TRABAJO FINAL DE CARRERA

TÍTULO: Comparativa de implementaciones de protocolos reactivos de encaminamiento en redes Ad-Hoc

AUTOR: Dídac Mediavilla Urra

DIRECTOR: Carles Gomez Montenegro

FECHA: 24 de febrero de 2006

Título: Comparativa de implementaciones de protocolos reactivos de encaminamiento en redes Ad-Hoc

Autor: Dídac Mediavilla Urra

Director: Carles Gomez Montenegro

Fecha: 24 de febrero de 2006

Resumen

Las redes de comunicación son una herramienta importante tanto para las empresas como para los particulares. Entre éstas tenemos las redes móviles MANET (*ad-hoc*). Este tipo de redes están en auge y, por lo tanto, es preciso el uso de unas normas (*protocolos*) para su correcto funcionamiento. Dichos protocolos realizan descubrimientos de rutas, encaminiento de datos y controlan el mantenimiento de dichas rutas. En este documento se verán los diferentes protocolos de encaminamiento reactivos para poder utilizar dichas redes y se realizará una comparación entre el DSR (*Dynamic Source Routing*) y el DYMO (*Dynamic MANET On-demand Routing*). Para poder hacer dicha comparación entre los diferentes protocolos, se realizan pruebas de latencia en el descubrimiento de una ruta, ancho de banda consumido por datos de control, consumo de batería y sobre todo, se tiene en cuenta el tiempo transcurrido en percibir un cambio de ruta, debido a cambios de topología de red, causados por la movilidad de terminales móviles. Estas pruebas se realizan con las implementaciones NIST-DYMO y DYMOUM para el protocolo DYMO y DSR-UU para el DSR.

The communication networks are an important tool both for companies and individuals. Among them, the mobile networks MANET *(ad-hoc)* are at their peak nowadays. Therefore, it is necessary to make use of some regulations or procedures *(protocols)* in order to obtain a correct performance. These protocols are able to discover routes, route data information between nodes and control the maintenance of these routes as well. In this study, different reactive routing protocols will be analyzed and a comparison between DSR (*Dynamic Source Routing*) and DYMO (*Dynamic MANET On-demand Routing*) will be carried out. In order to achieve that comparison, tests of latency in the route discovery, throughput consumed by control information, battery consumption and specially the time needed to notice a change in a route due to a change in the network topology have been fulfilled and analyzed. Finally, just mention that all the tests were realized using NIST-DYMO and DYMOUM implementations for DYMO protocol and DSR-UU implementation for DSR protocol.

AGRADECIMIENTOS

Quisiera agradecer a toda la gente que con su apoyo, dedicación y esfuerzo han ayudado a la realización de este trabajo.

Ante todo, destacar la labor realizada por mi director del trabajo Carles Gomez Montenegro que me ha guiado y se ha prestado a cualquier consulta en todo momento.

Quiero dar gracias a Vanessa por la dedicación prestada en la redacción del presente trabajo y por el esfuerzo que supone aguantarme en todo momento, "*te quiero peke*".

Agradecer a los "compañeros" y en algunos casos amigos de la EPSC: Tomás Pistolas, Omar Jamonero, Eloy Encias, Fernando Sebas, Mario Salido, Carlos Truchero, Juanma kakillas y Alex pajaro por acompañar, en esas largas tardes en la Universidad y por compartir una parte de mi vida. Me gustaría mencionar a Sebas que me ha introducido en el mundo de Linux y sin su ayuda, éste no podría haberse acabado.

Me encantaría agradecer, a mis padres el esfuerzo realizado conmigo, ya que me han ayudado no solo en este momento, sino en innumerables de mi vida.

El último grupo a quien tengo que agradecer de todo corazón, son a mis amigos de toda la vida: Marcel, David, Alberto, Xavi, Raul, Copi, Victor, Ceju y sus respectivas novias.

Por último agradezco al tribunal, la lectura del trabajo.

ÍNDICE

INTRODUCCIÓN

Las redes informáticas están alcance de todo el mundo hoy en día, esto es debido en gran parte a la aparición de Internet. Gracias a ésta han ido surgiendo distintos tipos de redes. Las redes wireless se están introduciendo en el mercado rápidamente a causa de la gran demanda. Dentro de este tipo de redes están las redes Ad-Hoc que no precisan una infraestructura previa, por lo que dos o más usuarios que tengan terminales móviles pueden realizar una red en cualquier momento para satisfacer las necesidades de comunicación de datos. La principal ventaja que tienen estas redes es que pueden abarcar una larga distancia, ya que los terminales actúan como enrutadores. Por ejemplo si alguno de ellos tiene conexión a Internet, los otros pueden acceder a ella gracias a ese nodo.

En este trabajo nos centramos en las redes wireless, más concretamente en las redes Ad-Hoc o redes MANET (*Mobile Ad-Hoc Networks*), los terminales son capaces de actuar como receptores, transmisores o enrutadores. Los nodos al ser móviles pueden moverse por un espacio libre, dando lugar a topologías distintas en todo momento. Es una topología dinámica, debido a cortes en enlaces, y a la aparición de nuevos enlaces. Para ello son necesarios los protocolos de encaminamiento en redes Ad-Hoc, ya que los existentes para redes fijas no son adecuados para este tipo de redes. De los protocolos existentes nos centramos en los reactivos. En tales protocolos si un nodo no envía información no incurre en mensajes de control, con lo que se tiene un ahorro de recursos de red. Especialmente en este documento se tratan los protocolos DSR (*Dynamic Source Routing*) (*véase [1]*) y DYMO (*Dynamic MANET On-demand Routing*) (*véase [2]*), los cuales se encuentran en proceso de estandarización por el grupo de trabajo MANET.

El documento se divide en 6 capítulos. En el primero, se definen las redes Ad-Hoc y sus principales características. También se mencionan los principales protocolos existentes para este tipo de redes y sus posibles aplicaciones en el ámbito real. En el segundo capítulo se describe el funcionamiento del protocolo DSR y sus principales implementaciones. Ya en el tercer capítulo, se especifica el funcionamiento del protocolo reactivo DYMO y los principales parámetros que utiliza éste. El cuarto, define los escenarios utilizados en este documento e indica como se tienen que crear las redes con las implementaciones escogidas. En el quinto capítulo, se especifican las pruebas con las implementaciones y se comentan los principales resultados obtenidos. Ya por último, en el último capítulo se realiza una conclusión final del trabajo y se especifican líneas futuras de éste.

CAPÍTULO 1. INTRODUCCIÓN A LAS REDES AD-HOC

1.1 Orígenes y definición

Las redes móviles MANET (*mobile Ad-Hoc Network*) o también conocidas como redes Ad-Hoc (*vease Fig.1.1*), son redes inalámbricas en las que tienen que existir dos o más terminales móviles que pueden comunicarse entre ellos sin la necesidad de utilizar ningún tipo de infraestructura. Estos terminales pueden actuar como receptores, emisores o pueden hacer una función de router. Estas redes pueden comunicarse con diferentes redes que no sean Ad-Hoc, es decir, pueden comunicarse con redes fijas dando lugar a redes híbridas. En las redes Ad-Hoc la movilidad de los nodos hace que su topología cambie continuamente, por lo tanto es necesario un protocolo específico de redes Ad-Hoc dinámico que descubra, mantenga y borre las rutas obsoletas continuamente.

