ugr | Universidad de Granada

ESTUDIOS DE INGENIERÍA
DE TELECOMUNICACIÓN

PROYECTO FIN DE CARRERA

PROTOCOLOS DE ENCAMINAMIENTO IP EN DISPOSITIVOS MÓVILES PARA CREAR REDES MANETs

CURSO:11/12

JOSÉ MANUEL PÉREZ HUESO

El tribunal constituido para la evaluación del PFC titulado:

PROTOCOLOS DE ENCAMINAMIENTO IP EN DISPOSITIVOS MÓVILES PARA CREAR REDES MANETs

Realizado por el alumno: José Manuel Pérez Hueso.

Y dirigido por el tutor: Jorge Navarro Ortiz.

Ha resuelto asignarle la calificación de:

- ☐ SOBRESALIENTE (9 – 10 puntos)
- ☐ NOTABLE (7 – 8.9 puntos)
- ☐ APROBADO (5 – 6.9 puntos)
- ☐ SUSPENSO

Con la nota[1]: ☐ puntos.

El Presidente: _____

El Secretario: _____

El Vocal: _____

Granada, a de de 201 .

[1] Solamente con un decimal.

ugr | Universidad de Granada

ESTUDIOS DE INGENIERÍA DE TELECOMUNICACIÓN

PROTOCOLOS DE ENCAMINAMIENTO IP EN DISPOSITIVOS MÓVILES PARA CREAR REDES MANETs

REALIZADO POR:
José Manuel Pérez Hueso

DIRIGIDO POR:
Jorge Navarro Ortiz

DEPARTAMENTO:
Teoría de la Señal, Telemática y Comunicaciones

Granada, Enero de 2012

PROTOCOLOS DE ENCAMINAMIENTO IP EN DISPOSITIVOS MÓVILES PARA CREAR REDES MANETs

José Manuel Pérez Hueso

PALABRAS CLAVE:

Redes MANET, redes ad-hoc, AODV, Bada, protocolos de encaminamiento, dispositivos móviles.

RESUMEN:

En los últimos años se ha producido un vertiginoso aumento en el tráfico de datos por parte de dispositivos móviles, fruto de las constantes mejoras de éstos en cuanto a capacidades de procesamiento y conectividad se refiere. Tal es el caso que *smartphones* o *tablets* actuales permiten tener una conexión a Internet mediante las tecnologías 3G o WiFi.

Aprovechando las funcionalidades de la tecnología WiFi, en este trabajo se pretende implementar un protocolo de nivel de red que permita crear una red MANET entre varios dispositivos y comprobar su correcto funcionamiento mediante dos aplicaciones diseñadas para este fin. Estas aplicaciones son una aplicación de eco y una aplicación de chat. La primera servirá para mostrar el correcto funcionamiento de las rutas creadas, mientras que la segunda permitirá intercambiar mensajes de texto entre los usuarios que formen parte de la red. Debido a la forma de operar de las redes MANET, estos mensajes de texto se podrán enviar tanto a dispositivos que se encuentren conectados directamente como a otros que necesiten de elementos intermedios para encaminar los mensajes. Es decir, todos los dispositivos podrán actuar como origen y destino además de poseer capacidades para funcionar como *routers* intermedios.

La aplicación se ha desarrollado en un sistema operativo de reciente aparición denominado Bada. Samsung, una de las empresas más importantes en el ámbito de los dispositivos móviles, es la marca propietaria de Bada.

IP ROUTING PROTOCOLS FOR MOBILE DEVICES TO CREATE MANET NETWORKS

José Manuel Pérez Hueso

KEYWORDS:

MANET, ad-hoc networks, AODV, Bada, routing protocols, mobile devices.

ABSTRACT:

In recent years there has been a rapid increase in data traffic from mobile devices as a result of steady improvements in terms of their processing capabilities and connectivity. Therefore, current smartphones or tablets constitute examples of how to have an Internet connection via 3G or WiFi technologies.

Considering the capabilities of the WiFi technology, we propose an implementation of a network level protocol that allows the creation of a MANET between several devices. We have developed two applications with the purpose of verifying this implementation. In particular, we use an echo application to check the proper behaviour of the routing protocol, and a chat application to exchange text messages between users that are part of the network. Due to how MANET networks operate, text messages may be sent to devices with a direct link or to devices not directly connected that require intermediate elements to forward the messages. Therefore, all devices may act as source and destination, in addition of working as intermediate routers.

The application was developed in a new operating system called Bada. Samsung, one of the most important companies in the mobile industry, is the owner of Bada.

D. Jorge Navarro Ortiz

Profesor del departamento de Teoría de la Señal, Telemática y Comunicaciones de la Universidad de Granada, como director del Proyecto Fin de Carrera de D. José Manuel Pérez Hueso

Informa:

que el presente trabajo, titulado:

Protocolos de encaminamiento IP en dispositivos móviles para crear redes MANETs

Ha sido realizado y redactado por el mencionado alumno bajo nuestra dirección, y con esta fecha autorizo a su presentación.

Granada, a _____ de _____ de 20___

Fdo.:_____

Los abajo firmantes autorizan a que la presente copia de Proyecto Fin de Carrera se ubique en la Biblioteca del Centro y/o departamento para ser libremente consultada por las personas que lo deseen.

Granada, a _____ de _____ de 20___

(Firma y números de DNI del alumno y el tutor)

AGRADECIMIENTOS

Al finalizar esta etapa de mi vida quiero agradecer a todos los que de alguna forma me han ayudado a conseguirlo y por supuesto, agradecer a Jorge, mi tutor, por brindarme la posibilidad de realizar este proyecto y aconsejarme siempre de manera adecuada.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

GLOSARIO

3G	Tercera generación de tecnologías para comunicaciones móviles.
AODV	*Ad-hoc On Demand Vector,* protocolo para crear una red MANET.
ABR	*Associativity Based Routing*, protocolo para crear una red MANET.
CBRP	*Cluster Based Routing Protocol*, protocolo para crear una red MANET.
DARPA	*Defense Advanced Research Projects Agency*, agencia de investigación en proyectos avanzados de defensa del Departamento de Defensa de Estados Unidos.
DHCP	*Dynamic Host Configuration Protocol*, protocolo destinado a la configuración automática de equipos en una red IP.
DNS	*Domain Name System*, sistema de nomenclatura jerárquica que permite traducir nombres inteligibles en direcciones IP.
DSDV	*Destination Sequenced Distance Vector*, protocolo para crear una red MANET.
DSR	*Dynamic Source Routing*, protocolo para crear una red MANET.
HTTP	*Hypertext Transfer Protocol*, protocolo de transferencia de hipertexto.
IEEE	*Institute of Electrical and Electronics Engineers*, Instituto de Ingenieros Eléctricos y Electrónicos.
IETF	*Internet Engineering Task Force*, organización internacional de normalización en el ámbito de Internet.
IP	*Internet Protocol,* protocolo del nivel tres del modelo OSI (capa de red). También corresponde con la etiqueta numérica que identifica de manera lógica y jerárquica a una interfaz de un dispositivo dentro de una red que utilice el protocolo IP.
MANET	*Mobile Ad-hoc Networks*, redes móviles descentralizadas que se crean para una tarea concreta.
OLSR	*Optimized Link State Routing*, protocolo para crear una red MANET.
OSI	*Open System Interconnection,* marco de referencia para la definición de

arquitecturas de interconexión de sistemas de comunicaciones.

TCP *Transmission Control Protocol*, protocolo de comunicación orientado a conexión y fiable del nivel de transporte.

TTL *Time To Live*, campo que permite establecer un límite en el número de enrutadores que un paquete IP puede atravesar.

UDP *User Datagram Protocol*, protocolo de comunicación no orientado a conexión y no fiable del nivel de transporte basado en el intercambio de datagramas.

WRP *Wireless Routing Protocol*, protocolo para crear una red MANET.

ZRP *Zone Routing Protocol*, protocolo para crear una red MANET.

1 Introducción

Durante este primer capítulo se pretende hacer una introducción al concepto de red MANET, explicar las motivaciones para realizar este estudio y presentar la estructura de los demás apartados del proyecto.

1.1 Motivación y objetivos

Como se indicará de manera más precisa en el apartado 1.4, el auge de las comunicaciones inalámbricas es un hecho incuestionable. Por lo tanto, serán necesarias aplicaciones y protocolos que hagan posible y mejoren la interconexión entre dispositivos móviles.

Debido a la movilidad, no siempre se dispondrá de una infraestructura fija, por lo que se debe pensar en otros mecanismos de interconexión. Aún disponiendo de movilidad y de infraestructuras fijas, como la comunicación de un móvil con su estación base, se puede pensar en la posibilidad de tener aplicaciones que no dependan de un servicio ofrecido por una operadora. Los protocolos que utilicen los dispositivos para comunicarse entre sí, tienen que tener en cuenta la naturaleza cambiante de una topología móvil. Habrá dispositivos que cambien de lugar, que desaparezcan o que se incorporen dinámicamente, por lo que no siempre será adecuado tener actores principales y secundarios, sino que en muchos casos será interesante que estén conectados de manera descentralizada (ad-hoc).

Mediante el presente trabajo, se pretende implementar un protocolo de comunicación ad-hoc para poder ser probado y utilizado en un dispositivo real. Antes de escoger uno en concreto, se han analizado las distintas soluciones existentes. El dispositivo para el cual se programará el protocolo es el Samsung Wave GT-S8500 cuyo sistema operativo es Bada [1]. Durante el capítulo 6 se señalarán algunas limitaciones encontradas en la arquitectura de Bada y cómo se han solventado. Además, realizar una aplicación en un sistema operativo emergente constituye otra de las motivaciones y objetivos del proyecto ya que es propiedad de una de las marcas más importantes en el mercado de la telefonía móvil, Samsung. Según un informe reciente realizado por la empresa Gartner [2], el mayor fabricante a nivel mundial de móviles es Nokia, con una cuota de mercado del 23.9%, seguido de Samsung con el 17.8% (0.6 puntos más que el año pasado). Los siguientes en la lista son LG (4.8%), Apple (3.9%) y ZTE (3.2%). Si se centra la atención en las ventas de *smartphones*, los vendidos con sistema operativo

Android alcanzaron los 60.4 millones (52.5% de cuota de mercado). Apple vendió 17.2 millones de móviles inteligentes con su sistema operativo (15%), mientras que Blackberry situó su porcentaje de ventas en el 11% del total. Los dispositivos de Samsung con sistema operativo Bada tuvieron una cuota de mercado del 2.2% con un crecimiento de 0.9 puntos.

A propósito del auge del número de dispositivos móviles vendidos, se presentará en el apartado 1.4 un estudio que muestra la evolución que se presume que ocurra entre 2010 y 2015. Los resultados muestran un aumento vertiginoso en el número de unidades fabricadas y en la cantidad de datos mandados a través de estos elementos tecnológicos. Por esto, resulta de gran interés la investigación en tipos de protocolos que les proporcionen funcionalidades adicionales.

Del mismo modo, la cantidad de servicios ofrecidos por aplicaciones que basen su funcionamiento en la forma de actuar de las redes MANETs constituye otra motivación importante. En los apartados 1.2 y 1.3 se explicará en qué se basan y ejemplos de posibles aplicaciones.

1.2 ¿Qué es una red MANET?

1.2.1 Definición e historia

MANET es la forma abreviada de "***M**obile **A**d-hoc **NET**works*" y puede definirse como redes en las que todos los dispositivos (nodos) se encuentran en igualdad de condiciones y se conectan para una actividad concreta. Esta igualdad de condiciones significa que no hay elementos que tengan funcionalidades específicas, como enrutadores (*routers*) o destinos finales. De este modo, todos pueden ser *routers*, origen y destino de las comunicaciones.

En este tipo de redes, cada dispositivo debe ser capaz de conocer automáticamente cuál es el camino para llegar al destino final. Existen diferentes formas de establecer estas rutas según el protocolo que se utilice para formar la red MANET (en el capítulo 2, se discutirán los métodos que se han estudiado para formar las tablas de encaminamiento).

Por ello, los dispositivos que formen parte de una red de estas características, deberán estar dotados de mayor capacidad de procesamiento. La incorporación de *smartphones* y *tablets* hacen posible tener un escenario que propicia realizar este tipo de comunicaciones inalámbricas ad-hoc.

Con respecto a los inicios, según [3], la retransmisión multi-salto tiene sus orígenes en el año 500 antes de Cristo. Darío I (522-486 aC), rey de Persia, ideó un sistema de comunicación revolucionario que se usaba para enviar mensajes y noticias desde su capital a las provincias remotas de su imperio. Consistía en un sistema de rutas en las que hombres colocados estratégicamente en estructuras altas podían comunicarse mediante voz e ir transmitiendo el mensaje salto a salto. Este método era 25 veces más rápido que un mensajero de la época.

Ya en la época actual, en 1970 Norman Abramson, junto con su grupo de investigación, idearon un sistema de comunicación (AlohaNet) para poner en contacto las distintas sedes de la Universidad de Hawaii de manera inalámbrica, debido a que estaban ubicadas en diferentes islas. El gran reto conseguido con este resultado, fue el hecho de conseguir un método con varios nodos accediendo (acceso múltiple) a un único canal compartido. En principio, la

comunicación estaba ideada para ser de un único salto, pero el hecho de poder compartir el canal de manera eficiente, hizo que otros proyectos pudieran tener a AlohaNet como punto de partida para nuevas investigaciones.

Tal es el caso de PRNET, un proyecto promovido por la agencia de investigación en proyectos de defensa DARPA [4], que buscaba crear una red de paquetes inalámbrica para aplicaciones militares. La idea inicial era usar un control centralizado, pero se derivó hacia un sistema distribuido y multi-salto que pudiera operar en un área geográfica amplio. El sistema se diseñó para auto-organizarse, auto-configurarse y detectar la conectividad a nivel radio. Su funcionamiento estaba gobernado por la operación de un protocolo de rutas dinámico sin el soporte de una infraestructura fija.

Además de este proyecto, durante la década de los 80 se llevaron a cabo numerosas investigaciones en aplicaciones militares. Este hecho propició la necesidad de tener estándares relativos a esta forma de comunicación. Por ello, se creó el grupo de trabajo de la *Internet Engineering Task Force* (IETF) denominado MANET [5], encargado de la estandarización de protocolos en las redes ad-hoc inalámbricas.

1.3 Aplicaciones de las redes MANETs

En este apartado se presentan algunos escenarios en los que el despliegue de una red ad-hoc puede tener ventajas significativas.

1.3.1 Aplicaciones militares

Uno de los campos en los que se está invirtiendo más dinero en proyectos relacionados con el mundo de las redes MANETs es en el ámbito militar.

Muchas operaciones militares se caracterizan por el hecho de que un grupo de soldados, o grupo de vehículos, realizan una serie de operaciones en un determinado lugar y deben estar comunicados entre ellos. En general esta comunicación puede hacerse con un punto central que actúa de antena para dar cobertura al resto de elementos, pero esta aproximación presenta varios inconvenientes:

- Si falla el punto o los puntos de acceso, la comunicación falla.
- Cuanto más alejado se esté del punto de acceso, más potencia se necesita para transmitir, lo que conlleva mayor consumo energético.
- Si el enemigo destruye el punto central, se pierde la comunicación.

Entonces, se podría pensar en un sistema que pudiese crear por sí solo una infraestructura independiente en el que cada elemento del grupo militar pudiera actuar como antena. Esto puede tener los siguientes beneficios:

- Aunque falle algún elemento, los demás pueden comunicarse entre ellos.

- Como la comunicación se establece salto a salto entre elementos adyacentes del grupo, la potencia de transmisión no tendrá que ser tan elevada, ahorrando así energía.

- Debido a la naturaleza de las redes MANETs, cuando se pierde una ruta se intenta buscar un camino alternativo para establecer la comunicación. Este hecho dota de robustez a la red, aunque también puede perderse la conexión si fallasen varios nodos del grupo de forma que se imposibilitase llegar a dispositivos con los que no se tiene una visión directa.

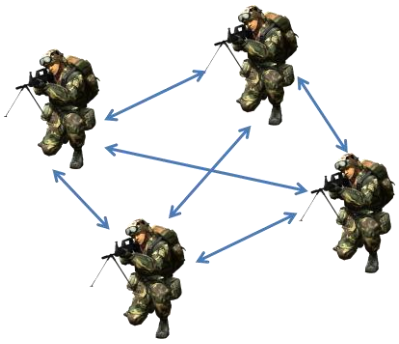

Figura 1.1: *Disposición de red MANET*

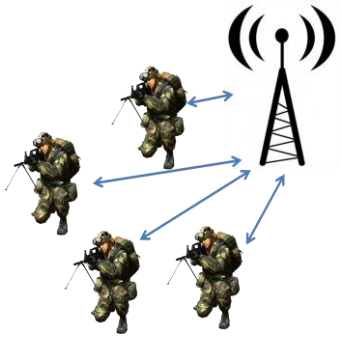

Figura 1.2: *Disposición de comunicación tradicional*

Por otro lado, el grupo que esté realizando alguna misión tiene que estar en comunicación con un campamento central. Para esto, uno de ellos puede ser el que sirva de puente entre el grupo y la base, pudiendo existir un equipo de respaldo para que en caso de que falle, sea otro componente el que permita la comunicación hasta el campo base.

1.3.2 Catástrofes

Otra aplicación importante que tienen las redes MANETs, es la posibilidad de crear una infraestructura de comunicación en lugares donde hayan ocurrido catástrofes y no estén disponibles comunicaciones tradicionales (Internet, móvil, línea telefónica fija…). Al crear una nueva red de comunicaciones en la zona afectada, se podría mejorar la organización y cooperación entre los equipos de rescate con otros servicios y acelerar las alertas.

Un ejemplo de este tipo de aplicaciones es LifeNet [6], un software de código abierto que permite a usuarios de ordenadores portátiles o móviles crear una red ad-hoc basada en WiFi sin utilizar ninguna estación base. Es una forma de proporcionar conectividad en áreas donde las infraestructuras de comunicaciones se hayan destruido o no existan. Puede ser utilizado para mejorar las actividades coordinadas entre los grupos que proporcionen asistencia como pueden ser personal médico, policía o equipos de rescate. Por otro lado, existen poblaciones en países subdesarrollados o en vías de desarrollo que se encuentran en zonas remotas donde no ha sido posible proporcionar conectividad debido a la falta de recursos. Esas áreas son muy vulnerables a las catástrofes debido a la falta de comunicación a tiempo. Según la información obtenida en la referencia apuntada al comienzo de este párrafo, Lifenet puede llenar este vacío. Además, puede ser utilizado en redes de sensores para monitorizar bosques donde exista una probabilidad alta de que se produzcan incendios o para monitorizar la contaminación del aire y ofrecer alertas de manera rápida.

1.3.3 Campus inalámbricos

Centrando la atención en la vida estudiantil se puede observar que la comunidad universitaria está organizada en campus, conjunto de terrenos y edificios que pertenecen a una universidad. Si se camina con un dispositivo móvil por los edificios de las diferentes facultades, se puede tener acceso por medio de tecnología WiFi a contenidos de Internet, pero no así si te encuentras en lugares como jardines al aire libre, en los que normalmente la cobertura es muy pobre. Si entre todos los usuarios pudiesen crear una red que permitiese llegar hasta el punto de acceso más cercano en el edificio, aumentaría la cobertura global sin tener que dotar a la universidad de ningún otro dispositivo adicional.

Figura 1.3: *Ejemplo de aplicación MANET (campus)*

En este sentido, en la Figura 1.3 se muestra una posible comunicación entre varios grupos de estudiantes. Entre ellos podrían formar estructuras de red autónomas y establecer conexión con una infraestructura fija instalada en uno de los edificios.

En referencia a la forma de modelar estas comunicaciones, un artículo [7] muestra una solución para simular modelos de movilidad en redes ad-hoc pensados para los campus. En él se propone un modelo realista donde se puede definir la posición, forma y tamaño de los obstáculos. Además, se utilizan una serie de caminos predefinidos para la interconexión de facultades que pertenezcan a un mismo campus.

1.3.4 Otro tipo de redes MANETs, las VANETs

Dentro del grupo de las redes inalámbricas ad-hoc, existen numerosos estudios y proyectos sobre las denominadas redes VANETs (***V****ehicular* ***A****d-hoc* ***NET****works*) [8], un tipo de redes MANETs destinadas a dotar de información a los usuarios en el ámbito de la circulación vial.

Uno de los casos significativos en el que se puede mostrar su uso corresponde con la prevención de accidentes entre vehículos adyacentes. Este aspecto es de vital importancia ya que las estadísticas de la Dirección General de Tráfico de 2010 [9] muestran los siguientes datos:

- Accidentes por alcance: 15074.
- Accidentes múltiples o en caravana: 5062.
- Accidentes por vehículo estacionado o averiado: 600.

Por lo tanto, se trata de cifras lo suficientemente importantes como para pensar en medidas que pudiesen disminuirlas.

De este modo, tal y como se aprecia en la Figura 1.4, si los automóviles que estuviesen involucrados en un atasco mandaran señales advirtiendo a los demás de que están parados, los que viniesen a una velocidad alta y se pudiesen encontrar la vía sin tráfico fluido, podrían ser advertidos con suficiente antelación como para aminorar la marcha y que no hubiese colisiones en las zonas traseras del embotellamiento.

Figura 1.4: *Ejemplo de uso para evitar accidentes en atascos*

Este tipo de redes también son capaces de dar información sobre el estado de la calzada. Se pueden tener señales de tráfico que, además de informar visualmente sobre una posible prohibición o peligro, pudieran mandar mensajes a los vehículos que circulan por su alrededor. En tal sentido, si se sabe que la carretera está deslizante en un tramo determinado, se puede advertir a los que se vayan acercando al lugar de peligro y el aviso puede propagarse de la misma manera que el ejemplo de la Figura 1.4 para poner en alerta a usuarios que se encuentren todavía alejados del punto conflictivo. De esta manera, la conducción podría llegar a ser más segura y se podrían evitar múltiples accidentes.

1.4 Crecimiento de las comunicaciones móviles

Según un reciente informe [10] realizado por Cisco, una de las empresas líderes en el ámbito de las telecomunicaciones, el tráfico global de redes móviles se multiplicará por 26 entre 2010 y 2015. De hecho, el crecimiento del tráfico de datos móviles a nivel mundial seguirá un ritmo exponencial. La Figura 1.5 muestra el crecimiento del tráfico de datos móviles mientras que en la imagen de debajo, Figura 1.6, el crecimiento en el tráfico según el tipo de dispositivo. En todos los casos la tendencia es de forma ascendente a un ritmo muy alto.

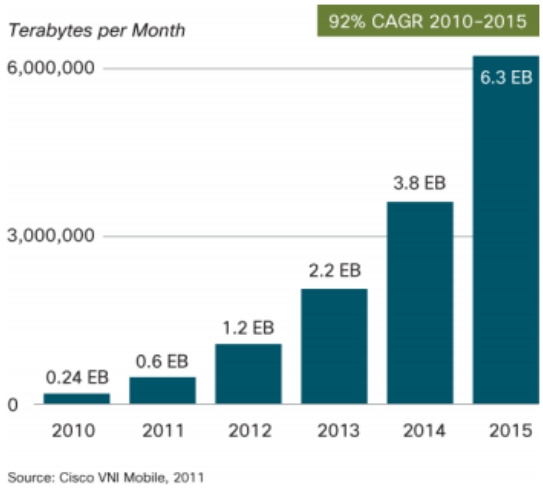

Figura 1.5: *Crecimiento comunicaciones móviles 2010-2015*

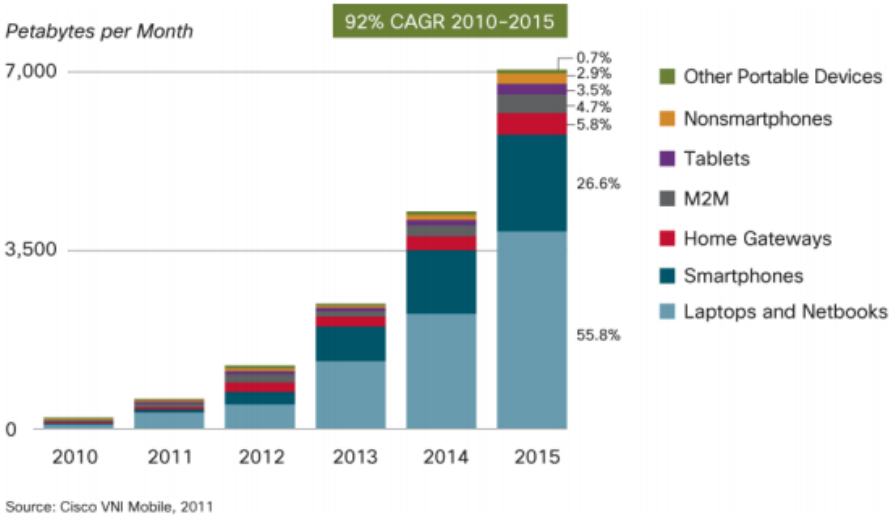

Figura 1.6: *Crecimiento por tipo de dispositivos 2010-2015*

Algunos datos de especial relevancia en este estudio son los siguientes:

- En 2010, el tráfico mundial de datos móviles superó en tres veces el tamaño de todo el tráfico global de Internet (fijo y móvil) generado en el año 2000.

- En 2015 habrá más de 5600 millones de dispositivos personales conectándose a redes móviles y 1500 millones de conexiones entre máquinas. Todos en conjunto, equivalen a una conexión móvil a la red por cada habitante del mundo.

- El tráfico móvil que se realizará desde dispositivos *tablets* se multiplicará por 205 entre 2010 y 2015.

- Para 2015, los dispositivos *tablets* generarán por sí solos más tráfico que la suma total de datos que han cruzado la Red móvil durante 2010. Serán responsables de un tráfico

de 248 petabytes mensuales en 2015, frente a los 237 petabytes mensuales generados en 2010 por todos los dispositivos de acceso.

Se han realizado y se realizan numerosos estudios sobre el auge de las comunicaciones inalámbricas en los últimos años para mostrar la penetración de esta forma de comunicación en nuestras vidas. Todos ellos reflejan una tendencia ascendente tanto en el número de dispositivos como en el tráfico que se va a producir en un futuro en las comunicaciones móviles.

Analizando estos datos, parece claro que existe una gran oportunidad de negocio para aquellas empresas u organizaciones que busquen realizar protocolos o aplicaciones destinados a los dispositivos de comunicaciones móviles emergentes. La razón es que tales elementos tecnológicos, como *smartphones* o *tablets*, poseen gran capacidad de procesamiento y de conectividad. Como se pueden utilizar para crear redes MANETs, son excelentes impulsores de esta forma de comunicación.

1.5 Estructura del documento

Este trabajo se estructura en nueve capítulos y un apéndice. En el actual, se hace una introducción general explicando las motivaciones para realizar este proyecto.

Durante el capítulo dos se hace una revisión de los distintos protocolos y soluciones que existen para conseguir el objetivo final, implementar una red MANET.

Ya en el capítulo tres, se explicará el protocolo elegido (AODV) y se expondrán ejemplos para su comprensión.

Durante el capítulo cuatro se presentarán los requisitos propios del programa a realizar, la planificación en cuanto a tiempo de la duración del proyecto y un análisis de los costes.

En el capítulo cinco se explicarán las características propias de la plataforma en la que se ha implementado el trabajo (Bada) y se introducirán aspectos a tener en cuenta a la hora de programar en dicho sistema operativo.

Durante los capítulos seis y siete se explicará la forma de diseñar e implementar el protocolo y dos aplicaciones creadas (aplicación de chat y aplicación de eco) para mostrar el correcto funcionamiento de AODV.

En el capítulo ocho se mostrarán las pruebas que se han realizado y los resultados obtenidos poniendo especial atención a la diferencia existente entre la comunicación de dos dispositivos que tienen visión directa y la comunicación de estos dos nodos cuando necesitan un elemento intermedio.

Por último, en el capítulo nueve se explicarán las conclusiones extraídas de la realización del proyecto y una serie de líneas de investigación futuras interesantes para mejorar la red MANET implementada.

2 Estado del arte

El capítulo 2 del presente trabajo se centra en la recopilación de información acerca de los diferentes protocolos y métodos para abordar el problema del encaminamiento en una red MANET. Se tratará de explicar los tres grandes grupos en los que podemos dividir las soluciones encontradas y, por último, se explicará cuál es la más conveniente según la topología.

2.1 Tipos de protocolos

En el ámbito de los tipos de protocolos existentes para crear redes MANETs se pueden diferenciar tres grandes grupos: protocolos proactivos, reactivos e híbridos. En el primer grupo, los nodos mantienen información de encaminamiento hacia todos los dispositivos de la red. En el segundo grupo, los nodos actualizan las tablas de encaminamiento solamente en caso de necesidad. Por último, el tercer grupo intenta incorporar las ventajas de los protocolos proactivos y reactivos.

2.2 Proactivos

Dentro de este grupo se encuadran aquellos protocolos que basan su funcionamiento en el intercambio periódico de mensajes para actualizar sus tablas de rutas. La característica principal es que, aunque no haya movilidad entre los dispositivos que componen la red o no cambie la topología, siempre va a haber un número mínimo de mensajes mandados para mantener la conexión y tener constancia de los nodos que forman la red.

A continuación se describen algunos ejemplos significativos de estos protocolos proactivos: DSDV, WRP y OLSR.

2.2.1 DSDV

DSDV (***D**estination **S**equenced **D**istance-**V**ector*) [11] basa su funcionamiento en la actualización de una tabla de rutas donde aparece cada destino con su distancia (número de saltos), su número de secuencia que indica cuál es la información más actualizada, y el siguiente

salto hacia el nodo final. Debido a que está dentro del grupo de proactivos, tiene siempre constancia de todos los dispositivos de la topología.

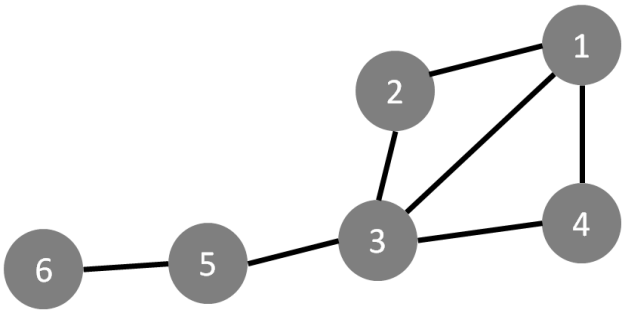

Figura 2.1: *Ejemplo de disposición de nodos para DSDV*

En la Figura 2.1 se muestra una topología de red que servirá para mostrar en la Tabla 2.1 las rutas de un nodo en concreto, en este caso el 1.

Destino	Siguiente Nodo	Distancia	Número de Secuencia
2	2	1	154
3	3	1	123
4	4	1	145
5	3	2	123
6	3	3	147

Tabla 2.1: *Tabla de encaminamiento del nodo 1*

En relación a los tipos de mensajes mandados, se utilizan dos tipos de actualizaciones. La primera se produce cuando no existen cambios por motivos de movilidad o rotura de algún enlace. Los mensajes mandados en este caso están destinados a informar sobre las rutas que posee cada nodo. En la segunda de las actualizaciones, que se produce si se pierde algún enlace, el dispositivo que se dé cuenta del suceso marca la distancia del destino como infinito (∞). Seguidamente, se va difundiendo la notificación por toda la red con el fin de volver a actualizar la relación de rutas alcanzables en cada nodo.

2.2.2 WRP

WRP (***W****ireless* ***R****outing* ***P****rotocol*) [12] se encuadra también dentro del grupo de los protocolos proactivos, pero se diferencia de DSDV en la forma de mantener las rutas y en los procedimientos de actualización. Mientras que DSDV solamente contiene una tabla, WRP mantiene varias con el fin de tener información más precisa. Las tablas que utiliza son: Tabla de distancias (DT), tabla de rutas (RT), tabla de coste de un enlace (LCT) y una lista de mensajes de retransmisión (MRL).

- Tabla de distancias (DT): Contiene la información de los vecinos de un determinado nodo.

- Tabla de rutas (RT): Contiene la información que tiene cada nodo para un destino determinado. Se guarda la distancia más corta, el nodo predecesor (penúltimo), el siguiente nodo y un indicador mostrando el estado del enlace.

- Coste (LCT): Contiene el coste que tiene el llegar hacia un destino (puede ser debido a diferentes métricas, como número de saltos o estado del enlace).

- Lista de mensajes de retransmisión (MRL): Se crea una entrada para cada mensaje de actualización que llegue de un vecino y tenga que ser retransmitido. Cada vez que se retransmite un mensaje, se reduce el contador asociado a la entrada correspondiente. Si ese contador llega a cero porque no se reciben actualizaciones, se ha perdido la comunicación con el dispositivo correspondiente a esa entrada.

