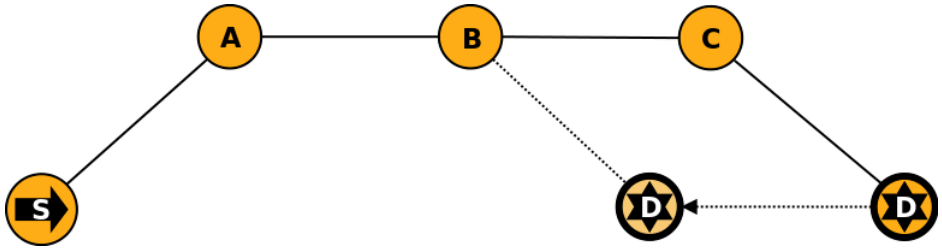


Figure 5.25: Escenario 4 de simulación: Red dinámica.

A continuación, se detallan las diferentes situaciones simuladas sobre este escenario y se comentan los resultados obtenidos.

1. **Límite de saltos de una ruta**.
   En la configuración de todos los nodos de la red se fija el número máximo de saltos en una ruta, `MAX_NUM_HOPS`, a tres. El `nodo S` tiene alarmas que enviar al `D` y para ello debe realizar un Descubrimiento de Ruta. Con la configuración inicial nunca encontrará dicho camino, ya que la distancia entre estos nodos es de cuatro saltos. En la simulación vemos lo siguiente:

```
1  fifo3_out: [0|Q|S|1304601056|D|S|1]      // RREQ1
2  // El nodo A lo recibe, ve que el campo 'hops' es menor
3  // que MAX_NUM_HOPS y lo reenvía
4  fifo2_out: [0|Q|S|1304601056|D|A|2]      // RREQ1
5  // El nodo B lo recibe, ve que el campo 'hops' es menor
6  // que MAX_NUM_HOPS y lo reenvía
7  fifo4_out: [0|Q|S|1304601056|D|B|3]      // RREQ1
8  // El nodo C lo recibe, ve que el campo 'hops' es igual
9  // que MAX_NUM_HOPS y no lo reenvía.
10 // Tras RREQ_TIMEOUT el nodo S repite la operación.
```

```
fifo3_out: [0|Q|S|1304601057|D|S|1]      // RREQ2
fifo2_out: [0|Q|S|1304601057|D|A|2]      // RREQ2
fifo4_out: [0|Q|S|1304601057|D|B|3]      // RREQ2
```

ROUTING TABLE			
DNID	**NNID**	**#SEQ**	**HOPS**
S	**B**	**1304601057**	**3**

Figure 5.26: Tabla de encaminamiento del nodo C.

Tras el mensaje RREQ del `nodo S`, los tres siguientes nodos crean una ruta hacia él, pero el tercero de ellos, en este caso el `nodo C`, compara el campo `hops` del mensaje RREQ con el parámetro `MAX_NUM_HOPS` de su configuración y, al ver que son iguales, ya no reenvía nuevamente el RREQ. Por ello, el `nodo D` nunca recibe la solicitud de ruta del `S` y la alarma o alarmas de éste no podrán ser entregadas. Como ya se comentó anteriormente, la utilización de una restricción de saltos para una ruta permite controlar la dispersión de los mensajes en la red.

2. **Dinamicidad de la red**.
   El `nodo D` que, anteriormente, no era alcanzable desde el `nodo S` se mueve y pasa a serlo.

```
// Se comienza con la primera configuración correspondiente
// a 'bash1.nodes'
$ ./bash_router.sh bash1.nodes
fifo3_out: [0|Q|S|1304601583|D|S|1]      // RREQ1 (NODO S)
fifo2_out: [0|Q|S|1304601583|D|A|2]      // RREQ1 (NODO A)
fifo4_out: [0|Q|S|1304601583|D|B|3]      // RREQ1 (NODO B)
fifo3_out: [0|Q|S|1304601584|D|S|1]      // RREQ2 (NODO S)
fifo2_out: [0|Q|S|1304601584|D|A|2]      // RREQ2 (NODO A)
fifo4_out: [0|Q|S|1304601584|D|B|3]      // RREQ2 (NODO B)
// Se cambia a la segunda configuración correspondiente
// a 'bash2.nodes'
$ ./bash_router.sh bash2.nodes
fifo3_out: [0|Q|S|1304601585|D|S|1]      // RREQ3 (NODO S)
fifo2_out: [0|Q|S|1304601585|D|A|2]      // RREQ3 (NODO A)
fifo4_out: [0|Q|S|1304601585|D|B|3]      // RREQ3 (NODO B)
// Cuando el nodo B reenvía el RREQ el nodo D lo recibe
// por fín, y responde con el correspondiente RREP
fifo1_out: [B|P|D|D|1304601586|1|S]      // RREP  (NODO D)
fifo4_out: [A|P|B|D|1304601586|2|S]      // RREP  (NODO B)
fifo2_out: [S|P|A|D|1304601586|3|S]      // RREP  (NODO A)
// El nodo S recibe el RREP del nodo A y manda su alarma
fifo3_out: [A|D|D|S|1|1304532488|S|41.2061|1.7300|87]   // ALARM1
// El nodo A reenvía la alarma 1 del nodo S
fifo2_out: [B|D|D|S|1|1304532488|S|41.2061|1.7300|87]   // ALARM1
// El nodo B reenvía la alarma 1 del nodo S
fifo4_out: [D|D|D|S|1|1304532488|S|41.2061|1.7300|87]   // ALARM1
// Cuando el nodo B reenvía la alarma 1 del nodo S, el
// nodo D la recibe y responde con el correspondiente
// acuse de recibo (ACK1)
fifo1_out: [B|A|S|1]    // ACK1 (NODO D)
```

```
31 fifo4_out: [A|A|S|1]     // ACK1 (NODO B)
32 fifo2_out: [S|A|S|1]     // ACK1 (NODO A)
33 // El nodo S recibe el ACK de su primera alarma y si tuviera
34 // más alarmas las seguiría enviando
```

ROUTING TABLE			
DNID	**NNID**	**#SEQ**	**HOPS**
S	**B**	**1304601585**	**3**

Figure 5.27: Tabla de encaminamiento del nodo D.

Cuando el `nodo D` cambia de posición recibe el RREQ del `nodo S` y responde con el correspondiente RREP, formando una ruta entre ambos. Una vez que el RREP llega a `S`, éste comienza a enviar sus alarmas que serán respondidas por `D` con acuses de recibo.

El protocolo de encaminamiento AODV-LAB soporta y detecta cambios en la topología de la red, de modo que, si existe algún camino posible entre dos nodos para comunicarse se descubrirá y podrá utilizarse.

3. **Envío de alarmas desde múltiples nodos**.
Utilizando la configuración definida por `bash2.nodes`, es decir, la que resulta tras el movimiento del `nodo D`, se envían dos alarmas desde cada nodo de la red. A continuación, se muestran la tabla de encaminamiento y el fichero `alarmReg.dat` generado por `D`.

alarm ID	timeStamp	alarmType	latitude	longitude	perConf
1	1304602164	B	41.2061	1.7300	87
2	1304602169	B	41.2061	1.7300	87
1	1304602164	A	41.2061	1.7300	87
1	1304602164	S	41.2061	1.7300	87
1	1304602164	C	41.2061	1.7300	87
2	1304602169	A	41.2061	1.7300	87
2	1304602169	C	41.2061	1.7300	87
2	1304602169	S	41.2061	1.7300	87

ROUTING TABLE			
DNID	**NNID**	**#SEQ**	**HOPS**
B	**B**	**1304602277**	**1**
A	**B**	**1304602278**	**2**
C	**B**	**1304602278**	**2**
S	**B**	**1304602277**	**3**

Figure 5.28: Tabla de encaminamiento del nodo D.

Al igual que sucedió en el escenario 3, se vuelve a observar que los nodos más próximos al colector entregan sus alarmas ántes que los más alejados.

5.4.6. Conclusiones

Como se comentaba al inicio de esta sección, el objetivo que se perseguía era constatar las principales funcionalidades deseadas, tanto del protocolo de encaminamiento AODV-LAB, como del módulo de entrega remota de datos del sistema LIDO DCL. Además de ello, la simulación también ha servido para verificar el comportamiento deseado en ciertas situaciones y, finalmente, lograr una mejor comprensión del sistema.

Sobre sus funcionalidades se puede afirmar que éste soporta:

- Descubrimiento de rutas hacia nodos vecinos.
- Descubrimiento de la ruta más corta entre dos nodos no vecinos.
- Redescubrimiento de una ruta alternativa entre dos nodos en caso de fallo prolongado de la actual. Este hecho, se vió que produce siempre la pérdida de una alarma en caso de que se estén enviando.
- Envío de alarmas y acuses de recibo entre dos nodos, o simultáneamente desde varios a un nodo colector. Capacidad de multisalto.
- Pérdidas temporales de la comunicación entre nodos sin extravío de datos. Esto es posible gracias a la utilización de acuses de recibo y a la posibilidad de reintentar el envío de alarmas tantas veces como se indique en `ALARM_RETRIES`.
- Envío de solicitudes y respuestas de posición sin necesidad de realizar un Descubrimiento de Ruta.
- Envío conjunto de alarmas y datos de posición entre varios nodos y un colector.
- Limitación del número máximo de saltos en una ruta para controlar la dispersión de los mensajes en la red.
- Dinamicidad de la red, es decir, detecta cambios en la topología y responde en consecuencia para mantener su operatividad. Caducidad de las entradas en la tabla de encaminamiento.
- Reseteo de los nodos de la red. Éste hecho puede producir que se entreguen alarmas repetidas pero, gracias a que el número de secuencia se inicializa con la función `time()`, al reiniciarse el nodo puede comenzar a operar sin ningún problema en la red.