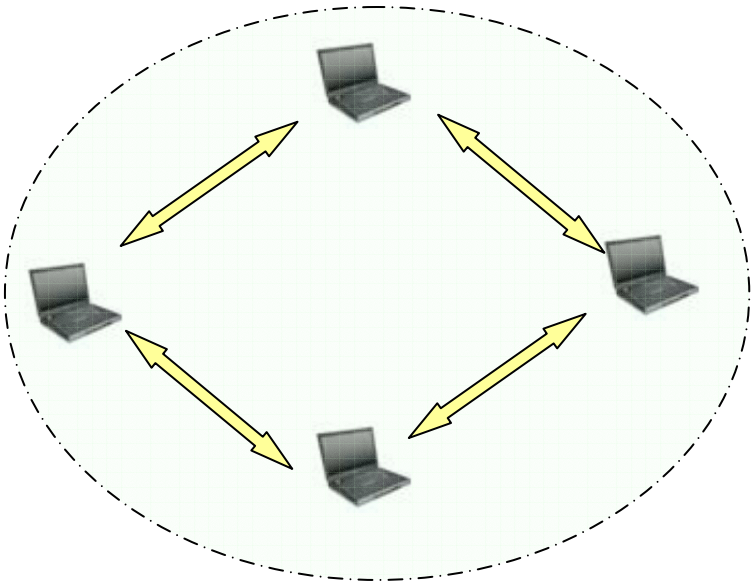


Fig.1.1 Topología de una red Ad-Hoc

El origen de las redes Ad-Hoc se remonta a los años 70 por la necesidad militar de la interconexión de diferentes terminales. Este tipo de redes fue implantado en EEUU por miedo a eliminar una base central de comunicaciones y no poder reencaminar la información a otros terminales. Con estas redes se pretendía transmitir información de forma rápida, estable y poder abarcar la mayor parte de rango posible sin necesidad de tener una infraestructura previa.

1.2 Características y problemas de las redes Ad-Hoc

Las principales características de las redes Ad-Hoc son que los terminales utilizan el medio inalámbrico, en el cual se pueden mover libremente, con lo que cambia la topología de la red constantemente y no necesitan ningún tipo de infraestructura previa para ser utilizada.

Otra característica principal que tienen las redes Ad-Hoc es que los terminales actúan como routers de manera transparente, para que el receptor obtenga la información destinada hacia él. Como hemos comentado anteriormente, dichas redes se pueden conectar a redes fijas utilizando un terminal de la red Ad-Hoc que actuará de Gateway, entonces éstas se dejarán de llamar Ad-Hoc y se denominarán Híbridas, ya que existe una parte MANET y otra parte fija.

Existen numerosos problemas en las redes Ad-Hoc que se están mejorando día a día, entre éstos tenemos como principales los que se exponen a continuación.

1.2.1 Arquitectura de protocolos

El protocolo TCP (*Transmision Control Protocol*) es un protocolo orientado a conexión, está diseñado para redes cableadas y por lo tanto en estas redes el índice de pérdidas de datos es muy bajo, con lo que la fiabilidad es alta. Cuando se detecta una pérdida de un paquete en las redes cableadas es mayoritariamente debido a la congestión de la red y TCP baja la tasa de emisión de datos. En cambio en las redes wireless el principal problema no es la congestión, sino la pérdida de datos, porque en estas redes hay una tasa de error en las comunicaciones un par de órdenes de magnitud superior, y la pérdida de datos es más comun que en las cableadas. Otro problema que aparece con las redes MANET son los continuos traspasos que hacen los terminales por su movilidad.

Dichas causas provocan que se aplique el algoritmo de control de congestión cuando no toca, por lo que TCP reduce la tasa de envío degradando el rendimiento.

Para un mejor funcionamiento de este protocolo se han realizado mejoras, como por ejemplo New Remo, SACKs, ELN (*véase [3]*), que se pueden aplicar a los sistemas wireless para un mejor funcionamiento del TCP. Otra mejora que se contempla es poner otra capa entre la capa IP y la TCP, la cual utiliza el protocolo ATCP (*Ad-Hoc Transmision Control Protocol)* (*véase [4*]) para mejorar el rendimiento de TCP.

Se escoge la arquitectura TCP/IP por la compatibilidad con Internet, pero esta arquitectura de protocolos no es la más adecuada para redes Ad-Hoc. Se ha demostrado que hay otros tipos de arquitecturas que son más idoneas para este tipo de redes.

1.2.2 Características de los dispositivos

Una de las principales características de los dispositivos de las redes Ad-Hoc es la escasa autonomía de sus baterías (aunque se están haciendo esfuerzos para que las baterías tengan mucha más vida), Este inconveniente se tiene que tener presente en el reenvío de información, debido a que no se tienen que hacer retransmisiones inútiles pues el envío de esta información consume batería. Además se tiene que tener presente para evitar el consumo de batería que los terminales entren en reposo cuando no sean utilizados. Otro inconveniente de estos terminales es que las memorias normalmente son pequeñas y la capacidad de proceso es bastante reducida.

1.2.3 Acceso al medio

El acceso al medio es un problema, ya que los terminales se disputan el mismo medio para poder transmitir, con lo que los terminales se tienen que poner de acuerdo para no interferirse entre ellos. Para ello hay un par de esquemas: Centralizado, en el que un terminal va asignando el tiempo para transmitir los terminales, o por contienda, que es el que se utiliza en este trabajo, donde los terminales emiten cuando es necesario. El principal problema son las posibles colisiones para acceder al medio.

Hay diferentes tipos de protocolos de acceso al medio como por ejemplo Bluetooh (*véase [5]*), Zigbee (*véase [6]*) o el IEEE 802.11 que es el más famoso, y el cual se utiliza en este documento. Este protocolo tiene un acceso llamado CSMA/CA (*Carrier Sense Multiple Acces – Collision Avoidance*) (*véase [7]*), además de este mecanismo el IEEE transmite los mensajes RTS/CTS (*Request To Send* / *Clear To Send*) que advierten a los otros terminales que hay una comunicación en curso y asimismo advierten de la duración de esta misma.

Dichos protocolos tienen su principal inconveniente en el terminal expuesto y el terminal oculto, como se puede ver en los siguientes apartados.

1.2.3.1 Terminal expuesto

En la Fig.1.2 se puede ver el problema del terminal expuesto; el nodo C está en medio de una transmisión con el nodo D, el nodo B no inicia otra transmisión ya que sino interfiere en la ya iniciada del terminal C. Realmente no es así, ya que la transmisión del nodo B se dirige hacia el nodo A, por lo que no interfiere entre la del C y la del D.