Para clarificar la actualización de las tablas de rutas en WRP, se ha utilizado el ejemplo que se muestra en la Figura 2.2.

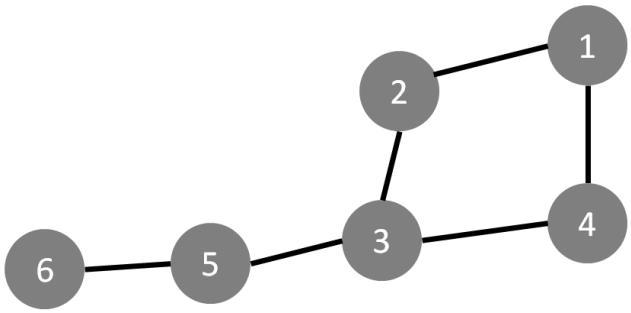

Figura 2.2: *Ejemplo de disposición de nodos para WRP*

La tabla de rutas para llegar al nodo 1 contenida en los demás dispositivos, corresponde ahora a la Tabla 2.2.

Nodo	Siguiente Nodo	Predecesor	Coste
2	1	2	-
3	1	2	-
4	1	4	-
5	3	2	-
6	5	2	-

Tabla 2.2: *Tabla de rutas hacia el nodo 1*

No se ha mostrado el coste porque éste depende de distintos factores. Nótese que ahora se ha especificado lo que contiene cada uno de los restantes nodos para llegar al 1, no la tabla de rutas de un elemento concreto.

Como se aprecia, el dispositivo número 1 tiene dos predecesores, el 2 y el 4. Dado que el protocolo es proactivo, los demás tienen información de que al nodo 1 se puede llegar por medio de 2 y de 4. Los nodos 3, 5 y 6 mandarían a través de 2 que es su predecesor, pero si éste fallara podrían conmutar al 4, puesto que conocen de su existencia y les llevaría igualmente al 1.

En consecuencia, la gran ventaja que tenemos ahora es que se conoce cuál es el nodo predecesor al destino final. De esta manera, si el predecesor se cayera, podríamos encaminar por otro predecesor de manera rápida, porque se tiene información de los dispositivos que rodean al destino. El resultado es que las rutas se pueden volver a obtener más rápidamente, pero con el coste de tener que usar más memoria y mensajes de actualización periódicos.

2.2.3 OLSR

OLSR (***O**ptimized **L**ink **S**tate **R**outing*) [13] [14] es otro protocolo dentro del grupo de los proactivos que se caracteriza por intentar optimizar el número de mensajes de control que se producen en la red.

El hecho de que todos los dispositivos estén continuamente retransmitiendo mensajes de control para informar sobre la topología es un proceso costoso desde el punto de vista de la señalización implicada. OLSR intenta solucionar este problema eligiendo estratégicamente un conjunto de nodos (MPRs, *MultiPoint Relays*) mediante los cuales se puede llegar a cualquier punto de la malla. Para clarificar este tipo de funcionamiento, en la Figura 2.3 se puede ver a) cómo sería el mecanismo de inundación de mensajes para informar sobre las tablas de rutas que posee cada nodo y b) cómo sería la elección de un número específico de MPRs que permitan llegar a los demás. La diferencia es notable. Mientras que en el primer caso todos los dispositivos retransmiten los mensajes de control, en el segundo se reduce el número de mensajes transmitidos ya que hay algunos nodos que no son MPRs.

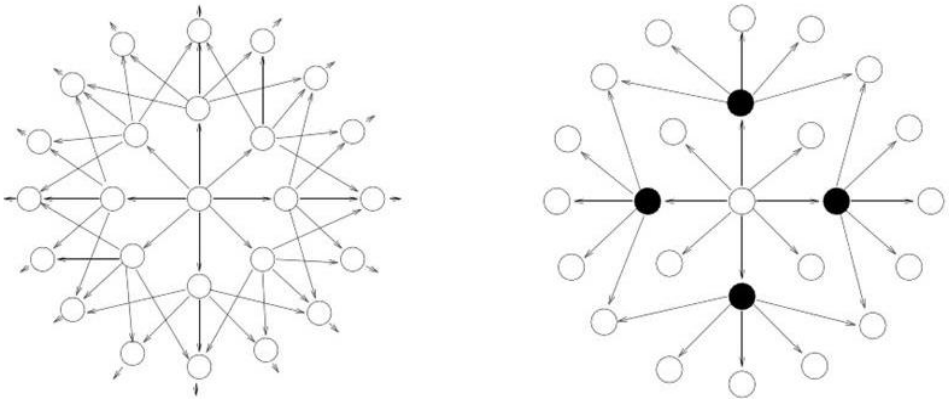

Figura 2.3: *Disposición de MPRs*

En este protocolo se utilizan dos mensajes de control para informar sobre el estado de cada nodo.

- Mensajes HELLO: Se mandan únicamente a los dispositivos situados a un salto. Informan sobre los vecinos del emisor del HELLO, es decir, los nodos a los que es capaz de llegar mediante un salto y también sobre los MPRs elegidos. De esta manera, el nodo que reciba un mensaje HELLO puede saber si ha sido elegido como MPR por parte de un vecino.

- Mensajes TC (*Topology Control*): Al contrario que los anteriores, que solamente se mandan a los destinos situados a un salto, éstos tienen como objetivo difundir el estado por toda la red. El mecanismo de difusión se hace teniendo en cuenta cuáles son los MPRs seleccionados para permitir una mayor eficiencia en los mensajes de control. Cada nodo incorpora en la notificación TC los vecinos que dispone en su tabla de rutas para que por medio de los MPRs se difundan de manera controlada.

2.3 Reactivos

Dentro de este grupo, se encuadran aquellos protocolos que basan su funcionamiento en la actualización de las rutas solamente cuando sean necesarias. La característica principal es que, si hay movilidad o cambio en la topología, los dispositivos no actualizan sus tablas de encaminamiento a menos que lo requieran. De hecho, no se conoce cómo es la red completa porque no se sabe cuáles son las tablas de los demás nodos, por lo que únicamente hay disponible información sobre las rutas activas.

Algunos ejemplos representativos son los protocolos AODV, DSR y ABR, que se explican a continuación.

2.3.1 AODV

AODV (***A**d-hoc **O**n-demand **D**istance **V**ector*) es uno de los que se encuadran dentro de los reactivos. Por lo tanto, no utiliza mensajes de control hasta que un nodo determinado no necesita una ruta hacia un destino. En el capítulo 3 se explica más detalladamente su funcionamiento debido a que ha sido el elegido para llevar a cabo la implementación de la red MANET.

Para crear y actualizar las entradas en las tablas de encaminamiento de cada dispositivo, este protocolo se vale de tres tipos de mensajes principales: mensaje de establecimiento de rutas o *Route Request* (RREQ), mensaje de respuesta o *Route Reply* (RREP) y mensaje de error o *Route Error* (RERR). Cuando un nodo quiere establecer una comunicación con otro dispositivo, difunde un mensaje RREQ a la dirección *broadcast* 255.255.255.255. La respuesta a este tipo de mensaje es el RREP que servirá para crear el camino entre los dos extremos. La peculiaridad de este último es que se envía directamente al siguiente salto que permite llegar hasta el dispositivo que inició el descubrimiento de rutas, a diferencia del RREQ que era *broadcast*. De esta manera, el RREP se encamina hasta que llega al origen. Por último, el RERR sirve para notificar de posibles caídas de enlaces o pérdidas de rutas.

Además de estos tres tipos, se utilizan mensajes HELLO para notificar sobre la disponibilidad de los diferentes dispositivos que forman parte de una ruta activa. Únicamente los nodos con al menos una ruta activa pueden enviar mensajes HELLO.

Por último, la métrica que se utiliza para decidir entre varias rutas que puedan llevar a un mismo destino es la cuenta de saltos entre los dos dispositivos extremos de la comunicación.

2.3.2 DSR

DSR (***D**ynamic **S**ource **R**outin*g) [15] [16] está diseñado para restringir el ancho de banda consumido por paquetes de control eliminando las actualizaciones periódicas. El mayor logro es que no se necesitan mensajes HELLO para mantener la conectividad local. Para obtener información, se aprovecha de los mensajes de datos mandados por el camino que se está creando. De este modo, lo que hace es extraer de estos mensajes toda la información relativa a los dispositivos origen y final de la comunicación. Además, al tratarse de un protocolo basado en *Source Routing*, en cada salto se va incorporando al paquete información de todos los dispositivos por los que se va pasando.

En relación a los mensajes utilizados para realizar el establecimiento de las rutas, se pueden distinguir tres tipos: mensaje de establecimiento de rutas o *Route Request* (RREQ), mensaje de respuesta o *Route Reply* (RREP) y mensaje de error o *Route Error* (RERR).

La forma de establecer las rutas y los tipos de mensajes mandados son similares a los de AODV. Las mayores diferencias entre los mismos residen en la forma de mantener la conectividad local y en el hecho de que DSR va incorporando en las cabeceras de los mensajes la dirección de cada nodo por el que se pasa.

2.3.3 ABR

ABR (***A**ssociativity-**B**ased **R**outing*) [17] basa su funcionamiento en la estabilidad entre los dispositivos. Cada nodo clasifica como estable o inestable los enlaces que tenga a su disposición en función de las pérdidas de mensajes HELLO que se produzcan.

Sobre la base de lo anteriormente planteado, cuando un dispositivo requiere una ruta hacia un destino, manda un mensaje de petición de rutas RREQ. Una vez recibido, cada nodo intermedio retransmite el mensaje incorporando información acerca de sus enlaces. El destino final espera durante un periodo de tiempo delimitado para recibir varias de estas peticiones, las analiza viendo cuál ha transcurrido por nodos más estables y escoge la que vaya a proporcionar una comunicación más fiable. En el caso en que dos rutas sean igual de seguras, se elige el que tenga menor número de saltos. Si también coincidieran en el número de saltos, se utiliza un mecanismo aleatorio para dilucidar el camino final.

En este sentido, en la Figura 2.4 se muestra un ejemplo de topología que permite comprender el funcionamiento de ABR.

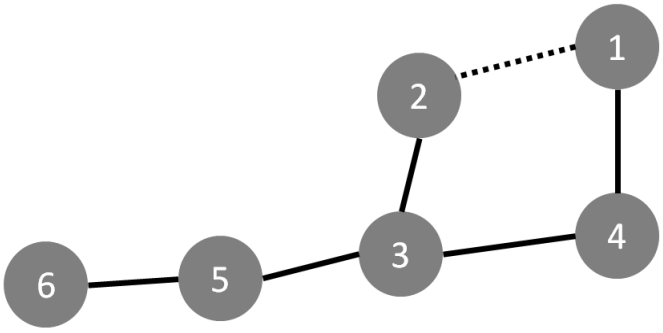

Figura 2.4: *Ejemplo de protocolo ABR*

En este contexto, si el nodo 6 quisiese establecer una comunicación con el nodo 1, mandaría un RREQ que se propagaría a través de 5 y de 3. A partir del dispositivo número 3 el RREQ seguiría dos caminos, uno por 2 y el otro por 4. Cuando ambos RREQs lleguen a 1, éste comprueba por dónde ha pasado el mensaje y se da cuenta que el que ha seguido el camino superior (a través del nodo 2) ha transcurrido por un nodo menos fiable, por lo que la comunicación final se hará mediante el camino inferior (nodo 4).

2.4 Híbridos

Dentro de este apartado se encuadran aquellos protocolos que basan su funcionamiento en una mezcla de los dos tipos explicados en las secciones 2.2 y 2.3 (proactivos y reactivos).

En este sentido, intentan incorporar lo mejor de los dos tipos. En general, lo que se pretende es dividir la red en grupos en los que de forma local se utilice un protocolo proactivo, mientras que para conocer las rutas hacia los demás grupos se utilice uno reactivo. De esta manera, localmente se tendrá constancia de todos los nodos que forman el grupo en el que se encuentra el dispositivo. Debido a este hecho, no se tendrá que inundar la totalidad de la topología con mensajes de control para conocer los caminos hacia las demás comunidades.

A continuación se describe el protocolo ZRP como un ejemplo de este tipo de protocolos.

2.4.1 ZRP

ZRP (***Z**one **R**outing **P**rotocol*) [18] es un ejemplo de protocolo híbrido. Como tal, utiliza algún tipo de encaminamiento proactivo en una zona delimitada y para el exterior, uno reactivo. Este protocolo está pensado para redes grandes con una movilidad elevada.

De hecho, su principal logro es que la zona delimitada no es un elemento estático, sino que va modificándose según las condiciones de movilidad y tráfico. Los nodos, por sí solos, van eligiendo cuáles son los dispositivos periféricos que permiten llegar a los demás nodos fuera de la comunidad. Para ello, el radio (número de saltos máximo dentro de una zona) se configura automáticamente.

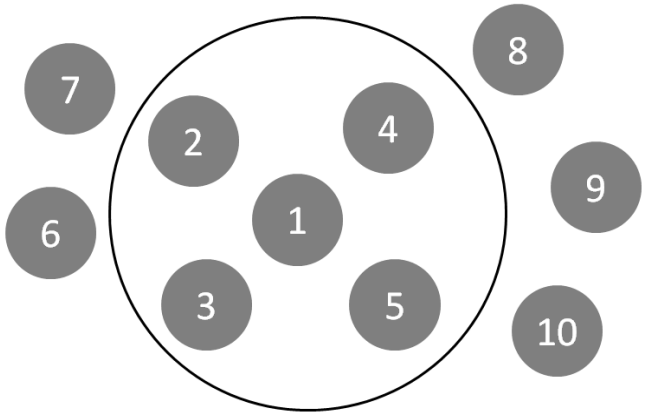

Figura 2.5: *Ejemplo ZRP*

En la Figura 2.5 se puede apreciar como los nodos 1, 2, 3, 4 y 5 pertenecen a la misma comunidad. La distancia desde el nodo central (1) hasta los periféricos es un salto. Los dispositivos que funcionan como frontera de la zona son 2, 3, 4 y 5. Éstos últimos son los que permitirán llegar hacia el resto que se encuentran fuera de la zona por medio de un protocolo reactivo. Por el contrario, en el interior se utilizará un protocolo proactivo para el establecimiento y conocimiento de las rutas.

2.5 Comparación

La Tabla 2.3 muestra una comparativa de las principales características de los protocolos reactivos y proactivos [19].

Características	Proactivos	Reactivos
Estructura	Plana y en algunos casos jerárquica (OLSR).	En la mayoría de los casos plana, excepto en CBRP (*Cluster Based Routing Protocol*) [20].
Disponibilidad de las rutas	Siempre disponibles.	Se determinan cuando se necesiten.
Volumen de tráfico de control	Alto.	Menor que los protocolos proactivos.
Actualizaciones periódicas	Se utilizan. Algunos protocolos tienen mecanismos para usar las actualizaciones en función de la necesidad (OLSR).	No se utilizan. Únicamente para comprobar la disponibilidad de los nodos vecinos.
Sobrecarga por datos de control	Alta.	Baja.
Tiempo en obtener una ruta	Bajo.	Alto.
Requerimientos de almacenamiento	Altos.	Depende del número de rutas que se mantengan. Menor que los protocolos proactivos.
Ancho de banda necesario	Alto.	Bajo.
Energía consumida	Alta.	Baja.
Nivel de retardo	Bajo si las rutas están determinadas.	Mayor que en los proactivos.
Problemas de escalabilidad	Por encima de los 100 nodos.	Por encima de varios cientos de nodos.

Tabla 2.3: *Comparación entre proactivos y reactivos*

Según esta tabla, los protocolos proactivos mantienen las rutas hacia todos los dispositivos de la red. De esta manera, si se quisiese iniciar una comunicación con cualquiera, las direcciones estarían disponibles y el proceso sería más rápido. Sin embargo, los protocolos reactivos no tienen conocimiento de los demás dispositivos. Por ello, cuando quieren establecer una ruta tienen que enviar mensajes de control y las respuestas a éstos son las que le van a proporcionar un camino válido. De esta forma, en general, el retardo será superior al de los protocolos proactivos.

De acuerdo con el ancho de banda consumido por los mensajes de control, queda claro que los protocolos proactivos tienen peor eficiencia ya que envían continuamente mensajes periódicos que aumentan la carga en la red. Los reactivos en cambio, no envían mensajes si no necesitan establecer un camino.

De igual forma, los reactivos son más eficientes en cuanto a la memoria consumida. Los proactivos tienen un conocimiento más amplio de la red, pero a cambio tienen que consumir más memoria. Como tienen que mandar un número mayor de mensajes de control y las tablas de encaminamiento serán más costosas, el consumo energético también será mayor, factor a tener en cuenta en dispositivos móviles en los que la duración de la batería es un aspecto importante.

En base a estas características, se ha optado por utilizar para nuestro escenario un protocolo reactivo, ya que el trabajo está pensado para su uso en teléfonos móviles en los que la duración de la batería y las capacidades de procesamiento son limitadas.

Dentro de los protocolos reactivos, DSR y AODV se utilizan para comparar prestaciones con otros tipos de protocolos ya que ofrecen buenas prestaciones en cuanto al retardo en el establecimiento de las rutas, *troughput* y carga de la red debido a los mensajes de control. Dado que AODV utiliza únicamente mensajes periódicos cuando se tiene alguna ruta activa, no aumenta considerablemente el número de paquetes de control que emplea DSR. Este hecho lo hace más robusto en cuanto a posibles caídas de los enlaces, algo importante en el restablecimiento de las rutas. De esta manera, se ha elegido AODV como el protocolo utilizado para realizar la red MANET.

A continuación, durante el capítulo 3 se explicará con detalle su funcionamiento.

3 Protocolo AODV

Durante este capítulo se pretende explicar el funcionamiento de uno de los protocolos utilizados para construir una red MANET y que se ha elegido para la realización del presente trabajo. Se trata de AODV (***A****d-hoc* ***O****n-demand* ***D****istance* ***V****ector*) [21]. En adelante, se explicará el funcionamiento general y tres ejemplos de uso para su mejor entendimiento.

3.1 Introducción a AODV

AODV ha sido el protocolo elegido para desarrollar este trabajo ya que, según se ha argumentado en el apartado 2.5, las características de los mecanismos reactivos parecen más idóneas para nuestro escenario. Además, AODV ofrece mejores resultados en cuanto a *throughput* y retardo que otros protocolos (DSR, OLSR, OSPFv2 y ZRP) [22].

En dicho protocolo, la prioridad a la hora de elegir una ruta se realiza por medio de la cuenta de saltos hacia el destino. Como se ha mencionado anteriormente, se encuadra dentro de los protocolos reactivos, lo que significa que no actualiza el contenido de las tablas de rutas hasta que no es estrictamente necesario. De esta forma, reduce el número de notificaciones mandadas en el caso en el que no hubiese movilidad.

Con respecto a la variedad de mensajes intercambiados, se pueden diferenciar tres tipos distintos: RREQ, RREP y RERR. El primero está destinado a la búsqueda de rutas, el segundo es una respuesta a un mensaje RREQ y el tercero es el mensaje de error que notificará la caída de algún enlace.

En relación a su transporte, AODV utiliza UDP y los puertos origen y destino 654.

3.2 Tipos de mensajes AODV

Para la explicación del protocolo se van a mostrar los tres tipos de mensajes básicos necesarios para hacer posible las distintas funcionalidades que tiene AODV.

3.2.1 Mensaje RREQ

El mensaje RREQ (***R**oute **REQ**uest*) es el encargado de solicitar una ruta cuando un nodo necesite establecer una comunicación con otro dispositivo. En la Figura 3.1 se muestran los campos de un mensaje RREQ.

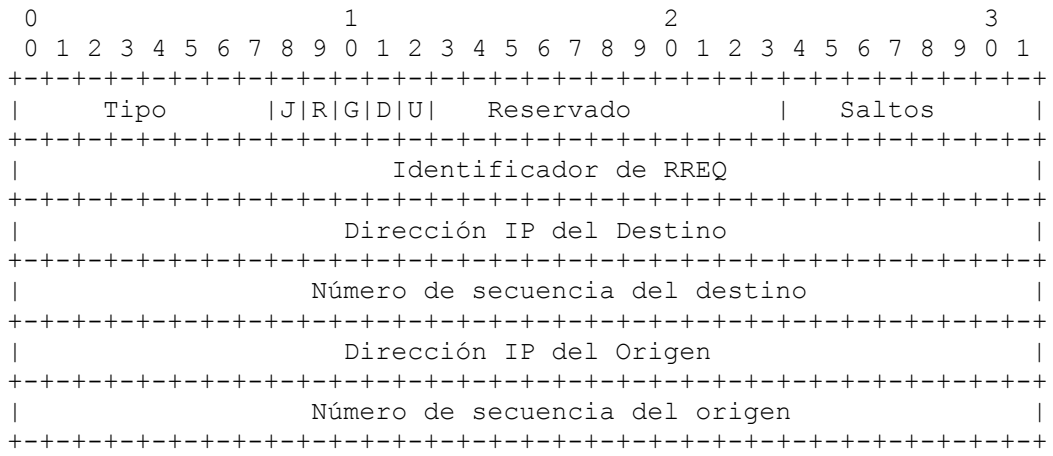

Figura 3.1: *Campos de un mensaje RREQ*

- Tipo: Para los RREQs vale 1. Se utiliza para identificar la clase de mensaje AODV y para diferenciar entre RREQ, RREP y RERR.

- Bits indicadores: Los dos primeros (J y R) están destinados a escenarios *multicast*, el tercero (G) se utiliza para solicitar un mensaje RREP, el cuarto (D) para indicar que responda el destino y el último (U) indica que el número de secuencia no se conoce.

- Saltos: Número de saltos que hay hacia el destino. El origen pondrá este campo con el valor 0 y cada salto que vaya propagando un mensaje RREQ deberá incrementarlo en una unidad.

- Identificador de RREQ (RREQ_ID): Está destinado a evitar bucles en la búsqueda de rutas, ya que si un nodo ha enviado un RREQ determinado, llevará asociado un RREQ_ID para que no haya retransmisiones si recibiese el mismo RREQ por parte de otro dispositivo.

- Dirección IP del destino: Dirección IP para la cual se está solicitando una ruta.

- Número de secuencia del destino: Número administrado por cada nodo destinado a tener siempre rutas más actualizadas.

- Dirección IP del origen: Dirección IP del primer emisor del RREQ.

- Número de secuencia del origen: Número de secuencia propio del origen.

Una vez que se solicita una ruta hacia un elemento determinado de la red, se espera obtener una respuesta que permita saber si es posible tener una conexión activa con dicho

dispositivo. Por ello, en el siguiente apartado se van a explicar las características de la notificación que permite responder a un RREQ.

3.2.2 Mensaje RREP

El paquete RREP (***R**oute **REP**ly*) es la respuesta a un RREQ. De esta manera, el RREQ será el encargado de solicitar una ruta y el RREP responderá siguiendo el camino inverso. En particular, los campos de un mensaje RREP se muestran en la Figura 3.2.

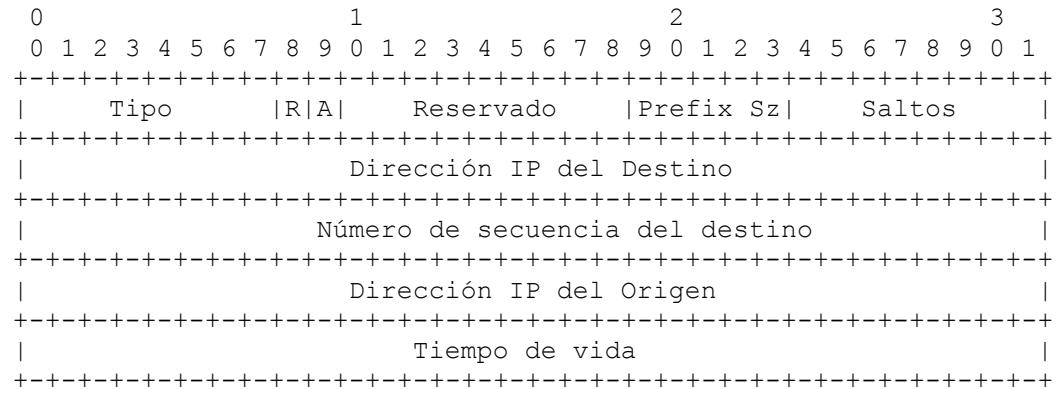

Figura 3.2: *Mensaje RREP*

- Tipo: Para los mensajes RREP vale 2. Se utiliza para identificar la clase de mensaje AODV y para diferenciar entre RREQ, RREP y RERR.

- Bits indicadores: El primero (R) está destinado a escenarios *multicast* mientras que el segundo (A) solicita una confirmación.

- Saltos: Número de saltos que hay hacia el dispositivo final de la comunicación. El destino pondrá este campo con el valor 0 y cada salto que vaya propagando un mensaje RREP deberá incrementarlo en una unidad.

- Dirección IP del destino: Dirección IP del nodo que era el destino en el mensaje RREQ, es decir, el elemento para el que se estaba solicitando una ruta. No es el destino del mensaje RREP.

- Número de secuencia del destino: Número administrado por cada nodo destinado a tener siempre rutas actuales.

- Dirección IP del origen: Dirección IP del dispositivo que fue el origen del mensaje RREQ anterior, es decir, el nodo que estaba solicitando una ruta hacia el destino. En este caso, no se refiere al origen del mensaje RREP.

- Tiempo de vida: Tiempo de vida de la ruta que se ha solicitado.

Una vez que se tienen establecidas las tablas de encaminamiento, se debe tener en cuenta la posibilidad de que ocurran fallos en la topología, de ahí el siguiente apartado donde se explicará el tipo de notificación destinada a este hecho.

3.2.3 Mensaje RERR

El mensaje RERR (***R**oute **ERR**or*) está pensado para informar a los dispositivos vecinos de que existen enlaces rotos a los que no se puede llegar por medio del nodo que envía la notificación. Así, los campos de este mensaje se muestran en la Figura 3.3.

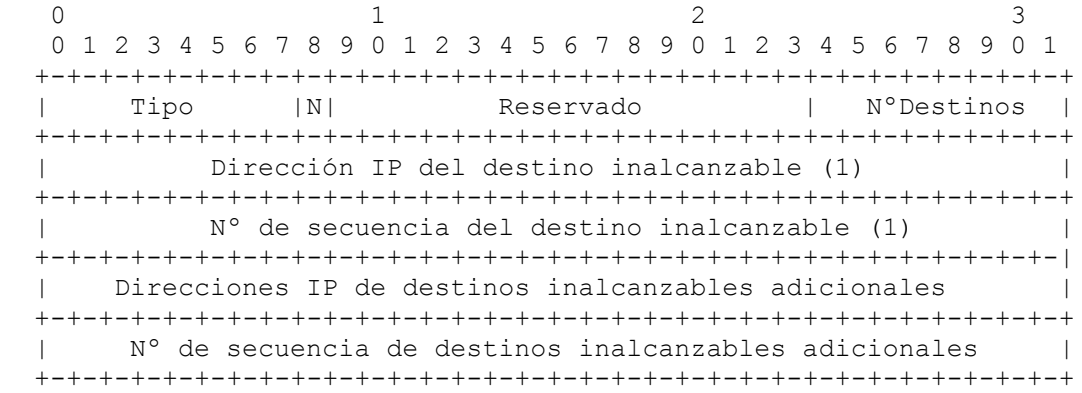

Figura 3.3: *Mensaje RERR*

En este caso la explicación es la siguiente:

- Tipo: Para los mensajes RERR vale 3. Se utiliza para identificar la clase de mensaje AODV y para diferenciar entre RREQ, RREP y RERR.

- N: Bit para indicar que no se borren las rutas.

- Nº Destinos: Número de destinos inalcanzables.

- Dirección IP del destino inalcanzable (1): Dirección IP para el nodo que se ha perdido la comunicación.

- Número de secuencia del destino inalcanzable (1): Número de secuencia del dispositivo para el que se ha perdido la comunicación.

- Direcciones IP de destinos inalcanzables adicionales: Se introducirán las direcciones que se hayan perdido una por una como se ha introducido la anterior. Por ello, se coloca primero la dirección y posteriormente el número de secuencia del destino perdido.

- Número de secuencia de destinos inalcanzables adicionales: Se introducirán los números de secuencia de los destinos inalcanzables adicionales al primero.

El número que se especifica en el campo *Nº Destinos* debe coincidir con el número de destinos introducidos en el paquete RERR, puesto que si no coinciden, el paquete estará mal formado y no se procesará de manera correcta. Por otra parte, éste es el único mensaje del que

en un principio, no se sabe su tamaño inicial. El número final de bytes dependerá del número de destinos inalcanzables que posea el nodo que envía el mensaje RERR.

3.3 Modo de operación AODV

Cuando un dispositivo necesita una ruta para una dirección y en su tabla de encaminamiento no posee entrada alguna para realizar la comunicación, debe enviar un mensaje tipo RREQ preguntando a los demás nodos de la red si alguno sabe llegar hasta el destino que se requiere.

Antes de mandar la petición de rutas, el origen guarda el RREQ_ID para evitar volver a procesar las retransmisiones del mismo RREQ que difunden los dispositivos vecinos. Adicionalmente, este campo se incrementa en una unidad para diferenciar entre peticiones diferentes. Respecto al número de secuencia del destino, se utiliza el último conocido para la dirección buscada y se copia en el campo correspondiente del RREQ. En cambio, el número de secuencia del origen es el propio del nodo que inicia el descubrimiento de rutas. Cada vez que se mande un mensaje de petición de rutas se aumenta el número de secuencia del origen y se copia el de destino en el RREQ. De esta manera, cada dispositivo intermedio podrá comparar estos campos con los que tenga almacenados en sus tablas de encaminamiento y actualizarlos a la versión más reciente (el número de secuencia mayor).

Ya en un dispositivo intermedio, cuando se recibe un RREQ se crea o actualiza una ruta al nodo previo. Además, se comprueba si se ha recibido un RREQ del mismo destino y con el mismo RREQ_ID. Si es así, el paquete se descarta. De no ser así, lo primero que se hace es incrementar la cuenta de saltos del paquete para que cuando llegue al destino, éste tenga conocimiento de cuántos nodos intermedios existen.

El mensaje RREQ se reenvía a la dirección *broadcast* 255.255.255.255 hasta llegar al nodo final. Éste, al ver en el campo *Dirección IP del destino* su propia dirección IP, deberá actuar enviando un RREP al nodo que inició el descubrimiento de ruta. El RREP deberá tener la cuenta de saltos iniciada a cero y se irá incrementando a medida que vaya atravesando la red hasta llegar al origen.

A diferencia de los RREQs que son *broadcast*, las respuestas son *unicast*. Cuando un RREP llega a un nodo, actualiza o crea una ruta hacia el dispositivo anterior del que ha recibido el RREP. Tras esto, incrementa la cuenta de saltos para que cuando llegue al origen, éste sepa el número de dispositivos que ha atravesado el mensaje. Si el número de secuencia es mayor o igual que el que tenemos almacenado y la cuenta de saltos es menor, se actualizan los datos y se envía el RREP al nodo siguiente que nos permite llegar hasta el origen del RREQ.

Cuando el RREP llega al nodo que solicitó la ruta mediante un RREQ, éste mira en su tabla si ya tiene una activa para el destino solicitado. Si la tuviese pero el número de saltos del mensaje RREP fuese menor, actualiza su tabla de rutas. Si la cuenta es mayor, no la actualiza. Por el contrario, si no se dispone de ninguna entrada ya almacenada, se guarda con la cuenta de saltos que nos indica el RREP recibido.

Una vez que se tiene una ruta activa para un destino, cada elemento de la comunicación envía mensajes HELLO a la dirección *broadcas*t 255.255.255.255 para que sus vecinos sepan

que esa ruta está disponible. Este mensaje HELLO es un mensaje RREP con el parámetro TTL (*Time To Live*) del paquete IP con valor 1.

Por otra parte, cuando se pierde la conectividad hacia el siguiente salto o expira el tiempo de vida de una entrada en la tabla de encaminamiento, el dispositivo que ha perdido la ruta deberá notificar a los demás para que dejen de enviar datos a través de él. Este proceso se realiza cuando se detecta que no hay conectividad, cuando se recibe un paquete de datos para el que no se sabe encaminar o cuando se obtiene un RERR de otro dispositivo. En este último caso, si se recibe un mensaje de datos para el que no se tiene una entrada, el RERR será *unicast* directamente destinado al dispositivo que envió el paquete. En cambio, si se detecta que se ha perdido un vecino, se enviará un RERR *broadcast* para notificar al mayor número de elementos y que se actualicen rápidamente las nuevas tablas de encaminamiento.