Además, se ha verificado el comportamiento deseado frente a la siguientes situaciones:

- Agotamiento de los reintentos de envío de la última alarma del fichero `alarm.dat`. En tal caso, ésta se descarta y no se realiza ninguna operación más.
- Agotamiento de los reintentos de envío de una alarma del fichero `alarm.dat` que tiene, además de ésta, otras pendientes de entregar. En este caso, se descarta la que agotó los reintentos y, antes de continuar enviando nuevas alarmas, se realiza un Descubrimiento de Ruta nuevo, ya que el camino anterior no fué válido.
- Un nodo con alarmas pendientes de entregar inicia periódicamente un Descubrimiento de Ruta, hasta que este finalice con éxito y pueda entregar sus alarmas.
- Descarte de mensajes no dirigidos al propio nodo.
- Actualización de las tablas de encaminamiento.
- Influencia del parámetro `ALARM_TIMEOUT` en el tráfico de la red.

Por todo ello, se considera que el software implementado para este primer prototipo de red ad hoc de datos cumple los objetivos deseados y, por lo tanto, será válido para continuar con las pruebas. Se destaca el enfoque del mismo hacía un uso óptimo de los recursos, característica muy importante en un medio aislado como es el mar. No obstante, si se diera continuidad a este proyecto, mejorar el software sería un objetivo importante a alcanzar, trabajando sobre: los protocolos de encaminamiento, las funcionalidades del módulo de entrega remota de datos, la mejora del código, o todos ellos.

Chapter 6

Prueba final del prototipo de red ad hoc de datos

Lo importante es no dejar
de hacerse preguntas.
Albert Einstein

Resumen: Este capítulo describe una serie de pruebas realizadas con el prototipo final de red ad hoc de datos, concretamente utilizando 3 nodos para constituir la red. Los resultados observados muestran algunas situaciones nuevas respecto a las simulaciones. Se confirma el funcionamiento de la red y algunas de las funcionalidades deseadas, pero es necesario profundizar en la realización de este tipo de pruebas, a ser posible en el mar, para: lograr una configuración de los nodos tanto a nivel de hardware como de software más óptima, probar funcionalidades que requieren un mayor número de nodos y detectar posibles fallos del código y corregirlos.

6.1. Introducción

Una vez finalizado el desarrollo de las distintas partes del prototipo por separado, se ha procedido a la integración de las mismas según el diseño realizado inicialmente. Este capítulo documenta una serie de pruebas realizadas con tres nodos que emplean el software desarrollado y el hardware seleccionado y probado anteriormente en este proyecto.

Los nodos se han colocado de manera alineada en tierra firme formando un enlace bidireccional en el que, como se muestra en la Figura 6.1, los `nodos S` y `D` requieren del `A` para poderse comunicar entre sí. Los nodos se han formado de la siguiente manera:

- `Nodo D`. Actúa como colector o destino de todas las alarmas. Consiste en un módulo de comunicación (XBee-PRO 868 + Antena + Conversor UART-USB) conectado por el puerto USB a un ordenador de mesa con la distribución Ubuntu 11.04 de GNU/Linux.
- `Nodos A y S`. Actúan como nodos generadores de alarmas, que sería el equivalente a una boya operando en el mar. Cada uno consiste en un ordenador portatil con Ubuntu 10.04 y un módulo de comunicación conectado por el puerto USB.

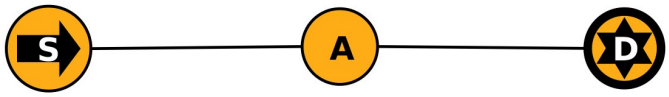


Figure 6.1: Escenario de pruebas: Enlace bidireccional entre tres nodos alineados.

El firmware de todos los XBee-PRO 868 se ha mantenido configurado sin reenvíos a nivel de MAC (RR=0 y MT=0). La potencia utilizada ha sido la mínima (PL=0) ya que las pruebas se han realizado con los nodos relativamente próximos entre sí. Los parámetros de configuración del software de todos los nodos han sido:

- `SERIAL_NAME = ´/dev/ttyUSB0´`
- `SERIAL_BAUD_RATE = 9600`
- `NODE_ID = ´S´, ´A´ o ´D´`. Cada uno el suyo.
- `SINK_NODE_ID = ´D´`
- `MAX_NUM_HOPS = 3`
- `LIFETIME_RTENTRY = 300`. Valor en segundos.
- `ALARM_RETRIES = 2`
- `ALARM_TIMEOUT = 12`. Valor en segundos.
- `RREQ_TIMEOUT = 12`. Valor en segundos.

6.2. Primera prueba

Ha consistido en el envío de 4 alarmas desde en `nodo S` al `D`.

Durante su operación cada nodo va imprimiendo mensajes por pantalla indicando los datos que recibe y los que envía. De esta manera es posible, al finalizar cada prueba, observar cómo se ha desarrollado todo. Se han registrado los mensajes del `nodo D` y a continuación se presentan comentados, permitiendo así seguir y entender lo sucedido en el proceso de entrega de las 4 alarmas.

```
1  // Se recibe un RREQ del nodo S a través del A. Se puede observar que
       el número de saltos es 2.
2  Serial data received (24 bytes): [0|Q|S|1310577406|D|A|2]
3  RREQ message received!
4
5  // Se guarda la ruta hacia el nodo S y se responde al RREQ con un
       RREP unicast.
6  Unicast RREP message (24 bytes): [A|P|D|D|1310577407|1|S]
7  // Se imprime por pantalla la tabla de rutas y de registro de RREQs.
8  ************************************************
9  ***                ROUTING TABLE               ***
10 *-----------------------------------------------*
11 *    DNID     NNID         SEQN     HOPS         *
12 *-----------------------------------------------*
13 *     S       A      1310577406      2           *
```

```
14 ***********************************************
15 ***************************************
16 ***             RREQ TABLE            ***
17 *-------------------------------------*
18 *    SNID         SeqNum                *
19 *-------------------------------------*
20 *     S           1310577406            *
21 ***************************************
22
23 // Se recibe el RREP que el nodo A envía al S, y como es unicast se
      descarta.
24 Serial data received (24 bytes): [S|P|A|D|1310577407|2|S]
25 This message is not for me.
26
27 // Llega la primera alarma. El nodo S la mandó al A, y éste al D.
28 Serial data received (42 bytes):
29     [D|D|D|S|1|1310564064|S|41.2061|1.7300|87]
30 DATA message received!
31
32 // Se imprime la información de la alarma recibida.
33 Alarm received: alID: 1         timeStmp: 1310564064    alType: S
34                 Lat: 41.2061    Lon: 1.7300             PerConf: 87
35
36 // Se envía el acuse de recibo (ACK) de la primera alarma.
37 Unicast ACK message (9 bytes): [A|A|S|1]
38
39 // Se vuelve a recibir la primera alarma. Como no llegó el ACK
      reenviado por el nodo A, se sabe que ese mensaje falló entre el D
      y el A y, por eso, tras ALARM_TIMEOUT el nodo S vuelve a enviar su
       primera alarma.
40 Serial data received (42 bytes):
41     [D|D|D|S|1|1310564064|S|41.2061|1.7300|87]
42 DATA message received!
43 Alarm received: alID: 1         timeStmp: 1310564064    alType: S
44                 Lat: 41.2061    Lon: 1.7300             PerConf: 87
45
46 // Se envía el ACK de la primera alarma.
47 Unicast ACK message (9 bytes): [A|A|S|1]
48
49 // Llega el ACK reenviado por el nodo A. Como va dirigido a S, el
      nodo D lo descarta.
50 Serial data received (9 bytes): [S|A|S|1]
51 This message is not for me.
52
53 // Llega la alarma 2 que el nodo S envía al A. Esto significa que por
       algún motivo los nodos S y D se encuentra uno dentro del área de
      cobertura del otro pero, como el mensaje es unicast, el nodo D lo
      descarta.
54 Serial data received (42 bytes):
55     [A|D|D|S|2|1310564070|S|41.2061|1.7300|87]
56 This message is not for me.
57
58 // Ahora se recibe la alarma 2 reenviada por el nodo A. Como D es el
      destinatario se registra la alarma y se imprime por pantalla.
59 Serial data received (42 bytes):
60     [D|D|D|S|2|1310564070|S|41.2061|1.7300|87]
```

```
61 DATA message received!
62 Alarm received: alID: 2          timeStmp: 1310564070    alType: S
63                  Lat: 41.2061    Lon: 1.7300             PerConf: 87
64
65 //Se envía el ACK de la segunda alarma.
66 Unicast ACK message (9 bytes): [A|A|S|2]
67
68 // Llega el ACK reenviado por el nodo A. Como va dirigido a S, el
      nodo D lo descarta.
69 Serial data received (9 bytes): [S|A|S|2]
70 This message is not for me.
71
72 // Se recibe la alarma 3.
73 Serial data received (42 bytes):
74     [D|D|D|S|3|1310564077|S|41.2061|1.7300|87]
75 DATA message received!
76 Alarm received: alID: 3          timeStmp: 1310564077    alType: S
77                  Lat: 41.2061    Lon: 1.7300             PerConf: 87
78
79 // Envio de ACKs por los nodos D y A.
80 Unicast ACK message (9 bytes): [A|A|S|3]
81 Serial data received (9 bytes): [S|A|S|3]
82 This message is not for me.
83
84 // Se recibe la alarma 4.
85 Serial data received (42 bytes):
86     [D|D|D|S|4|1310564082|S|41.2061|1.7300|87]
87 DATA message received!
88 Alarm received: alID: 4          timeStmp: 1310564082    alType: S
89                  Lat: 41.2061    Lon: 1.7300             PerConf: 87
90
91 // Envío de ACKs por los nodos D y A.
92 Unicast ACK message (9 bytes): [A|A|S|4]
93 Serial data received (9 bytes): [S|A|S|4]
94 This message is not for me.
```