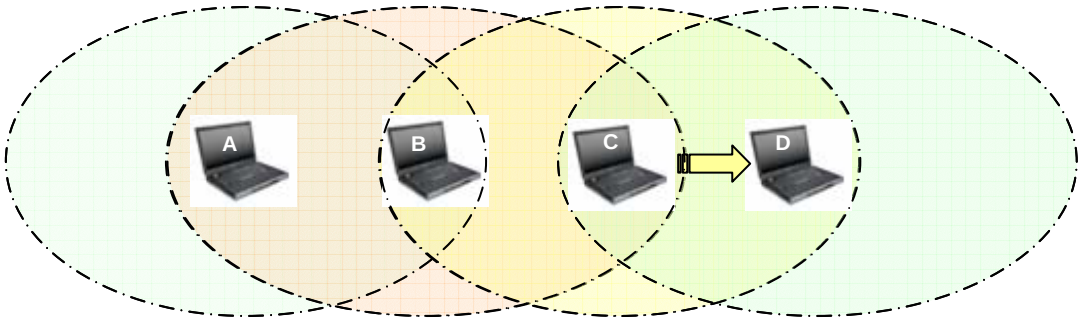


Fig.1.2 Terminal expuesto

1.2.3.2 Terminal oculto

El problema del terminal oculto se da cuando el terminal A quiere transmitir al terminal B, comienza a transmitir ya que el medio está libre y en el mismo momento C quiere enviar datos a B, y al no estar en el mismo rango de cobertura A y C, comienza a transmitir a B. Con lo que en B se produce una colisión y no recibe ninguna de las dos tramas, tal y como se puede observar en la Fig.1.3.

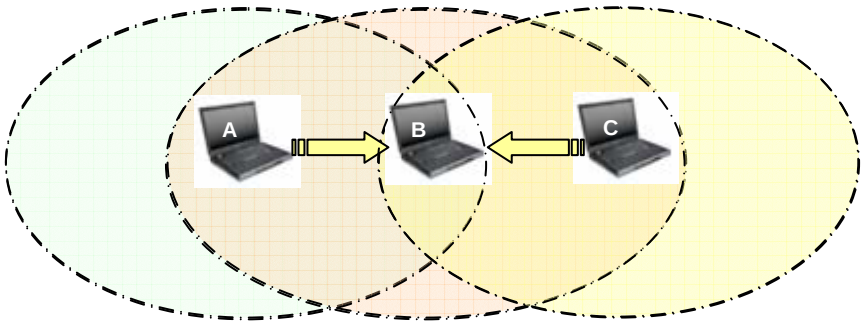


Fig.1.3 Terminal oculto

Para combatir este problema el nodo A antes de empezar a transmitir envía un mensaje RTS, B responde al nodo A con un mensaje CTS que llegará a toda su cobertura, por lo que el terminal C no transmite mensajes a B durante un cierto tiempo, cuando A recibe la trama CTS entonces puede empezar a enviar la información sin que haya colisión.

1.2.4 Direccionamiento

El direccionamiento de las redes Ad-hoc no es un tema cerrado, ya que actualmente no existe ningún tipo de mecanismo para realizar la

autoconfiguración, ejemplo de esto es el DHCP[1] (*Dynamic Host Configuration Protocol*) existente en redes de infraestructura o en redes fijas.

1.2.5 Topología y encaminamiento

El movimiento de los terminales hace que la topología vaya cambiando continuamente y por lo tanto los terminales vayan creando enlaces y destruyéndolos dinámicamente, a medida que se van moviendo los terminales.

El encaminamiento no es el mismo que en las redes cableadas, que tienen que pasar por un elemento central para poder enviar información. Este elemento central hace las veces de enrutador. En cambio en las redes MANETs no hay ningún elemento central que actúe como router, sino que todos los nodos pueden actuar en algún momento de router, elemento transmisor o receptor. Por lo que el enrutado lo realiza el terminal ejecutando un protocolo de encaminamiento específico de redes Ad-Hoc.

1.2.6 Seguridad de la red

La seguridad en las MANETs es un tema a pulir ya que en las redes inalámbricas, al tener un medio compartido al que cualquiera puede tener acceso, es más fácil adquirir la información por terceras personas que no en medios cableados. Una solución es que los protocolos de MANET hagan una función de seguridad, siempre y cuando no saturen con información de control, pues estamos condicionados a la batería de los terminales. Actualmente se está evaluando el protocolo SAODV (*Secure Ad-Hoc On-Demand Vector*) (*véase [8]*)

1.3 Encaminamiento

Como ya hemos comentado anteriormente en las redes Ad-Hoc son necesarios unos protocolos diferentes a los utilizados en las redes cableadas. En este tipo de redes tenemos dos grandes subgrupos de protocolos en los que se utilizan el vector distancia (*véase [9]*) y los algoritmos basados en el estado de enlace *(véase [10]*); en los primeros, podemos destacar el protocolo RIP (*Routing Information Protocol*), en el otro extremo tenemos como protocolo predominante el OSPF (*Open Shortest Path Fist*).

Los protocolos de las redes MANETs los podemos clasificar en tres grandes bloques como veremos a continuación. Estos protocolos han nacido ante la necesidad de controlar el encaminamiento en las redes Ad-Hoc, teniendo en

[1] Es un protocolo de red en el que un servidor provee los parámetros de configuración a los terminales conectados a la red informática que los requieran.

cuenta el movimiento de los terminales, la batería de los mismos y su limitación en ancho de banda.

1.3.1 MANET IETF

El IETF (*Internet Engineering Task Force*) MANET working group (*véase [11]*) es el encargado de analizar los problemas en las redes Ad-Hoc y de observar su rendimiento. Este organismo es el encargado de especificar los protocolos candidatos a ser estandarizados. En este grupo existen dos grandes categorías y una que es mixta, tal y como se puede ver en la Fig.1.4 con sus principales protocolos.

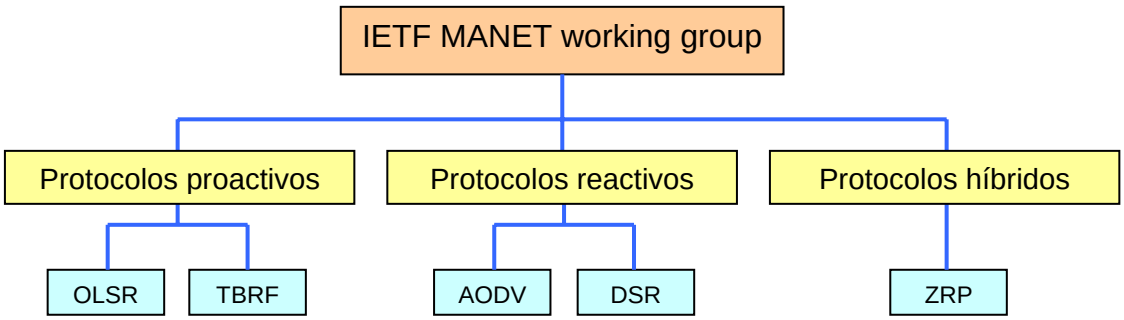


Fig.1.4 Clasificación realizada por el IETF MANET working group, junto con algunos ejemplos representativos

1.3.1.1 Protocolos proactivos

Los protocolos proactivos son aquellos que tienen periódicamente actualizadas las tablas de encaminamiento de todos los nodos de la red aunque no se estén enviando información. Cuando hay un cambio de topología la tabla de encaminamiento se actualiza y el protocolo escoge el camino más óptimo para enviar la información. Esto es debido al intercambio de mensajes de control de los protocolos, con lo que hay un gasto importante de ancho de banda y batería ya que se envían periódicamente tales mensajes.