Cuando el origen de la comunicación recibe el RERR, decide si es necesario comenzar una nueva búsqueda. En este sentido, si tiene que seguir manteniendo una comunicación con el destino, iniciará un nuevo proceso de descubrimiento de rutas mandando un RREQ que tendrá que ser contestado por medio de un RREP adecuado.

3.4 Escenarios de aplicación AODV

Para facilitar la comprensión de AODV, se va a proceder a explicar a través de diferentes escenarios la operación del protocolo.

3.4.1 Descubrimiento de rutas

El primer escenario consiste en un descubrimiento de rutas entre dos dispositivos pertenecientes a la Figura 3.4.

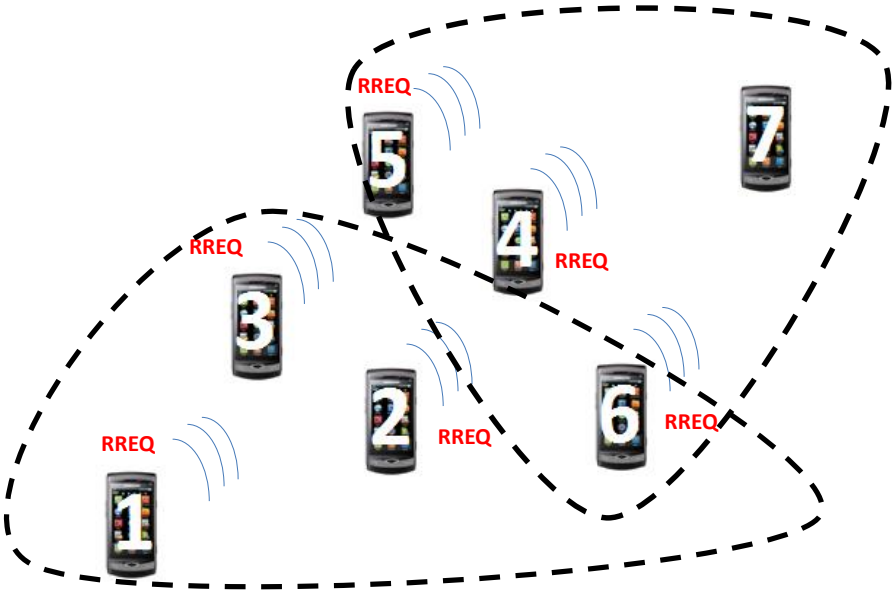

Figura 3.4: *Escenario de descubrimiento de rutas*

De acuerdo con la figura presentada, se va a llamar a los móviles por el número colocado en sus pantallas. En el escenario planteado, el móvil 1 quiere establecer una ruta hacia el móvil 7. El primero está en el rango de cobertura de los móviles 2, 3 y 6 mientras que el segundo tiene visión directa con 4, 5 y 6.

1. El móvil 1 envía un RREQ *broadcast* con la dirección de destino número 7.

2. El mensaje RREQ lo reciben tanto 2, 3 como 6.

3. Los móviles que reciben el RREQ actualizan o crean una entrada en sus tablas de rutas hacia el nodo 1, que fue el que les envió el mensaje.

4. Los móviles 2, 3 y 6 incrementan la cuenta de saltos del RREQ en uno y vuelven a difundir el RREQ actualizado.

5. El RREQ enviado por 2 y 3 les llega a 4 y 5, que realizan la misma operación que realizaron 2 y 3 cuando recibieron el RREQ de 1. Establecen una ruta hacia el nodo anterior y vuelven a mandar el RREQ.

6. El RREQ que envía 6, le llega directamente a 7 que, mirando la IP destino que trae el mensaje, se da cuenta de que es el nodo para el que se está solicitando una ruta y responderá de manera *unicast* con un RREP a 6.

7. Cada mensaje RREQ generado por los nodos intermedios les llega a los demás que están en su rango de cobertura pero, como llevan el mismo RREQ_ID (identificador de RREQ) que el que ellos han difundido, no lo procesan y lo descartan.

8. A 7 llegan también los RREQs provenientes de 4 y 5, pero no les contesta porque ve que ya ha contestado a un RREQ para el mismo origen y con una cuenta de saltos menor. Conviene recordar que la cuenta de saltos que le llegará a 7 en este caso será de 2 y la que llegó por medio del nodo 6 era de uno.

9. El nodo 6 recibe el RREP y establece una ruta hacia el nodo 7 que es el nodo del que ha recibido el RREP. Como cuando le llegó el RREQ desde 1, creó una entrada en su tabla de encaminamiento hacia dicho nodo, mira en el RREP cuál fue el origen del RREQ y le reenvía el mensaje aumentando la cuenta de saltos en una unidad.

10. Finalmente el RREP llega hasta el nodo 1 que ya tiene la ruta que había solicitado hacia el nodo 7. El siguiente salto será el móvil 6.

11. Una vez que se establecen las rutas, los nodos se mandan HELLOs periódicos para comprobar la conectividad local. En este momento, se puede decir que la ruta que se solicitó se ha establecido correctamente.

3.4.2 Fallo en uno de los dispositivos

En este caso el escenario consiste en la pérdida de uno de los nodos. Se parte de la situación que se ha producido anteriormente, es decir, 1 y 7 están conectados mediante 6.

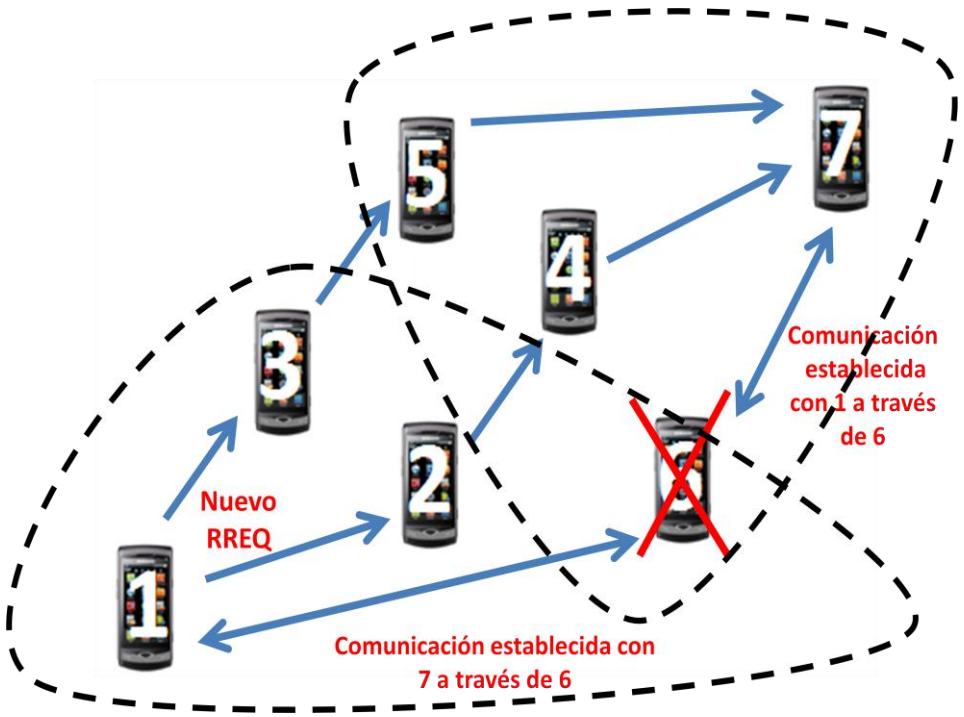

Figura 3.5: *Escenario de pérdida de rutas*

El móvil 6 sufre un fallo y se pierde la comunicación entre el móvil 1 y 7 por lo que ocurre lo siguiente:

1. El nodo 1 y el nodo 7 se dan cuenta de que el móvil 6 está inoperativo debido a la pérdida de mensajes HELLO y mandan un RERR indicando que el móvil 6 es el destino inalcanzable.

2. El móvil 1 necesita un camino para seguir comunicándose con el móvil 7. Por ello, inicia un nuevo descubrimiento de rutas con un RREQ que llega hasta 2 y 3.

3. Los móviles 2 y 3 propagarán el RREQ como se explicó en el primer ejemplo hasta llegar al 7, que responderá con un RREP.

4. Cuando llegue la respuesta al móvil 1, ya tendrá una nueva vía para poder comunicarse con 7.

5. Como las dos rutas van a tener la misma cuenta de saltos, dependerá de qué mensaje llegue primero para escoger una u otra.

3.4.3 Datos a dispositivo incorrecto

En el anterior ejemplo se ha mostrado el mensaje de error que se manda en el caso de que falle un móvil. Ahora se pretende mostrar qué pasaría si algún dispositivo recibe un mensaje de datos hacia un destino para el que no se tiene ninguna entrada en la tabla de encaminamiento.

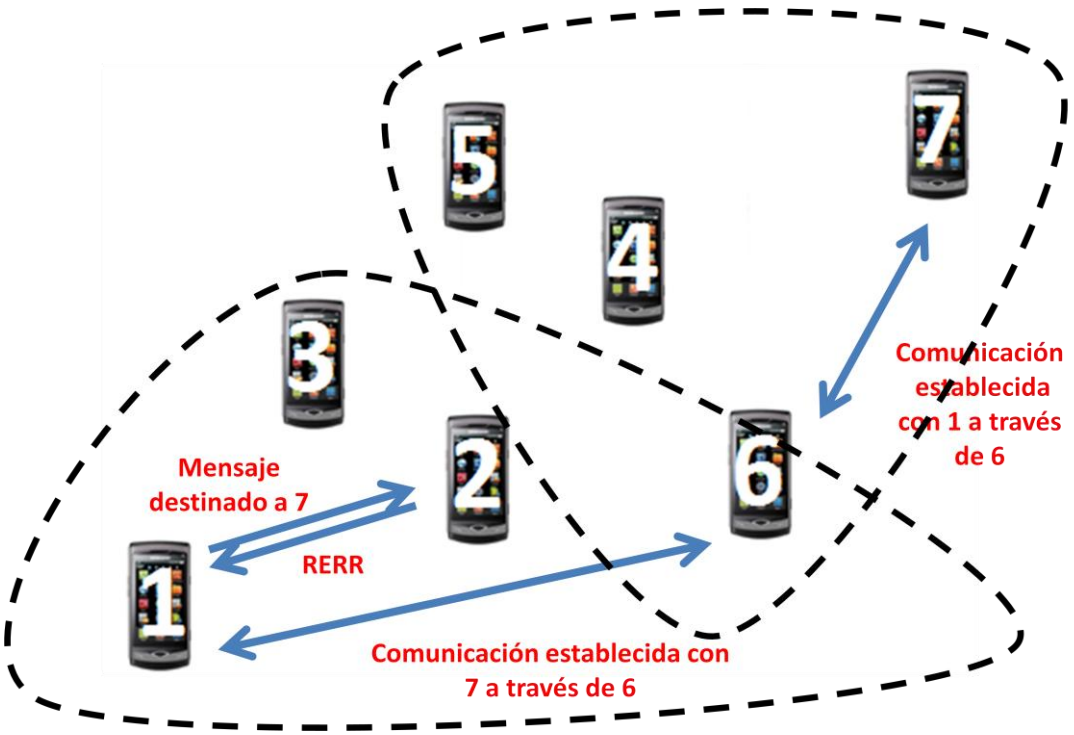

Figura 3.6: *Escenario de ruta errónea*

Se tiene establecida una comunicación entre el móvil 1 y 7 por medio de 6 y ocurre lo siguiente:

1. El móvil 1, por error, manda un mensaje de datos destinado al móvil 7 a través de 2.

2. El móvil 2, al procesarlo, ve que no tiene un camino activo hacia 7, por lo que no puede reenviar el mensaje.

3. El móvil 2 envía un mensaje RERR *unicast* al móvil 1 indicándole que el número 7 es un destino inalcanzable para él.

4. El móvil 1 recibe el RERR por parte de 2 y borra la entrada que, por error, tenía almacenada para llegar al nodo 7.

5. Como ya tiene una conexión establecida por medio de 6, no necesita enviar un nuevo RREQ para solicitar una nueva ruta.

4 Requisitos del sistema y costes de la aplicación

Durante el capítulo 4 se explicarán las distintas funcionalidades que debe tener la aplicación, los recursos humanos y técnicos utilizados, así como un estudio del coste y tiempo empleados.

4.1 Requisitos funcionales

Para la especificación de los requisitos funcionales se han utilizado un conjunto de diagramas *sysML*. De este modo, se ha divido el sistema en paquetes de funciones que se van desgranando desde el caso más general hasta el más específico.

En la Figura 4.1 se puede distinguir el esquema completo de requisitos y las relaciones entre ellos. De esta forma, se tienen tres requisitos fundamentales que satisfacen las necesidades de poder unirse a una red ad-hoc (*Conectar_con_Red*), poder establecer una conexión con un destino determinado (*Modo_Red*) y establecer un intercambio de mensajes de tipo chat con los usuarios con los que se tengan conexiones activas (*Modo_Chat*). Además, cada uno de estos tres se dividirán en nuevos requisitos que también se explicarán de manera detallada.

En relación a lo representado en la figura mencionada, se debe decir que en los paquetes que dependen del principal no se ha especificado el requisito concreto que deben cumplir, ya que se explicará más detalladamente en los sucesivos diagramas. En cambio, para mayor comodidad a la hora de situar un elemento concreto, se han numerado jerárquicamente mediante el identificador *id*.

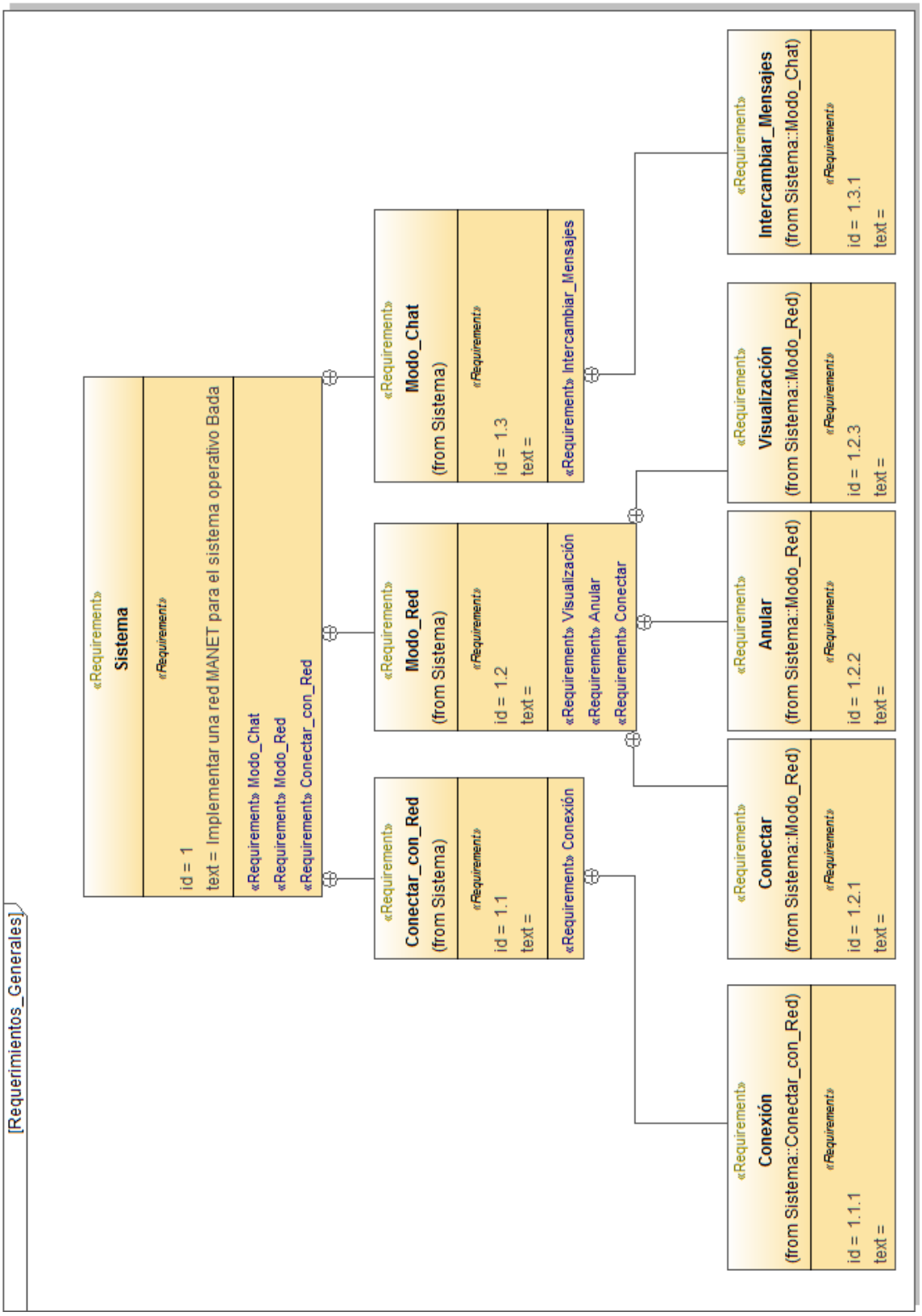

Figura 4.1: *Esquema general de requisitos*

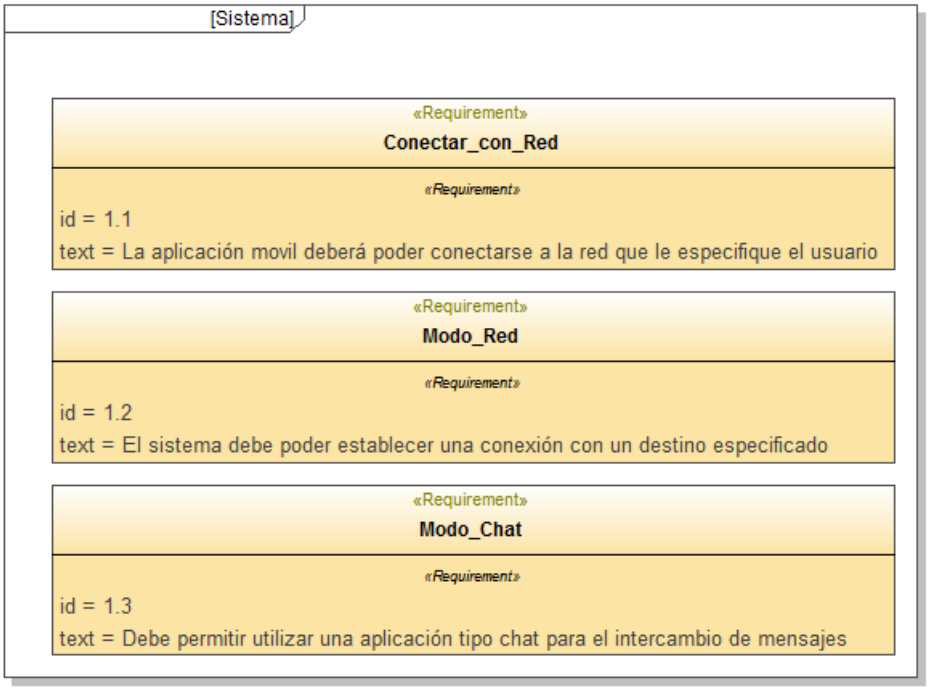

Figura 4.2: *Requisitos del sistema*

En la Figura 4.2 se pueden ver las tres funcionalidades que debe tener la aplicación. En la primera se muestra la posibilidad de acceder a un grupo ad-hoc por medio de la especificación de un nombre de red. Además de esta primera funcionalidad se incorporan dos modos de funcionamiento: uno para establecer las rutas y el otro para poder intercambiar mensajes con un dispositivo concreto.

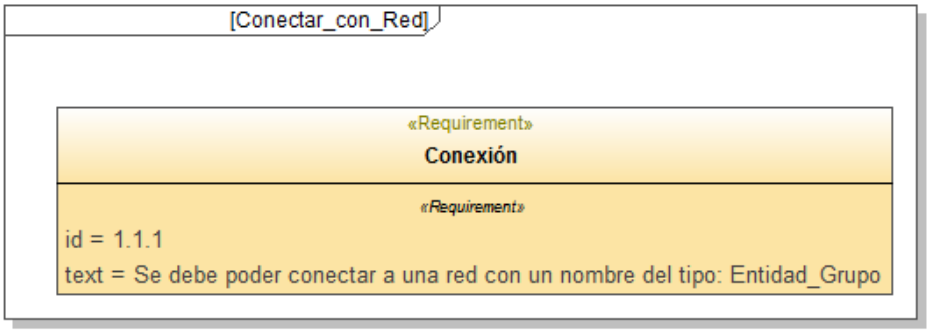

Figura 4.3: *Requisitos de la conexión*

Ya dentro del paquete *Conectar_con_Red*, en la Figura 4.3 se da a conocer la funcionalidad primera al iniciar la aplicación, que consiste en tener la posibilidad de elegir el nombre de la red a la que conectarse. Nótese que debe ser de la forma *Entidad_Grupo*, donde *Entidad* y *Grupo* pueden ser cualquier tipo de carácter. El tipo de nombre debe ser así (dos nombres separados por un guión bajo) porque viene impuesto por la API utilizada en Bada.

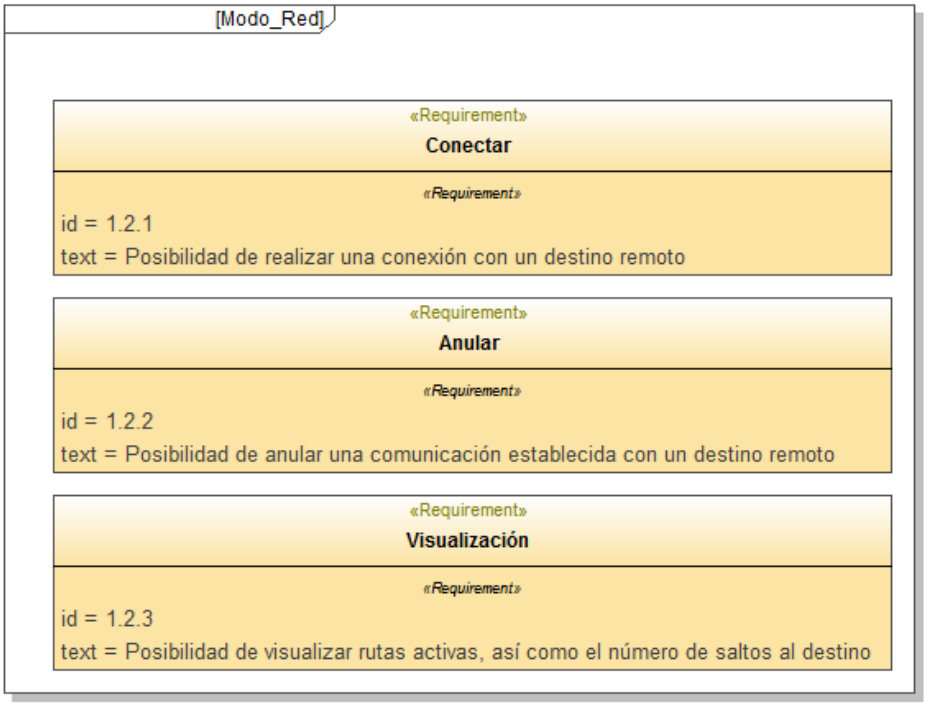

Figura 4.4: *Requisitos del Modo Red*

Continuando con el *Modo_Red*, en la Figura 4.4 se especifican los requisitos funcionales que debe tener. Uno de ellos es que debe ser capaz de establecer las rutas específicas hacia el destino que se desee. Además, el usuario debe poder anular una conexión para testear el restablecimiento de rutas. Por último, se debe tener la funcionalidad de poder observar cuáles son las conexiones activas existentes. El interactuar con las rutas activas disponibles será posible debido a la aplicación de eco que se debe crear para que las entradas de las tablas de encaminamiento se mantengan siempre actualizadas.

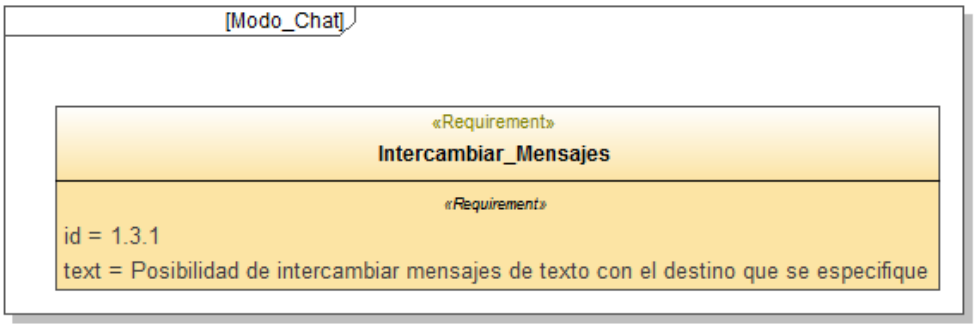

Figura 4.5: *Requisitos del Modo Chat*

Para finalizar, con el *Modo_Chat* se tiene que poder mantener una comunicación textual con el usuario que se especifique dentro de las rutas activas que se conozcan. Este tipo de comunicación sirve de ejemplo de aplicación real sobre la red MANET desplegada, así como para comprobar su correcto funcionamiento.

4.2 Requisitos no funcionales

A continuación se especifican los requisitos no funcionales del proyecto. A diferencia de los funcionales, estos requisitos no exigirán ningún tipo de funcionalidad pero serán indispensables a la hora de realizar la aplicación.

Por un lado, se programarán las partes específicas del protocolo de red intentando que sean lo más independientes posibles de la aplicación creada, con el fin de proporcionar la mayor extensibilidad posible.

Por otro lado, la interfaz de red para la interconexión de los dispositivos será WiFi.

El sistema operativo para el que se realizará la aplicación será Bada. Debido a esto, el lenguaje de programación utilizado será C++, puesto que es el utilizado en su plataforma de desarrollo.

Por último, la interfaz de usuario debe ser intuitiva y de fácil manejo. Debe permitir a un usuario utilizar la aplicación después de haber leído el Apéndice A, donde se explica el modo de funcionamiento.

4.3 Recursos humanos

- D. Jorge Navarro Ortiz como tutor del proyecto y profesor del Departamento de Teoría de la Señal, Telemática y Comunicaciones, perteneciente a la Escuela Técnica de Ingenierías Informática y de Telecomunicación de la Universidad de Granada.
- José Manuel Pérez Hueso como autor del proyecto y alumno de la titulación Ingeniería de Telecomunicación de la Universidad de Granada.

4.4 Dispositivos y herramientas utilizadas

Para el desarrollo, evaluación y prueba del presente trabajo se han utilizado los siguientes dispositivos:

Dispositivo
Ordenador Acer Aspire 5920G
Ordenador Acer Aspire 5742G
Móvil Samsung Wave GT-S8500

Tabla 4.1: *Dispositivos utilizados en el desarrollo del proyecto*

Las herramientas software utilizadas han sido las siguientes:

Herramienta	Función
Microsoft Office Word 2007	Realización de la memoria
Plataforma de desarrollo de Bada	Implementación de la aplicación
Argo UML	Creación de diagramas de estado
Star UML	Creación de diagramas de relación entre clases
Paint	Creación y retoque de figuras
GanttProject	Creación del diagrama de planificación

Tabla 4.2: *Herramientas utilizadas en el desarrollo del proyecto*

4.5 Planificación del proyecto

Para realizar la planificación se ha dividido el trabajo en 5 tareas diferentes: búsqueda de información, diseño, implementación, pruebas y redacción de la memoria. En este sentido, en la Figura 4.6 se muestra la asignación de actividades en el orden cronológico en el que se ha desarrollado el proyecto. En la fila colocada debajo de los meses se muestra el número de semana del año, en total, 26 semanas comenzando en la número 22 y acabando en la 48.

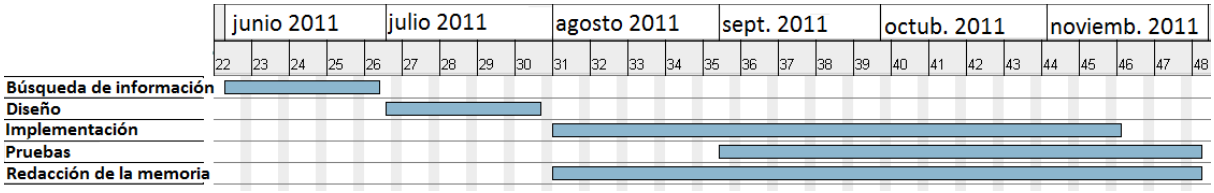

Figura 4.6: *Planificación de las tareas que componen el proyecto*

Búsqueda de información: Es el período de tiempo en el que se recopilan los datos necesarios para saber qué solución se va a llevar a cabo. Por ello, en esta fase se hace el estudio del estado del arte. (1 mes)

Diseño: Período en el que se decide cuál va a ser la solución adoptada (AODV) y se analiza cómo se puede implementar. (1 mes)

Implementación: Llevar a cabo lo que se ha diseñado en la anterior etapa. Es de señalar que en esta fase, además de implementar lo que se ha diseñado, también se diseñan posibles problemas que surgen durante la implementación. (3 meses y medio)

Pruebas: Una vez que se empieza a realizar la aplicación, se irá testeando y probando para analizar si se van cumpliendo los objetivos. Además, cuando se acaba la implementación, los últimos 15 días se dedican a realizar pruebas que se mostrarán en el capítulo 8 de la memoria.

Redacción de la memoria: Una vez que se ha analizado el estado del arte y se ha diseñado la aplicación, se tienen datos suficientes como para empezar a redactar la memoria del proyecto. En todo caso, se irán realizando los capítulos en función de los avances en la implementación, siendo una tarea realizada en paralelo

4.6 Costes

En este apartado del trabajo se pretende realizar una estimación en cuanto a costes se refiere de la totalidad del proyecto. Para ello, se va a diferenciar entre los costes humanos y los costes de los dispositivos y herramientas utilizadas.

Centrando la atención en los recursos humanos, se tiene en cuenta una jornada laboral de 8horas/día de lunes a viernes.

Actividad	Tiempo (horas)	Coste (€/h)	Total (€)
Búsqueda de información	176	20	3520
Diseño	168	20	3360
Implementación	370	20	7400
Pruebas	63	20	1260
Redacción de la memoria	225	20	4500
Total	1002	20	20040

Tabla 4.3: *Costes de recursos humanos en el proyecto*

En cuanto a los costes de los dispositivos y herramientas utilizadas, se debe señalar que se han utilizado dos ordenadores y el móvil Samsung Wave GT-S8500. Para realizar los cálculos se estima que la vida media de un ordenador es de 4 años y la de un móvil de 2 años. Para mostrar este hecho, en la columna de la derecha se indica el coste debido al uso de cada dispositivo en el tiempo que dura el proyecto.

Dispositivo	Coste de la unidad (€)	Coste en el proyecto (€)
PC1	800	200
PC2	800	200
Móvil	330	82.5
Total	1930	482.5

Tabla 4.4: *Costes de los dispositivos*

Recapitulando, el coste total se muestra en la Tabla 4.5 donde se han unido los costes por dispositivos y los recursos humanos. En cuanto a las herramientas utilizadas, todas ellas han sido gratuitas excepto Microsoft Office Word 2007 cuya licencia viene incluida en el precio del ordenador Acer Aspire 5920G utilizado.

Asunto	Coste (€)
Recursos humanos	20040
Dispositivos	482.5
Total	20522.5

Tabla 4.5: *Coste total del proyecto*

5 Bada

Durante el presente capítulo se pretende presentar las características más importantes de la plataforma en la que se ha desarrollado el proyecto. Además, en la parte final se explicarán algunos detalles importantes a tener en cuenta a la hora de programar la aplicación.

5.1 ¿Qué es Bada?

Bada significa océano en coreano. La marca Samsung, propietaria de este sistema operativo ideado para *smartphones*, eligió este nombre por la variedad ilimitada de aplicaciones que se pueden crear.

Bada incorpora una plataforma de desarrollo gratuita para que los programadores interesados puedan crear aplicaciones. El lanzamiento oficial se produjo el 17 de noviembre de 2009 [23].

En cuanto a los dispositivos que utilizan este sistema operativo, el primer móvil que lo incorporó fue el Samsung Wave GT-S8500, que se presentó en el *Mobile World Congress* [24] de Barcelona en Febrero de 2010. Actualmente, se comercializan siete modelos diferentes [25] que integran Bada.

5.2 Plataforma de desarrollo

La plataforma de desarrollo gratuita se puede encontrar en el sitio web oficial de Bada [1]. Proporciona un editor C++, compilador, depurador de fallos, un creador de la interfaz gráfica y un simulador que permite poder probar las aplicaciones que se creen directamente en el dispositivo móvil concreto al que vayan dirigidas. Está basada en Eclipse CDT [26].