Fichero `alarmReg.dat` generado por el `nodo D`:

alarm ID	timeStamp	alarmType	latitude	longitude	perConf
1	1310564064	S	41.2061	1.7300	87
1	1310564064	S	41.2061	1.7300	87
2	1310564070	S	41.2061	1.7300	87
3	1310564077	S	41.2061	1.7300	87
4	1310564082	S	41.2061	1.7300	87

Todas las alarmas han llegado correctamente. Al igual que se comprobó en la simulación (ver Subsección 5.4.3), los nodos de la red son capaces de descubrir rutas hacia otros no vecinos y, de igual manera, hacerles llegar alarmas y acuses de recibo. Por lo tanto, la red soporta el multisalto. De esta prueba, únicamente, se destacan dos situaciones:

- Se ha registrado una pérdida temporal de la comunicación entre los `nodos D` y `A`, probablemente debida a la variabilidad del canal radio. Este efecto ha provocado la pérdida de un acuse de recibo. Esta situación ya se contempló en el desarrollo del software y, como se ha visto, el sistema la soporta sin mayor problema. Al no

recibir el `nodo S` su acuse de recibo tras `ALARM_TIMEOUT`, como en la configuración `ALARM_RETRIES` es dos, se reintenta el envío de la alarma, en este caso la 1. La segunda vez todo va correctamente. Por este motivo dicha alarma aparece repetida en el fichero `alarmReg.dat` generado por el `nodo D`.

- En cierto momento, el `nodo D` ha recibido un mensaje del `S`, que inicialmente se intentó colocar lo suficientemente separado para que esto no ocurriera. Al ser un mensaje unicast dirigido al `nodo A` el `D` lo ha descartado sin más. De este caso se destaca que el área de cobertura varía y, por lo tanto, dos nodos situados a una distancía próxima al límite de éste pueden tener una comunicación intermitente. Para garantizar un comportamiento más estable habrá que dejar cierto margen respecto al límite.

6.3. Segunda prueba

Ha consistido en el envío de 4 alarmas desde el `nodo S` y otras 4 desde el `A`. Además, durante el proceso de envío de las alarmas, el `nodo D` ha realizado una solicitud de posición mediante el envío en broadcast de un mensaje PREQ.

Nuevamente se han registrado los mensajes del `nodo D` para posteriormente analizar lo sucedido durante la prueba.

```
1  // Lee del puerto serie un RREQ del nodo A y otro del S y los pone en
       la cola de mensajes.
2  Serial data received (48 bytes): [0|Q|A|1310570952|D|A|1]
3                                   [0|Q|S|1310578153|D|A|2]
4
5  // Coge de la cola de mensajes el RREQ del nodo A, guarda la ruta y
      responde con un RREP unicast.
6  RREQ message received!
7  Unicast RREP message (24 bytes): [A|P|D|D|1310570953|1|A]
8  ************************************************
9  ***                ROUTING TABLE               ***
10 *-----------------------------------------------*
11 *    DNID    NNID        SEQN    HOPS           *
12 *-----------------------------------------------*
13 *     A      A     1310570952     1              *
14 ************************************************
15 ****************************************
16 ***             RREQ TABLE            ***
17 *--------------------------------------*
18 *    SNID        SeqNum                *
19 *--------------------------------------*
20 *     A          1310570952            *
21 ****************************************
22
23 // Coge de la cola de mensajes el RREQ del nodo S, guarda la ruta y
      responde con un RREP unicast.
24 RREQ message received!
25 Unicast RREP message (24 bytes): [A|P|D|D|1310578154|1|S]
26 ************************************************
27 ***                ROUTING TABLE               ***
28 *-----------------------------------------------*
29 *    DNID    NNID        SEQN    HOPS           *
```

```
*-----------------------------------------------*
*     A     A     1310570952     1              *
*     S     A     1310578153     2              *
*************************************************
*****************************************
***             RREQ TABLE            ***
*---------------------------------------*
*    SNID        SeqNum                 *
*---------------------------------------*
*     A          1310570952             *
*     S          1310578153             *
*****************************************

// Lee del puerto serie la primera alarma de los nodos A y S y las
   pone en la cola de mensajes.
Serial data received (84 bytes):
    [D|D|D|A|1|1310564064|A|41.2061|1.7300|87]
    [D|D|D|S|1|1310564064|S|41.2061|1.7300|87]

// Coge la primera alarma del nodo A de la cola de mensajes, la
   registra y manda su correspondiente ACK.
DATA message received!
Alarm received: alID: 1         timeStmp: 1310564064    alType: A
                Lat: 41.2061    Lon: 1.7300             PerConf: 87
Unicast ACK message (9 bytes): [A|A|A|1]

// Coge la primera alarma del nodo S de la cola de mensajes, la
   registra y manda su correspondiente ACK.
DATA message received!
Alarm received: alID: 1         timeStmp: 1310564064    alType: S
                Lat: 41.2061    Lon: 1.7300             PerConf: 87
Unicast ACK message (9 bytes): [A|A|S|1]

// El nodo D manda un mensaje de solicitud de posición, PREQ.
PREQ message sent (22 bytes): [0|W|D|1310578155|D|1]

// Lee del puerto serie la segunda alarma del nodo A y el ACK de la
   primera alarma del S reenviado por el nodo A.
Serial data received (51 bytes):
    [D|D|D|A|2|1310564070|A|41.2061|1.7300|87]
    [S|A|S|1]

// Coge la segunda alarma del nodo A de la cola de mensajes, la
   registra y manda su correspondiente ACK.
DATA message received!
Alarm received: alID: 2         timeStmp: 1310564070    alType: A
                Lat: 41.2061    Lon: 1.7300             PerConf: 87
Unicast ACK message (9 bytes): [A|A|A|2]

// Coge el ACK de la primera alarma del nodo S reenviado por el A y
   lo descarta.
This message is not for me.

// Lee del puerto serie la segunda alarma del nodo S, el PREP del A y
    el reenvío del PREQ hecho también por el nodo A.
Serial data received (103 bytes):
```

```
79      [D|D|D|S|2|1310564070|S|41.2061|1.7300|87]
80      [D|C|D|1310570975|A|+90.1234|-127.4321]
81      [0|W|D|1310578155|A|2]
82
83  // Coge la segunda alarma del nodo S de la cola de mensajes, la
       registra y manda su correspondiente ACK.
84  DATA message received!
85  Alarm received: alID: 2         timeStmp: 1310564070    alType: S
86                  Lat: 41.2061    Lon: 1.7300             PerConf: 87
87  Unicast ACK message (9 bytes): [A|A|S|2]
88
89  // Coge el PREP del nodo A de la cola de mensajes y lo registra.
90  PREP message received!
91  PREP message received:  TS: 1310570975    NID: A
92                          LAT: +90.1234     LON: -127.4321
93
94  // Coge el reenvío del PREQ hecho por el nodo A, y no hace nada ya
       que él es el nodo que hizo el PREQ.
95  PREQ message received!
96
97  // Lee del puerto serie la tercera alarma del nodo A y el ACK de la
       segunda alarma del S reenviado por el nodo A.
98  Serial data received (51 bytes):
99      [D|D|D|A|3|1310564077|A|41.2061|1.7300|87]
100     [S|A|S|2]
101
102 // Coge la tercera alarma del nodo A de la cola de mensajes, la
       registra y manda su correspondiente ACK.
103 DATA message received!
104 Alarm received: alID: 3         timeStmp: 1310564077    alType: A
105                 Lat: 41.2061    Lon: 1.7300             PerConf: 87
106 Unicast ACK message (9 bytes): [A|A|A|3]
107 // Coge el ACK de la segunda alarma del nodo S reenviado por el A y
       lo descarta.
108 This message is not for me.
109
110 // Lee del puerto serie la tercera alarma del nodo S y la cuarta
       alarma del A
111 Serial data received (84 bytes):
112     [D|D|D|S|3|1310564077|S|41.2061|1.7300|87]
113     [D|D|D|A|4|1310564082|A|41.2061|1.7300|87]
114
115 // Coge la tercera alarma del nodo S de la cola de mensajes, la
       registra y manda su correspondiente ACK.
116 DATA message received!
117 Alarm received: alID: 3         timeStmp: 1310564077    alType: S
118                 Lat: 41.2061    Lon: 1.7300             PerConf: 87
119 Unicast ACK message (9 bytes): [A|A|S|3]
120
121 // Coge la cuarta alarma del nodo A de la cola de mensajes, la
       registra y manda su correspondiente ACK.
122 DATA message received!
123 Alarm received: alID: 4         timeStmp: 1310564082    alType: A
124                 Lat: 41.2061    Lon: 1.7300             PerConf: 87
125 Unicast ACK message (9 bytes): [A|A|A|4]
126
```

```
127 // Lee del puerto serie el ACK de la tercera alarma del nodo S
       reenviado por el A y lo descarta.
128 Serial data received (9 bytes): [S|A|S|3]
129 This message is not for me.
130
131 // Lee del puerto serie la cuarta alarma del nodo S, la registra y
       manda su correspondiente ACK.
132 Serial data received (42 bytes):
133     [D|D|D|S|4|1310564082|S|41.2061|1.7300|87]
134 DATA message received!
135 Alarm received: alID: 4          timeStmp: 1310564082    alType: S
136                  Lat: 41.2061    Lon: 1.7300             PerConf: 87
137 Unicast ACK message (9 bytes): [A|A|S|4]
138
139 // Lee del puerto serie el ACK de la cuarta alarma del nodo S
       reenviado por el A y lo descarta.
140 Serial data received (9 bytes): [S|A|S|4]
141 This message is not for me.
142
143 // Vuelve a recibir la cuarta alarma del nodo S, la registra de nuevo
        y vuelve a mandar su correspondiente ACK. Recibir esta alarma de
       nuevo significa que el ACK que reenvió el nodo A no llego al S, y
       éste tras esperar ALARM_TIMEOUT volvió a mandar la alarma 4.
144 Serial data received (42 bytes):
145     [D|D|D|S|4|1310564082|S|41.2061|1.7300|87]
146 DATA message received!
147 Alarm received: alID: 4          timeStmp: 1310564082    alType: S
148                  Lat: 41.2061    Lon: 1.7300             PerConf: 87
149 Unicast ACK message (9 bytes): [A|A|S|4]
150
151 // Recibe el ACK de la cuarta alarma del nodo S reenviado por el A y
       lo descarta.
152 Serial data received (9 bytes): [S|A|S|4]
153 This message is not for me.
154
155 // Tras un periodo de tiempo sin recibir nada, puede suceder que el
       nodo A se haya caído y el S y D no puedan comunicarse, o que el
       los nodos hayan terminado de enviar sus alarmas y estén a la
       espera de que aparezcan nuevas.
```