Entre los protocolos estandarizados por el IETF MANET WG está el OLSR (*Optimized Link State Routing Protocol*) (*véase [12]*) y el TBRF (*Topology Dissemination Based on Reversed–Path*) (*véase [13]*) como más importantes. Actualmente está como candidato para ser estandarizado el OLSRv2 (*véase [14]*) que es una versión ampliada del OLSR.

1.3.1.2 Protocolos reactivos

Este grupo de protocolos únicamente tiene rutas de encaminamiento en sus tablas cuando un nodo origen tiene que realizar una comunicación con otro nodo de la red. Con estos protocolos, al iniciar una comunicación y no tener la ruta para llegar a un nodo destino, se envía un mensaje de descubrimiento de ruta. Cuando recibe la respuesta, añade esta ruta en su tabla de enrutamiento, y es entonces cuando es posible la comunicación con dicho nodo. El principal inconveniente es la latencia a la hora de descubrir una nueva ruta al principio de las comunicaciones, pero a su vez mejoran los recursos de red y energéticos de los terminales. Dentro de los protocolos reactivos existen dos clases de protocolos:

- *Basados en la fuente*: La ruta de los nodos por donde tiene que ir la información es almacenada en la cabecera de los paquetes, por lo tanto los nodos intermedios no necesitan tablas de encaminamiento, ya que consultan la cabecera de dichos paquetes para reenviarlos al destino. Esta solución en redes extensas no es aconsejable ya que la cabecera aumenta por cada nodo que pase y se pierde ancho de banda. Otro problema es que en una red amplia hay mucha movilidad y se puede perder fácilmente el enlace.
- *Salto a salto*: La ruta en este caso la escoge cada nodo en cualquier momento, ya que cuando enviamos la información, el paquete contiene la dirección de final y la dirección del siguiente nodo por el que se quiere pasar. En este caso se adapta más rápido a los cambios de topología, pero hay un gasto superior de recursos de los nodos intermedios ya que tienen que almacenar en tablas de encaminamiento las rutas correspondientes.

Como protocolos estandarizados por el IETF MANET WG, existe el AODV (*Ad-hoc On Demand Distance Vector*) y el DSR (*Dynamic Source Routing Protocol*) (*véase [15]*). Este último no está estandarizado, aunque actualmente existe un RFC experimental, que explica el funcionamiento y contiene los principales valores recomendables. Como RFC Satndards Track, del IETF tenemos el DYMO (*Dynamic MANET On-demand Routing*), estos dos protocolos se describen extensamente a lo largo de este documento.

1.3.1.3 Protocolos híbridos

Estos protocolos son una mezcla entre los dos tipos anteriormente comentados, y utilizan las ventajas de los dos. Los protocolos dividen las redes Ad-Hoc en diferentes zonas, los nodos que están próximos utilizan un encaminamiento proactivo, mientras que los que están alejados utilizan encaminamiento reactivo. El protocolo propuesto en el marco de el IETF es el ZRP (*The Zone Routing Protocol*) (*véase [16]*)

1.3.2 OSPF – MANET

El protocolo OSPF (*Open Shortest Path First*) (*véase [17]*) está orientado hacia redes cableadas por lo que en sí no es un protocolo adecuado para redes Wireless.

Se ha creado un grupo de trabajo que desarrolla dicho protocolo para redes Ad-Hoc, autorizando un estándar que intenta adaptar el tamaño de los paquetes hellos, así como también optimizar el método de difusión del estado.

1.3.3 6LOWPAN IETF

6LOWPAN perteneciente al IETF, es un grupo de trabajo que nació ante la necesidad de comunicar pequeños dispositivos (*sensores*). Un objetivo del grupo es adaptar a estos entornos los protocolos de routing de MANET. Estos protocolos tienen que tener en cuenta las características de los elementos interconectados entre sí, ya que éstos poseen un nivel de procesado muy bajo, la cual cosa conlleva a que los paquetes deben ser lo más reducidos posible. Por otro lado se tiene que tener en cuenta el poco ancho de banda disponible de estos dispositivos y sobre todo la limitación en cuanto a autonomía se refiere (*batería limitada*). El grupo 6LOWPAN trabaja también para que los protocolos puedan utilizar IPv6, ya que la tecnología de estos dispositivos avanza rápidamente.

Principalmente 6LOWPAN trabaja sobre los protocolos DYMO y AODV para adaptarlos a este tipo de redes, dando lugar a los protocolos DYMO-low *(véase [18])* y LOAD *(véase [19]).*

1.3.4 Conclusiones

Tal y como hemos podido ver, cada tipo de los protocolos comentados con anterioridad tiene su ventaja y desventaja. En redes de sensores o de pequeños terminales en los que la capacidad computacional esté limitada se utilizará protocolos de la 6LOWPAN, ya que son específicos para dichas redes.

Cuando la capacidad computacional crece y disponemos de un ancho de banda superior, se tiene que escoger entre los protocolos proactivos o reactivos.

Cuando se esta utilizando un tráfico de red grande se escogen los protocolos proactivos, ya que así, los nodos no tienen que esperar el descubrimiento de ruta al iniciar una transmisión, en cambio, en redes que el patrón de tráfico es a ráfagas muy distanciadas, se escogerán los protocolos reactivos, ya que no importa esta pérdida de tiempo, porque los nodos utilizan la red mucho menos que en el otro caso. Si la movilidad de los nodos es muy acentuada, entonces se tiene que escoger un protocolo proactivo, ya que éste tiene un mapa de la

topología periódicamente actualizado, mientras que si los terminales no se mueven mucho la elección sería un protocolo reactivo. Se pueden tener más clasificaciones para escoger un tipo de protocolo u otro, como por ejemplo la medida de la red, la batería de los terminales, etc. En cualquier caso, no existe ninguna implementación que sea óptima en todos los escenarios disponibles.

1.4 Aplicaciones

La tendencia a utilizar cada vez más dispositivos inalámbricos nos da un sinfín de aplicaciones posibles para estas redes, a continuación se citan los importantes.