Para presentar la plataforma de desarrollo, primero se muestra la Figura 5.1 donde se diferencian cuatro espacios que ofrecen diferentes funcionalidades y posteriormente se explica para qué sirven.

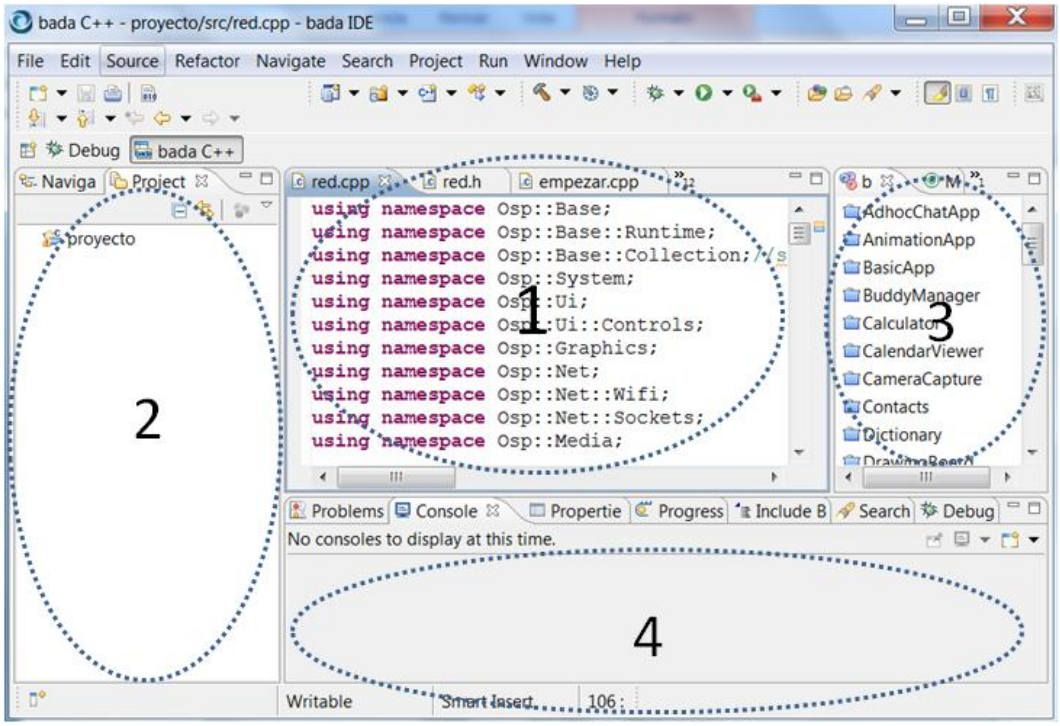

Figura 5.1: *Plataforma de desarrollo de aplicaciones*

Espacio número 1: Zona reservada para la visualización e implementación de las clases que componen la aplicación.

Espacio número 2: Zona reservada para la visualización de los proyectos que se tengan activos en el actual espacio de trabajo. En el caso de la Figura 5.1 únicamente se tiene uno activo.

Espacio número 3: Área en la que se dispone de ejemplos de proyectos completos que podemos añadir (Figura 5.2) al espacio de trabajo (espacio número 2). Con el conjunto de ejemplos que se dispone, se muestra un amplio abanico de las posibilidades que permite la plataforma.

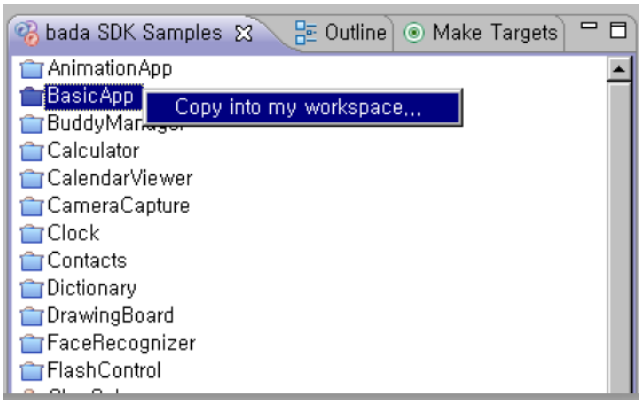

Figura 5.2: *Espacio con ejemplos de aplicaciones*

Espacio número 4: Zona en la que aparecerán avisos y notificaciones en referencia a la compilación y ejecución de los programas.

Respecto al sitio web destinado a los desarrolladores, éste ofrece una amplia variedad de foros en los que se discute sobre posibles dificultades o dudas acerca de la forma de implementar y desarrollar casos específicos. Por otra parte, se proporciona una gran cantidad de documentación con respecto al funcionamiento y la manera de programar en esta arquitectura.

En este mismo sentido, cada desarrollador puede crear una cuenta necesaria para poder subir las aplicaciones al SamsungApps, que es el único mercado posible para este tipo de terminales Samsung. Además, es el único sitio desde el que se pueden instalar aplicaciones a excepción del caso de depuración por parte del programador. Por otro lado, esta cuenta permitirá al desarrollador especificar las aplicaciones que quiere implementar y obtener de esta manera el *manifest.xml* adecuado. Este archivo es indispensable a la hora de realizar cualquier proyecto, puesto que define la forma y privilegios que se van a tener. En la Figura 5.3 se pueden encontrar los parámetros a modificar que permitirán realizar ciertas tareas. Por ejemplo, si no se quiere que una aplicación utilice WiFi, se puede especificar aquí, concretamente dentro del grupo *Osp::Net*.

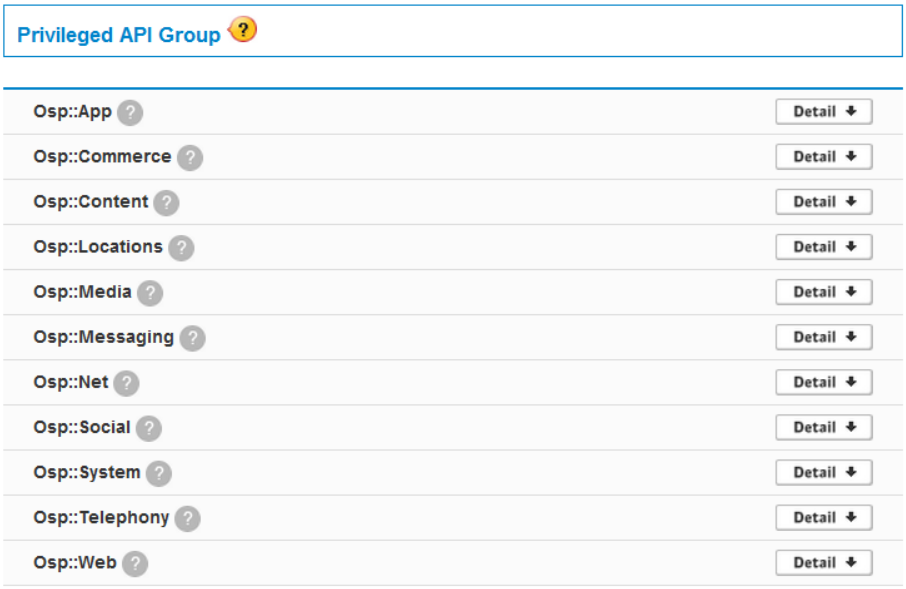

Figura 5.3: *Especificación de los privilegios de la aplicación*

Una vez que se eligen las características y privilegios que se quieran tener, se obtiene el mensaje de la Figura 5.4. En este caso se ha creado un proyecto llamado *Ejemplo* en la versión 1.0.0 y se ofrece la posibilidad de obtener el *manifest.xml* correspondiente a la aplicación registrada.

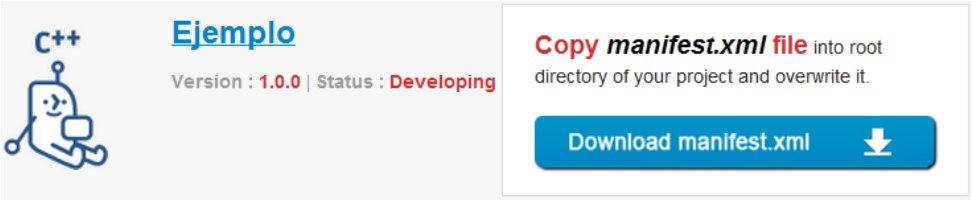

Figura 5.4: *Obtención del manifest.xml*

Para resaltar la importancia de este archivo, se puede ver en la Figura 5.5 cómo cada vez que se quiere crear un proyecto nuevo dentro de la plataforma de desarrollo, se pide introducir un *manifest.xml* ya guardado o que se cree accediendo a la cuenta del usuario en el sitio web de Bada.

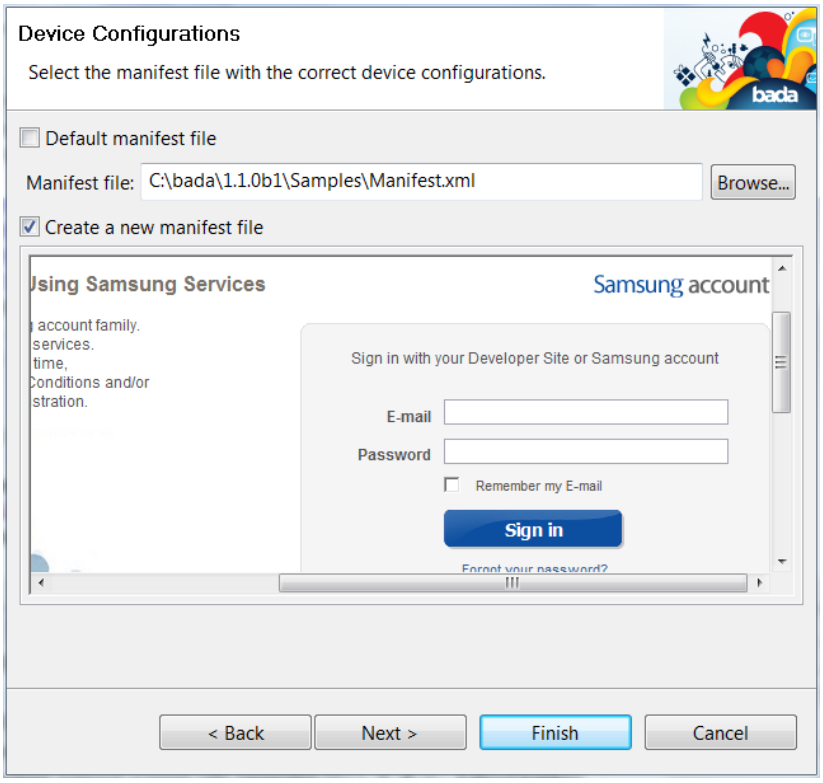

Figura 5.5: *Especificación del manifest.xml en el proyecto creado*

En cuanto a las pruebas realizadas, conviene destacar que se proporciona la funcionalidad de comprobar la implementación en el simulador propio de la plataforma o testearla directamente en un dispositivo real. Para ello, hay que instalar el certificado digital del desarrollador en el móvil, necesario para poder realizar la instalación en modo depuración. Este certificado está disponible en la ruta \<BADA_SDK_HOME>\Tools\sbuild\rootCACert.cer y se debe copiar en la carpeta *Otros* dentro de *Galería*. Una vez copiado, se debe instalar en el móvil tal como se indica en la Figura 5.6.

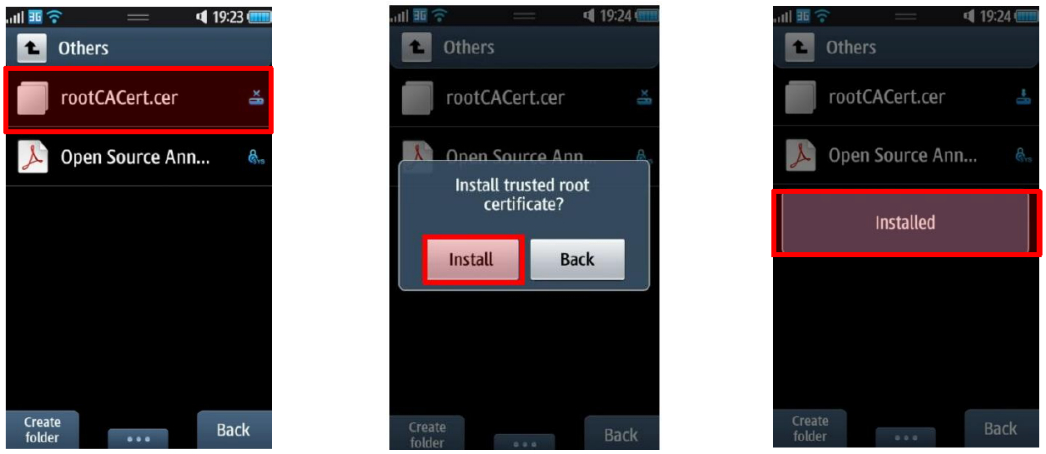

Figura 5.6: *Instalación de certificado en el móvil*

5.3 Características de programación

A continuación se mostrarán las características a tener en cuenta a la hora de programar en el sistema operativo que se está explicando.

5.3.1 Características generales

En Bada existen dos tipos de aplicaciones. Por un lado encontramos las *aplicaciones base* que son almacenadas en la ROM y no se pueden borrar por medio del gestor del dispositivo móvil. Ejemplos de éstas son los contactos o el reproductor de música. Por otro lado, se encuentran aquéllas que pueden ser instaladas y eliminadas a las que comúnmente se les llama *aplicaciones Bada*.

En este sentido, solamente una *aplicación Bada* puede ejecutarse simultáneamente en la versión actual del Sistema Operativo. Si se comienza a ejecutar otra cuando ya hay una operativa, la primera acaba su ejecución para poder dar paso a la segunda. Esto quiere decir que una *aplicación Bada* puede ejecutarse simultáneamente con múltiples *aplicaciones base* pero no con otra *aplicación Bada*. Por lo tanto, el *multi-tasking* (procesar múltiples tareas al mismo tiempo) únicamente está permitido entre una *aplicación Bada* y *aplicaciones base*.

Centrando la atención en características propias de programación, existe una clase que se considera raíz de la mayoría de las demás: se trata de *Object*, que tiene como misión hacer que sea más fácil definir los comportamientos y características compartidas entre dichas clases.

Una cualidad significativa es la presencia de *Listeners* (apartado 5.3.2) o clases encargadas de estar esperando algún evento que se produzca en la aplicación desarrollada. Estos *Listeners* implementan métodos que se ejecutan de manera asíncrona a modo de hebras durante la ejecución del programa. Dichos métodos, pertenecientes a clases que se encargan de implementar por sí solas soluciones a los posibles eventos que puedan ocurrir en una aplicación, actúan de la forma en que se especifica en la Figura 5.7. Se comienzan a ejecutar de manera asíncrona e independiente al modo de ejecución normal del programa y, cuando termina la acción a realizar, se finaliza la ejecución de la hebra que se había lanzado de manera paralela al código.

Figura 5.7: *Ciclo de funcionamiento de los eventos producidos*

En cuanto a la forma de utilizar los tipos de datos, cada uno de ellos implementará métodos encargados de dotar de funcionalidad a las variables. Por lo tanto, no existen propiedades de las variables ya que cada tipo de dato tiene implementados métodos propios en Bada para realizar las funcionalidades que se le permita. Por ejemplo, el tipo de dato Boolean tiene implementados, entre otros, los métodos operator!= (para comprobar si dos datos de tipo Boolean son distintos) y ToString (para convertir a un tipo de dato String).

En cuanto a la representación de lo implementado, existen tres capas básicas para el manejo y el control de las ventanas. La principal es el *Frame*, ya que solamente puede haber una capa de estas características por programa. La siguiente capa es el *Form* o forma, cuya misión es albergar controles. Puede haber más de una forma por aplicación y cada una tiene un área de indicación, una barra de título y teclas de control. La última capa corresponde con los controles que se pueden añadir a la forma. En la Figura 5.8 se puede ver una representación de las tres capas explicadas.

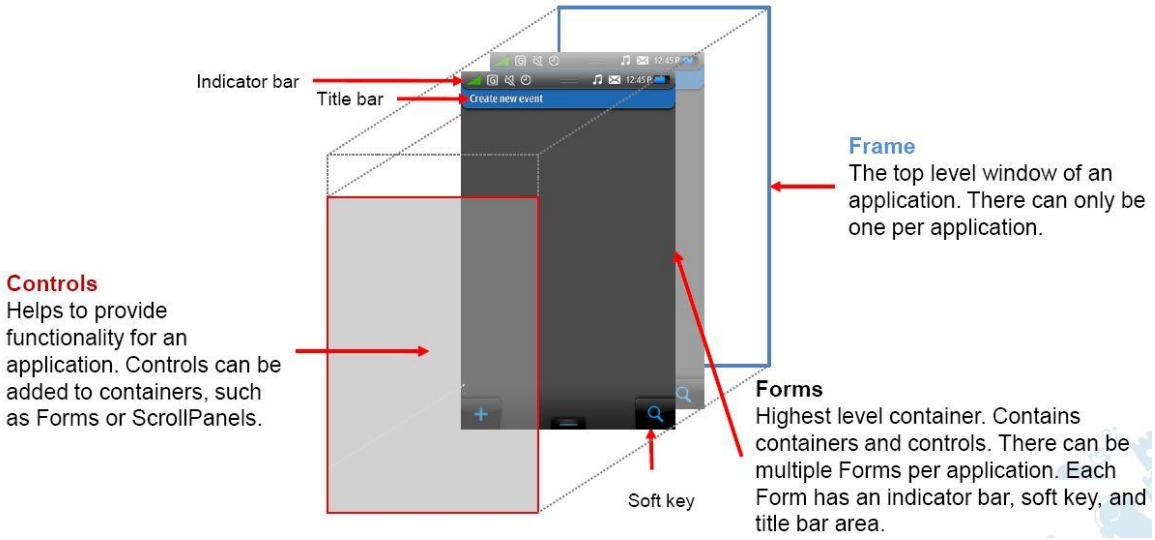

Figura 5.8: *Representación de capas en la interfaz gráfica de Bada*

5.3.2 Especificación de *Listeners*

Cuando se está ejecutando una aplicación se deben definir los *Listeners* necesarios para atender los posibles eventos que puedan ocurrir. Poniendo de ejemplo el programa realizado durante el presente trabajo, se definen los siguientes *Listeners*:

```
public Osp::Net::Wifi::IAdhocServiceEventListener,
public Osp::Ui::IActionEventListener,
public Osp::Net::Sockets::ISocketEventListener,
public Osp::Base::Runtime::ITimerEventListener,
public Osp::Ui::IExpandableItemEventListener,
public Osp::Net::INetConnectionEventListener
```

Figura 5.9: *Listeners de la aplicación creada*

Cada uno de estos *Listeners* es una clase que posee métodos esperando que ocurra algún evento ante el que puedan reaccionar. Aunque no se utilicen, hay que definirlos en el programa que se cree. A continuación se muestra cada clase con sus correspondientes métodos.

IAdhocServiceEventListener: Es la clase encargada de administrar una red ad-hoc. Mediante ella se puede crear la red con un nombre determinado o destruirla, bien cuando no se utilice o bien cuando finalice el programa. Los métodos propios que hay que definir son los que se muestran en la Figura 5.10.

```
virtual void OnAdhocServiceStarted(const Osp::Net::NetConnection*
pNetConnection, result r);
virtual void OnAdhocServiceStopped(result r);
virtual void OnMessageReceived(const Osp::Base::String& peerName, const
Osp::Base::String& message);
```

Figura 5.10: *Métodos de la clase IadhocServiceEventListener*

Método	Funcionalidad
OnAdhocServiceStarted	Es el método encargado de gestionar la comunicación ad-hoc en su inicio. Cuando se establece la conexión, se ejecuta la funcionalidad que se haya implementado en su interior.
OnAdhocServiceStopped	Método que se encarga de terminar una conexión ad-hoc.
OnMessageReceived	Método que notifica la llegada de algún mensaje de un vecino.

Tabla 5.1: *Explicación de los métodos de la clase IadhocServiceEventListener*

IActionEventListener: Es la clase encargada de gestionar los eventos que ocurren cada vez que se pulsa algún control de la aplicación. El método que se encarga de atender los eventos que se puedan producir es el que se muestra en la Figura 5.11.

```
virtual void OnActionPerformed(const Osp::Ui::Control& source, int actionId);
```

Figura 5.11: *Método de la clase IActionEventListener*

Método	Funcionalidad
OnActionPerformed	Es el método encargado de gestionar todos los eventos provenientes de controles. Cada control tendrá un identificador de acción asociado (*actionId*) y según se actúe, se ejecutará el código asociado a cada identificador.

Tabla 5.2: *Explicación del método de la clase IActionEventListener*

ISocketEventListener: Es la clase encargada de administrar toda la funcionalidad que se permite a los *sockets*. Se pueden definir como UDP, TCP, que tengan funcionalidad *broadcast* y multitud de opciones más configurables que se pueden encontrar en la ayuda de la plataforma de desarrollo. En este caso, los métodos que se tienen que implementar son los que se muestran en la Figura 5.12.

```
virtual void OnSocketAccept(Socket& socket);
virtual void OnSocketClosed(Socket& socket1, NetSocketClosedReason reason);
virtual void OnSocketConnected(Socket& socket1);
virtual void OnSocketReadyToReceive(Socket& socket);
virtual void OnSocketReadyToSend(Socket& socket1);
```

Figura 5.12: *Métodos de la clase ISocketEventListener*

Método	Funcionalidad
OnSocketAccept	Se notifica que se ha recibido una nueva conexión desde un nodo vecino.
OnSocketClosed	Se notifica al dispositivo local el cierre de un *socket* establecido con algún dispositivo.
OnSocketConnected	Se notifica la conexión exitosa establecida con otro *socket*. Es propio del protocolo TCP.
OnSocketReadyToReceive	Notifica al *socket* que se está listo para recibir los datos.
OnSocketReadyToSend	Notifica al *socket* que se está listo para enviar los datos.

Tabla 5.3: *Explicación de los métodos de la clase ISocketEventListener*

ITimerEventListener: Clase que tiene la misión de crear y gestionar el uso de los temporizadores o *timers*. El método encargado de implementar las funcionalidades es el que se muestra en la Figura 5.13.

```
virtual void OnTimerExpired(Osp::Base::Runtime::Timer& timer);
```

Figura 5.13: *Método de la clase ITimerEventListener*

Método	Funcionalidad
OnTimerExpired	Método que se ejecuta cuando expira el tiempo de vida asociado a un *timer*. Todo el código asociado a la funcionalidad de los *timers* tiene que implementarse en su interior. Para volver a iniciarlos hay que incluir las líneas necesarias dentro de este método.

Tabla 5.4: *Explicación del método de la clase ITimerEventListener*

IExpandableItemEventListener: Permite la interacción con las listas desplegables que se definan. Cada vez que se pulse en un icono de uno de estos controles, se llamará a los métodos de esta clase para llevar a cabo la acción que corresponda. En la Figura 5.14 se muestran los dos que permiten esta funcionalidad.

```
virtual void OnItemStateChanged(const Osp::Ui::Control& source, int mainIndex,
int subIndex, int itemId, Osp::Ui::ItemStatus status);
virtual void OnItemStateChanged(const Osp::Ui::Control& source, int mainIndex,
int subIndex, int itemId, int elementId, Osp::Ui::ItemStatus status);
```

Figura 5.14: *Métodos de la clase IExpandableItemEventListener*

Método	Funcionalidad
OnItemStateChanged	Método que se ejecuta cuando se realiza alguna acción en la lista desplegable que se haya implementado. Se recoge el menú principal (*mainIndex*), el submenú (*subIndex*), el estado (*status*) y lo único en que se diferencian los dos métodos es en el elemento concreto pulsado dentro del menú o submenú (*elementId*).

Tabla 5.5: *Explicación de los métodos de la clase IExpandableItemEventListener*

INetConnectionEventListener: Clase que se encarga de administrar la red que se crea y a la que se está conectado.

```
virtual void  OnNetConnectionResumed (NetConnection &netConnection);
virtual void  OnNetConnectionStarted (NetConnection &netConnection, result r);
virtual void  OnNetConnectionStopped (NetConnection &netConnection, result r);
virtual void  OnNetConnectionSuspended (NetConnection &netConnection);
```

Figura 5.15: *Métodos de la clase INetConnectionEventListener*

Método	Funcionalidad
OnNetConnectionResumed	Notifica que la red se ha recuperado del estado suspendido. Una aplicación podrá enviar o recibir datos a través de un *socket* o vía HTTP a partir de que se ejecute este método.
OnNetConnectionStarted	Se notifica que la conexión de red se ha creado con éxito. A partir de ahora, una aplicación puede recibir o mandar datos.
OnNetConnectionStopped	Se notifica que una conexión de red ha sido cerrada y desconectada.
OnNetConnectionSuspended	Notifica que una conexión ha cambiado su estado o está suspendida. Una aplicación no debe enviar ni recibir datos hasta que se pase a *Resumed* (primer método explicado).

Tabla 5.6: *Explicación de los métodos de la clase INetConnectionEventListener*

6 Diseño e implementación de AODV

Durante el capítulo 6 se explicará el diseño y la forma de llevar a cabo la implementación del protocolo elegido para el desarrollo del proyecto. Además, se comenzará con un análisis del diseño general del programa.

6.1 Análisis general

A continuación se va a mostrar el diagrama general de clases utilizado para el diseño de la aplicación. No se han incluido los métodos y atributos de la clase *red* porque se hará con más detalle posteriormente. Desde el punto de vista del programa, las clases *empezar* y *empezarForm1* no tienen especial relevancia dado que se crean automáticamente cuando se comienza a crear un proyecto basado en un *Form*, pero conviene destacar que desde *empezar* se podría tener control de las acciones en función del estado del móvil (*OnForeground*, *OnBackground*, *OnLowMemory*, *OnBatteryLevelChanged*, *OnScreenOn*, *OnScreenOff*). Por otro lado, *empezarForm1* crea el *Form* del programa y permite intercambiar diferentes tipos de *Form* que se estén utilizando. Sin embargo, debido a la dependencia de variables importantes como las propias de la creación de la red ad-hoc con este control, no se podría hacer que el programa interactuara con varios *Forms* ya que cada vez que se le diese el control a uno habría que reiniciar la red ad-hoc, cuya configuración no es instantánea. Por lo tanto, se ha optado por realizar toda la aplicación en una única forma.

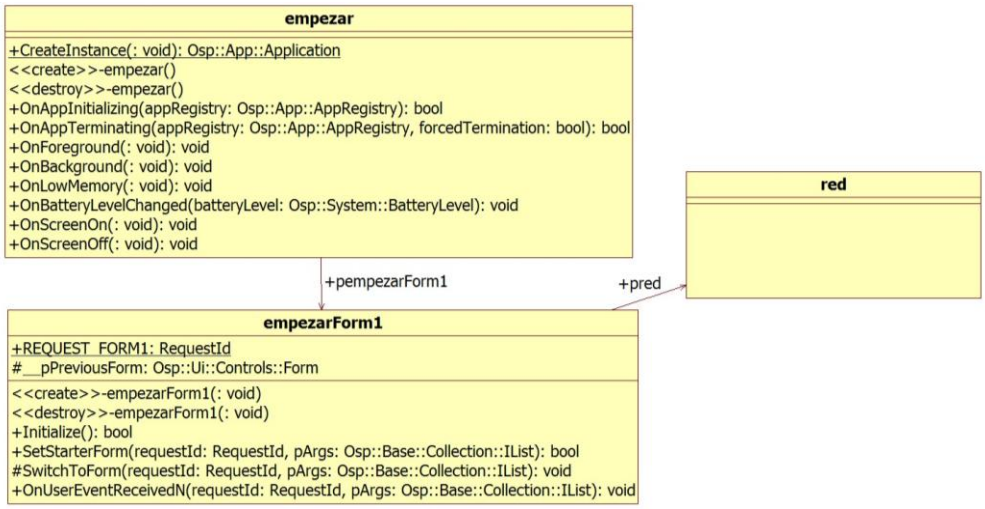

Figura 6.1: *Diagrama general de clases*

red
+Destino_IP: String +Destino_Seq: unsigned long +Destino_Saltos: unsigned int +Siguiente_Salto: String +Rutas_Inactivas: String +Destino_Seq_In: unsigned long +Destino_Saltos_In: unsigned int +RREQID: unsigned long +tiempo: Osp::Base::Runtime::Timer +tiempo_Hello: Osp::Base::Runtime::Timer +__pTimer: Osp::Base::Runtime::Timer +Estado: unsigned int +Hello_Recibidos: unsigned int +Orden_Nodo: int +RREQ_Retry: int +direccionesDatos: String +Mi_Numero_Seq: unsigned long +Mi_Ip: String +RREQ_ID: unsigned long +Hello_mandado: unsigned int +__pSocket: Osp::Net::Sockets::Socket +__SocketAplicacion: Osp::Net::Sockets::Socket +__pLocalPoint: Osp::Net::NetEndPoint +RutaOk: RequestId +RutaEncontrada: RequestId +RutaHecha: RequestId +RutaBusca: RequestId #ID_BUTTON_SEND: int #ID_BUTTON_ENVIAR: int #ID_BUTTON_EMPEZAR: int #ID_OPTION_ITEM1: int #ID_OPTION_ITEM2: int #ID_OPTION: int #ID_OPTION_ITEM3: int #ID_OPTION_ITEM4: int #ID_OPTION_ITEM5: int #ID_FORMAT_STRING: int #ID_FORMAT_BITMAP: int #ID_ITEM_MAIN0: int #ID_ITEM_MAIN1: int #ID_ITEM_MAIN0_SUB0: int #ID_ITEM_MAIN0_SUB1: int #ID_ITEM_MAIN1_SUB0: int #ID_ITEM_MAIN1_SUB1: int #ID_BUTTON_CLOSE_POPUP: int #ID_BUTTON_CLOSE_POPUP1: int #ID2_ITEM_MAIN0_SUB0: int #ID2_ITEM_MAIN0: int -__pMainItemFormat: Osp::Ui::Controls::CustomListItemFormat -__pSubItemFormat: Osp::Ui::Controls::CustomListItemFormat -__pMainItemFormat2: Osp::Ui::Controls::CustomListItemFormat -__pSubItemFormat2: Osp::Ui::Controls::CustomListItemFormat -pExpandableList: Osp::Ui::Controls::ExpandableList -pExpandableList2: Osp::Ui::Controls::ExpandableList -__pPopup: Osp::Ui::Controls::Popup -__pPopup1: Osp::Ui::Controls::Popup -__pOptionMenu: Osp::Ui::Controls::OptionMenu -__AreaEscribir: Osp::Ui::Controls::EditArea -__AreaEscribir2: Osp::Ui::Controls::EditArea -__pEscribirDireccion: Osp::Ui::Controls::EditField -__pFrame1: Osp::Ui::Controls::Frame -__pBotonEmpezar: Osp::Ui::Controls::Button -__pCompanyNameLabel: Osp::Ui::Controls::Label -__pAppNameLabel: Osp::Ui:Controls::Label -__pPeerNameLabel: Osp::Ui::Controls::Label -__pCompanyNameLabel: Osp::Ui::Controls::Label -__pCompanyNameField: Osp::Ui::Controls::EditField -__pAppNameField: Osp::Ui::Controls::EditField

-__pPeerNameField: Osp::Ui::Controls::EditField -__pAdhocProgressPopupLabel: Osp::Ui::Controls::Label -__pAdhocProgressPopup: Osp::Ui::Controls::Popup -accountManager: NetAccountManager -pNetConnection: NetConnection -ModoChat: int -ModoExplorar: int -ModoTodos: int -Mi_nombre: String -Destinos: String -Nombres: String -Destinos2: String -Nombres2: String -control_numero: int -posicion_Lista: unsigned int -Rutas: String
<<create>> red(: void) <<destroy>> red(: void) +Initialize(: void): bool +OnInitializing(: void): result +OnTerminating(: void): result +OnActionPerformed(source: Osp::Ui::Control, actionId: int): void +OnAdhocServiceStarted(pNetConnection: Osp::Net::NetConnection, r: result): void +OnAdhocServiceStopped(r: result): void +OnMessageReceived(peerName: Osp::Base::String, message: Osp::Base::String): void +OnSocketAccept(socket: Socket): void +OnSocketClosed(socket1: Socket, reason: NetSocketClosedReason): void +OnSocketConnected(socket1: Socket): void +OnSocketReadyToReceive(socket: Socket): void +OnSocketReadyToSend(socket1: Socket): void +OnNetConnectionResumed(netConnection: NetConnection): void +OnNetConnectionStarted(netConnection: NetConnection, r: result): void +OnNetConnectionStopped(netConnection: NetConnection, r: result): void +OnNetConnectionSuspended(netConnection: NetConnection): void +MandaRREP(RA: int, saltos: unsigned int, Dest_Seq: unsigned long, Life_time: unsigned long, IP_Destino: String, IP_Origen: String, destino: NetEndPoint): void +RecibeRREP(recibe_RREP: ByteBuffer, direccionAnterior: String): void +MandaRREQ(Flg: int, saltos: unsigned int, RREQid: unsigned long, Dest_Seq: unsigned long, Orig_Seq: unsigned long, IP_Destino: String, IP_Origen: String, destino: NetEndPoint): void +RecibeRREQ(recibe_RREQ: ByteBuffer, direccionAnterior: String): void +MandaRERR(DestinoPerdido: String, indice: int, direccionAnterior: String): void +RecibeRERR(recibe_RREQ: ByteBuffer, direccionAnterior: String): void +RecibePropuestas(requestId: RequestId, pArgs: Osp::Base::Collection::IList): void +BuscarDireccion(direccion: String): result +OnTimerExpired(timer: Osp::Base::Runtime::Timer): void +NotificaRuta(requestId: RequestId, DireccionIP: String): void +DevolverIndice(direccionIP: String): int +GuardarRuta(indice: int): void +MensajeOriginal(): void +BorrarEntrada(direccion: String): void +MandaDatos(IP_Destino: String, IP_Origen: String, Datos: String, destino: NetEndPoint, indice: int): void +RecibeDatos(recibe_Datos: ByteBuffer, direccionAnterior: String, longitud: unsigned long, indice: int): void +RecibeDatos2(recibe_Datos: ByteBuffer, direccionAnterior: String, longitud: unsigned long, indice: int): void +RecibeNotificacion(requestId: RequestId, destino: NetEndPoint): void +OnItemStateChanged(source: Osp::Ui::Control, mainIndex: int, subIndex: int, itemId: int, status: Osp::Ui::ItemStatus): void +OnItemStateChanged(source: Osp::Ui::Control, mainIndex: int, subIndex: int, itemId: int, elementId: int, status: Osp::Ui::ItemStatus): void +crearEntrada(direccionIP: String): void +crearEntrada2(direccionIP: String): void +MuestraDatos(direccion: String, indice: int): void

Figura 6.2: *Clase red*

Conviene destacar que el programa contiene tres partes bien diferenciadas. Por un lado, encontramos las funcionalidades propias del protocolo AODV. Por otro, las funcionalidades de las aplicaciones de chat y de eco creadas. Por último, se diferencian las funcionalidades propias de la interacción del usuario con el móvil.