Fichero `alarmReg.dat` generado por el `nodo D`:

alarm ID	timeStamp	alarmType	latitude	longitude	perConf
1	1310564064	A	41.2061	1.7300	87
1	1310564064	S	41.2061	1.7300	87
2	1310564070	A	41.2061	1.7300	87
2	1310564070	S	41.2061	1.7300	87
3	1310564077	A	41.2061	1.7300	87
3	1310564077	S	41.2061	1.7300	87
4	1310564082	A	41.2061	1.7300	87
4	1310564082	S	41.2061	1.7300	87
4	1310564082	S	41.2061	1.7300	87

Fichero `positionReg.dat` generado por el `nodo D`:

timeStamp	SNID	latitude	longitude
1310570975	A	+90.1234	-127.4321

En este caso, también han llegado todas las alarmas correctamente al nodo colector, sin embargo, la información de posición del `nodo S` no lo ha hecho. Se destacan los siguientes sucesos:

- En todos los casos se reciben primero las alarmas del `nodo A`, que es el más cercano al colector. En caso de existir más nodos, o de haber rutas con más saltos, este efecto puede verse aumentado, pudiendo influir en el correcto funcionamiento de la herramienta. De todos modos, como ya se ha reseñado varias veces en la memoria, el sistema de alarmas de eventos acústicos avisará de la presencia de la fuente sonora en un área y, por lo tanto, el número de alarmas y la frecuencia de aparición de las mismas no se espera que sea tan elevado.
- El `nodo D` no ha recibido el PREP del `S`. Mirando con detalle el flujo de mensajes, en la línea 81, vemos que el `nodo A` reenvía correctamente el PREQ, por consiguiente, el mensaje se ha perdido entre este nodo y el `S`. Nuevamente se vuelve a detectar una caída temporal de la comunicación. En este caso, para obtener la información de localización del `nodo S`, el colector debería volver a enviar el PREQ con el gasto de recursos que ello supone.
- La cuarta alarma del `nodo S` llega repetida debido a la pérdida de un ACK, concretamente entre el `S` y el `A` (línea 140).

6.4. Tercera prueba

Como se comentó en la Subsección **??**, Yapicioglu y Oktug (2009) proponían en las conclusiones de su estudio sobre la influencia de la altura de las olas en las redes de sensores en el mar la utilización de boyas únicamente enrutadoras o repetidoras, sin sistema sensor. En esta prueba se ha desconectado el ordenador portatil del `nodo A` y se ha configurado éste en modo *loopback*, tal y como se hizo para el estudio de alcance. De esta manera, el nodo únicamente actúa como repetidor de todos los mensajes que reciba.

En la prueba el `nodo S` envía 4 alarmas al colector. A continuación, se muestran los mensajes que este último ha generado en el proceso.

```
1  // El nodo D recibe un RREQ del S, pero el contador de saltos se
       encuentra a 1. Esto se debe a que este mensaje a sido repetido por
        el nodo A, sin haberle incrementado el contador de saltos. El
       nodo D guarda la ruta, y responde con un RREP.
2  Serial data received (24 bytes): [0|Q|S|1310578564|D|S|1]
3  RREQ message received!
4  Unicast RREP message (24 bytes): [S|P|D|D|1310578565|1|S]
5  ************************************************
6  ***                ROUTING TABLE               ***
7  *-----------------------------------------------*
8  *    DNID    NNID        SEQN    HOPS           *
9  *-----------------------------------------------*
10 *     S     S     1310578564     1              *
11 ************************************************
12 ***              RREQ TABLE             ***
```

```
*----------------------------------------*
*    SNID        SeqNum                  *
*----------------------------------------*
*     S          1310578564              *
******************************************
// El nodo D recibe su propio RREP cuando éste es repetido por el
   nodo A, pero lo descarta ya que este tipo de mensaje es unicast y
   su DNID es el nodo S.
Serial data received (24 bytes): [S|P|D|D|1310578565|1|S]
This message is not for me.

// Recibe la primera alarma del nodo S, la registra y responde con el
    correspondiente ACK.
Serial data received (42 bytes):
    [D|D|D|S|1|1310564064|S|41.2061|1.7300|87]
DATA message received!
Alarm received: alID: 1         timeStmp: 1310564064    alType: S
                Lat: 41.2061    Lon: 1.7300             PerConf: 87
Unicast ACK message (9 bytes): [S|A|S|1]

// Repetición del ACK por el nodo A descartada.
Serial data received (9 bytes): [S|A|S|1]
This message is not for me.

// Recibe la segunda alarma del nodo S, la registra y responde con el
    correspondiente ACK.
Serial data received (42 bytes):
    [D|D|D|S|2|1310564070|S|41.2061|1.7300|87]
DATA message received!
Alarm received: alID: 2         timeStmp: 1310564070    alType: S
                Lat: 41.2061    Lon: 1.7300             PerConf: 87
Unicast ACK message (9 bytes): [S|A|S|2]

// En este caso el nodo A no ha repetido el ACK. Esto significa que
   no lo ha recibido por una pérdida temporal de la comunicación.
   Tras ALARM_TIMEOUT el nodo S vuelve a enviar su segunda alarma.
   Ésta es repetida por el nodo A y, finalmente, la recibe el D. La
   registra nuevamente y responde con el correspondiente ACK.
Serial data received (42 bytes):
    [D|D|D|S|2|1310564070|S|41.2061|1.7300|87]
DATA message received!
Alarm received: alID: 2         timeStmp: 1310564070    alType: S
                Lat: 41.2061    Lon: 1.7300             PerConf: 87
Unicast ACK message (9 bytes): [S|A|S|2]

// Repetición del ACK por el nodo A descartada.
Serial data received (9 bytes): [S|A|S|2]
This message is not for me.

// Recibe la tercera alarma del nodo S, la registra y responde con el
    correspondiente ACK.
Serial data received (42 bytes):
    [D|D|D|S|3|1310564077|S|41.2061|1.7300|87]
DATA message received!
Alarm received: alID: 3         timeStmp: 1310564077    alType: S
                Lat: 41.2061    Lon: 1.7300             PerConf: 87
```

```
60 Unicast ACK message (9 bytes): [S|A|S|3]
61
62 // Repetición del ACK por el nodo A descartada.
63 Serial data received (9 bytes): [S|A|S|3]
64 This message is not for me.
65 // Recibe la cuarta alarma del nodo S, la registra y responde con el
       correspondiente ACK.
66 Serial data received (42 bytes): [D|D|D|S|4|1310564082|S
       |41.2061|1.7300|87]
67 DATA message received!
68 Alarm received: alID: 4          timeStmp: 1310564082     alType: S
69                  Lat: 41.2061    Lon: 1.7300              PerConf: 87
70 Unicast ACK message (9 bytes): [S|A|S|4]
71
72 // Repetición del ACK por el nodo A descartada.
73 Serial data received (9 bytes): [S|A|S|4]
74 This message is not for me.
```

Fichero `alarmReg.dat` generado por el `nodo D`:

alarm ID	timeStamp	alarmType	latitude	longitude	perConf
1	1310564064	S	41.2061	1.7300	87
2	1310564070	S	41.2061	1.7300	87
2	1310564070	S	41.2061	1.7300	87
3	1310564077	S	41.2061	1.7300	87
4	1310564082	S	41.2061	1.7300	87

El `nodo D` ha recibido todas las alarmas de `S` correctamente. Se ha vuelto a detectar una pérdida temporal de la comunicación, teniendo que reenviarse una alarma tras el `ALARM_TIMEOUT` correspondiente.

Utilizar boyas repetidoras puede ser una buena solución para aumentar la distancia entre boyas sensoras, en el caso de que eso interese. Estas boyas requerirán muy poca electrónica; se eliminará el subsistema sensor y el módulo de procesado de audio del computacional, así podrá utilizarse un subsistema de energía reducido.

6.5. Discusión de resultados

El objetivo de la realización de esta prueba final ha sido verificar el correcto funcionamiento de todas las partes de este proyecto integradas: módulo de comunicación del subsistema radio y módulo de entrega remota de datos y protocolo de encaminamiento AODV-LAB del subsistema computacional. El resultado, como se ha ido comentado en cada una de las pruebas, ha sido positivo. Únicamente se destaca la pérdida de algún paquete, suceso común en los sistemas de comunicación inalámbrica. De este modo, queda implementada una herramienta o prototipo a partir del cual se puede continuar experimentando y trabajando para ir logrando cada vez una solución mejor.

En base a los resultados obtenidos se pueden extraer una serie de conlusiones más concretas:

- El proceso de Descubrimiento de Ruta con 3 nodos funciona correctamente.
- La red soporta multisalto y realiza adecuadamente la entrega de alarmas.