- *Aplicaciones militares*. El origen de estas redes viene de aplicaciones militares, por lo que hay numerosas aplicaciones en campos de batalla de difícil acceso donde no haya infraestructura previa, estas redes se pueden montar entre tanques, aviones y otros elementos móviles.
- *Redes de difícil acceso*. Estas aplicaciones se realizan en lugares donde no es posible o no es económico instalar una red cableada, debido a la topología del terreno, por lo cual es más cómodo apostar por una red Ad-Hoc.
- *Servicio de emergencia.* Estas aplicaciones son necesarias en caso de desastre natural (huracanes, inundaciones, etc), ya que no es posible disponer de una red cableada o una infraestructura previa.
- *Redes Mesh*. Las redes Mesh son redes Ad-Hoc donde distintos nodos están conectados mediante una topología punto a punto, y se utilizan nodos intermedios para llegar al destino si éste no está en el área de cobertura, para ello se necesitan los protocolos de encaminamiento reactivos comentados. Como aplicación resaltamos la comunicación en grandes áreas de cobertura mediante saltos.
- *Redes de sensores*. Este tipo de redes son las que crecen más rápido actualmente, ya que hay una infinidad de aplicaciones posibles, como pueden ser, redes de pequeños sensores para usos domóticos para controlar dispositivos domesticos desde un elemento central, otra aplicación ya disponible es la monitorización del medio con pequeños sensores. A nivel industrial también es útil, ya que estas redes pueden tener controlado el inventario de una fábrica en todo momento. Por último comentar que las redes de sensores pueden ser útiles en el campo de la medicina ya que hay estudios que se realizan con nanosensores que pueden detectar el grado de bacterias en una zona o incluso pueden supervisar diversas funciones del cuerpo humano.

Todas estas aplicaciones son posibles gracias a que los terminales son cada vez más pequeños, con más autonomía y coste significativamente bajo. Los fabricantes están realizando esfuerzos en investigación y desarrollo de nuevas aplicaciones orientadas a nuevos mercados en auge.

CAPÍTULO 2. EL PROTOCOLO DSR

El encaminamiento en las redes Ad-Hoc es un asunto en continuo desarrollo, tal y como hemos podido observar. Una posible solución es utilizar el protocolo reactivo DSR que no consume muchos recursos de red, aunque a la hora de descubrir una ruta introduce una latencia inicial. Actualmente existen varias implementaciones de este protocolo. La principal de ellas es la DSR-UU implementada por la universidad de Uppsala.

2.1 Visión general

El protocolo reactivo DSR (*Dynamic Source Routing Protocol*) es un protocolo creado específicamente para redes Ad-Hoc, únicamente envía información cuando es preciso, con lo que hay un ahorro de energía de red, liberando ancho de banda y ahorrando batería. DSR también incorpora un mecanismo para evitar la formación de bucles. Es compatible con IPv6. Como contrapartida tenemos la latencia inicial al descubrir una ruta, y al ser un protocolo basado en la fuente, la cabecera del paquete va creciendo a medida que va pasando por nodos, cosa que conlleva a perder un preciado ancho de banda.

DSR aún no ha sido estandarizado por el IETF MANET working group aunque está en proceso. Actualmente el último draft realizado es el número 10, que data del 19 de julio del 2004. Este último draft ha sido realizado por David B. Jonson de la Universidad de Rice, David A. Maltz de la Universidad de Carnegie Mellon y de Yih-Chun Hu de la Universidad de Rice.

Este protocolo se puede estructurar en dos mecanismos: el de descubrimiento de ruta y el de mantenimiento de la misma. El primero realiza la búsqueda de la ruta y tiene los paquetes RREQ (*Route Request*) para buscar la ruta, si no la dispone en la tabla de encaminamiento y el RREP (*Route Reply*) que responde al RREQ sobre el descubrimiento de una ruta. El segundo mecanismo tiene el RRER (*Route Error*), que indica la caída de una ruta y el ACK, que realiza el mantenimiento de la ruta periódicamente. En el anexo A se muestra un resumen de los formatos de paquetes utilizados por el protocolo.

2.2 Funcionamiento

2.2.1 Descubrimiento de ruta

Este mecanismo se ejecuta cuando el emisor no tiene en su tabla de encaminamiento la ruta hacia el nodo destino al que va dirigido la información, si es no es así el nodo emisor hace uso de su tabla de encaminamiento.

En el caso de que no tenga la ruta en la tabla de encaminamiento el nodo emisor tiene que utilizar el mecanismo Route Discovery para encontrar dinámicamente una nueva ruta hacia el nodo destino; el emisor envía un paquete RREQ en modo broadcast[2]. Este paquete contiene la dirección del nodo fuente, la dirección del nodo destino y un identificador del paquete RREQ. En dichos paquetes se irán insertando las direcciones de los nodos intermedios tal y como se puede ver en la Fig.2.1. Si el nodo solicitado en el paquete RREQ esta activo en la tabla de encaminamiento, entonces envía el paquete de respuesta RREP con una copia del registro de la ruta acumulada en el RREQ, más la ruta que contenga su tabla de encaminamiento. Si en su defecto es el nodo destino, únicamente transmite el paquete RREP con una copia del registro de las rutas acumuladas en el RREQ más la suya.

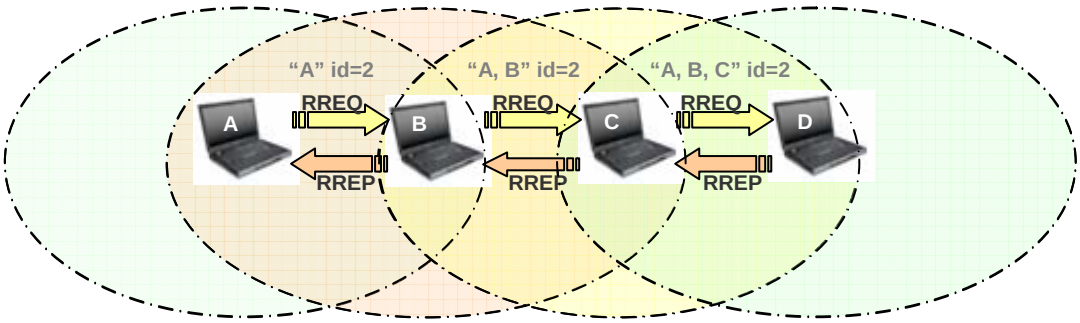


Fig.2.1 Descubrimiento en una red Ad-Hoc con el protocolo DSR

En el caso de que el nodo intermedio no conozca el destino, debe introducir su dirección en el registro de las rutas acumuladas del paquete RREQ y debe reenviar en modo broadcast el RREQ. La operación se repite con todos los nodos hasta llegar al nodo final, como se puede ver en la Fig.2.1. Una vez llegue al destino éste deberá enviar el paquete RREP, el cual tiene que ser enviado por el mismo camino que le indica el RREQ pero en sentido inverso, pudiendo utilizar rutas bidireccionales.

El último caso que se puede plantear, es que una vez haya llegado un paquete RREQ a un nodo con un número de identificación, nos reenvíen otro paquete RREQ con el mismo identificador y de dirección de destino, entonces dicho nodo descarta el paquete, ya que es una retransmisión del nodo origen o de un nodo próximo a éste.

Existe la opción "*promiscuous listening*" que hace que los nodos puedan recibir y procesar paquetes de control y de datos no destinados a ellos, para poder tener información de las rutas sin necesidad de encaminar una transmisión. Otra opción existente que tiene DSR es elegir la opción más adecuada para enviar datos, es decir, la primera ruta en establecerse será la que se siga, por el número de saltos o bien seguir otros criterios. Por defecto, la ruta se establece con el primer paquete RREP recibido.