A continuación se va a explicar la funcionalidad de todos los métodos del programa. En apartados posteriores se entrará en más detalle para explicar toda la funcionalidad que tienen a la hora de ser utilizados para el protocolo AODV (apartado 6.2) o para las aplicaciones de chat y de eco (capítulo 7). Sin embargo, aquéllos que sean propios de Bada e implementen funcionalidades propias de clases que funcionan como *Listeners* fueron explicados en el capítulo 5.

Método	Funcionalidad
MandaRREP	Método encargado de formar y mandar un mensaje RREP.
RecibeRREP	Método que actúa cuando se recibe un mensaje RREP.
MandaRREQ	Método encargado de formar y mandar un mensaje RREQ.
RecibeRREQ	Método que actúa cuando se recibe un mensaje RREQ.
MandaRERR	Método encargado de mandar un mensaje RERR. Debe diferenciar si mandar a un destino o a todos.
RecibeRERR	Método encargado de actuar cuando se recibe un mensaje RERR. Deberá borrar las rutas inalcanzables por el nodo que reciba el mensaje.
RecibePropuestas	Método para actualizar los destinos disponibles en la lista desplegable.
BuscarDireccion	Método llamado cuando se quiere establecer contacto con un destino.
NotificaRuta	Notifica el establecimiento de una ruta indicada por su dirección IP.
DevolverIndice	Devuelve la posición en la lista de destinos del elemento seleccionado en la lista desplegable.
GuardarRuta	Método para guardar las rutas que ya no se encuentran activas en las variables que se encargan de gestionar las rutas inactivas.
MensajeOriginal	Método llamado cuando no hay ningún destino activo y en la lista desplegable se quiere que aparezca el mensaje por defecto (Conexiones).
BorrarEntrada	Método llamado cuando se quiere borrar un destino de la lista desplegable.
MandaDatos	Se encarga de enviar los datos que se le pasen como argumento al destino deseado.
RecibeDatos	Método encargado de recibir los mensajes propios de las aplicaciones que no son AODV.
RecibeDatos2	Método encargado de recibir los datos destinados a explorar el número de vecinos que se encuentran en el rango de cobertura de un dispositivo.

Tabla 6.1: *Métodos del programa*

Método	Funcionalidad
RecibeNotificacion	Notifica el establecimiento de una ruta por su IP y puerto.
CrearEntrada	Se encarga de crear una entrada en la lista desplegable cuando se consigue establecer una ruta activa.
CrearEntrada2	Se encarga de crear una entrada en la lista desplegable de exploración de vecinos.
MuestraDatos	Método llamado para mostrar estadísticas de funcionamiento del protocolo AODV.

Tabla 6.2: *Continuación de la Tabla 6.1(Métodos del programa)*

A continuación se presentan los atributos de la clase en función de la funcionalidad para la que estén dirigidos. En un principio se empieza describiendo las variables propias del protocolo AODV.

Atributo	Funcionalidad
Destino_IP	Variable encargada de guardar los destinos que se tengan activos.
Destino_Seq	Variable encargada de guardar el número de secuencia de los destinos activos.
Destino_Saltos	Variable que almacena el número de saltos necesarios para llegar a un destino.
Siguiente_Salto	Variable que guarda el siguiente salto al que hay que mandar un mensaje que va dirigido a un destino determinado.
Rutas_Inactivas	Variable que guarda los destinos que han estado activos, pero que han expirado.
Destino_Seq_In	Variable que guarda el número de secuencia de los destinos inactivos.
Destino_Saltos_In	Variable que almacena el número de saltos necesarios para llegar a un destino inactivo.
RREQID	Variable que se encarga de almacenar el identificador de RREQ asociado a una petición de rutas con el fin de no procesar dos veces la misma.
tiempo	Variable que almacena los *timers* de AODV.
tiempo_Hello	Variable que inicia y actualiza el *timer* HELLO.
Estado	Variable que indica si el estado de una ruta es activo o inactivo.
Hello_Recibidos	Variable que permite saber el número de mensajes HELLO recibidos por parte de un dispositivo y por tanto, poder determinar si existe algún enlace roto.
Orden_Nodo	Variable que sirve para saber si el dispositivo es el origen de la comunicación o no.
Mi_Numer_Seq	Variable encargada de administrar el número de secuencia propio de cada dispositivo.

Tabla 6.3: *Atributos propios de AODV*

Atributo	Funcionalidad
Mi_Ip	Variable que guarda la IP propia de cada dispositivo.
RREQ_ID	Variable que guarda el identificador de RREQ propio de cada dispositivo.
Hello_mandado	Variable que permite saber si hay alguna ruta activa para empezar a mandar mensajes HELLO o si no hay ninguna para no hacerlo.
__pSocket	*Socket* de AODV. Enviará y escuchará peticiones en el puerto 654.
__pLocalPoint	Variable que guarda la dirección *broadcas*t 255.255.255.255 para ser utilizada en los mensajes que no vayan destinados a un dispositivo concreto.

Tabla 6.4: *Continuación de la Tabla 6.3(Atributos propios de AODV)*

Siguiendo con la descripción de las variables de la clase, se especifican aquéllas que están relacionadas con las aplicaciones de chat y de eco. Durante el capítulo 7 se explicará con más detalle cuál ha sido la forma de diseñarlas e implementarlas.

Atributo	Funcionalidad
Mi_nombre	Variable para guardar el nombre propio de cada dispositivo.
Destinos	Variable que guarda las direcciones IP de los vecinos de un dispositivo cada vez que se pregunta por ellos.
Nombres	Variable que guarda los nombres de los vecinos de un dispositivo cada vez que se pregunta por ellos.
Destinos2	Igual que Destinos pero en lugar de guardarlos cada vez que se pregunta por ellos, Destinos2 consiste en un histórico de direcciones IP que se han ido almacenando cada vez que se preguntaba.
Nombres2	Igual que Nombres pero en lugar de guardarlos cada vez que se pregunta por ellos, Nombres2 consiste en un histórico de nombres que se han ido almacenando cada vez que se preguntaba.
__SocketAplicacion	*Socket* de las aplicaciones de eco y chat. Enviará y escuchará en el puerto 15000.
direccionesDatos	Variable para almacenar el destino con el que intercambiar mensajes de datos. Se guarda el vecino seleccionado en el *Modo Chat*.
__pTimer	*Timer* utilizado para buscar periódicamente a los vecinos que se tengan al alcance.

Tabla 6.5: *Atributos propios de las aplicaciones*

Por último, se presentan aquellas variables que hacen posible la interacción del usuario con el programa y a su vez de éste con el usuario, ya que los resultados del protocolo AODV y de las aplicaciones de chat y de eco también deberán ser mostrados para su visualización.

Atributo	Funcionalidad
ID_BUTTON_SEND	Identificador del botón de buscar una ruta.
ID_BUTTON_ENVIAR	Identificador del botón que permite enviar un mensaje de datos.
ID_BUTTON_EMPEZAR	Identificador del botón que inicializa la aplicación.
ID_OPTION	Identificador del menú de opciones.
ID_OPTION_ITEM1	Identificador del *Modo Red*.
ID_OPTION_ITEM2	Identificador del *Modo Chat*.
ID_OPTION_ITEM3	Identificador del *Modo Conexiones*.
ID_OPTION_ITEM4	Identificador del *Modo Explorar*.
ID_OPTION_ITEM5	Identificador del *Modo Mensaje a Todos*.
ID_FORMAT_STRING	Identificador que permite saber cuándo se pulsa sobre el texto de una lista desplegable.
ID_FORMAT_BITMAP	Identificador que permite saber cuándo se pulsa sobre el área de la lista desplegable que no es texto.
ID_ITEM_MAIN0	Identificador del menú principal de la lista desplegable.
ID_ITEM_MAIN0_SUB0	Identificador del primer elemento del submenú de la lista desplegable.
ID_ITEM_MAIN0_SUB1	Identificador del segundo elemento del submenú de la lista desplegable.
ID_BUTTON_CLOSE_POPUP	Identificador de cierre del mensaje que informa sobre una IP incorrecta introducida en el área de búsqueda.
ID_BUTTON_CLOSE_POPUP1	Identificador de cierre del mensaje de notificaciones generales.
ID2_ITEM_MAIN0	Identificador del menú principal de la lista desplegable de los vecinos encontrados.
ID2_ITEM_MAIN0_SUB0	Identificador del primer elemento del submenú de la lista desplegable de los vecinos encontrados.
__pMainItemFormat	Variable que especifica las características del menú principal de la lista desplegable.
__pSubItemFormat	Variable que especifica las características del submenú de la lista desplegable.
__pMainItemFormat2	Variable que especifica las características del menú principal de la lista desplegable propia de la exploración de vecinos.

Tabla 6.6: *Atributos propios de la interfaz y de la funcionalidad entre métodos*

Atributo	Funcionalidad
__pSubItemFormat2	Variable que especifica las características del submenú de la lista desplegable propia de la exploración de vecinos.
pExpandableList	Variable que permite construir la lista desplegable que informará sobre las conexiones activas.
pExpandableList2	Variable que permite construir la lista desplegable que informará sobre los vecinos del dispositivo.
__pPopup	Variable que permite hacer un aviso cuando se introduzca una IP incorrecta.
__pPopup1	Variable que permite hacer notificaciones al usuario de carácter general.
__pOptionMenu	Variable que permite construir el menú de opciones que albergará los diferentes modos que posee la aplicación.
__AreaEscribir	Variable que permite construir el área donde se escribirán las notificaciones propias del protocolo AODV.
__AreaEscribir2	Variable que permite construir el área donde se escribirán los intercambios de mensajes de texto entre los usuario de la aplicación de chat.
__pEscribirDireccion	Variable que permite escribir la dirección IP que se quiere buscar si se está en el *Modo Red* o el texto que queramos enviar si se está en el *Modo Chat*.
__pFrame1	*Frame* principal.
__pBotonEmpezar	Botón que permite iniciar la aplicación una vez que se ha especificado el nombre de la red y el nombre del usuario.
__pCompanyNameLabel	Variable que permite mostrar el texto junto al área de escritura de la primera parte del nombre de la red.
__pAppNameLabel	Variable que permite mostrar el texto junto al área de escritura de la segunda parte del nombre de la red.
__pPeerNameLabel	Variable que permite mostrar el texto junto al área de escritura del nombre de usuario.
__pCompanyNameField	Variable que permite crear un área donde escribir la primera parte del nombre de la red.
__pAppNameField	Variable que permite crear un área donde escribir la segunda parte del nombre de la red.
__pPeerNameField	Variable que permite crear un área donde escribir el nombre de usuario.
__pAdhocProgressPopupLabel	Variable que contiene el texto que se mostrará cuando se esté creando la red ad-hoc.

Tabla 6.7: *Continuación de la Tabla 6.6 (Atributos propios de la interfaz y de la funcionalidad entre métodos)*

Atributo	Funcionalidad
__pAdhocProgressPopup	Variable que hará posible la visualización de la notificación que demuestra que se está creando una red ad-hoc.
accountManager	Variable que permite gestionar la red creada.
pNetConnection	Variable que permite realizar acciones sobre la red como crearla o esperar que llegue un evento.
RutaOk	Identificador de ruta encontrada para el método *NotificaRuta.*
RutaEncontrada	Identificador de ruta encontrada para el método *RecibeNotificacion.*
RutaHecha	Identificador de ruta creada para el método *RecibePropuestas.*
RutaBusca	Identificador para buscar una nueva dirección.
ModoChat	Variable que especifica que se está en el *Modo Chat* de la aplicación.
ModoExplorar	Variable que especifica que se está en el *Modo Explorar* de la aplicación.
ModoTodos	Variable que especifica que se está en el modo *Mensaje a Todos* de la aplicación.
Rutas	Variable que permite tener una copia de las rutas activas que se tienen en la lista desplegable.

Tabla 6.8: *Continuación de la Tabla 6.6 (Atributos propios de la interfaz y de la funcionalidad entre métodos)*

6.2 Diseño del protocolo AODV

A partir de ahora, se entrará en profundidad en el protocolo AODV. Para satisfacer sus funcionalidades, la aplicación debe tener seis métodos independientes que se encarguen de cada uno de los mensajes que se puedan enviar. Esto es así porque se tienen 3 tipos diferentes de mensajes (RREQ, RREP y RERR) y para cada uno de ellos se debe tener un método de envío y otro de recepción. De esta manera, se necesitan 6 métodos: *MandaRREQ*, *RecibeRREQ*, *MandaRREP*, *RecibeRREP*, *MandaRERR* y *RecibeRERR*.

MandaRREQ: Método encargado de formar el mensaje RREQ y enviarlo a la dirección 255.255.255.255 para su difusión por la red.

RecibeRREQ: Método encargado de recibir un mensaje RREQ. Debe decidir si seguir propagando la petición de rutas o responder.

MandaRREP: Es el método llamado por el programa cuando el dispositivo necesita dar una respuesta a una petición de rutas. A diferencia del *MandaRREQ* que es *broadcast* (255.255.255.255), aquí se tiene que buscar cuál es el dispositivo concreto al que se tiene que enviar el mensaje.

RecibeRREP: Método que actúa cuando se recibe un mensaje RREP y que deberá comprobar si es el destino final o intermedio para seguir encaminando.

MandaRERR: Método llamado cuando se detecta una rotura de un enlace o se recibe un mensaje de datos para el que no se tiene una ruta activa. Deberá diferenciar entre enviar a todos los dispositivos para propagar rápidamente la información o enviar solamente a uno.

RecibeRERR: Es el método que se llama cuando se recibe un mensaje RERR. Deberá anular las rutas correspondientes a los destinos que no se encuentren operativos.

Otros dos métodos importantes en el diseño de la aplicación, pero que no están relacionados con los tipos de mensajes AODV, son *BuscarDirección* y *GuardarRuta*.

BuscarDirección: Método llamado cuando se necesita establecer una comunicación con un destino. Debe comprobar si la conexión está ya establecida o si es necesario realizarla.

GuardarRuta: Método que actúa cuando expira una entrada de la tabla de encaminamiento y se tienen que guardar los datos de ésta en las variables que se encargan de gestionar la tabla de rutas inactivas.

6.3 Almacenamiento de la información en AODV

En este apartado se va a mostrar la forma de guardar los datos de las rutas. Para ello se diferenciará entre los relativos a las activas, inactivas y los *timers*.

6.3.1 Opciones de ruta activa

En todos los casos, la manera de almacenar las opciones consiste en una matriz con diferentes tipos de datos. En cada fila se insertarán los distintos elementos que queremos guardar y cada columna nos marcará a qué destino en concreto se refiere la comunicación. Se puede ver más detallado en la Tabla 6.9.

VARIABLES	CONTENIDO DE LOS VECTORES				
Destino_IP	IP1	IP2	IP3	. . .	IPN
Siguiente_Salto	IP1	IP2	IP3	. . .	IPN
Destino_Saltos	-	-	-	. . .	-
Estado	-	-	-	. . .	-
Destino_Seq	-	-	-	. . .	-
RREQID (Identificador de RREQ)	-	-	-	. . .	-
Orden_Nodo	-	-	-	. . .	-
Hello_Recibidos	-	-	-	. . .	-

Tabla 6.9: *Contenido de los vectores usados en AODV*

A continuación se explicará con más detalle cada uno de los campos contenidos en la matriz de la Tabla 6.9.

Destino_IP: Es la variable encargada de guardar las direcciones IP de los destinos con los que se tiene una conexión activa.

Siguiente_Salto: Es la variable encargada de guardar la dirección IP del siguiente nodo al que hay que dirigirse para llegar al destino final encuadrado en la variable *Destino_IP* dentro de la misma columna.

Destino_Saltos: Variable destinada a guardar el número de dispositivos que hay que atravesar para llegar el destino especificado en la misma columna de la variable *Destino_IP*.

Estado: Permite saber si el destino especificado en la misma columna de la variable *Destino_IP* está en modo activo o no. El estado 1 será modo inactivo y el estado 2 será modo activo.

Destino_Seq: Permite guardar el número de secuencia asociado al destino guardado en la misma columna de la variable *Destino_IP*.

RREQID: Campo en el que se almacena un identificador asociado a cada petición de ruta (mensaje RREQ), para no volver a procesarla.

Orden_Nodo: Permite saber si un dispositivo es el origen de la comunicación establecida con el destino que se almacena en la misma columna de la variable *Destino_IP*.

Hello_Recibidos: Campo en el que se van contabilizando el número de mensajes HELLO que se pierden por parte de los vecinos de un nodo.

6.3.2 Opciones de ruta inactiva

Además, se tienen tablas de rutas hacia los destinos que no están activos. Se puede ver más detallado en la Tabla 6.10.

VARIABLES	CONTENIDO DE LOS VECTORES				
Rutas_Inactivas	IP1	IP2	IP3	. . .	IPN
	IP1	IP2	IP3	. . .	IPN
Destino_Saltos_In	-	-	-	. . .	-
Destino_Seq_In	-	-	-	. . .	-

Tabla 6.10: *Contenido de los vectores de rutas inactivas en AODV*

Rutas_Inactivas: Se trata de una matriz de dos filas. La primera corresponde a los destinos con los que se ha tenido conexión pero que han expirado o se han perdido. La segunda muestra el siguiente salto que se tenía hacia un destino ya inactivo.

Destino_Saltos_In: Número de saltos que se tenía hacia el destino inactivo situado en la misma columna.

Destino_Seq_In: Número de secuencia que se tenía para el destino inactivo situado en la misma columna.

6.3.3 Almacenamiento de temporizadores

Por otra parte, también se incluye la forma de guardar los temporizadores (*timers*) que se tienen para cada destino. Se ha dejado la variable *Destino_IP* para una mejor comprensión de las tablas. En el apartado 6.1 se vieron las distintas variables utilizadas en AODV. En este caso, los *timers* corresponden con la variable *tiempo,* que es una matriz como la que se muestra a continuación.

TIMERS	CONTENIDO DE LOS VECTORES				
Destino_IP	IP1	IP2	IP3	. . .	IPN
Timer Mantenimiento Ruta Activa (TMRA)	TMRA1	TMRA2	TMRA3	...	TMRAN
Timer Ruta Activa (TRA)	TRA1	TRA2	TRA3	. . .	TRAN
Timer Búsqueda de Ruta (TBR)	TBR1	TBR2	TBR3	. . .	TBRN

Tabla 6.11: *Contenido de los Timers usados en AODV*

Timer Mantenimiento Ruta Activa (TMRA): *Timer* destinado a mantener una conexión periódica con el destino especificado en la misma columna de la variable *Destino_IP*. Cada vez que expira, se manda un mensaje al destino correspondiente. Su periodicidad es de 500ms.

Timer Ruta Activa (TRA): *Timer* que especifica el tiempo que una ruta hacia un destino debe estar activa si no se ha recibido ningún mensaje de control o de datos. Este temporizador expira a los tres segundos.

Timer Búsqueda de Ruta (TBR): *Timer* asociado al tiempo que se debe estar esperando a recibir una respuesta (RREP) cuando se manda una petición de rutas (RREQ). Expira a los dos segundos de haber mandado el primer RREQ.

6.4 Diagramas de estados de las distintas funcionalidades de AODV

A continuación se van a presentar los distintos escenarios posibles en la aplicación en función del mensaje que se reciba o que se intente mandar. Para ello se ha hecho uso de un conjunto de diagramas de estado que resumirán las acciones a realizar según corresponda en cada caso.

6.4.1 Inicio de conexión

En la Figura 6.3 se muestra el mecanismo llevado a cabo a la hora de iniciar una conexión.

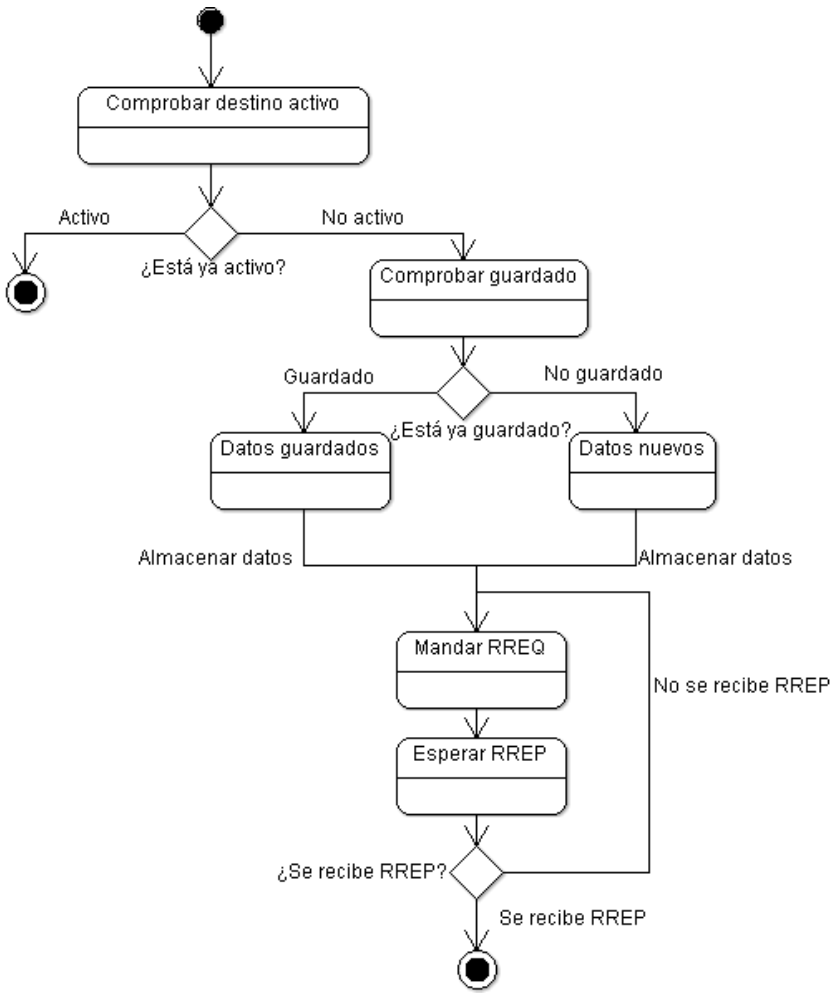

Figura 6.3: *Diagrama de estados al buscar una ruta*

Mediante el diagrama mostrado en la Figura 6.3 se pueden ver las diferentes funcionalidades a la hora de mandar un mensaje RREQ si el que lo hace es el nodo origen de la comunicación.

Cuando un dispositivo desea hacer una petición de rutas para encontrar un destino, comprueba si el destino está ya activo. Si está activo no se hace nada, puesto que ya se dispone de la ruta buscada. Por el contrario, si no está activo, se comprueba si la ruta se había tenido antes almacenada o no. Si se había tenido almacenada, se cargan los datos guardados (último número de secuencia conocidos para ese destino). Si no se había tenido almacenada, se activa el bit U (indicador de desconocimiento del número de secuencia). Tras esto, se crea el mensaje RREQ y se manda. Una vez finalizado el proceso, se espera una respuesta RREP y si no se recibe, se vuelve a mandar el mensaje RREQ. Si se recibe un mensaje RREP, no se mandan más RREQs.

6.4.2 Recepción de RREQ

Una vez que se ha mandado un RREQ, éste llegará a los dispositivos que se encuentren dentro de la cobertura del emisor. Se debe realizar una serie de acciones para atender las opciones que se pueden dar cuando se recibe. En la Figura 6.4 se muestra un diagrama de estados detallado para comprender la funcionalidad de recibir un mensaje RREQ.

Cuando se recibe el mensaje, lo primero que se hace es extraer los datos propios del protocolo. Se comprueba cuál es el identificador de RREQ (RREQ_ID) y el origen de la comunicación con el fin de saber si ya se había procesado. Si éste es el caso, se descarta y no se vuelve a retransmitir.

Una vez que se pasa a analizarlo, se comprueba si ya se tiene una ruta guardada y activa. Si es así, también se descarta. Si no se posee una entrada activa, se incorpora a la tabla de encaminamiento una ruta hacia el nodo del que proviene el RREQ y se comprueba si dicho nodo es el origen de la comunicación. Si no es así, se debe crear también una ruta hacia el dispositivo origen teniendo como siguiente salto al que envía el RREQ.

Por último, se comprueba si el nodo al que le llega el mensaje debe responder o si, por el contrario, tiene que seguir difundiendo el RREQ. Si se está en el primer caso, se comienza un proceso de respuesta mandando un mensaje *unicast* RREP destinado al dispositivo que inició la búsqueda de rutas. En cambio, en el segundo caso, se incrementa la cuenta de saltos en uno y se difunde de nuevo el mensaje.

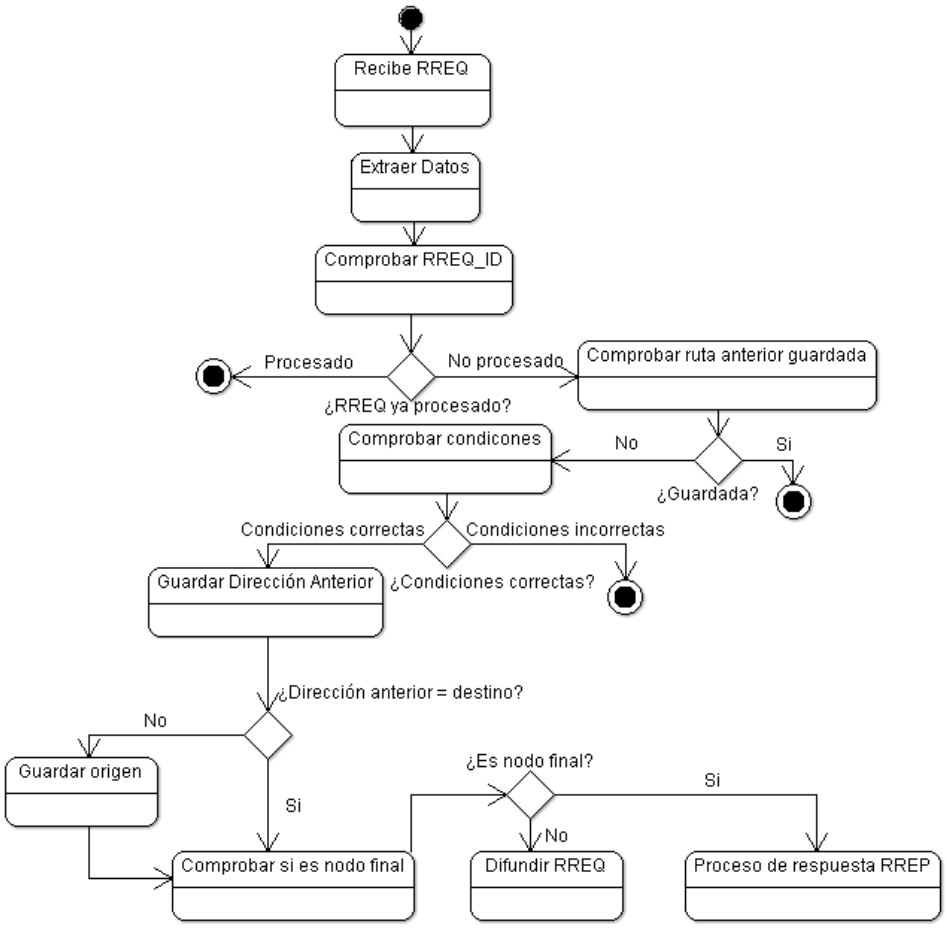

Figura 6.4: *Diagrama de estados al recibir un RREQ*

6.4.3 Recepción de RREP

Una vez que se manda un mensaje de respuesta a una petición de rutas (RREP), los nodos intermedios y final irán recibiéndolo y tendrán que tomar algunas decisiones.

Lo primero que se realiza es la extracción de los datos para saber quién es el origen y final de la comunicación. Se comprueba si el número de secuencia del destino es mayor y la cuenta de saltos es menor que la que ya se tenía almacenada. Si se cumplen estas condiciones, se procede a crear una entrada en la tabla de rutas hacia el dispositivo del que proviene el RREP. Acto seguido se comprueba si la dirección anterior es el destino. Si no lo fuese, habría que incorporar en la tabla de encaminamiento la dirección del destino.

Por otro lado, cuando se recibe un RREP pueden darse dos casos: que el dispositivo que recibe el mensaje sea el origen de la comunicación (el que inició la búsqueda con un RREQ) o que sea un elemento intermedio. En el caso de que sea el origen, se tiene la ruta deseada, por lo que se pueden empezar a mandar mensajes de datos. Si se trata de uno intermedio, se construye un mensaje RREP sumando a la cuenta de saltos una unidad para que cuando llegue al origen, se conozcan los dispositivos intermedios que tiene el camino por el que ha pasado el mensaje.

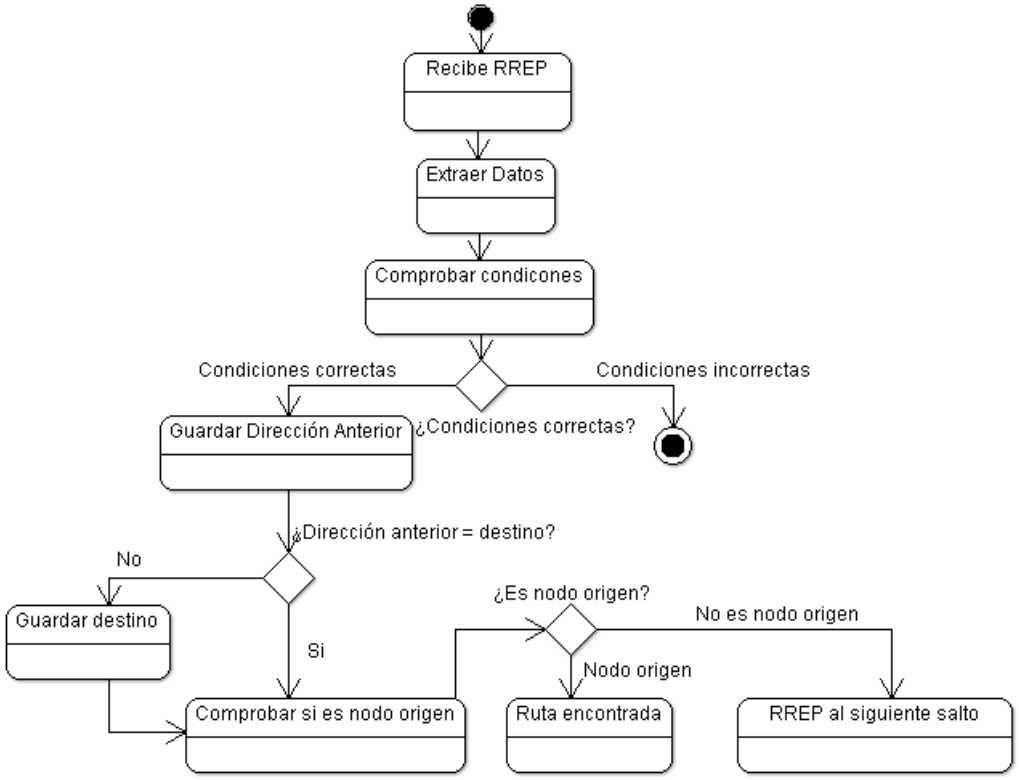

Figura 6.5: *Diagrama de estados al recibir un RREP*

6.4.4 Recepción de RERR

Cuando se pierde un enlace en una ruta activa o se reciben datos para un destino del que no se dispone una entrada en la tabla de encaminamiento, se debe mandar un mensaje de error (RERR) indicando los destinos a los que no se puede llegar. Si lo que se recibe es un mensaje de datos para el que no se sabe llegar al destino, la respuesta debe ser un RERR *unicast*. Sin embargo, si lo que se recibe es un RERR, la respuesta será *broadcast* (255.255.255.255). Una

vez que se manda, los dispositivos en la cobertura del emisor lo recibirán y tendrán que actuar en función del caso en el que se esté.