- Las pérdidas temporales de comunicación entre nodos debido a la variabilidad del canal radio están presentes en todos los casos. En la mayoría de las pruebas realizadas la capacidad de reenvío de alarmas, configurable a nivel de software, permite solventar estos problemas sin un consumo elevado de recursos. Exísten algunas excepciones, como es el caso de la pérdida de mensajes de posición, en las que, si se produce un fallo, la nueva solicitud debe hacerse manualmente por el usuario. En este caso, reenviar un PREQ implica que nuevamente todos los nodos que lo reciban envíen su PREP, siendo redundante para todos aquellos que la primera vez lo hicieron correctamente.
- El área de cobertura de un nodo es variable. Por ello, a la hora de colocar los nodos habrá que dejar un determinado margen respecto al límite medido.
- La operación con nodos repetidores en la red es viable y permite aumentar la distancia entre boyas de forma económica.

Para pruebas futuras se considera apropiado:

- Utilizar una población de nodos algo mayor. Permitirá realizar pruebas de descubrimiento de rutas óptimas, y analizar en profundidad el efecto del número de saltos (`MAX_NUM_HOPS`) entre un nodo fuente y el colector.
- Estudiar la opción del firmware de utilizar repeticiones a nivel de MAC. Esta caraterística puede disminuir el efecto de la pérdida temporal de comunicación entre nodos, ya sea: reduciendo la carga computacional que implica, el número de paquetes redundantes que se introducen a la red, o el tiempo empleado en solventar la pérdida de mensajes. Concrétamente, las repeticiones a nivel de MAC parecen más apropiadas para pérdidas breves de la comunicación y los reintentos a nivel de red para periodos más largos sin conexión. Debe prestarse atención a la restricción del ciclo de trabajo que existe en la banda, ya que estas repeticiones estarán introduciendo más tráfico al medio.
- Realizar pruebas en el mar. Podrán detectarse efectos nuevos a los registrados en tierra, o de mayor magnitud. También permitirá cuantificar la variabilidad del área de cobertura de los nodos.
- Utilizar un tráfico de datos real o acorde con el que generará el sistema de alarmas.
- Estudiar en detalle las opciones de configuración de los nodos y ver su efecto en la red para lograr un funcionamiento más óptimo.

Chapter 7

Conclusiones y trabajo futuro

Parecía que habíamos llegado al final del camino y
resulta que era sólo una curva abierta a otro
paisaje y a nuevas curiosidades.

El año de la muerte de Ricardo Reis
José Saramago

7.1. Conclusiones

Este proyecto ha tenido como meta el dar una respuesta a la necesidad manifiesta de aportar soluciones viables y económicamente sostenibles para la utilización de redes inalámbricas de sensores en el medio marino, orientadas a la monitorización de cetáceos, fuentes acústicas, u otras aplicaciones.

Se ha tratado de abarcar los diferentes campos implicados en este ámbito de actuación con el objetivo de formar una base sólida y completa sobre la que continuar trabajando.

Atendiendo a los objetivos definidos en el primer capítulo de la presente memoria se extraen las siguientes conclusiones:

1. La implementación de redes ad hoc con capacidad sensora en el medio marino es viable y sostenible económicamente utilizando tecnologías estandarizadas, o con elevada presencia en el mercado.

   En función de la aplicación, o servicio que deba soportar la red, se optará por una u otra tecnología teniendo en cuenta que:

   - Existe un compromiso entre velocidad de envío de datos y alcance de la señal.
   - Cuanto menor sea el consumo energético del sistema de comunicación menor será el coste final de cada nodo.

   En este proyecto se optó por el desarrollo de una red capaz de dar soporte a un servicio de alarmas, lo que supone un flujo bajo de datos y permite maximizar la distancia entre nodos.

2. Se ha decidido operar en la banda ICM de 869.40 a 869.65 MHz, donde: está permitido emplear una PIRE considerablemente superior a otras bandas, existen módulos económicos y de bajo consumo, y el compromiso entre velocidad de envío

de datos y alcance permite la implementación de una red ad hoc y distancias entre nodos de algunos kilómetros. Concretamente se ha optado por los módulos XBee-PRO 868.

3. La solución propuesta responde a una arquitectura modular, lo que ha facilitado su integración con el resto del sistema LIDO DCL ya desarrollado.

4. Se ha realizado un estudio teórico de un enlace radioeléctrico en entorno marino empleando el modelo de tierra plana. El resultado obtenido ha sido que el alcance teórico para los módulos XBee-PRO 868 es de 5.6 kilómetros. Finalmente, se realizaron medidas reales sobre la superficie del mar logrando buenas prestaciones a distancias de hasta aproximadamente 2 kilómetros, suficiente para el servicio de alarmas.

   La desviación entre el alcance teórico y el empírico se debe a que el modelo de tierra plana supone el mar como una superficie lisa y no contempla otros factores que están presentes en el caso real, como: pérdidas de potencia en los equipos, variación de la altura relativa de las antenas, difracción por barcos u el propio oleaje, atenuación por gases o partículas en suspensión, etc.

5. Se ha desarrollo en C/Linux un módulo de entrega remota de datos, encargado de la gestión de las alarmas en el nodo local y de su correcto envío al colector. Este módulo emplea el protocolo de encaminamiento AODV-LAB, también implementado en el proyecto.

6. Se ha simulado la red recreando varios escenarios y situaciones distintas para verificar el correcto funcionamiento del software desarrollado. Los resultados obtenidos confirmaron las prestaciones deseadas.

Todo esto integrado ha dado lugar a un prototipo de red ad hoc de datos que servirá como herramienta para continuar experimentando y trabajando en esta línea. Por ello, se considera cumplido el objetivo general planteado para este proyecto.

7.2. Trabajo futuro

A lo largo de esta memoria se han apuntado las siguientes líneas de actuación orientadas a la mejora y continuación del proyecto:

1. **Realización de más medidas de alcance sobre la superficie el mar**. Emplear los módulos XBee-PRO 868 situados en boyas o, al menos, en una embarcación. Tratar de aproximarse lo máximo posible a la situación final del sistema. Tomar datos durante un trayecto pudiendo registrar más claramente el efecto del aumento de la distancia en el enlace. Repetir las medidas en tierra firme para determinar la magnitud de efectos como: la reflexión en el agua salada, la variación de la altura relativa de las antenas, la atenuación por particulas en suspensión, etc.

2. **Evaluación de otras tecnologías de comunicación**. Realizar medidas en entorno marino con equipos que operen en otras bandas de frecuencia. Elaborar un informe de medidas y resultados empíricos con diferentes tecnologías que permita plantearse el desarrollo de nuevas aplicaciones, o servicios para la mitigación del

impacto de la contaminación acústica en el medio marino y otras herramientas para el estudio de este medio. Se propone la utilización de: módulos RF en la banda 433 MHz, Wi-Fi en 2.4 y 5.6 GHz, y WiMAX en 5.6 GHz.

3. **Realización de un seguimiento del mercado**. Conviene estar atento a la evolución de los módulos de comunicación. Las redes de sensores y ad hoc son un tema muy activo de investigación en varias universidades, y cada vez hay más aplicaciones comerciales basadas en ellas. Constantemente aparecen en el mercado módulos con nuevas prestaciones. Concretamente en la banda 868 MHz están empezando a aparecer unidades con soporte de topología mesh, lo que simplifica considerablemente el desarrollo de aplicaciones.

4. **Profundización en el estudio teórico de enlaces radioeléctricos en el mar**. Ahondando en este estudio se podrá lograr un modelo teórico más fiel a la realidad que el utilizado en el proyecto. Deberá tener en cuenta, entre otros, los efectos comentados en el punto 4 de las conclusiones, cuantificando su magnitud.

5. **Evaluación de otros protocolos de encaminamiento**. Esta evaluación permitirá realizar una elección más precisa según la aplicación que interese. Para ello se pueden emplear simuladores de redes, como NS-2, obteniendo resultados cuantitativos de varios protocolos de encaminamiento que ayudarán a establecer una comparativa.

6. **Módulo de entrega remota de datos**. En este ámbito se podrían añadir nuevas funcionalidades, mejoras, correciones y trabajar en la compresión de datos para reducir el tráfico de la red, entre otros.

7. **Realización de pruebas reales de operación de la red**. Estas pruebas permitirán:

   - Evaluar distintas configuraciones con un número más elevado de nodos.
   - Estudiar el efecto en la red de los diferentes parámetros de configuración de los módulos.
   - Realizar pruebas en el mar con un tráfico real de alarmas.

Se considera importante continuar con estas líneas de investigación con el objetivo de mitigar la problemática medioambiental comentada en el Capítulo 1, a través de soluciones tecnológicas viables y sostenibles en todos sus aspectos.

Apéndice A

Presupuesto

EJECUCIÓN MATERIAL		
Compra de ordenador personal (software incluido)		2.000 €
Alquiler impresora láser durante 6 meses		260 €
Material oficina		300 €
Material técnico		
Módulo XBee-PRO 868 (Venco Electrónica)	(x3)	180 €
Conversor UART-USB (Lextronic)	(x3)	70 €
Antena 4.5 dBi (Cooking-Hacks)	(x3)	30 €
Cable miniUSB-USB (Cooking-Hacks)	(x3)	10 €
TOTAL EJECUCIÓN MATERIAL		2.850 €

GASTOS GENERALES	
13 % sobre Ejecución Material	370,5 €

BENEFICIO INDUSTRIAL	
6 % sobre Ejecución Material	171 €

HONORARIOS PROYECTO	
1500 horas a 15 €/ hora	22.500 €

MATERIAL FUNGIBLE	
Gastos de impresión	150 €
Encuadernación	10 €
TOTAL MATERIAL FUNGIBLE	160 €

PRESUPUESTO TOTAL	
Presupuesto total sin I.V.A.	26.051,5 €
Presupuesto total con I.V.A. (18 %)	30.740,8 €

Barcelona, Septiembre de 2011
El Ingeniero Jefe de Proyecto

Fdo.: Ramiro Utrilla Gutiérrez
Ingeniero Superior de Telecomunicación

Apéndice B

Código MATLAB

En este apéndice se incluye el código MATLAB utilizado en el Capítulo 4 para el estudio teórico de un enlace radioeléctrico en el mar.