[2] Broadcast dígase en castellano la difusión que es producida cuando una fuente envía datos a todos los dispositivos dentro de un cierto rango.

2.2.2 Almacenamiento de las rutas

Una vez aprendida una ruta gracias al paquete RREQ, se guarda en una caché temporal, la cual tiene un tiempo asignado para borrarse en el caso de no utilizarse. Este temporizador tiene asignados 300 segundos según la especificación de DSR para que la ruta caduque y poder borrarla, en el momento que se utilice de nuevo una ruta que está en la caché, se inicializa el temporizador, con lo que se logra un ahorro de memoria para las rutas no utilizadas.

2.2.3 Mantenimiento de ruta

El mecanismo de mantenimiento de ruta es el encargado de avisar si una ruta se rompe. Cuando se está transmitiendo, cada nodo es el encargado de controlar el estado del tramo en el que se encuentra, es decir, verifica que el siguiente nodo está recibiendo la información, en la Fig.2.2 se puede ver como se transmite información del nodo A hacia el nodo D. En este caso, el terminal A es responsable de la entrega de la información del nodo B, el nodo B es el encargado de la entrega de los paquetes del C y a su vez el C se encarga que le llegue la información a D correctamente.

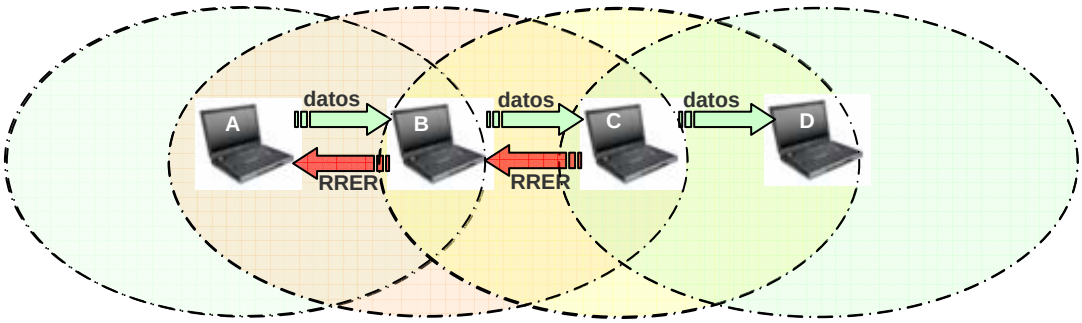


Fig.2.2 Mecanismo de mantenimiento de ruta

En el momento que se descubre que una ruta se rompe, el nodo que lo descubre envía un paquete RRER al nodo emisor. En la fig.2.2 se puede ver como el nodo C detecta la pérdida de ruta y envía un RRER hacia el emisor. Este nodo procesa el RRER y borra la ruta de su caché de encaminamiento. Si hay otra ruta en la caché se envían los datos por dicha ruta, en caso contrario se realiza el mecanismo de descubrimiento de ruta.

En el draft se utilizan tres formas para el descubrimiento de rotura de una ruta:

2.2.3.1 Reconocimientos de la capa de enlace

Estos reconocimientos se realizan desde la capa de enlace y se utilizan siempre que es posible. A medida que envían datos, el siguiente receptor, ya

sea el final o sea un nodo enrutador, tiene que ir recibiendo reconocimientos a sus paquetes enviados a nivel 2. El emisor no debe borrar ningún paquete hasta que no llegue su reconocimiento del siguiente nodo.

En el caso que se detecten pérdidas de reconocimiento se realiza el cambio de ruta. Como inconveniente tenemos que la implementación se tiene que realizar a nivel 2 (*capa de enlace*), y por lo tanto tendría que implementar todas las tarjetas de red que queremos utilizar, ya que son las que actúan a este nivel, por ello, no es viable. El mecanismo de reconocimiento a nivel 2 es el que proporciona mejor rendimiento, ya que no introduce ningún paquete de control en la red debido a que utilizan los existentes a nivel de enlace, con lo que se consigue una liberación de recursos de red.

2.2.3.2 Reconocimientos pasivos

Este mantenimiento se puede utilizar cuando los reconocimientos de capa de enlace no están disponibles.

El mecanismo que utiliza es el siguiente: Un terminal envía un paquete a un nodo adyacente al que no sea el destino final, cuando lo recibe, tiene que reenviar el paquete según su tabla de encaminamiento. Como el medio es compartido por todos los nodos, el paquete que ha reenviado el nodo intermedio, será recibido por el terminal que inicio la transmisión y por el que indica en la tabla de encaminamiento del nodo intermedio. De esta forma el nodo transmisor se asegura que al siguiente nodo le ha llegado el paquete transmitido por él.

Para detectar una pérdida de ruta se tiene en cuenta el tiempo de reconocimiento pasivo. Cuando el temporizador expira se reenvía el paquete sin necesidad de notificarlo a la capa de red. Si no se recibe ninguna retransmisión después de un número de intentos, se crea el paquete RRER para notificarlo al emisor.

2.2.3.3 Reconocimientos de la capa de red

Este modo de detección de rupturas de rutas se realiza cuando no hay ningún otro mecanismo de detección posible. Es el más razonable para una implementación real, por los motivos de viabilidad indicados en 2.2.3.1

El nodo que transmite tiene que solicitar un reconocimiento en la capa de red (*nivel 3*) del siguiente nodo cada MaintHoldoffTime. Para realizar tal función el nodo tiene que introducir en la cabecera de opciones del paquete DSR la opción de petición de reconocimiento. En el momento que recibe un paquete que contenga una petición de reconocimiento éste tiene que mirar si contiene su dirección IP para procesarla y enviarle respuesta con un reconocimiento, en

caso contrario no se procesa el paquete. Una vez que pase un tiempo RTT[3] (*basado en el algoritmo de estimación del RTT que se usa en TCP*) y no llegue ningún paquete de reconocimiento, se realizarán 2 retransmisiones de petición de reconocimiento más, para asegurarse que es una ruptura. En este caso el nodo que se percata de la ruptura envía al anterior un paquete RRER indicándole la ruptura del enlace.

2.2.4 Prevención de tormentas RREQ

La capacidad que tienen los nodos para contestar una petición de ruta, puede dar lugar a tormentas de RREQ, es decir, todos los nodos que tienen en su caché la ruta solicitada en un RREQ pueden enviar una respuesta, dando lugar a un gasto de banda preciado y a posibles colisiones.