Lo primero que se tiene que hacer al recibir un RERR es comprobar la longitud para saber cuántos destinos inalcanzables tiene el dispositivo del que proviene el mensaje. Conviene recordar que los RERR no tienen un tamaño fijo, por lo que se tiene que comprobar el número de direcciones que se reciben.

Una vez que se sabe cuántas IPs se han perdido, se comprueba si son destinos del nodo receptor. Si alguno no es destino, no se hace nada con ese mensaje. Pero para los que sí lo son, se borran y se fabrica un nuevo RERR que incorpore las direcciones a las que no se puede acceder.

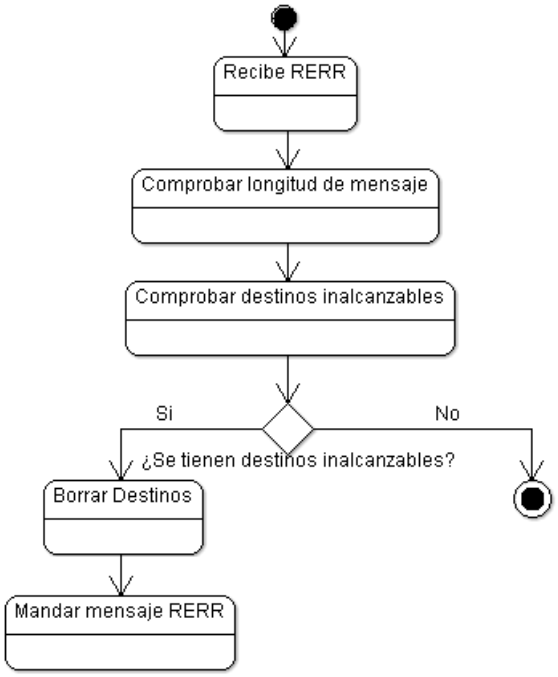

Figura 6.6: *Diagrama de estados al recibir un RERR*

6.4.5 Timer TBR

Unos elementos muy importantes en la solución al problema son los *timers* o temporizadores, que ofrecen funcionalidades periódicas necesarias a la hora de mantener, iniciar o terminar una comunicación.

En la Figura 6.7 se muestra el primer *timer* que se analiza y que corresponde con el de búsqueda de rutas (TBR). Cada vez que se inicie un proceso de descubrimiento de rutas mandando un RREQ se debe activar. Este temporizador expira cada 2 segundos, pero si se recibe respuesta, el *timer* se cancela y se comienza a comprobar si el siguiente nodo hasta llegar al destino final está operativo (*Método de activación Timer HELLO*). En cambio, si no se recibe respuesta, se repite el proceso hasta un máximo de nueve veces para subsanar el hecho de que se hubieran podido perder paquetes durante el trayecto y no se hubiera contestado a la petición de rutas hecha en el origen.

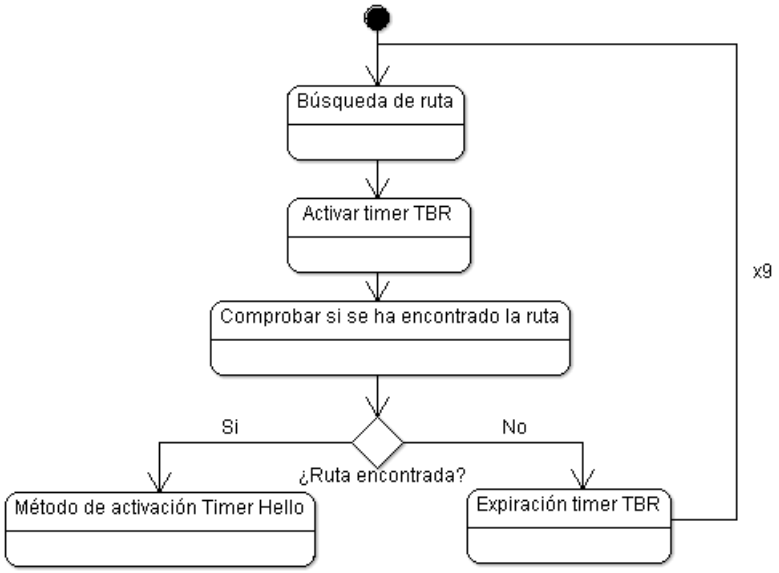

Figura 6.7: *Timer de Búsqueda de Rutas (TBR)*

6.4.6 Timer HELLO

El siguiente *timer* es el encargado de comprobar la disponibilidad de los vecinos en rutas activas (*Timer HELLO*) y expira cada segundo. Cada vez que lo hace, se comprueba si se han recibido mensajes HELLO por parte de los vecinos. Si no se recibiesen dos seguidos, se debe mandar un mensaje RERR. Para activar este *timer* hace falta que al menos haya una ruta activa, dado que en el momento en el que no se disponga de ninguna, se cancela. Este mecanismo es propio de un protocolo reactivo ya que únicamente se envían mensajes de control cuando realmente se necesitan.

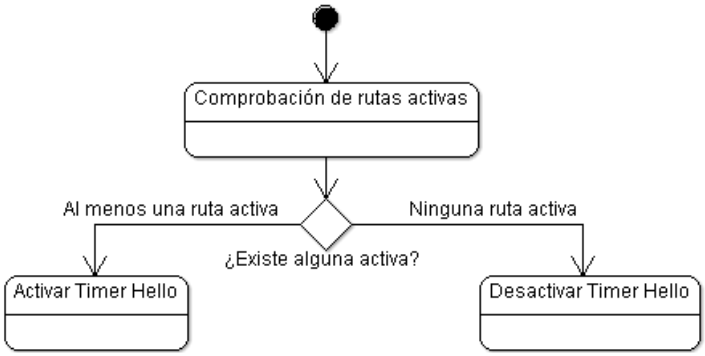

Figura 6.8: *Timer Hello*

6.4.7 Timer TMRA

El *Timer de Mantenimiento de Rutas Activas* (TMRA) es el encargado de mantener una conexión constante con el destino que se especifique. En cada origen de la comunicación se reactiva cada medio segundo y se manda un mensaje de mantenimiento de rutas. Cada vez que expira se vuelve a reactivar y se vuelve a enviar el mensaje. Este proceso se repite indefinidamente hasta que el origen anule la conexión establecida.

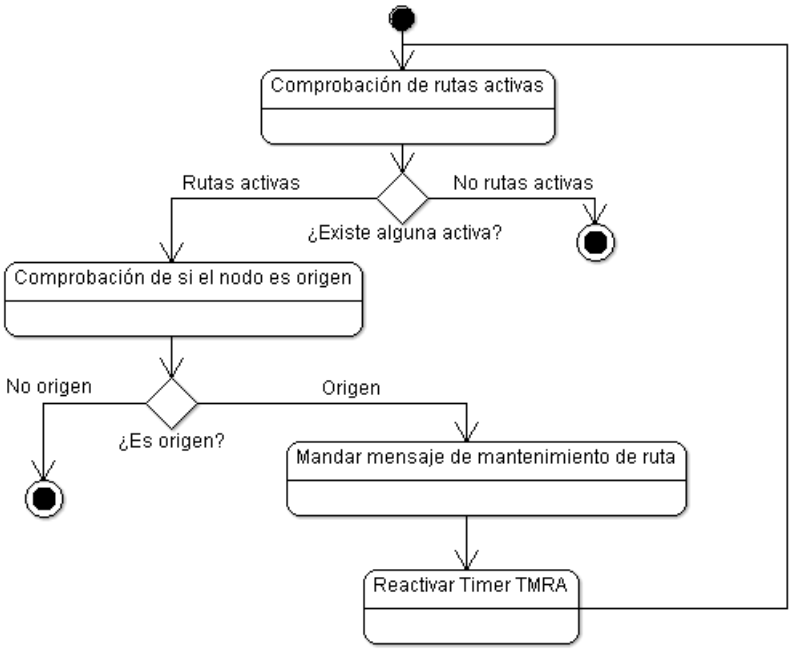

Figura 6.9: *Timer de Mantenimiento de Rutas Activas (TMRA)*

6.4.8 Timer TRA

El *Timer de Ruta Activa* (TRA) es el encargado de saber si no se recibe ningún mensaje de control ni de datos realizando una comprobación cada tres segundos. Si expira es porque no se ha recibido ningún tipo de mensaje, por lo que se comienza un proceso de borrado para indicar que se ha perdido la comunicación.

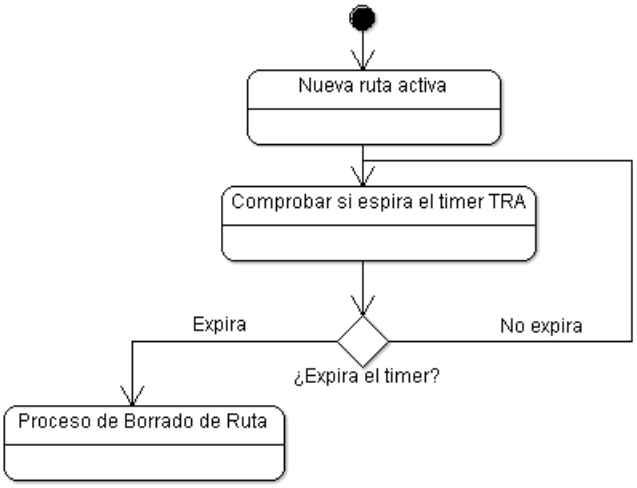

Figura 6.10: *Timer de Rutas Activas*

6.5 Limitaciones en el diseño

Según el modelo OSI, se pueden diferenciar siete capas con funcionalidades diferentes para la interconexión de equipos en red.

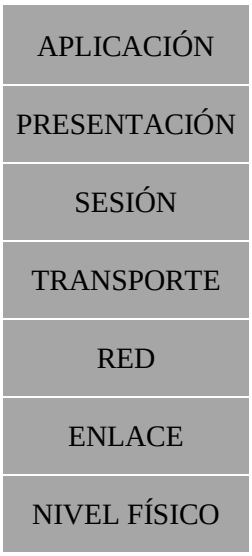

Figura 6.11: *Capas del modelo OSI*

Al implementar un protocolo IP de descubrimiento de rutas, el trabajo se centraría en el nivel 3 (capa de RED). De esta manera, cuando un dispositivo quisiese enviar un mensaje a otro por medio de alguna aplicación, miraría sus tablas de rutas y formaría un paquete IP en el que la dirección origen fuese la suya propia y la dirección final, la del dispositivo final. En cada salto que se produjese, los dispositivos intermedios mirarían el destino del paquete y comprobando sus tablas de rutas irían encaminando hasta llegar al elemento final. El último de la comunicación observaría que es el destino del paquete y actuaría según la naturaleza del mensaje.

Sin embargo, existe una limitación importante en Bada a la hora de abordar el problema del encaminamiento IP, ya que no se permite modificar las tablas de rutas del dispositivo. Ante tal circunstancia se ha pensado en la siguiente solución: implementar el protocolo AODV encima de la capa de aplicación. Así, los mensajes de la capa de aplicación deben incluir la dirección origen y destino (extremos de la comunicación) antes que el mensaje propio que se quiera enviar. En la Figura 6.12 se muestra cómo se deben enviar los mensajes en una aplicación que funcionase en base al protocolo implementado.

Dirección IP Destino	Dirección IP Origen	Mensaje propio de la aplicación

Figura 6.12: *Disposición de los datos en la capa de aplicación*

De esta forma, cuando un dispositivo reciba el mensaje mirará cuál es la dirección de destino (extraída del contenido de la capa de aplicación) y, según la tabla de rutas creada, encaminará al siguiente salto (cambiando los campos dirección origen y dirección destino de la cabecera IP), pero la información no cambiará. Durante todo el recorrido los campos *Dirección IP Destino*, *Dirección IP Origen* y *Mensaje propio de la aplicación* (incluidos en la capa de aplicación) permanecerán inalterados.

Otra limitación impuesta en el diseño es el campo *Time To Live* (TTL) de un paquete IP. El protocolo lo utiliza para no expandir los mensajes más allá del límite impuesto. Como se

explicó en el capítulo 3, los mensajes HELLO son un tipo de mensajes RREP (mensaje de respuesta) en los que el TTL tiene el valor 1. Al programar esta funcionalidad, se puso de manifiesto que en el simulador no se puede cambiar de manera dinámica el TTL de un paquete, puesto que está funcionando bajo un sistema operativo (Microsoft Windows) que al final es el responsable de mandar el mensaje a la red. Debido a este hecho, se ha modificado el mensaje enviado. Para el protocolo implementado, un mensaje HELLO es un mensaje RREP con las mismas direcciones origen y destino (la dirección del que manda el mensaje). Cuando cualquier nodo reciba un mensaje RREP en el que la dirección origen y la dirección destino son las mismas, lo identificará y además sabrá quién lo envió para así actualizar el contador de HELLOs recibidos por parte de ese dispositivo.

Por otro lado, Bada tiene métodos específicos para crear una red ad-hoc. Sin embargo, estos métodos no funcionan en el simulador aunque sí en el dispositivo móvil. Para subsanar este hecho, se ha utilizado un código distinto para su ejecución en los PCs y en el móvil. Cabe destacar que los cambios son muy sencillos. La única diferencia entre la versión para el simulador y para el dispositivo son las siguientes líneas que sirven para crear la red ad-hoc mencionada.

```
__pAdhocService     = new AdhocService(*this);

String companyName = String(__pCompanyNameField->GetText());
String appName     = String(__pAppNameField->GetText());

r =__pAdhocService ->StartAdhocService(companyName, appName);
if(IsFailed(r))
{
  AppLog("comienzo incorrecto");
}
```

Figura 6.13: *Creación de una red ad-hoc*

En los ordenadores, estas líneas estarán comentadas para que no se ejecuten. Nótese que, debido a la API de Bada, la red que se crea es de la forma *companyName_appName*. Este hecho se tuvo en cuenta en el apartado 4.1 para la especificación de requisitos de la conexión, donde se indicó que el nombre de la red tendría la forma *Entidad_Grupo*.

6.6 Implementación de los métodos

En este apartado se presentarán las partes de la implementación más importantes para crear y esperar que expire un *timer*, así como comprender la forma de encapsulación y extracción de los datos en los mensajes.

6.6.1 Implementación de los Timers

Para la implementación de todos los tipos de *timers*, se ha utilizado la clase *Timer* que viene incorporada en Bada y que dispone de tres métodos: construir, empezar y cancelar.

La forma de crear e inicializar un *timer* en Bada es la que se puede observar en la Figura 6.14.

```
__pTimer=new Timer; // Creación del objeto
r = __pTimer->Construct(*this); // Construcción del Timer
if (IsFailed(r))
{
	AppLog("Fallo en timer");
}
r=__pTimer->Start(4000); // Inicialización del Timer
if(r!=E_SUCCESS)
{
	AppLog("Fallo en timer");
}
```

Figura 6.14: *Creación de un timer en Bada*

Una vez que se inicializa, hay que esperar a que se ejecute el método *OnTimerExpired*. Como se puede ver en la Figura 6.15, recibe como argumento un objeto del tipo *Timer* y lo que se hace es buscar el que ha expirado para realizar alguna acción.

```
void
red::OnTimerExpired(Timer& timer)
 {
Boolean final=false;
for (int i=0;i<longitud && final==false;i++)
 {
	for (int j=0;j<4 && final==false;j++)
	{
	if (tiempo[i][j].Equals(timer))
		{
		  final = true;
		}
	}
}
```

Figura 6.15: *Método OnTimerExpired (búsqueda del timer que ha expirado)*

En este apartado conviene recordar con qué frecuencia se utiliza cada tipo concreto. El que se repite con más frecuencia es TMRA (*Timer de Mantenimiento de Ruta Activa*), ya que cuando se tiene una ruta activa expira cada 500ms. El siguiente en frecuencia es el temporizador TRA (*Timer de Ruta Activa*), que lo hace cada tres segundos. El último es TBR (*Timer de Búsqueda de Ruta*), que se utiliza únicamente cuando se está buscando una ruta y no se recibe respuesta. De este modo, la matriz de *timers* quedaría dispuesta según se especificó en la Tabla 6.11 de la parte de diseño.

Por otra parte, el *timer* HELLO es común a todos los destinos por lo que únicamente se va a necesitar uno, que expirará cada segundo.

6.6.2 Método MandaRREQ

A continuación se detalla la forma de implementar el método *MandaRREQ*.

```
void
red::MandaRREQ(int Flg, unsigned int saltos,unsigned long RREQid, unsigned
long Dest_Seq, unsigned long Orig_Seq, String IP_Destino, String IP_Origen,
NetEndPoint* destino)
{
  …
}
```

Figura 6.16: *Método MandaRREQ*

Se puede observar que los argumentos que recibe son los campos de un mensaje RREQ: bits indicadores (*Flg*), número de saltos (*saltos*) , identificador de RREQ (*RREQid*), número de secuencia del destino (*Dest_Seq*), número de secuencia del origen (*Orig_Seq*), dirección IP del destino (*IP_Destino*) y dirección IP del origen (*IP_Origen*). También recibe una dirección a la que enviar el mensaje que siempre será la dirección de difusión 255.255.255.255.

Ya en la Figura 6.17 se puede ver que la longitud del mensaje es fija (24 bytes) como se deriva de la sentencia *sBuffer.Construct(24)*. Se dividen las direcciones de origen y destino en 4 bytes diferentes que incluirán las direcciones IP correspondientes. Seguidamente, se van introduciendo cada uno de los datos en el orden establecido en el mensaje RREQ y se manda. Nótese que al insertar el identificador de RREQ y los números de secuencia, se tiene que llamar a la función *HtoNL* para transformar la secuencia de bits de *Little-Endian* a *Big-Endian*, que es el formato que se utiliza en las transferencias de datos.

```
// Buffer donde meter los datos
      ByteBuffer  sBuffer;
// Tamaño del buffer
       sBuffer.Construct(24);

// Campo tipo del mensaje
       sBuffer.SetByte(tipo);
// Campo de indicadores
       sBuffer.SetByte(flags);
       sBuffer.SetByte(prefijo);
// Cantidad de nodos atravesados
       sBuffer.SetByte(cuenta_saltos);
// Identificador de RREQ
       sBuffer.SetLong(cambio.HtoNL(RREQid));
// Dirección de destino
       sBuffer.SetByte(d1);
       sBuffer.SetByte(d2);
       sBuffer.SetByte(d3);
       sBuffer.SetByte(d4);
// Número de secuencia del destino
       sBuffer.SetLong(cambio.HtoNL(Dest_Seq));
// Dirección de origen
       sBuffer.SetByte(od1);
       sBuffer.SetByte(od2);
       sBuffer.SetByte(od3);
       sBuffer.SetByte(od4);
// Número de secuencia del origen
       sBuffer.SetLong(cambio.HtoNL(Orig_Seq));

// Envío de los datos
       sBuffer.Flip();
       res = __pSocket->SendTo(sBuffer, *destino);
       sBuffer.Clear();
```

Figura 6.17: *Encapsulado de los datos*

6.6.3 Método RecibeRREQ

A continuación se especifica la parte más significativa del método *RecibeRREQ*. En este caso, los únicos argumentos que recibe son un *buffer* (contiene el mensaje RREQ) y el nodo que lo envía (*direccionAnterior*). Se puede ver cómo se tienen que ir extrayendo cada uno de los campos que se introdujeron en el *MandaRREQ*, haciendo la conversión de *Big-Endian* a *Little-Endian* en los casos que corresponda.

```
void
red::RecibeRREQ(ByteBuffer& recibe_RREQ, String direccionAnterior)
{
// Extracción del primer byte
       byte ra;
       recibe_RREQ.GetByte(1,ra);
       unsigned int Cra;
       Cra=ra;
// Extracción del número de saltos
       byte saltos;
       recibe_RREQ.GetByte(3,saltos);
       unsigned int Csaltos;
       Csaltos=saltos;
// Extracción del identificador de RREQ
       SocketUtility transformar;
       transformar.Construct();
       long int cogeRREQ;
       unsigned long cogeRREQ2;
       recibe_RREQ.GetLong(4,cogeRREQ);
       cogeRREQ2=cogeRREQ;
       cogeRREQ2=transformar.NtoHL(cogeRREQ2);
// Extracción de la dirección IP del destino
       byte ipout;
       String ipDestino;
       recibe_RREQ.GetByte(8,ipout);
       ipDestino.Append(ipout);
       ipDestino.Append(".");
       recibe_RREQ.GetByte(9,ipout);
       ipDestino.Append(ipout);
       ipDestino.Append(".");
       recibe_RREQ.GetByte(10,ipout);
       ipDestino.Append(ipout);
       ipDestino.Append(".");
       recibe_RREQ.GetByte(11,ipout);
       ipDestino.Append(ipout);
// Extracción del número de secuencia del destino
       long int cogeSeq;
       unsigned long cogeSeq2;
       recibe_RREQ.GetLong(12,cogeSeq);
       cogeSeq2=cogeSeq;
       cogeSeq2=transformar.NtoHL(cogeSeq2);
// Extracción de la dirección IP del origen
       String ipOrigen;
       recibe_RREQ.GetByte(16,ipout);
       ipOrigen.Append(ipout);
       ipOrigen.Append(".");
       recibe_RREQ.GetByte(17,ipout);
       ipOrigen.Append(ipout);
       ipOrigen.Append(".");
       recibe_RREQ.GetByte(18,ipout);
       ipOrigen.Append(ipout);
       ipOrigen.Append(".");
       recibe_RREQ.GetByte(19,ipout);
       ipOrigen.Append(ipout);
// Extracción del número de secuencia del origen
       long int ocogeSeq;
       unsigned long ocogeSeq2;
       recibe_RREQ.GetLong(20,ocogeSeq);
       ocogeSeq2=ocogeSeq;
       ocogeSeq2=transformar.NtoHL(ocogeSeq2);
        …
}
```

Figura 6.18: *Método RecibeRREQ (extracción de datos)*

Nótese como en el caso de las direcciones de origen y destino, se transforman los bytes en un dato del tipo *String* para poder trabajar posteriormente con ellos. Una vez que se tienen todas las especificaciones del mensaje, se actúa como se indicó en el apartado de diseño apartado 6.4.2.

6.6.4 Método MandaRREP

A continuación se muestra el método *MandaRREP*. A partir de aquí, se va a omitir la forma de encapsular y extraer los datos porque ya se han especificado en los apartados 6.6.2 y 6.6.3.

```
void
red::MandaRREP(int flg, unsigned int saltos, unsigned long Dest_Seq, unsigned
long Life_time, String IP_Destino, String IP_Origen, NetEndPoint* destino)
{
…
}
```

Figura 6.19: *Método MandaRREP*

En este caso, lo más significativo se encuentra en el valor de *NetEndPoint* destino*, ya que debe contener la dirección del siguiente salto al que hay que enviar el mensaje RREP. Este campo coincidirá con la dirección del origen (*IP_Origen*) si el siguiente nodo es directamente el origen de la comunicación.

6.6.5 Método RecibeRREP

Al igual que en el método *RecibeRREQ*, los únicos parámetros que recibe son un *buffer* que contiene el mensaje RREP y la dirección de la que proviene.

```
void
red::RecibeRREP(ByteBuffer& recibe_RREP, String direccionAnterior)
{
…
}
```

Figura 6.20: *Método RecibeRREP*

La forma de proceder es la que se indicó en el apartado 6.4.3, aunque cabe destacar la forma de actuar con los mensajes HELLO.

```
        for (int i=0;i<longitud && ipDestino==ipOrigen;i++)
        {
                if (Estado[i]==2)
                {
                 Hello_recibidos[i]=3;
                }
        }
```

Figura 6.21: *Actualización de Hellos recibidos*

Para identificar los mensajes HELLO, se comprueba si las direcciones de origen (*ipOrigen*) y de destino (*ipDestino*) son iguales. Además, si el estado es 2 quiere decir que la ruta está activa, por lo que se vuelve a activar la cuenta de mensajes HELLO recibidos.

6.6.6 Método MandaRERR

El método *MandaRERR* debe diferenciar entre dos tipos de casos: si tiene que enviar el mensaje a todos los dispositivos o si tiene que enviarlo únicamente a un nodo. Por este motivo, recibe un argumento nuevo (*int índice*). Si *índice* vale 1, el mensaje se enviará a la dirección *broadcast* 255.255.255.255. Si *índice* vale 2, se enviará el RERR a la dirección indicada en la variable *direccionAnterior*.

```
void
red::MandaRERR(String DestinoPerdido, int indice, String direccionAnterior)
{
…
}
```

Figura 6.22: *Método MandaRERR*

La diferencia a la hora de implementar uno y otro caso se muestra en la Figura 6.23.

```
////////////////////// índice=1 ////////////////////////////
      sBuffer.Flip();
// __pLocalPoint es la dirección 255.255.255.255
       __pSocket->SendTo(sBuffer, *__pLocalPoint);
       sBuffer.Clear();
////////////////////// índice=2 ////////////////////////////
       Ip4Address dir_uni(direccionAnterior);
       unsigned short port = 654;
       NetEndPoint  point (dir_uni,port);
       NetEndPoint *point2=new NetEndPoint(point);
       sBuffer.Flip();
       __pSocket->SendTo(sBuffer, *point2);
       sBuffer.Clear();
```

Figura 6.23: *Diferencia entre enviar a uno o a todos los dispositivos*

6.6.7 Método RecibeRERR

El método *RecibeRERR* recibe un *buffer* que contiene un mensaje RERR y la dirección de la que se recibe (*direccionAnterior*). De esta manera, *direccionAnterior* va a servir para saber que mediante ese nodo no se puede acceder a los destinos que se especifiquen dentro del mensaje RERR y, por lo tanto, habrá que actuar como se especifica en el apartado de diseño 6.4.4.

```
void
red::RecibeRERR(ByteBuffer& recibe_RERR, String direccionAnterior)
{
…
}
```

Figura 6.24: *Método RecibeRERR*

7 Diseño e implementación de las aplicaciones

Durante el capítulo 7 se explicará cómo se han diseñado e implementado dos aplicaciones para testear y probar el correcto funcionamiento del protocolo AODV. En concreto, se trata de una aplicación de eco y otra de chat.

En la primera parte se hará mención a la forma de diseñar un servicio para el mantenimiento de las rutas creadas por AODV. Se explicarán los métodos empleados, los mensajes, las diferentes funcionalidades y la forma de llevar a cabo la implementación.

En la segunda parte se presentará cómo se ha abordado el diseño y la implementación de la aplicación que permitirá enviar y recibir mensajes de texto dentro de una comunidad de usuarios. Al igual que en la aplicación de eco, se explicarán los métodos empleados, los mensajes, las diferentes funcionalidades y cómo se ha llevado a cabo la implementación.

7.1 Aplicación de Eco

Para comprobar el correcto funcionamiento del protocolo AODV se ha diseñado una aplicación de eco. Es conveniente recordar que AODV es reactivo, lo que quiere decir que no mantiene en las tablas de rutas nada más que las conexiones que tenga activas. Por lo tanto, para la comprobación de las rutas activas existentes es necesario que haya una aplicación que mantenga la conexión.

La forma de actuar es sencilla: cuando se dispone de una ruta activa, el origen de la comunicación comienza a mandar mensajes periódicos de datos para que la conexión se mantenga activa. Los dispositivos intermedios encaminan el paquete y el final lo devolverá al origen. Este proceso se repite continuamente hasta que el usuario que inició la conexión decida eliminarla.

En cuanto a la forma de enviar los mensajes, se utiliza UDP en el puerto 15000 tanto si se actúa como emisor o receptor.

7.1.1 Tipos de métodos necesarios

Para tener una aplicación de eco se necesitan dos métodos, uno para enviar datos y otro para recibirlos.

MandaDatos: Método encargado de mandar los datos de mantenimiento de rutas al siguiente salto. Antes de enviar, debe comprobar si el siguiente salto en la comunicación está activo.

RecibeDatos: Método encargado de analizar la información de la aplicación de eco que llega a un dispositivo. Analizará si es el nodo final de la comunicación o si es intermedio. En el primer caso, devolverá el mensaje al origen y en el segundo, mirará las tablas de rutas creadas con AODV y encaminará al siguiente dispositivo.

7.1.2 Forma del mensaje de la aplicación de eco

Como se especificó en el apartado 6.5 del capítulo de diseño de AODV, el protocolo está implementado en la capa de aplicación del modelo OSI. Esto quiere decir que dentro del mensaje se deben mandar las direcciones origen y final para que, salto a salto, se pueda producir el enrutamiento. La forma del mensaje es la que aparece en la Figura 7.1.

1Byte	4 Bytes	4Bytes	
Tipo	Dirección IP Destino	Dirección IP Origen	Mensaje propio de la aplicación

Figura 7.1: *Disposición de los datos en la aplicación de eco*

Tipo: Tiene el valor 8. Se utiliza para diferenciar el tipo de mensaje con la otra aplicación implementada (chat).

Dirección IP Destino: Contiene la dirección IP del destino de la comunicación. Este campo debe permanecer inalterado hasta encontrar el punto final.

Dirección IP Origen: Contiene la dirección IP del origen de la comunicación. Este campo debe permanecer inalterado hasta encontrar el punto final.

Mensaje propio de la aplicación: Se manda un dato. Se ha elegido "5" como el dato a enviar.

7.1.3 Diagrama de estados de la aplicación de eco

Para comprender la funcionalidad de la aplicación, se debe indicar que se tienen dos casos diferentes. Por un lado, se puede ser el origen de la comunicación y por tanto, se envían mensajes periódicamente para mantener la conexión activa. Por el contrario, si no se es origen, se debe actuar según corresponda a un dispositivo intermedio o final.

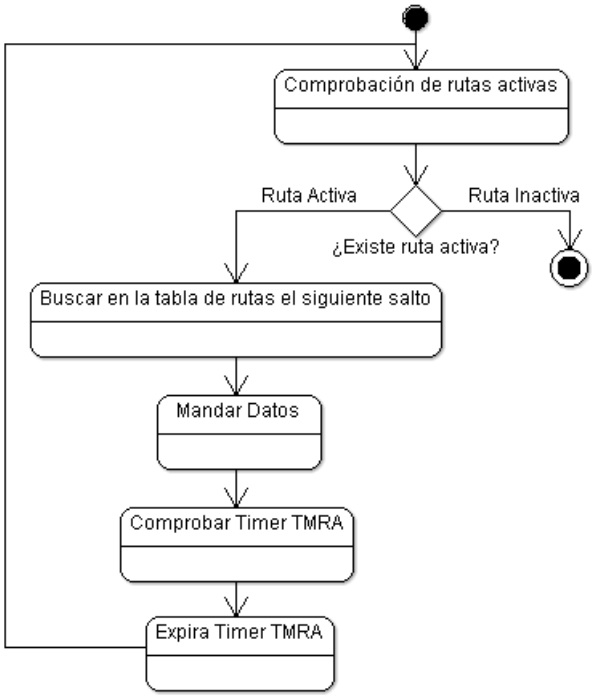

Figura 7.2: *Actuación del nodo origen de la aplicación de eco*

La Figura 7.2 muestra cuál es la actuación del nodo origen. Cuando se tiene una ruta activa se comprueba si el siguiente salto también lo está y se mandan los datos para que se comiencen a encaminar hasta el destino. Una vez mandados se comprueba si expira el TMRA (*Timer de Mantenimiento de Ruta Activa*). Cuando expira, se vuelve a comenzar el proceso para volver a mandar datos periódicos que serán encaminados hacia la dirección IP destino que se especifique.