B.1. freeSpace.m

```
%----------------------------%
% Ramiro Utrilla Gutiérrez  %
% Julio 2011                %
%----------------------------%
close all
clear all
clc

% RADIO LINK - FREE SPACE PROPAGATION
% Definitions
d = [1000:100:1000*10^3]; % m
freq = 869.525 * 10^6; % Hz
speed = 3 * 10^8; % m/s
lambda = speed / freq; % m
Pt = 25; % dBm
Gt = 4.5; % dBi
Gr = 4.5; % dBi
Pr = zeros(1,length(d)); % dBm
sens = -104*ones(1,length(d)); % dBm

% Free Space Path Loss
Lbf = zeros(1,length(d));
Lbf = 32.45 + 20*log10(d/(10^3)) + 20*log10(freq/(10^6)); % dB

% Receive Power
Pr = Pt + Gt - Lbf + Gr; % dBm

% Figure 1. Free Space Path Loss
figure
semilogx(d/(10^3), Lbf, 'b', 'linewidth', 2);
grid on
set(gca,'XTickLabel',{'1','10','100','1.000'})
xlabel('\bfDistancia [km]')
ylabel('\bfL_{bf} [dB]')
```

```
35 title(sprintf('Pérdida básica de propagación en condiciones de
      espacio libre \n Frecuencia de operación: 869.525 MHz'));
36
37 % Figure 2. Receive Power
38 figure
39 semilogx(d/(10^3), Pr, 'b', 'linewidth', 2);
40 grid on
41 set(gca,'XTickLabel',{'1','10','100','1.000'})
42 xlabel('\bfDistancia [km]')
43 ylabel('\bfP_{r} [dBm]')
44 title(sprintf('Potencia disponible en el receptor en condiciones de
      espacio libre \n P_{t}: %d dBm; G_{t}: %g dBi; G_{r}: %g dBi', Pt,
       Gt, Gr));
45 hold on
46 semilogx(d/(10^3), sens, 'r', 'linewidth', 2);
47 legend('Potencia disponible en el receptor','Sensibilidad del
      receptor')
48 hold off
```

B.2. twoRayModel.m

```
1  %----------------------------%
2  % Ramiro Utrilla Gutiérrez  %
3  % Julio 2011                %
4  %----------------------------%
5  clear all
6  close all
7  clc
8
9  % RADIO LINK - TWO RAY PROPAGATION MODEL
10 % Definitions
11 d = [1:.01:10000]; % m
12 f = 869.525 * 10^6; % Hz
13 c = 3 * 10^8; % m/s
14 lambda = c/f; % m
15 Pt = 25; % dBm
16 Gt = 4.5; % dBi
17 Gr = 4.5; % dBi
18 ht = 2; % m
19 hr = 2; % m
20 beta = pi; % rad
21 modR = 1;
22 lb = zeros(1,length(d));
23 Lb = zeros(1,length(d)); % dB
24 Lbf = zeros(1,length(d)); % dB
25 Lb2 = zeros(1,length(d)); % dB
26 Pr = zeros(1,length(d)); % dBm
27 sens = -104*ones(1,length(d)); % dBm
28
29 % Two Ray Model Path Loss
30 delta = (4*pi*ht*hr)./(lambda*d);
31 for i=1:length(d)
32     lb(i) = ((4*pi*d(i))/lambda)^2 / (2*(1-cos(delta(i))));
33     Lb(i) = 10*log10(lb(i));
```

```
34 end
35
36 % Free Space Path Loss
37 Lbf = 32.45 + 20*log10(d/(10^3)) + 20*log10(f/(10^6)); % dBm
38
39 % Two Ray Model Path Loss for long distances
40 Lb2 = 40*log10(d/(10^3)) - 20*log10(ht*hr) + 120;
41
42 % Receive Power
43 Pr = Pt + Gt - Lb + Gr; % dBm
44
45 % Figure 1. Two Ray Path Loss 1
46 figure
47 semilogx(d, Lb, 'linewidth', 2)
48 grid on
49 set(gca,'XTickLabel',{'1','10','100','1.000','10.000'})
50 xlabel('\bfDistancia [m]')
51 ylabel('\bfPérdida básica de propagación [dB]')
52 title(sprintf('Pérdida básica de propagación \n f: 869.525 MHz; h_{t
      }: %d m; h_{r}: %d m', ht, hr));
53 hold on
54 semilogx(d, Lbf,'r', 'linewidth', 2);
55 legend('Modelo de tierra plana','En condiciones de espacio libre',2)
56 hold off
57
58 % Figure 2. Two Ray Path Loss 2
59 figure
60 semilogx(d, Lb, 'linewidth', 2)
61 grid on
62 set(gca,'XTickLabel',{'1','10','100','1.000','10.000'})
63 xlabel('\bfDistancia [m]')
64 ylabel('\bfPérdida básica de propagación [dB]')
65 title(sprintf('Pérdida básica de propagación \n f: 869.525 MHz; h_{t
      }: %d m; h_{r}: %d m', ht, hr));
66 hold on
67 semilogx(d, Lb2, 'k', 'linewidth', 2);
68 semilogx(d, Lbf,'r', 'linewidth', 2);
69 legend('Modelo de tierra plana','Modelo de tierra plana largas
      distancias','En condiciones de espacio libre',2)
70 hold off
71
72 % Figure 3. Receive Power
73 figure
74 semilogx(d/(10^3), Pr, 'b', 'linewidth', 2);
75 grid on
76 set(gca,'XTickLabel',{'1','10','100','1.000','10.000'})
77 xlabel('\bfDistancia [m]')
78 ylabel('\bfP_{r} [dBm]')
79 title(sprintf('Potencia disponible en el receptor con modelo de
      tierra plana \n f: 869.525 MHz; P_{t}: %d dBm; G_{t}: %g dBi; G_{r
      }: %g dBi; h_{t}: %d m; h_{r}: %d m', Pt, Gt, Gr, ht, hr));
80 hold on
81 semilogx(d/(10^3), sens, 'r', 'linewidth', 2);
82 legend('Potencia disponible en el receptor','Sensibilidad del
      receptor')
83 hold off
```

B.3. measurements.m

```
%---------------------------%
% Ramiro Utrilla Gutiérrez  %
% Julio 2011                %
%---------------------------%
clear all
close all
clc

% RADIO LINK - TWO RAY PROPAGATION MODEL
% Definitions
d = [1:.001:10]; % m
f = 869.525 * 10^6; % Hz
c = 3 * 10^8; % m/s
lambda = c/f; % m
Pt = 25; % dBm
Gt = 4.5; % dBi
Gr = 4.5; % dBi
ht = 2; % m
hr = 2; % m
beta = pi; % rad
modR = 1;
lb = zeros(1,length(d));
Lb = zeros(1,length(d)); % dB
day1e1 = zeros(1,5); % dBm
day2e1 = zeros(1,5); % dBm
day1e2 = zeros(1,5); % dBm
day2e2 = zeros(1,5); % dBm
day1e3 = zeros(1,5); % dBm
day2e3 = zeros(1,5); % dBm
Pr = zeros(1,length(d)); % dBm
sens = -104*ones(1,length(d)); % dBm
link1 = 1.84*ones(1,5); % km
link2 = 3.06*ones(1,5); % km
link3 = 3.5*ones(1,5); % km

% Two Ray Model Path Loss
delta = (4*pi*ht*hr)./(lambda*(d*10^3));
for i=1:length(d)
    lb(i) = ((4*pi*(d(i)*10^3))/lambda)^2 / (2*(1-cos(delta(i))));
    Lb(i) = 10*log10(lb(i));
end

% Receive Power
Pr = Pt + Gt - Lb + Gr; % dBm

% Measurements
day1e1 = [-95; -93; -96; -103; -97];
day2e1 = [-92; -98; -95; -102; -97];

% Hypothetical Link 2
space1 = Pr(841) - Pr(2061);
day1e2 = day1e1 - space1;
day2e2 = day2e1 - space1;
```

```
54
55  % Hypothetical Link 2
56  space2 = Pr(841) - Pr(2501);
57  day1e3 = day1e1 - space2;
58  day2e3 = day2e1 - space2;
59
60  % Figure 1. Receive Power and Link 1 Measurements
61  figure
62  semilogx(d, Pr, 'b', 'linewidth', 2);
63  grid on
64  set(gca,'XTickLabel',{'1','10'})
65  xlabel('\bfDistancia [km]')
66  ylabel('\bfP_{r} [dBm]')
67  title(sprintf('Potencia disponible en el receptor. ENLACE 1 \n f:
       869.525 MHz; P_{t}: %d dBm; G_{t}: %g dBi; G_{r}: %g dBi; h_{t}: %
       d m; h_{r}: %d m', Pt, Gt, Gr, ht, hr));
68  hold on
69  semilogx(link1, day1e1, 'kx', 'linewidth',2);
70  semilogx(link1, day2e1, 'mo', 'linewidth',2);
71  semilogx(d, sens, 'r', 'linewidth', 2);
72  legend('Modelo de tierra plana','Medidas de la jornada 1','Medidas de
        la jornada 2','Sensibilidad del receptor')
73  hold off
74
75  % Figure 2. Receive Power and Hypothetical Link 2 Measurements
76  figure
77  semilogx(d, Pr, 'b', 'linewidth', 2);
78  grid on
79  set(gca,'XTickLabel',{'1','10'})
80  xlabel('\bfDistancia [km]')
81  ylabel('\bfP_{r} [dBm]')
82  title(sprintf('Potencia disponible en el receptor. ENLACE 2 \n f:
       869.525 MHz; P_{t}: %d dBm; G_{t}: %g dBi; G_{r}: %g dBi; h_{t}: %
       d m; h_{r}: %d m', Pt, Gt, Gr, ht, hr));
83  hold on
84  semilogx(link2, day1e2, 'kx', 'linewidth',2);
85  semilogx(link2, day2e2, 'mo', 'linewidth',2);
86  semilogx(d, sens, 'r', 'linewidth', 2);
87  legend('Modelo de tierra plana','Medidas hipotéticas de la jornada 1'
       ,'Medidas hipotéticas de la jornada 2','Sensibilidad del receptor'
       )
88  hold off
89
90  % Figure 3. Receive Power and Hypothetical Link 3 Measurements
91  figure
92  semilogx(d, Pr, 'b', 'linewidth', 2);
93  grid on
94  set(gca,'XTickLabel',{'1','10'})
95  xlabel('\bfDistancia [km]')
96  ylabel('\bfP_{r} [dBm]')
97  title(sprintf('Potencia disponible en el receptor. ENLACE 3 \n f:
       869.525 MHz; P_{t}: %d dBm; G_{t}: %g dBi; G_{r}: %g dBi; h_{t}: %
       d m; h_{r}: %d m', Pt, Gt, Gr, ht, hr));
98  hold on
99  semilogx(link3, day1e3, 'kx', 'linewidth',2);
100 semilogx(link3, day2e3, 'mo', 'linewidth',2);
```

```
101 semilogx(d, sens, 'r', 'linewidth', 2);
102 legend('Modelo de tierra plana','Medidas hipotéticas de la jornada 1'
       ,'Medidas hipotéticas de la jornada 2','Sensibilidad del receptor'
       )
103 hold off
```