En el ejemplo de la Fig.2.3 se puede observar como el nodo A quiere transmitir al nodo G; los nodos B, C, D, E y F reciben la petición de ruta y cada uno tiene en su caché una posible ruta hasta el nodo G:

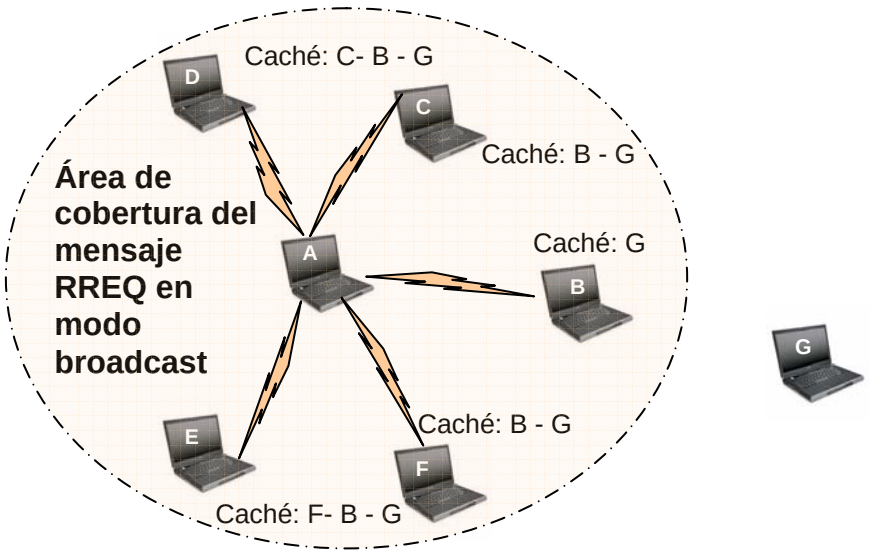


FIG.2.3 Ejemplo de prevención de tormentas

Los nodos enviarían la respuesta de solicitud de ruta al mismo tiempo ya que están a una distancia similar y el mensaje ha sido broadcast, la cual cosa decaería en múltiples colisiones.

Para corregir dicho problema, los receptores del RREQ esperan un tiempo aleatorio (*dependiendo del número de saltos*), para enviar la respuesta, si durante ese tiempo reciben un RREP de otro nodo, no envían su propia respuesta, ya que asumen que el emisor ya tiene una ruta preestablecida.

[3] Tiempo de ida y vuelta de un paquete

2.3 Parámetros y valores por defecto para DSR

En este apartado se exponen los parámetros y los valores por defecto del protocolo de encaminamiento DSR, según su especificación, como se pude ver en la Tabla 2.1:

Tabla 2.1 Configuración por defecto del protocolo DSR

Nombre de la variable	Cantidad	Unidades
DiscoveryHopLimit	255	Saltos
BroadcastJitter	10	Milisegundos
RouteCacheTimeout	300	Segundos
SendBufferTimeout	30	Segundos
RequestTableSize	64	Nodos
RequestTableIds	16	identificadores
MaxRequestRexmt	16	Retransmisiones
MaxRequestPeriod	10	Segundos
RequestPeriod	500	Milisegundos
NonpropRequestTimeout	30	Milisegundos
RexmtBufferSize	50	Paquetes
MaintHoldoffTime	250	Milisegundos
MaxMaintRexmt	2	Retransmisiones
TryPassiveAcks	1	Intento
PassiveAckTimeout	100	Milisegundos
GratReplyHoldoff	1	Segundo
MAX_SALVAGE_COUNT	15	Salvar

2.4 Implementaciones del DSR

Existen varias implementaciones para el protocolo DSR. Las más significativas son las siguientes:

- DSR University of Colorado at Boulder (*S.O: Linux*)
  - ✓ Creada por la Universidad de Colorado, únicamente es posible su utilización con sistemas operativos linux, con kernel 2.4.2 o superiores.
  - ✓ Creada a partir del draft número 8, datado el 15 de abril del 2003.
  - ✓ Homepage: http://pecolab.colorado.edu/

- MicroSoft Research Mesh Conectivity Layer (MCL) (*S.O: Windows XP*)
  - ✓ Crea una capa superior en el protocolo sobre Windows XP.
  - ✓ Es un derivado del DSR llamado Link Quality Source Routing (LQSR).
  - ✓ Investigado por Microsoft para el encaminamiento en redes Mesh.
  - ✓ Homepage: http://research.microsoft.com/mesh/

- Monarch Project (*S.O: FreeBSD*)
  - ✓ Esta implementación está un poco anticuada, ya que está realizada a partir del draft número 1.
  - ✓ Da soporte a FreeBSD 3.3 y 2.2.7 únicamente.
  - ✓ Homepage: http://www.monarch.cs.rice.edu/dsr-impl.html

- PicoNet II (*S.O: Linux*)
  - ✓ La implementación sólo puede ser ejecutada con kernels 2.4.
  - ✓ No implementa la versión 6 de IP.
  - ✓ Utiliza la versión 5 del draft de DSR.
  - ✓ Homepage: http://piconet.sourceforge.net/thesis/main.html

- DSR-UU (*S.O: Linux*)
  - ✓ Implementación actual, que puede utilizar kernels 2.4 y 2.6 y el simulador de red NS – 2.
  - ✓ Crea una interfaz virtual de red.
  - ✓ Todo el código escrito en C
  - ✓ Homepage: http://core.it.uu.se/AdHoc/ImplementationPortal

En este trabajo de final de carrera, se ha escogido la implementación de la Universidad de Uppsala (*DSR-UU*). El estudio del protocolo se realiza con un sistema operativo Linux Fedora Core, con el kernel 2.6.9, ya que es una versión estable y ya probada para el estudio de pruebas similares a las que se realizan.

Se escoge la aplicación de Uppsala debido a pruebas realizadas con otras implementaciones de esta Universidad. Utiliza reconocimientos en la capa de red, está escrito en lenguaje C el cual nos permite una asimilación rápida del contenido de la implementación y deja manipular los principales valores del protocolo y lo más importante, es una aplicación que utiliza la última versión del draft.

La implementación del DSR-UU ha sido realizada por Erick Nordström, también autor del AODV-UU, posiblemente la implementación más popular del protocolo AODV, esta ha sido otro punto a favor por lo que se ha escogido esta implementación. Es la primera versión editada del DSR-UU y por lo tanto tiene algún error en la implementación, que se tiene que pulir.

CAPÍTULO 3. EL PROTOCOLO DYMO

El protocolo DYMO (*Dynamic MANET On-demand*) es un protocolo reactivo que está en fase de desarrollo, por lo que actualmente no está estandarizado, pero está previsto que así lo sea en un futuro, tiene categoría de Standards Track. Hereda características de sus antecesores AODV y DSR, al ser un protocolo reactivo introduce una latencia al descubrir una ruta, pero lo contrarresta con el poco tráfico de control utilizado en la red. En este capitulo se explica el funcionamiento del protocolo y sus implementaciones.

3.1 Visión general

El protocolo DYMO es un protocolo reactivo que no envía paquetes de control si no está realizando funciones de encaminamiento, transmisión o recepción de información, lo que es bueno para redes en la que los recursos son escasos. Dicho protocolo es compatible con las versiones 4 y 6 del protocolo IP, tiene un mecanismo para la prevención de bucles. Actualmente se está desarrollando una versión reducida para redes de sensores, llamada DYMO-LOW, donde el objetivo principal a tener en cuenta es el bajo consumo de energía y la reducida capacidad de proceso que tienen estos sensores.