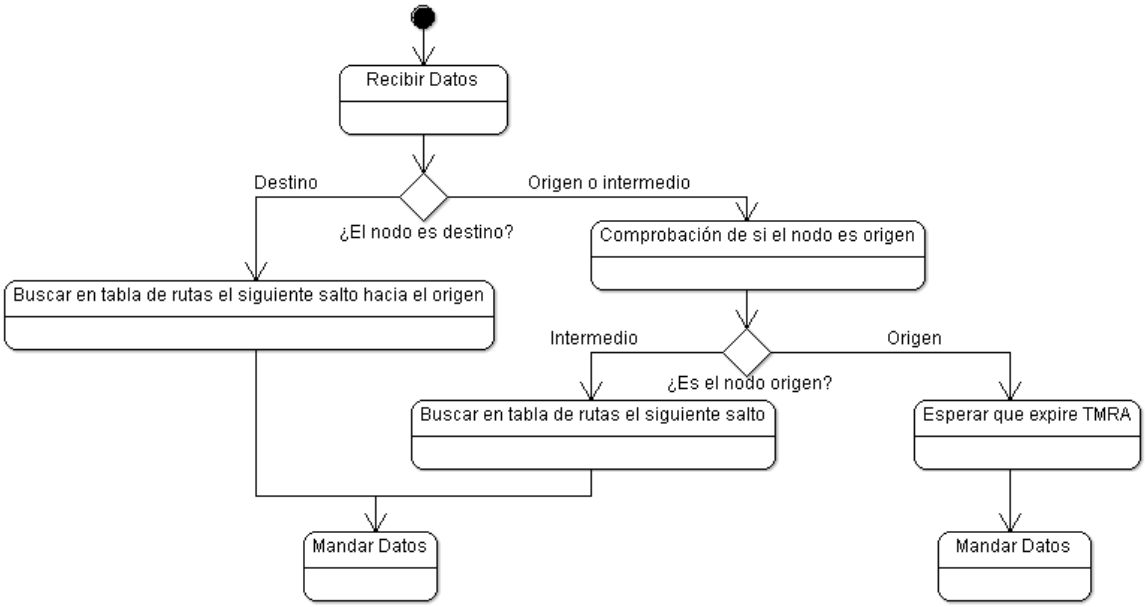

Figura 7.3: *Actuación ante la llegada de un mensaje de la aplicación de eco*

En la Figura 7.3 se muestra la forma de actuación ante la llegada de un paquete de eco. Se comprueba si el nodo es el destino de la comunicación. Si es así, se busca en la tabla de rutas el dispositivo siguiente para llegar al elemento de la red que mandó el mensaje y se envía. Si no se trata del destino, existen dos posibilidades, ser origen o intermedio. Si se trata de un dispositivo intermedio, éste busca en sus tablas de rutas, creadas por AODV, cuál es el siguiente nodo al que tiene que enviar. Si al que le llega el mensaje es el nodo origen, se habrá producido el camino de ida y vuelta y esperará hasta el siguiente momento en el que debe enviar los datos (cuando expira el TMRA).

7.1.4 Implementación del método MandaDatos

Como se especificó en el apartado 7.1.2, al mandar un mensaje de datos de la aplicación de eco, se necesita enviar la dirección del destino (*IP_Destino*), la del origen (*IP_Origen*) y los datos (*Datos*). Las otras dos variables que se especifican en la Figura 7.4 son *destino* (siguiente salto) e *índice* (variable para saber cuál va a ser el primer byte del mensaje). Si *índice* vale 1, el primer byte será 8, como corresponde a un mensaje de datos de eco. Los casos de *índice* 2 y 3 se estudiarán en el apartado 7.2.5 (chat). No se incorpora la forma de encapsular los datos porque ya se estudió en el apartado 6.6.2.

```
void
red::MandaDatos(String IP_Destino, String IP_Origen, String Datos,
NetEndPoint* destino, int indice)
{
       byte primerByte=0;
       if (indice == 1)
       {
        primerByte=8;
       }
       if (indice == 2)
       {
        primerByte=7;
       }
       if (indice == 3)
       {
        primerByte=0;
       }
…
}
```

Figura 7.4: *Método MandaDatos*

7.1.5 Implementación del método RecibeDatos

RecibeDatos es el método llamado cuando se recibe un mensaje de eco. En cuanto a los argumentos que recibe, *Recibe_Datos* es un *buffer* que contiene el mensaje completo. La variable *direccionAnterior* contiene la dirección de la que proviene. El tercer argumento (*longitud*) muestra la longitud de los datos contenidos e *índice* sirve para diferenciar entre los datos de tipo chat y los de eco. Una vez que se recibe un paquete, se actúa como se indicó en la Figura 7.3 del apartado de diseño.

```
void
red::RecibeDatos(ByteBuffer& recibe_Datos, String direccionAnterior, unsigned
long longitud, int indice)
{
…
}
```

Figura 7.5: *Método RecibeDatos*

7.2 Aplicación de Chat

Se ha creado otra aplicación que pone de manifiesto la usabilidad del protocolo implementado. Se trata de una aplicación de chat que funciona bajo AODV y que tiene la funcionalidad de explorar qué vecinos se encuentran próximos y establecer una conexión con ellos. Además, es capaz de mandar mensajes a todos los dispositivos con los que se tengan rutas activas o únicamente al que se seleccione.

Existen dos mensajes propios de esta aplicación: el primero permite la búsqueda de vecinos y el segundo es un mensaje de texto propio de una aplicación de intercambio de mensajes.

Se utiliza UDP y el puerto 15000 como origen y destino de la comunicación.

7.2.1 Tipos de métodos necesarios

Se ha diseñado la aplicación de forma que se necesiten tres métodos para hacer posible la funcionalidad necesaria.

MandaDatos: Es el método encargado de enviar el mensaje al siguiente salto en la comunicación. Debe comprobar si dicho salto está activo.

RecibeDatos: Método encargado de recibir los mensajes propios de la aplicación, es decir, el intercambio de texto entre dos usuarios.

RecibeDatos2: Método encargado de recibir los datos destinados a explorar el número de vecinos que se encuentran en el rango de cobertura del origen.

7.2.2 Forma del mensaje de la aplicación de chat

Como se especificó en el apartado 6.5 del capítulo de diseño de AODV, el protocolo está implementado en la capa de aplicación del modelo OSI. Esto quiere decir que dentro del mensaje propio de la aplicación se deben mandar las direcciones origen y final para que se pueda producir el enrutamiento salto a salto. La forma del mensaje es la que se muestra en la Figura 7.6.

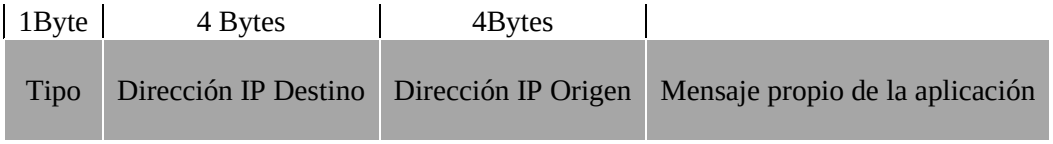

Figura 7.6: *Disposición de los datos en la aplicación de chat*

Tipo: Tiene el valor 7 para el intercambio de mensajes de texto y 0 para la exploración de vecinos.

Dirección IP Destino: Contiene la dirección IP del destino de la comunicación. Este campo debe permanecer inalterado hasta encontrar el punto final. Si el mensaje es de tipo exploración de vecinos, el valor es la dirección de difusión 255.255.255.255.

Dirección IP Origen: Contiene la dirección IP del origen de la comunicación. Este campo debe permanecer inalterado hasta encontrar el punto final.

Mensaje propio de la aplicación: Si el tipo vale 7, se mandan los datos de texto que se especifiquen. Si el tipo vale 0, se manda un dato. Se ha elegido "5" como el dato a enviar. Para responder a un mensaje de exploración de vecinos (tipo 0) se introduce también el nombre del dispositivo que responde.

7.2.3 Diagrama de estados del intercambio de mensajes de texto

En el caso de esta aplicación existen dos situaciones que se pueden dar. La primera corresponde al momento cuando se quiere enviar un mensaje a un destino determinado y la segunda cuando se recibe un mensaje de texto. Cada una de ellas será explicada mediante la Figura 7.7 y la Figura 7.8.

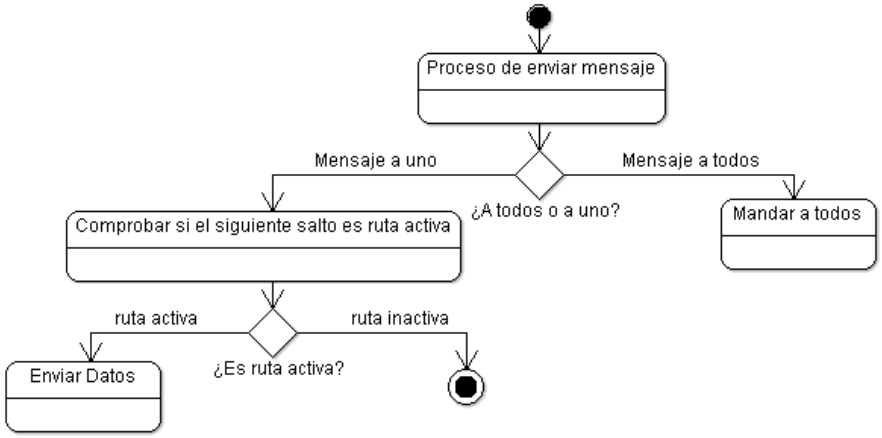

Figura 7.7: *Forma de actuación a la hora de enviar un mensaje de texto*

Cuando se quiere enviar un mensaje, lo primero que se debe comprobar es si va destinado a todas las rutas activas o únicamente a un dispositivo. En el primer caso se envía el mensaje a todas las rutas. Sin embargo, si va destinado a un nodo, se comprueba si el siguiente salto está activo y se envía.

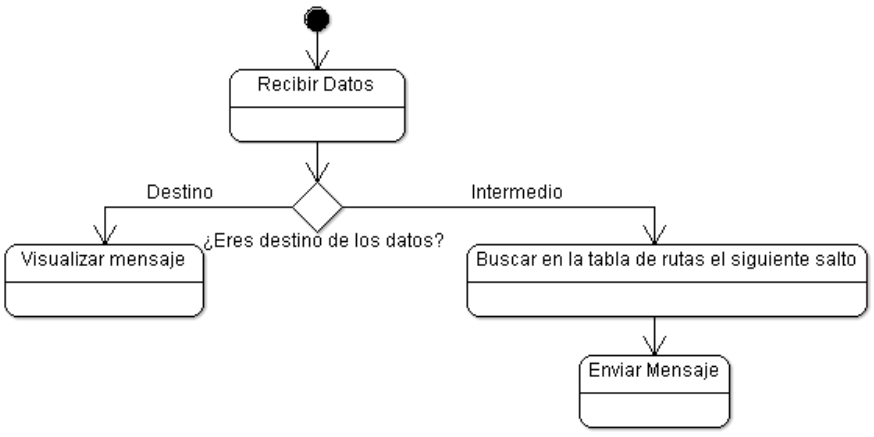

Figura 7.8: *Forma de actuación a la hora de recibir un paquete de datos de chat*

En la Figura 7.8 se muestra la forma de actuar cuando se recibe un paquete de datos. En este caso se comprueba si el dispositivo es el destino o un nodo intermedio. Si ocurre lo primero, se visualiza el mensaje y si sucede lo segundo, se sigue encaminando hasta que llegue al destino final.

7.2.4 Diagrama de estados de la búsqueda de vecinos

En la Figura 7.9 se muestra la forma de actuar para buscar y actualizar la lista de los vecinos que se encuentren. Para ello existe un *timer (__pTimer)* que expira cada cuatro segundos. Cada vez que lo hace se manda un mensaje con el campo *Tipo* igual a cero y se espera la contestación de los dispositivos que estén en el alcance del emisor. Se guardan los nombres y las direcciones de los que respondan y se comienza de nuevo el proceso esperando a que expire *__pTimer* para volver a mandar un mensaje.

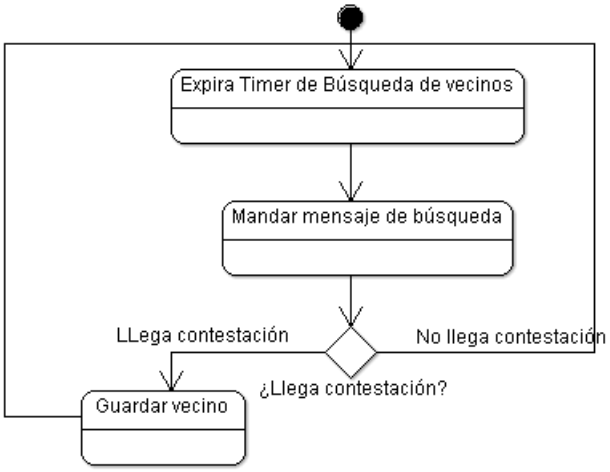

Figura 7.9: *Forma de actuación a la hora de explorar vecinos*

7.2.5 Implementación del método MandaDatos

El método *MandaDatos* es el mismo que se estudió en el apartado 7.1.4 para el caso de la aplicación de eco. En la Figura 7.4 se podía observar cómo en función del valor de una variable (*índice*) se decidía el tipo de mensaje a enviar.

Índice = 2: Se está en el caso de envío de mensajes de texto, por lo que el primer byte debe tener el valor 7.

Índice = 3: Se está en el caso de exploración de vecinos, por lo que el primer byte debe tener el valor 0.

7.2.6 Implementación del método RecibeDatos

El método *RecibeDatos* es el mismo que se estudió en el apartado 7.1.5 para el caso de la aplicación de eco. En este caso también existe una variable (*índice*) que especifica si se recibe un mensaje de eco o de datos.

Índice = 1: Se está en el caso de la aplicación de eco.

Índice = 2: Se está en el caso de la aplicación de chat.

7.2.7 Implementación del método RecibeDatos2

La implementación del método *RecibeDatos2* es la que se muestra en la Figura 7.10. Este método, encargado de recibir los datos destinados a explorar el número de vecinos que se encuentran en el rango de cobertura, recibe el mensaje (*recibe_Datos*) del nodo anterior, el dispositivo que lo envía (*direccionAnterior*), la longitud (*longitud*) y un *índice* para saber que es un mensaje de tipo exploración de vecinos. La forma de actuación es la que se muestra en el apartado de diseño en la Figura 7.9.

```
void
red::RecibeDatos2(ByteBuffer& recibe_Datos, String direccionAnterior, unsigned
long longitud, int indice)
{
…
}
```

Figura 7.10: *Método RecibeDatos2*

8 Pruebas realizadas y resultados obtenidos

Durante el capítulo 8 se pretenden mostrar las pruebas realizadas y los resultados obtenidos. Para ello, se han escogido dos escenarios distintos. En el primero, los dispositivos se encontrarán en un área reducida y tendrán acceso directo al nodo que se desee. En el segundo, se mostrarán los resultados en una topología en la que los mensajes tengan que ser encaminados por algún nodo intermedio.

8.1 Dispositivos en el mismo rango de cobertura

El primer conjunto de pruebas realizadas corresponden al escenario que se observa en la Figura 8.1. En este escenario se dispone de dos ordenadores (PC1 y PC2) y un móvil. Los tres nodos se encuentran en un área en el que cada uno tiene cobertura hacia los otros dos.

Con esta topología se van a probar las siguientes situaciones: 1) creación de una conexión; 2) interrupción y restablecimiento de una conexión; 3) notificación de una conexión perdida; y 4) envío de un mensaje de chat por parte del PC1 a sus vecinos (PC2 y móvil).

Figura 8.1: *Escenario de pruebas en el mismo rango de cobertura*

8.1.1 Creación de una conexión

En este apartado se va a realizar la conexión entre dos nodos de los que se muestran en la Figura 8.1. En este caso, el móvil será el encargado de solicitar una conexión con el PC1.

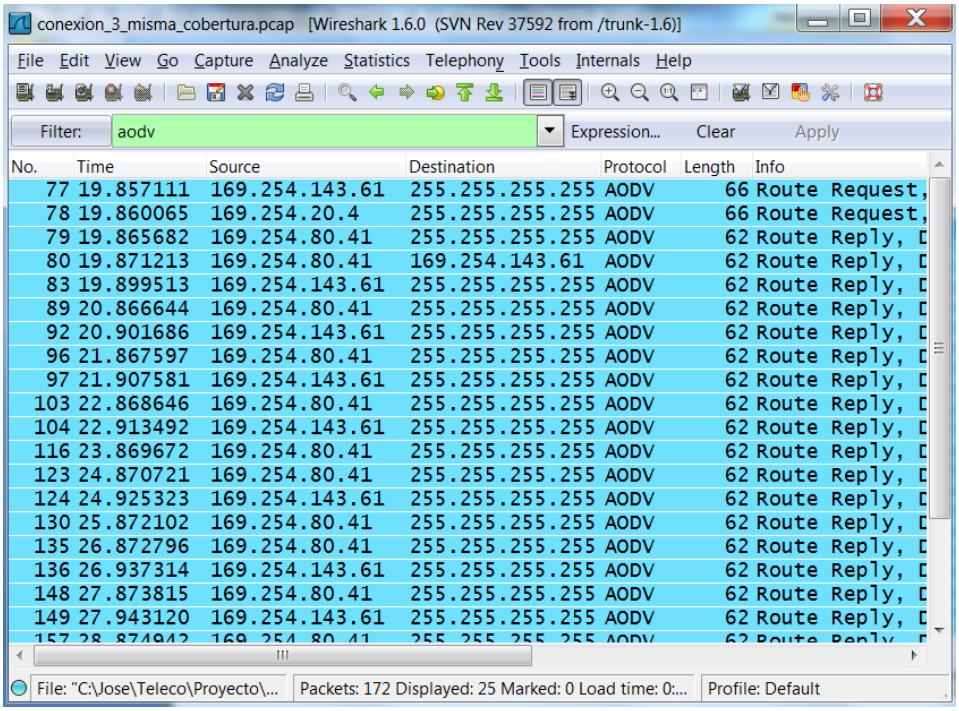

Figura 8.2: *Ejemplo de conexión entre dos dispositivos con visión directa*

Para su análisis se van a identificar los paquetes por su número (columna *No* en *Wireshark*). La captura muestra únicamente los mensajes AODV, tal y como indica el campo *Filter* de la Figura 8.2.

Paquete 77: El móvil (169.214.143.61) realiza una petición de ruta hacia el PC1 (169.254.80.41). Como se trata de un mensaje RREQ, la dirección destino es 255.255.255.255.

Paquete 78: El PC2 recibe el paquete 77 y ve que se está solicitando una ruta hacia el PC1. Ya que no dispone de ninguna entrada para ese destino, debe preguntar también cómo llegar hasta el PC1 por lo que vuelve a enviar el RREQ, pero con la cuenta de saltos aumentada en una unidad.

Una vez que los dos paquetes RREQ llegan al destino final (PC1), éste comprueba cuál de ellos tiene una cuenta de saltos menor. Lógicamente, el mensaje que llega directamente desde el móvil solamente requiere un salto mientras que el que ha pasado por el PC2 requiere dos saltos. De esta manera, el PC1 elige como ruta la que le conecta directamente con el móvil.

Paquete 79: Como el PC1 ya tiene decidido que va a tener una conexión activa con el móvil, empieza a mandar mensajes HELLO *broadcast* para que sus vecinos se percaten de que está activo.

Paquete 80: El PC1 manda de manera *unicast* al móvil (169.254.143.161) la respuesta a la petición de rutas que solicitó en el paquete 77.

A partir del paquete 80 se pueden observar mensajes periódicos HELLO entre el móvil y el PC1. Nótese que no aparece ningún mensaje HELLO proveniente del PC2 porque no tiene ninguna ruta activa.

De esta manera, se ha mostrado cómo se busca y se encuentra una ruta cuando se está en el mismo rango de cobertura. Además, se pone de manifiesto que el protocolo elige de manera adecuada cuál es el camino más corto hacia el destino, ya que la ruta que se pretendía crear a través del PC2 no se ha realizado.

8.1.2 Interrupción de una conexión

Mediante el siguiente ejemplo, se pretende demostrar cómo el protocolo es capaz de actuar ante los posibles fallos que se puedan producir intentando buscar un camino hacia un destino cuando éste se pierda. En la Figura 8.3 se muestra la topología de la red que se va a utilizar para este caso. El móvil y el PC1 tienen inicialmente una conexión activa, que a partir de un cierto instante se pierde.

Figura 8.3: *PC1 interrumpe la conexión con el móvil*

Los paquetes a los que hay que prestar atención en la Figura 8.4 son aquéllos cuyo identificador varía desde 197 hasta 215. A partir del paquete 197 solamente manda mensajes HELLO el móvil (169.254.143.161), ya que el PC1 no está activo.

Paquete 212: En algún momento anterior, en el móvil expira el temporizador de ruta activa. Dado que el móvil quiere seguir conectado, comienza un nuevo proceso de descubrimiento mandando un RREQ que pregunte por el PC1.

A partir de aquí, el proceso es similar al del apartado anterior. El PC2 vuelve a mandar un RREQ para buscar también al destino, pero el PC1 solamente responde al móvil (paquete 215) porque nuevamente tiene una cuenta de saltos menor que el camino que se crearía a través del PC2. Tras volver a establecer la conexión, es significativo que el PC1 y el móvil vuelven a mandarse mensajes HELLO periódicos (tras el paquete 218). Este hecho corrobora el restablecimiento correcto de la ruta perdida, ya que dichos mensajes únicamente se mandan cuando se dispone de una ruta activa hacia algún destino concreto.

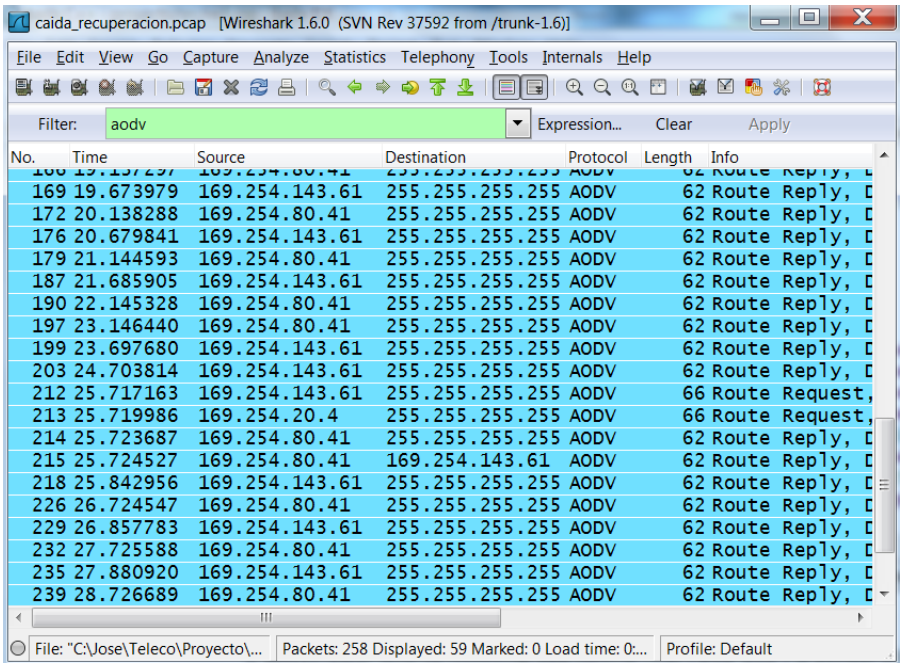

Figura 8.4: *Recuperación de la conexión perdida*

8.1.3 Notificación de conexión perdida

Mediante el siguiente ejemplo se pretende enseñar cómo cuando un nodo pierde un destino y no recupera la comunicación, notifica a los demás mandando un mensaje de error (RERR).

Según el diseño que se ha hecho, cuando un dispositivo establece una comunicación con un nodo final, ésta se mantendrá operativa a menos que el origen decida cancelarla. Si este hecho ocurre, el destino no recuperará la ruta, pero notificará a los demás que se ha perdido la conexión. La Figura 8.3 muestra el escenario al que corresponde la prueba realizada, pero el que interrumpe la conexión es el móvil.

Los paquetes más significativos son los que tienen los identificadores desde 242 a 264. Se puede observar en la Figura 8.5 cómo a partir del 242 no se reciben mensajes HELLO por parte del móvil (169.254.143.61), ya que la conexión se ha perdido.

Tal como se vio en el apartado 3.4.2, cuando se agota el tiempo de ruta activa o no se puede advertir de la presencia del nodo vecino, se manda un mensaje RERR. Esto le ocurre al PC1, que envía el paquete 264 tal como se describe a continuación.

Paquete 264: El PC1 manda el RERR a la dirección 255.255.255.255 para que todo el que esté en su rango de cobertura se dé cuenta de la caída del enlace. Como estaba previsto, en el interior del mensaje se especifica que la dirección a la que no se puede acceder es la del móvil (169.254.143.61).

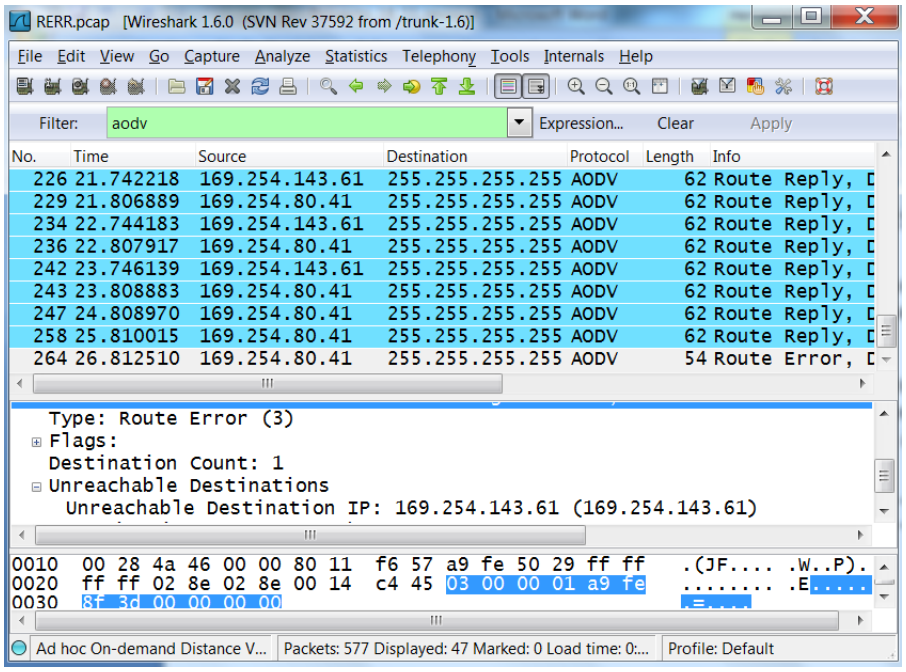

Figura 8.5: *Notificación de error*

8.1.4 Envío de un mensaje de chat a todos los vecinos

Mediante el siguiente ejemplo se pretende enseñar cómo la aplicación de chat puede enviar un paquete a todas las rutas que se tengan activas. Este hecho pone de manifiesto cómo una aplicación puede sustentarse en el protocolo creado y puede mandar información de manera correcta al destino deseado.

Figura 8.6: *Mensaje desde PC1 a los demás dispositivos*

En la captura tomada en la Figura 8.7 se muestran únicamente los mensajes mandados por las aplicaciones. Por eso, en el campo *Filter* se especifica que solamente se muestren los paquetes con puerto origen y destino 15000.

El PC1 (169.254.80.41) tiene una conexión activa con el PC2 (169.254.20.4) y con el móvil (169.254.143.61). A través de estas conexiones el PC1 manda el siguiente mensaje a todos los destinos: "cómo estáis todos, yo muy bien".

En este sentido, se pueden apreciar una serie de mensajes de datos con una longitud de 53 bytes que corresponden a paquetes destinados a mantener la conexión de las rutas y son enviados periódicamente.

Paquete 218: Es el paquete con el mensaje de chat que se manda al primer destino que se tiene almacenado, el móvil (169.254.143.61). La longitud es de 87 bytes.

Paquete 219: Es el paquete con el mensaje de chat que se manda al segundo destino que se tiene almacenado, el PC2 (169.254.20.4). La longitud es de 87 bytes.

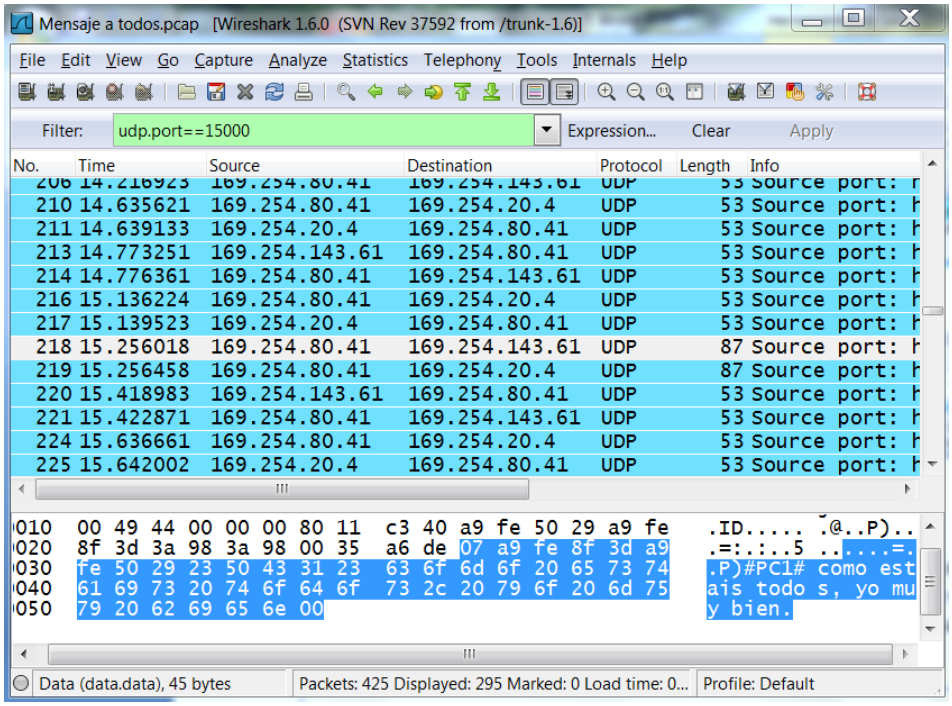

Figura 8.7: *Visualización de los mensajes mandados a todos los dispositivos*

8.2 Dispositivos sin visión directa

En esta segunda parte del capítulo se mostrarán los resultados obtenidos al analizar una topología como la que se muestra en la Figura 8.8 . El objetivo es que tanto el móvil como el PC2 tengan cobertura hacia el PC1, pero no entre sí.

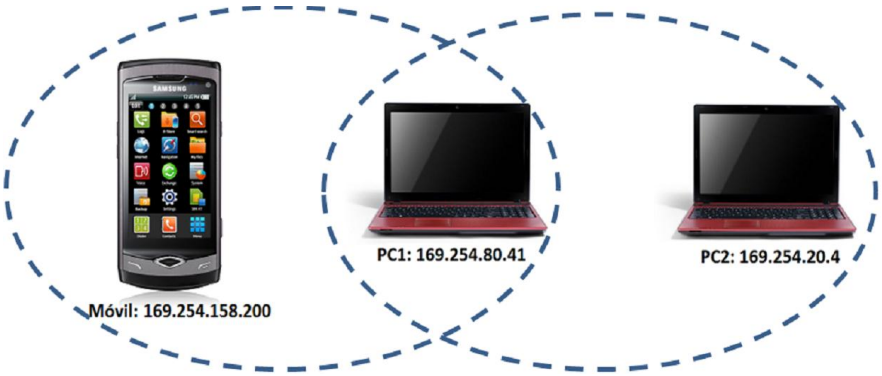

Figura 8.8: *Topología sin visión directa entre dispositivos*

8.2.1 Creación de una conexión

En esta sección se va a proceder a crear una conexión entre el móvil (169.254.158.200) y el PC2 (169.254.20.4). Para ello, harán uso de la conexión que pueden crear a través del PC1 (169.254.80.41).