Apéndice C

Cabecera y definición de funciones

En este apéndice se incluyen las cabeceras y declaraciones de las principales funciones implementadas en el sistema.

C.1. extractMessInfo

```
/*****************************************************************/
/* Take out the NNID and frameID headers of the message received  */
/*****************************************************************/
/* INPUT                                                         */
/*  -message:   Message received                                 */
/*                                                               */
/* OUTPUT                                                        */
/*  -message:   Message received without NNID and frameID fields. */
/*  -nnid:      NNID field                                       */
/*  -frameID:   frameID field                                    */
/*                                                               */
/* RETURNS                                                       */
/*  - ERROR if there is any problem in the process.              */
/*  - OK if the process is succesful.                            */
/*                                                               */
/*****************************************************************/
int extractMessInfo (char *nnid, char *frameID, char *message);
```

C.2. checkAlarm

```
/*****************************************************************/
/* Check if there are new alarms in the alarm.dat file.          */
/*****************************************************************/
/* INPUT                                                         */
/*  -lastAlarm: The alarmID of the last alarm used.              */
/*                                                               */
/* RETURNS                                                       */
/*  -ERROR if there is not more alarms now.                      */
/*  -The index of the line in the file with the next alarm to send*/
/*****************************************************************/
int checkAlarm (unsigned int lastAlarm);
```

C.3. sendLocalAlarm

```
/*****************************************************************/
/* Read an alarm in the 'alarmIndex' line from the 'alarm.dat'   */
/* file and send it.                                             */
/*****************************************************************/
/* INPUT                                                         */
/*  -dnid:       DNID field                                      */
/*  -nnid:       NNID field                                      */
/*  -nid:        SNID field                                      */
/*  -alarmIndex: The alarm index of the alarm to read.           */
/*                                                               */
/* RETURNS                                                       */
/*  -The alarmID of the alarm sended.                            */
/*  -0 if there is any problem in the process.                   */
/*****************************************************************/
unsigned int sendLocalAlarm (char *dnid, char *nnid, char *nid, int
   alarmIndex);
```

C.4. exeDATA

```
/*****************************************************************/
/* Execute the DATA reception process.                           */
/*****************************************************************/
/* INPUT                                                         */
/*  -nid:         NID of the own node.                           */
/*  -DATAmessage: DATA message received.                         */
/*  -lt_rte:      Lifetime of a routing table entry.             */
/*                                                               */
/* OUTPUT                                                        */
/*  -auxdnid:     DNID of the ACK, when a route has to be        */
/*                rediscovered.                                  */
/*                                                               */
/* RETURNS                                                       */
/*  -ERROR if there is any problem in the process.               */
/*  -OK if the process is succesful.                             */
/*****************************************************************/
int exeDATA(char *nid, char *DATAmessage, int lt_rte, char *auxdnid);
```

C.5. sendDATA

```
/*****************************************************************/
/* Send a unicast DATA message.                                  */
/*****************************************************************/
/* INPUT                                                         */
/*  -nnid:        Next node ID in the route.                     */
/*  -DATAmessage: The part of the message with the info.         */
/*****************************************************************/
void sendDATA (char *nnid, char *DATAmessage);
```

C.6. exeACK

```
/*****************************************************************/
/* Execute the ACK reception process.                            */
/*****************************************************************/
/* INPUT                                                         */
/*  -nid            NID of the own node.                         */
/*  -ACKmessage:    ACK message received.                        */
/*  -lt_rte:        Lifetime of a routing table entry.           */
/* RETURNS                                                       */
/*  -alarmID when the ACK messages corresponds with a local      */
/*   alarm sended.                                               */
/*  -0 in other cases.                                           */
/*****************************************************************/
unsigned int exeACK (char *nid, char *ACKmessage, int lt_rte);
```

C.7. sendACK

```
/*****************************************************************/
/* Send a unicast ACK message.                                   */
/*****************************************************************/
/* INPUT                                                         */
/*  -nnid           Next node ID in the route.                   */
/*  -ACKmessage:    The part of the message with the info.       */
/*****************************************************************/
void sendACK (char *nnid, char *ACKmessage);
```

C.8. serialData2queue

```
/*****************************************************************/
/* Read the serial data and put the messages received between    */
/* brackets ([Hello world!]) in the message queue.               */
/*****************************************************************/
/* INPUT                                                         */
/*  -serialData: Data readed in the serial port.                 */
/*  -Q:          Message queue.                                  */
/*****************************************************************/
void serialData2queue (char *serialData, queue Q);
```

C.9. initSeqNum

```
/*****************************************************************/
/* Inicialize the sequence number with time(NULL) for making a   */
/* strong system against resets.                                 */
/*****************************************************************/
void initSeqNum (void);
```

C.10. updateSeqNum

```
/*****************************************************************/
/* Update the sequence number.                                   */
/*****************************************************************/
/* INPUT                                                         */
/*  -seqN: The sequence number received in a RREQ or PREQ messages*/
/*                                                               */
/* RETURNS                                                       */
/*  -The sequence number updated.                                */
/*****************************************************************/
unsigned long int updateSeqNum (unsigned long int seqN);
```

C.11. checkRREQ

```
/*****************************************************************/
/* Check the RREQ table to verify that the RREQ is updated.      */
/*****************************************************************/
/* INPUT                                                         */
/*  -snid: Source node id of the RREQ message.                   */
/*  -seqN: The sequence number received in a RREQ message.       */
/*                                                               */
/* RETURNS                                                       */
/*  -OK to add the new entry.                                    */
/*  -ERROR to discard the new entry.                             */
/*****************************************************************/
int checkRREQ (char *snid, unsigned long int seqN);
```

C.12. exeRREQ

```
/*****************************************************************/
/* Execute the RREQ process.                                     */
/*****************************************************************/
/* INPUT                                                         */
/*  -nid:         NID of the own node.                           */
/*  -max_hops:    Maximum number of hops setted in the node.conf */
/*  -RREQmessage: RREQ message received.                         */
/*  -lt_rte:      Lifetime of a routing table entry.             */
/*                                                               */
/* RETURNS                                                       */
/*  -OK if the process is succesful.                             */
/*  -ERROR if there is any problem in the process.               */
/*                                                               */
/*****************************************************************/
int exeRREQ (char *nid, int max_hops, char *RREQmessage, int lt_rte);
```

C.13. sendRREQ

```
/*****************************************************************/
/* Send a broadcast RREQ message. If the sequence number in the  */
/* argument is '0', the RREQ is local and the function increase  */
/* the local seqNum. If the argument is different of 0, the RREQ */
/* is not local, it is a rebroadcast and it have its own sequence */
/* number.                                                       */
/*****************************************************************/
/* INPUT                                                         */
/*  -snid: SNID field.                                           */
/*  -seqN: The sequence number of the RREQ, or 0 if it is a local */
/*         RREQ which has to be sent.                            */
/*  -dnid: DNID field.                                           */
/*  -pnid: PNID field.                                           */
/*  -hops: The number of hops of the frame.                      */
/*                                                               */
/* RETURNS                                                       */
/*  -OK when make a RREQ rebroadcast.                            */
/*  -Sequence number when sends a local RREQ.                    */
/*****************************************************************/
unsigned long int sendRREQ (char *snid, unsigned long int seqN, char
   *dnid, char *pnid, int hops);
```

C.14. checkPREQ

```
/*****************************************************************/
/* Check the PREQ table to verify that the PREQ is updated.      */
/*****************************************************************/
/* INPUT                                                         */
/*  -snid: Source node id of the PREQ message.                   */
/*  -seqN: The sequence number received in a PREQ message.       */
/*                                                               */
/* RETURNS                                                       */
/*  -OK to add the new entry.                                    */
/*  -ERROR to discard the new entry.                             */
/*****************************************************************/
int checkPREQ (char *snid, unsigned long int seqN);
```

C.15. exePREQ

```
/*****************************************************************/
/* Execute the PREQ process.                                     */
/*****************************************************************/
/* INPUT                                                         */
/*  -nid:         NID of the own node.                           */
/*  -PREQmessage: PREQ message received.                         */
/*  -lt_rte:      Lifetime of a routing table entry.             */
/*                                                               */
```

```
/* RETURNS                                                          */
/*  -OK if the process is succesful.                                */
/*  -ERROR if there is any problem in the process.                  */
/*******************************************************************/
int exePREQ (char *nid, char *PREQmessage, int lt_rte);
```

C.16. sendPREQ

```
/*******************************************************************/
/* Send a broadcast PREQ message. If the sequence number in the    */
/* argument is '0', the PREQ is local and the function increase    */
/* the local seqNum. If the argument is different of 0, the PREQ   */
/* is not local, it is a rebroadcast and it have its own sequence  */
/* number.                                                         */
/*******************************************************************/
/* INPUT                                                           */
/*  -snid: SNID field.                                             */
/*  -seqN: The sequence number of the PREQ, or 0 if it is a local  */
/*         PREQ which has to be sent.                              */
/*  -pnid: PNID field.                                             */
/*  -hops: The number of hops of the frame.                        */
/*                                                                 */
/* RETURNS                                                         */
/*  -OK when make a PREQ rebroadcast.                              */
/*  -Sequence number when sends a local PREQ.                      */
/*******************************************************************/
unsigned long int sendPREQ (char *snid, unsigned long int seqN, char
   *pnid, int hops);
```

C.17. exeRREP

```
/*******************************************************************/
/* Execute the RREP process.                                       */
/*******************************************************************/
/* INPUT                                                           */
/*  -nid:         NID of the own node.                             */
/*  -max_hops:    Maximum number of hops setted in the node.conf   */
/*  -RREPmessage: RREP message received.                           */
/*  -lt_rte:      Lifetime of a routing table entry.               */
/*                                                                 */
/* RETURNS                                                         */
/*  -The sequence number (seqN > 0) when the RREP corresponds      */
/*   with one local RREQ.                                          */
/*  -0 in the other cases.                                         */
/*******************************************************************/
unsigned long int exeRREP (char *nid, int max_hops, char *RREPmessage
   , int lt_rte);
```

C.18. sendRREP

```
/*****************************************************************/
/* Send a unicast RREP message.                                  */
/*****************************************************************/
/* INPUT                                                         */
/*  -nnid: NNID field.                                           */
/*  -pnid: PNID field.                                           */
/*  -snid: SNID field.                                           */
/*  -seqN: The sequence number.                                  */
/*  -hops: The number of hops of the frame.                      */
/*  -dnid: DNID field.                                           */
/*****************************************************************/
void sendRREP (char *nnid, char *pnid, char *snid, unsigned long int
   seqN, int hops, char *dnid);
```

C.19. exePREP

```
/*****************************************************************/
/* Execute the PREP process.                                     */
/*****************************************************************/
/* INPUT                                                         */
/*  -nid:         NID of the own node.                           */
/*  -PREPmessage: PREP message received.                         */
/*  -lt_rte:      Lifetime of a routing table entry.             */
/* RETURNS                                                       */
/*  -OK if the process is succesful.                             */
/*  -ERROR if there is any problem in the process.               */
/*****************************************************************/
int exePREP (char *nid, char *PREPmessage, int lt_rte);
```

C.20. sendPREP

```
/*****************************************************************/
/* Send a unicast PREP message.                                  */
/*****************************************************************/
/* INPUT                                                         */
/*  -nnid:      NNID field.                                      */
/*  -dnid:      DNID field.                                      */
/*  -timeStamp: Timestamp in the PREP received, or 0 if it is a   */
/*              local PREP message.                              */
/*  -snid:      SNID field.                                      */
/*  -latitude:  Latitude in the PREP received, or NULL if it is a */
/*              local PREP message.                              */
/*  -longitude: Longitude in the PREP received, or NULL if it is a*/
/*              local PREP message.                              */
/*****************************************************************/
void sendPREP (char *nnid, char *dnid, unsigned long int timeStamp,
   char *snid, char *latitude, char *longitude);
```

C.21. checkNewRoute

```
/*****************************************************************/
/* Check if the new route is already in the routing table and    */
/* also check the update conditions.                             */
/*****************************************************************/
/* INPUT                                                         */
/*  -dnid:   Destination node of the route.                      */
/*  -seqn:   Sequence number of the route.                       */
/*  -hops:   The number of hops of the route till the own node.  */
/*  -lt_tre: Lifetime of a routing table entry.                  */
/* RETURNS                                                       */
/*  - OK when the routing table is OK to be updated.             */
/*  - ERROR if the routing table is already updated.             */
/*****************************************************************/
int checkNewRoute (char *dnid, unsigned long int seqn, int hops, int
   lt_rte);
```

C.22. newRoute

```
/*****************************************************************/
/* Add a new route to the routing table.                         */
/*****************************************************************/
/* INPUT                                                         */
/*  -dnid: Destination node of the route.                        */
/*  -nnid: Next node in the route to the DNID.                   */
/*  -seqn: Sequence number of the route.                         */
/*  -hops: The number of hops of the route till the own node.    */
/* RETURNS                                                       */
/*  -OK if the process is succesful.                             */
/*  -ERROR if there is any problem in the process.               */
/*****************************************************************/
int newRoute(char *dnid,char *nnid,unsigned long int seqn,int hops);
```

C.23. getRoute

```
/*****************************************************************/
/* Look for a valid route to the dnid.                           */
/*****************************************************************/
/* INPUT                                                         */
/*  -dnid:   Destination node of the route.                      */
/*  -lt_rte: Lifetime of a routing table entry.                  */
/* OUTPUT                                                        */
/*  -nnid:   Next node in the route to the DNID.                 */
/* RETURNS                                                       */
/*  -OK if the process is succesful.                             */
/*  -ERROR if there is any problem in the process.               */
/*****************************************************************/
int getRoute (char *dnid, int lt_rte, char *nnid);
```

Bibliografía

Y así, del mucho leer y del poco dormir, se le secó el celebro de manera que vino a perder el juicio.

Miguel de Cervantes Saavedra

Chakeres, I. D. y Belding-Royer, E. M. AODV routing protocol implementation design. En *Proceedings of the 24th International Conference on Distributed Computing Systems Workshops - W7: EC (ICDCSW'04) - Volume 7*, ICDCSW '04, páginas 698–703. IEEE Computer Society, Washington, DC, USA, 2004. ISBN 0-7695-2087-1.

Chakeres, I. D. y Klein-Berndt, L. AODVjr, AODV simplified. *SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 6, páginas 100–101, 2002. ISSN 1559-1662.

Das, S., Perkins, C. y Royer, E. Performance comparison of two on-demand routing protocols for ad hoc networks. En *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 1, páginas 3 –12 vol.1. 2000.

Domingo-Aladrén, M. C. *Diferenciación de servicios y mejora de la supervivencia en redes ad hoc conectadas a redes fijas*. Phd, Universitat Politècnica de Catalunya, 2005.

Gomez, C., Salvatella, P., Alonso, O. y Paradells, J. Adapting AODV for IEEE 802.15.4 mesh sensor networks: theoretical discussion and performance evaluation in a real environment. En *Proceedings of the 2006 International Symposium on on World of Wireless, Mobile and Multimedia Networks*, WOWMOM '06, páginas 159–170. IEEE Computer Society, Washington, DC, USA, 2006. ISBN 0-7695-2593-8.

Kawadia, V., Zhang, Y. y Gupta, B. System services for ad-hoc routing: architecture, implementation and experiences. En *Proceedings of the 1st international conference on Mobile systems, applications and services*, MobiSys '03, páginas 99–112. ACM, New York, NY, USA, 2003.

Klein-Berndt, L. *A quick guide to AODV routing*. Wireless Communications Technologies Group; National Institute of Standards and Technology, 2001. Disponible en `http://www.antd.nist.gov/wctg/aodv_kernel/` (último acceso, Abril, 2011).

Klein-Berndt, L. y Chakeres, I. D. *AODVjr. A simplified version of AODV*. Mobility Management and Networking Laboratory UC Santa Barbara; National Institute of Standards and Technology, 2002. Disponible en `http://www.ianchak.com/` (último acceso, Mayo, 2011).

KONG, P., WANG, H., GE, Y., ANG, C., WEN, S., PATHMASUNTHARAM, J., ZHOU, M. y DIEN, H. A performance comparison of routing protocols for maritime wireless mesh networks. En *Wireless Communications and Networking Conference, 2008. WCNC 2008. IEEE*, páginas 2170 –2175. 2008. ISSN 1525-3511.

PERKINS, C. y ROYER, E. Ad-hoc on-demand distance vector routing. En *Mobile Computing Systems and Applications, 1999. Proceedings. WMCSA '99. Second IEEE Workshop on*, páginas 90 –100. 1999.

PERKINS, C. E., BELDING-ROYERS, E. y DAS, S. R. Ad-hoc on-demand distance vector (AODV) routing. *Request For Comments Experimental 3561, Internet Engineering Task Force*, 2003.

ROYER, E. y TOH, C.-K. A review of current routing protocols for ad hoc mobile wireless networks. *Personal Communications, IEEE*, vol. 6(2), páginas 46 –55, 1999. ISSN 1070-9916.

WANG, W., YOUN, J.-H. y SHARIF, H. The implementation of an energy balanced routing protocol with dynamic power scaling in tinyos. En *Sensor Networks, Ubiquitous, and Trustworthy Computing, 2006. IEEE International Conference on*, vol. 2, páginas 262 –267. 2006.

YAPICIOGLU, T. y OKTUG, S. F. Effect of wave height to connectivity and coverage in sea surface wireless sensor networks. En *International Symposium on Computer and Information Sciences*, páginas 328–333. 2009.