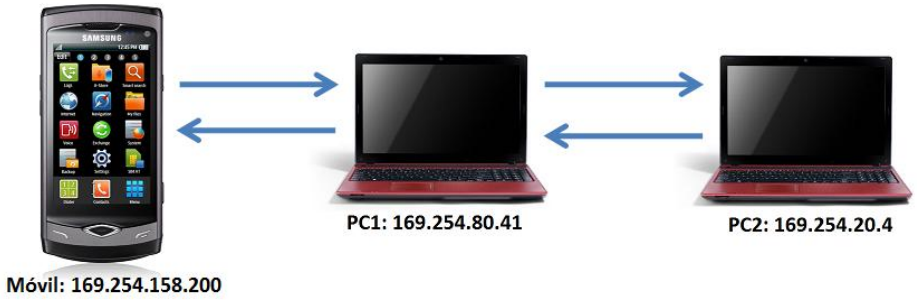

Figura 8.9: *Intercambio de mensajes entre el móvil y el PC2*

En la Figura 8.10 se puede ver cómo se establece la conexión entre los dos dispositivos de los extremos. Se analizará paquete a paquete el comportamiento del protocolo.

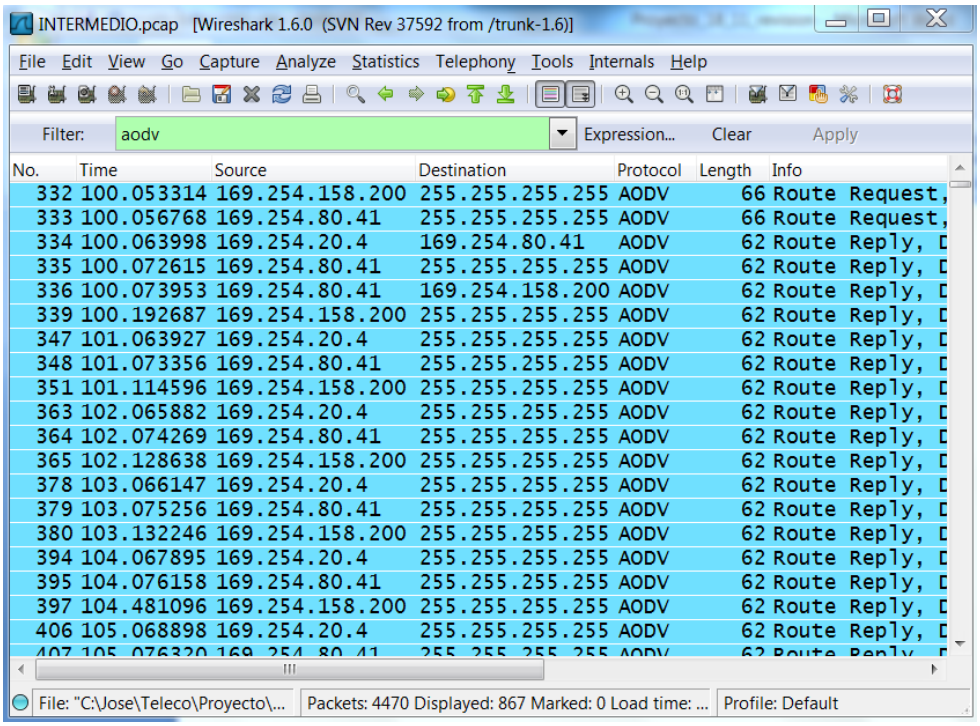

Figura 8.10: *Establecimiento de rutas sin visión directa*

Paquete 332: El móvil (169.254.158.200) quiere establecer una conexión con el PC2 (169.254.20.4) por lo que manda un RREQ preguntando por la dirección de este último.

Paquete 333: El PC1 es el dispositivo intermedio (169.254.80.41) que vuelve a difundir el RREQ con el fin de que llegue hasta el destino.

Paquete 334: El RREQ llega al destino (169.254.20.4), que contesta de forma *unicast* con un RREP directamente al nodo que le permite llegar hasta el origen, es decir, el intermedio (169.254.80.41).

Paquete 336: El PC1 (169.254.80.41) recibe el RREP del PC2 (169.254.20.4) y mira sus tablas de encaminamiento para dirigir el mensaje hasta el móvil (169.254.158.200), que fue el que originó el primer RREQ para establecer una conexión con el PC2.

Los siguientes mensajes son paquetes HELLO mandados para que cada uno sepa de la presencia activa de los demás nodos.

8.2.2 Mensajes de mantenimiento de ruta activa

En el apartado 8.2.1 se ha mostrado cómo se establece una ruta entre dos elementos que no tienen un camino directo. Una vez que se realiza esa conexión, la aplicación de eco debe empezar a funcionar para que se mantengan activas las rutas. En la Figura 8.11 se pone de manifiesto este hecho.

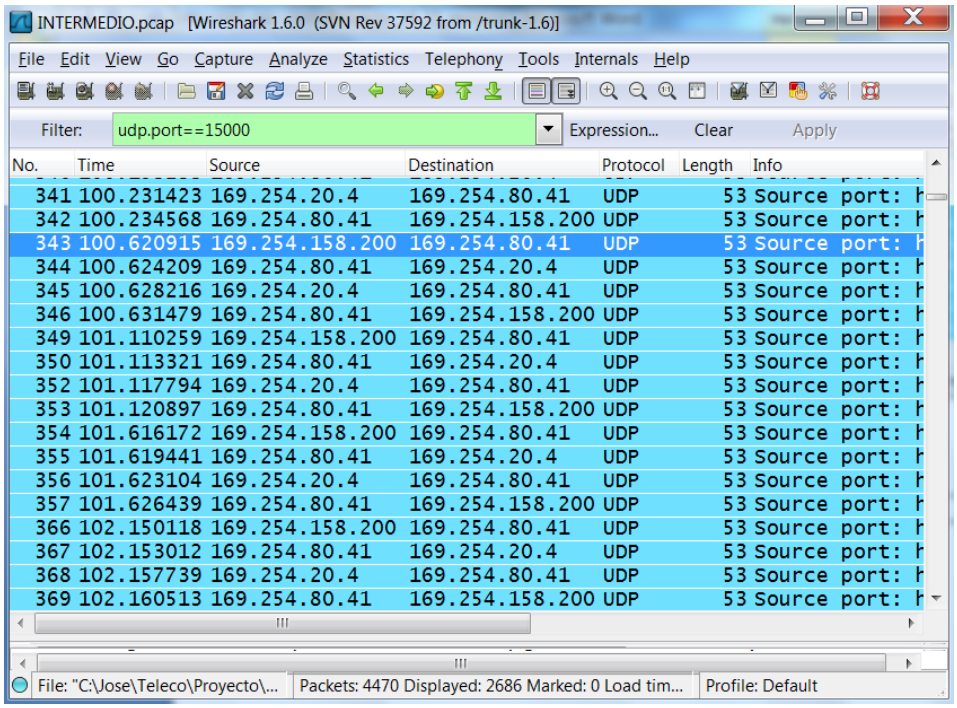

Figura 8.11: *Intercambio de mensajes de eco*

Paquete 343: El móvil (169.254.158.200) manda un mensaje de eco al destino con el que tiene una ruta activa, el PC2 (169.254.20.4). Como no tienen visión directa, el móvil mira su tabla de encaminamiento y lo envía a través del PC1 (169.254.80.41).

Paquete 344: El PC1 recoge el paquete enviado por el móvil y ve que está destinado al PC2. Por tanto, lo manda al siguiente salto, que directamente es el destino (169.254.20.4).

Paquete 345: El PC2 ve que es el destino de la comunicación y debe devolver el mensaje al origen. Así, busca en sus tablas cuál es el siguiente dispositivo para llegar al móvil y ve que es el PC1 (169.254.80.41), al que envía la información.

Paquete 346: El mensaje de vuelta llega al nodo intermedio que, al ver que va destinado hacia el origen de la comunicación (móvil), lo manda directamente. Así se completa el ciclo de ida y vuelta de la aplicación de eco.

En los siguientes paquetes se puede observar que esta secuencia se repite cíclicamente cada 500ms, ya que el origen vuelve a iniciar el proceso.

8.2.3 Mensaje tipo chat

El siguiente ejemplo muestra el correcto funcionamiento de una aplicación que tiene como objetivo mandar mensajes de texto y que necesita de algún tipo de protocolo de encaminamiento para poder alcanzar al destino correcto.

Para ello, se ha incluido una captura en la que se puede ver cómo se manda el mensaje "Hola pc2" desde el móvil (169.254.158.200) hacia el PC2 (169.254.20.4).

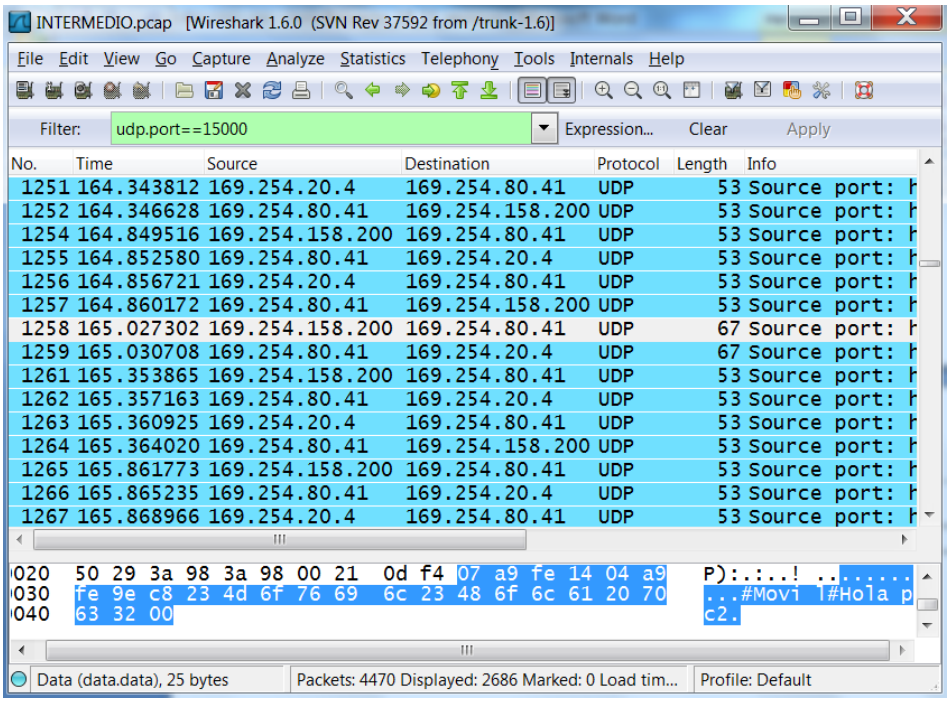

Figura 8.12: *Envío de mensaje de chat de un extremo a otro*

Paquete 1258: Mensaje de datos tipo chat iniciado por el móvil (169.254.158.200). Se manda al PC1 (169.254.80.41) ya que es el siguiente salto en la comunicación.

Paquete 1259: El 1258 llega al nodo intermedio y éste lo encamina hacia el siguiente dispositivo, el PC2 (169.254.20.4). De esta manera, se completa la comunicación entre el móvil y el destino.

En la parte inferior de la figura se puede distinguir el mensaje enviado. Los demás paquetes que se muestran corresponden a la aplicación de eco que se ha explicado en el apartado 8.2.2 destinados a mantener las rutas activas.

8.2.4 Pérdida de conexión

Para comprobar el envío de mensajes de error cuando se produce la caída de algún enlace, se ha utilizado la topología que se muestra en la Figura 8.13. El móvil y el PC2 tienen una conexión establecida que se mantiene a través de mensajes de eco periódicos.

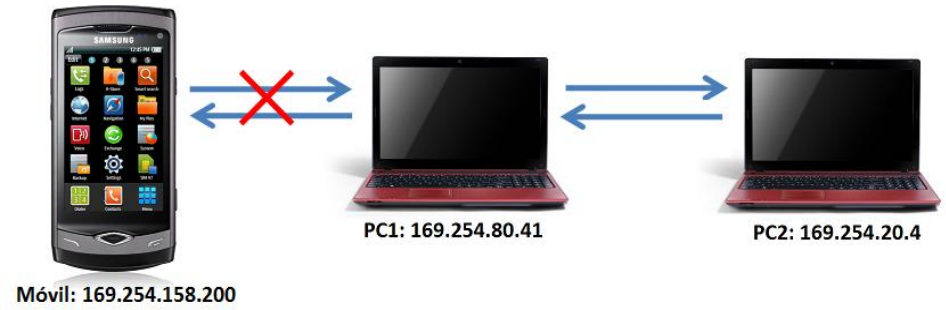

Figura 8.13: *Topología de pérdida de enlace*

En la Figura 8.14 se muestra el intercambio de mensajes AODV entre los tres dispositivos. Los mensajes de longitud 64 bytes son de tipo HELLO, enviados para que todos sepan que el nodo que los envía se encuentra operativo.

La prueba realizada consiste en eliminar desde el PC1 la conexión que se tiene hacia el móvil. Cuando esto suceda, no tendrá en las tablas de encaminamiento ninguna entrada que le notifique hacia dónde tiene que mandar un paquete que tenga como destino el móvil.

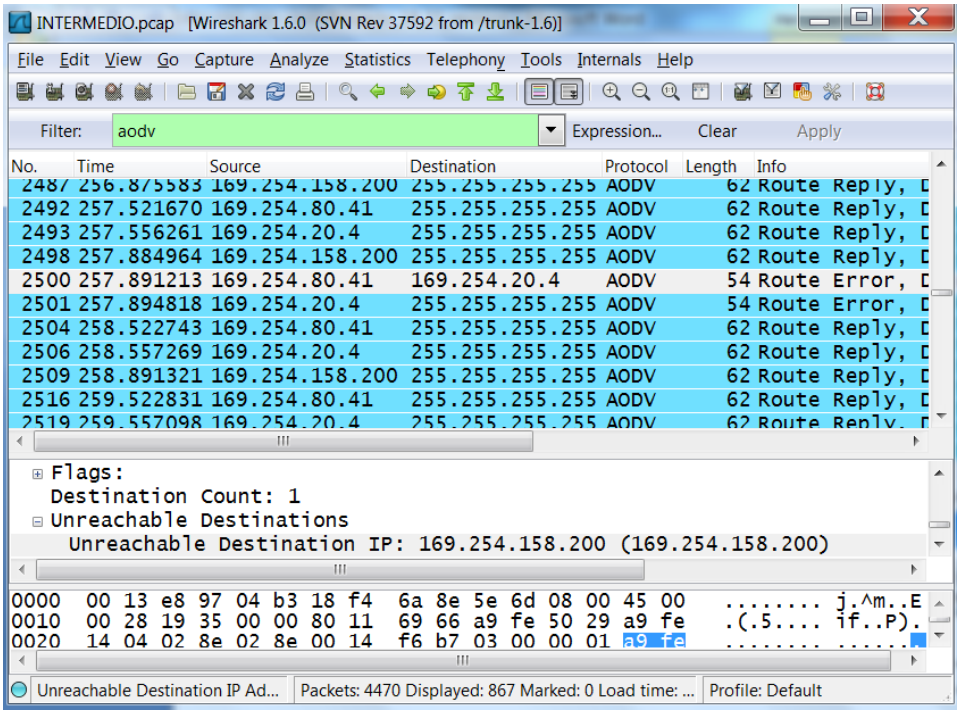

Figura 8.14: *Mensajes de error en una conexión multi-salto*

Paquete 2500: Se puede observar que se trata de un mensaje *unicast* destinado directamente desde el ordenador intermedio (169.254.80.41) hasta el que se encuentra en el extremo (169.254.20.4).

Este mensaje RERR *unicast* se manda porque el dispositivo intermedio (PC1) ha recibido un mensaje de datos por parte del ordenador del extremo (PC2) destinado al móvil. Como se ha cancelado el enlace entre el PC1 y el móvil, no se tiene ninguna ruta válida. Por lo tanto, se puede comprobar cómo el RERR lleva incorporado el destino al que no se puede acceder (169.254.158.200).

Paquete 2501: El ordenador del extremo (PC2) recibe el RERR por parte del PC1 y tras constatar que no puede llegar hasta el móvil, manda un mensaje de error, pero en este caso a la dirección 255.255.255.255 para que todos los que se encuentren en su radio de acción se percaten de la caída del enlace.

9 Conclusiones y trabajos futuros

En el capítulo 9 se van a explicar cuáles han sido las conclusiones extraídas del trabajo realizado y las líneas futuras de investigación y desarrollo que se podrían llevar a cabo.

9.1 Conclusiones

En este proyecto se ha diseñado una herramienta que permite crear una red MANET mediante dispositivos móviles y poder intercambiar mensajes de texto entre los nodos que la formen. Las contribuciones más destacables son las siguientes:

- Se han analizado las diversas aproximaciones existentes al problema, sus fortalezas y debilidades; en especial, se ha buscado la que pueda ofrecer mejores prestaciones en relación al dispositivo al que va dirigido, que en este caso es un teléfono móvil.

- Los tres grandes grupos de protocolos existentes son: proactivos, reactivos e híbridos. Los primeros (proactivos) utilizan mensajes de control periódicos para tener actualizadas las tablas de rutas, mientras que los últimos (híbridos) se utilizan para redes muy grandes. Por lo tanto, el protocolo elegido ha sido AODV que pertenece a los reactivos. En este grupo de protocolos no se mandan mensajes de control periódicos mientras que no se tenga una ruta activa, por lo que el consumo energético será menor, algo a tener en cuenta en dispositivos móviles. Además dentro del grupo de los reactivos, AODV destaca por su eficiencia en cuanto a *throughput*, retardo y recuperación de enlaces perdidos o rotos.

- El diseño del sistema está realizado en base a métodos lo más independientes posibles que permiten una mejor extensibilidad del programa.

- La solución propuesta se ha desarrollado en el sistema operativo Bada, propiedad de Samsung. Este sistema operativo ha salido al mercado recientemente y cuenta con el impulso de una de las marcas más importantes en el mercado de la telefonía móvil.

- Mediante el diseño e implementación del protocolo de encaminamiento presentado en este trabajo se han cumplido los requisitos establecidos al inicio del mismo. Debía ser capaz de crear y administrar una red MANET y utilizar algún tipo de

aplicación que permitiese corroborar el correcto funcionamiento. Por tanto, se han creado dos aplicaciones (aplicación de eco y aplicación de chat) y además se permite visualizar las conexiones que se tienen activas en un momento dado.

- Debido a una limitación importante en la API de Bada a la hora de abordar el problema del encaminamiento IP (no se permite modificar las tablas de rutas del dispositivo) se ha utilizado la siguiente solución: implementar el protocolo AODV encima de la capa de aplicación. Así, los mensajes de la capa de aplicación deben incluir la dirección origen y destino (extremos de la comunicación) antes que el mensaje propio que se quiera enviar. De esta forma, cada nodo comprobará estos campos y realizará el encaminamiento según las tablas almacenadas directamente en la aplicación.

- Mediante la aplicación de eco se ofrece la posibilidad de mantener las conexiones activas y que el usuario de la aplicación pueda visualizarlas. Si no se hubiese realizado esta funcionalidad, no sería posible ver las rutas activas existentes ya que, debido a la naturaleza reactiva de AODV, las entradas en la tabla de encaminamiento que no se utilicen se borran. Por lo tanto, la aplicación de eco envía un mensaje periódicamente al destino con el que se tenga una comunicación establecida para que, en cada salto, se vayan actualizando los temporizadores asociados al tiempo de ruta activo de las conexiones implicadas.

- En cuanto a la aplicación de chat, permite intercambiar mensajes de texto entre los usuarios que formen parte de la red. Debido a la naturaleza de las redes MANET, dichos mensajes podrán ser enviados a nodos con los que no se tenga una visión directa.

- Se han realizado pruebas con dispositivos que pueden enviarse mensajes directamente sin necesidad de encaminamiento intermedio y con nodos que necesitan de algún elemento intermedio para realizar la comunicación. En ambos casos las rutas en cada dispositivo se crean correctamente, eligiendo siempre el camino más corto para llegar hacia el destino deseado. Además, se ha probado la robustez a fallos eliminando enlaces y nodos de conexiones activas. Ante tal situación, el protocolo es capaz de recuperar la comunicación perdida.

9.2 Trabajos futuros

Quedan abiertas las siguientes líneas de trabajo:

- Como se explicó en el apartado 6.5, debido a las limitaciones impuestas por Bada se ha tenido que diseñar el protocolo en el nivel de aplicación. En el futuro se podría encapsular AODV en el nivel de red si la API del sistema de desarrollo permitiese modificar las tablas de encaminamiento del sistema. Como estas tablas se tienen guardadas de manera independiente, lo único que habría que hacer es copiarlas en las tablas de rutas propias del sistema operativo.

- Por otro lado, un aspecto que se podría estudiar es la seguridad. Ya que todos los dispositivos pueden ser nodos intermedios, pueden recibir información que sea confidencial y que no deberían poder conocer. En la implementación realizada, todos los elementos de la comunicación pueden tener acceso a la información que pase por ellos ya que no está cifrada.

- Otra línea de trabajo futuro sería la dotación a la red inalámbrica ad-hoc de las funcionalidades propias de una infraestructura fija. Se podrían incluir servidores DHCP que fuesen capaz de organizar y distribuir de manera eficiente y dinámica las direcciones IP de los dispositivos. Otro servicio que se podría incluir es el servicio de nombres de dominio (DNS), de forma que cuando un móvil se incorpore a la red se le identifique de manera unívoca e inteligible, teniendo en cuenta la naturaleza cambiante de la topología de este tipo de redes MANET.

- Otra posible tarea futura sería incluir un servicio que permita interconectar las redes autónomas que se crean a Internet. Para hacer posible una conexión total, alguno de los dispositivos debería tener acceso a la red global. Así, habría que estudiar la manera de hacer que, dinámicamente, se ofrezca a la red MANET la posibilidad de tener una conexión permanente a Internet.

- Por último, sería interesante extender la implementación a otros sistemas operativos actuales del mercado de las comunicaciones móviles, como son Android o iOS.

Apéndice A
Manual de Usuario

Durante este apéndice se va a indicar la manera de utilizar el programa a través de las diferentes pantallas y menús que ofrece la aplicación.

En la Figura A.1 se puede ver la pantalla de inicio.

El campo *Entidad* y *Grupo* es el nombre de la red ad-hoc que se va a crear. El nombre será del tipo Entidad_Grupo.

En el campo *Nombre* se especifica el nombre de usuario que se va a tener. En este caso PC1.

Una vez completados todos los apartados se pulsa el botón *CONECTAR* para crear la red especificada.

Figura A.1: *Pantalla de inicio*

Figura A.2: *Menú de opciones*

En la Figura A.2 se puede ver la aplicación inicializada con el menú de opciones desplegado. Se distinguen 5 opciones diferentes:

- Modo Red.
- Modo Chat.
- Conexiones.
- Examinar.
- Mensaje a todos.

La parte superior de la pantalla nos va a indicar en qué modo nos encontramos, en este caso *RED*, y la dirección IP propia del dispositivo (169.254.80.41).

Figura A.3: *Menú Examinar*

La Figura A.3 corresponde al menú *Examinar* de la pestaña de opciones.

En la parte superior se nos indica el título del menú (*EXPLORAR VECINOS*).

La parte superior de la pantalla (*Vecinos*) corresponde con los vecinos que se han encontrado. En este caso no se ha encontrado ninguno.

La parte inferior (*Conexiones*) son las conexiones activas que se van realizando con cada vecino. En este caso no existe ninguna conexión activa.

En la Figura A.4 se muestra el menú *Examinar* con vecinos encontrados.

Los vecinos son PC2 y Móvil. Cada uno de ellos contiene una pestaña desplegable que permitirá crear una conexión.

En la parte inferior se puede constatar que no existen conexiones activas en este momento.

Figura A.4: *Vecinos encontrados*

En la Figura A.5 se puede ver el menú *Examinar* con un vecino seleccionado. Como en este caso ya se han encontrado vecinos y se ha seleccionado uno para tener una conexión activa, no aparecen las pestañas iniciales (*Vecinos* y *Conexiones*). Para diferenciar ambas partes, existe un espacio (el que va desde Móvil a PC2) que delimitará las dos áreas. Los nuevos vecinos que se incorporen en la parte superior nunca ocuparán un espacio superior, en dirección descendente, que el de Móvil en la Figura A.5, pero como la zona de vecinos es una lista desplegable, el usuario podrá deslizar esa parte de la pantalla para ver los vecinos que posee. De igual manera, en la parte de abajo (*Conexiones*) se irán añadiendo pestañas inmediatamente después de PC2 a medida que se vayan seleccionando destinos. Como esta zona es también una lista desplegable, puede hacerse deslizar sin interferir con la parte de arriba y visualizar las conexiones activas.

El vecino que se ha seleccionado en este caso ha sido PC2. Para ello, se ha pulsado la pestaña *Seleccionar*.

La aplicación ha encontrado el destino y ha establecido una conexión que se puede ver en la parte inferior (PC2).

Figura A.5: *Vecino seleccionado*

Figura A.6: *Menú Chat*

La Figura A.6 corresponde con el menú *Modo Chat*.

En el título se especifica el modo en el que se está (*CHAT)* y el destino que se tiene seleccionado para la comunicación.

Como no hay destino seleccionado aparece *Seleccionar*.

El área en blanco servirá para escribir los mensajes.

Debajo de esta área aparecerán los mensajes recibidos.

En la parte inferior aparecen las conexiones que se tienen activas y que se pueden seleccionar para establecer una conversación.

Figura A.7: *Destino seleccionado*

En la Figura A.7 se puede ver un destino seleccionado.

El destino seleccionado ha sido PC2.

Al haber escogido PC2 (Figura A.7), en la parte superior correspondiente al título se especifica *CHAT | PC2*.

A partir de este momento se puede escribir en el área de escritura y utilizar el botón *Mandar* para enviar mensajes a PC2.

Figura A.8: *Visualización de destino*

Se puede distinguir el área de escritura (color blanco).

Se visualizan los mensajes entrantes (parte de abajo del área de escritura).

En este momento el dispositivo ha enviado a PC2 el mensaje "hola pc2". Después, ha recibido el mensaje "hola pc1" por parte de PC2. Por último, ha recibido el mensaje "hola pc1" por parte de Móvil.

En la parte inferior se puede seleccionar el destino con el que se quiere intercambiar mensajes.

Figura A.9: *Intercambio de mensajes*

Figura A.10: *Menú Mensaje a todos*

La Figura A.10 corresponde al menú *Mensaje a todos*.

Se puede apreciar que en la parte del título donde se especifica el destino seleccionado se especifica que el mensaje va destinado a *TODOS*.

Cualquier mensaje que se envíe en este modo irá destinado a todos los dispositivos que se tengan en la parte inferior en la lista desplegable.

En este caso el mensaje iría destinado a PC2 y a Móvil.

Figura A.11: *Menú Modo Red*

La Figura A.11 corresponde al menú *Modo Red*.

En el título se especifica el modo en el que se está (*RED*) y la dirección IP propia del dispositivo (169.254.80.41).

En el área de escritura se especifica *Escribir IP:* que será donde se introduzca la dirección que se desee buscar.

En el área de visualización se van mostrando los mensajes de las acciones que se llevan a cabo. Estas acciones son: buscar una dirección, establecer la conexión o anular una conexión.

En la parte inferior se encuentran las conexiones activas que se disponen. En este caso PC2 y Móvil.

Para buscar una dirección se debe pulsar el botón *Buscar* una vez que se haya introducido una IP correcta en el área de escritura.

La Figura A.12 corresponde con el menú *Conexiones*.

Es igual que el menú *Modo Red*, pero se eliminan las áreas de visualización para interactuar mejor con las conexiones.

En este momento existen dos conexiones activas, PC2 y Móvil.

En la Figura A.13 se visualizará el interior de los menús desplegables de cada conexión.

Figura A.12: *Menú Conexiones*

En la Figura A.13 se muestran los menús desplegables de las conexiones.

Se distinguen estadísticas como el siguiente salto y el número de saltos necesarios para llegar al destino.

La opción *Seleccionar* está anulada en este modo, porque es propia del chat.

La opción *Anular* permite romper enlaces y ver cómo el protocolo recupera las conexiones.

Figura A.13: *Opciones de visualización*

Figura A.14: *Opción deshabilitada*

En la Figura A.14 se puede ver el mensaje que se obtendría si hubiésemos pulsado la opción *Seleccionar* dentro del menú *Conexiones.*

Debido a que no es una funcionalidad propia de este modo, se indica mediante el mensaje mostrado (*PASAR A MODO CHAT*) dónde hay que acceder para hacer operativa la pestaña pulsada.

Se ha preferido mostrar siempre todas las opciones, aunque no estén operativas, para saber en cada momento todas las funcionalidades que se tienen. Además, en caso de pulsar por error sobre uno de estos controles se nos indicará el sitio exacto al que se debe entrar, haciendo así más intuitiva la labor del usuario.

Referencias

[1] BADA. [Online]. http://www.bada.com/

[2] Informe Gartner sobre ventas de móviles. [Online]. http://www.gartner.com/it/page.jsp?id=1848514

[3] C. Siva Ram Murthy and B. S. Manoj, *Ad Hoc Wireless Networks Architectures and Protocols*, quinta edición, ed. Prentice Hall Communications Engineering and Emerging Technologies Series, 2004.

[4] DARPA. [Online]. http://www.darpa.mil/

[5] IETF MANET. [Online]. http://datatracker.ietf.org/wg/manet/charter/

[6] LifeNet. [Online]. http://thelifenetwork.org/index.html

[7] Traore Soungalo, Wang Dong, and Liu Yulan, "Performance Evaluation of Campus Mobility Model in Mobile Ad Hoc Networks," *2009 International Conference on New Trends in Information and Service Science*, Pekín, Julio 2009.

[8] Redes VANETs. [Online]. http://vanet.info/

[9] Dirección General de Tráfico, "Anuario de accidentes 2010," 2010. [Online]. http://www.dgt.es/was6/portal/contenidos/es/seguridad_vial/estadistica/publicaciones/anuario_estadistico/anuario_estadistico014.pdf

[10] Cisco Systems. [Online]. http://www.cisco.com/web/ES/about/press/2011/11-02-01-vni-trafico-global-datos-moviles-se-multiplicara-por-26.html

[11] S. Taruna and G. N. Purohit, "Scenario Based Performance Analysis of AODV and DSDV in Mobile Adhoc Network," *Advances in Networks and Communications, Part 1*, vol. CXXXII, pp. 10-19, Enero 2011.

[12] Shree Murthy and Garcia-Luna-Aceves J. J., "An efficient routing protocol for wireless networks," *Mobile Networks and Applications*, vol. I, no. 2, pp. 183-197, 1996.

[13] Olsr.org. [Online]. http://www.olsr.org/

[14] T. Clausen and P. Jacquet, "Optimized Link State Routing Protocol (OLSR)," IETF , RFC 3626, 2003.

[15] D. Johnson, "The Dynamic Source Routing Protocol (DSR)," IETF, RFC 4728, 2007.

[16] Proyecto DSR. [Online]. http://piconet.sourceforge.net/

[17] CHAI-KEONG TOH, "Associativity-Based Routing for Ad-Hoc Mobile Networks," *Wireless Personal Communications*, vol. IV, no. 2, pp. 103-139, 1997.

[18] Marc R. Pearlman and Zygmunt J. Haas, "Determining the Optimal Configuration for the Zone Routing Protocol," *IEEE Journal on Selected Areas in Communications*, vol. XVII, no. 8, Agosto 1999.

[19] Shima Mohseni, Rosilah Hassan, Ahmed Patel, and Rozilawati Razali, "Comparative Review Study of Reactive and Proactive Routing Protocols in MANETs," *4th IEEE International Conference on Digital Ecosystems and Technologies*, Dubai, 12-15 Abril 2010.

[20] Ditipriya Sinha, Uma Bhattacharya, and Rituparna Chaki, "Trends in Network and Communications Part 2," *Communications in Computer and Information Science*, vol. 197, pp. 213-220, 2011.

[21] Nokia Research Center, "Ad hoc On-Demand Distance Vector (AODV) Routing," IETF, RFC 3561, 2003.

[22] J. Hsu, S. Bhatia, M. Takai, R. Bagrodia, and M.J. Acriche, "Performance of mobile ad hoc networking routing protocols in realistic scenarios," *Military Communications Conference, 2003. MILCOM 2003. IEEE*, vol. II, pp. 1268 - 1273, Octubre 2003.

[23] Lanzamiento de la plataforma Bada. [Online]. http://www.samsung.com/es/news/newsRead.do?news_group=corporatenews&news_type=activitynews&news_ctgry=&news_seq=17847

[24] Mobile World Congress. [Online]. http://www.mobileworldcongress.com/

[25] Modelos Móviles Bada. [Online]. http://www.bada.com/wave.html

[26] Eclipse. [Online]. http://eclipse.org/cdt/

[27] Artículo de crecimiento de ventas en smartphones. [Online]. http://www.20minutos.es/noticia/1220005/0/ventas/smartphones/moviles/

[28] Qué es Bada. [Online]. http://www.bada.com/whatisbada/index.html