Es un protocolo joven que no está del todo desarrollado. En la actualidad se está creando la versión 4 del draft, ya que el último draft está fechado a día 23 de octubre de 2005. En esta última entrega aún no se especifica del todo como se realiza el mantenimiento de la ruta. Esta especificación está desarrollada gracias a Ian Chakeres de Boeing, Elizabeth M. Belding-Royer de la Universidad de Santa Barbara y Charles Perkins del grupo de investigación de Nokia.

Básicamente DYMO tiene dos mecanismos de forma similar a otros protocolos reactivos como DSR o AODV: el de descubrimiento de ruta, que se utiliza cuando en la tabla de encaminamiento no está la ruta del nodo destino, y el de mantenimiento de ruta, que puede utilizar varias formas para descubrir si una ruta se rompe. Hay tres posibles mensajes de control; el RE (*Route Element*) que engloba el RREQ (*Route Request*) para descubrir una ruta, y el RREP (*Route Reply*) para contestar el descubrimiento de una nueva ruta; el mensaje RERR (*Route Error*) que indica una ruta errónea y el UERR (*Unsupported-element Error*) que es un mensaje de error no soportado por el protocolo. A éste último no le ha encontrado utilidad, por lo que en la última reunión del grupo de trabajo de MANET se ha decido suprimirlo del draft (*dicho cambio se verá reflejado en la versión del draft número 4*). El primero se envía en el descubrimiento de ruta y los dos últimos se utilizan para el mantenimiento de rutas. Dependiendo de la técnica de mantenimiento de las rutas empleada, también pueden existir mensajes de hellos o de reconocimiento, todos estos

paquetes son enviados en UDP por un puerto To Be Determined[4] (*TBD*). En el anexo B se muestra un resumen detallado de los formatos de los paquetes del protocolo.

3.1.1 Números de secuencia

El protocolo DYMO requiere que cada nodo de la red preserve su número de secuencia (*OwnSeqNum*) para asegurarse un mantenimiento de la red libre de bucles. Estos números de secuencia permiten que los nodos determinen el orden de descubrimientos de rutas, con lo que no permiten el uso de información caducada. Hay diferentes números de secuencia:

- El número de secuencia de la dirección del nodo destino de la tabla de encaminamiento de un nodo (*Route.SeqNum*).
- El número de secuencia del nodo destino cuando se envía un RE (*TargetSeqNum*). Si no tiene ningún número en su tabla, el valor que debe llevar es el de cero.
- El número de secuencia del nodo del bloque que transporta el RE (*RBNodeSeqNum*).
- El número de secuencia del nodo inalcanzable debido a que la ruta se ha roto (*UNodeSeqNum*), es transportado por el paquete RERR, en el caso que lo desconozca este será cero.

Para incrementar el OwnSeqNum se tiene que crear un RREQ o un RREP, y cumplir una de las dos condiciones siguientes:

- Que el TargetNumSeq sea superior al número de secuencia del nodo (*OwnSeqNum*).
- Que el TargetNumSeq sea igual al OwnSeqNum y el número de nodos por los que ha pasado un RE es menor al número de nodos intermedios por los cuales ha pasado un bloque hasta llegar al nodo.

3.1.2 Entradas en la tabla de encaminamiento

La tabla de encaminamiento es actualizada cuando se reciben paquetes de control del protocolo DYMO. Estos paquetes pueden contener información de una nueva ruta, actualización sobre un enlace o pueden indicar la ruptura de un enlace.

Cuando se recibe un RE y no se tiene constancia de la ruta especificada en el paquete, el nodo crea una nueva ruta. El nodo introduce en la tabla de enrutamiento los siguientes datos:

[4] El NIST-DYMO utiliza el puerto UDP número 462, mientras el DYMOUM el puerto UDP número 653

- La dirección IP del nodo destino.
- El número de saltos que hay entre el emisor y dicho nodo.
- El tiempo de vida de la ruta (*ROUTE_TIMEOUT*).
- La dirección del siguiente nodo hacia el destino.
- La interfaz por la cual reenvía los datos.
- El tamaño de la subred.
- El número de secuencia del nodo destino.
- Un indicador para saber si el nodo actúa como gateway[5].

En el caso de que una ruta exista en la tabla de encaminamiento se tiene que actualizar cuando:

- El temporizador no haya expirado y el número de saltos del paquete de control sea superior o igual al número que el nodo tiene en su tabla de encaminamiento.
- El temporizador ha expirado y el número del paquete es uno más que el que se tiene en la tabla.
- El número de secuencia del RE es superior que el que tiene el nodo.
- El número de secuencia del nodo destino en la tabla de enrutamiento no es conocido y se recibe un RE con un número de secuencia para ese destino.

Un nodo deja inactiva una ruta después de que pase un tiempo ROUTE_TIMEOUT, este temporizador se va reiniciando cada vez que el nodo reciba un paquete de datos y contenga esa ruta, ya que indica que está utilizando el enlace. Una vez que llegue a cero el temporizador ROUTE_TIMEOUT, la ruta pasa a estado inactivo durante un ROUTE_DELETE_PERIOD. Entonces, ésta no se puede utilizar y es borrada por el nodo cuando expira el temporizador.

Cuando un nodo indica la ruptura de un enlace o le llega un paquete RERR indicando la ruptura de un enlace que utilizaba, el nodo tiene que poner el temporizador ROUTE_TIMEOUT de la tabla de encaminamiento de dicho nodo al final, así se obliga a que la ruta pase a estado inactivo.

3.2 Funcionamiento

El funcionamiento del protocolo DYMO se divide en dos mecanismos básicos; descubrimiento de ruta y mantenimiento de ruta. En el primero se explica cómo se puede descubrir una ruta en la red, y en el segundo se expone la forma que tiene el protocolo para detectar rupturas en las rutas. De esta manera los nodos descubren otra forma de llegar al destino.

[5] Es un nodo en una red informática que sirve de punto de acceso a otra red.

3.2.1 Descubrimiento de ruta

Siempre que un nodo intenta enviar un paquete a un destino, el emisor comprueba que el destino esté en la tabla de encaminamiento. En el caso que ya exista esta ruta, el paquete envía la información al siguiente nodo basándose en la tabla. En el caso de que no esté, se realiza el mecanismo de descubrimiento de ruta.

Este mecanismo envía en modo broadcast el paquete RE indicando en un flag, que se trata de un mensaje de control RREQ. Cuando se crea este mensaje el OwnSeqNum se tiene que incrementar en una unidad. Los principales datos de este mensaje de control son:

- El TTL indica el número de saltos que le faltan para desechar el paquete, cuando es creado el valor es NET_DIAMETER.
- La dirección del nodo a la que se quiere enviar datos.
- El número de secuencia del nodo destino, en el caso que no se conozca este valor está a cero.
- Número de saltos por el que ha pasado el paquete (*THopCnt*).
- Estructuras de datos que informan del encaminamiento de una dirección (RBlock).

Cuando se crea un RREQ se tiene que crear el primer RBlock, el cual contiene:

- La dirección del nodo del RBlock al que pertenece.
- El número de secuencia del nodo que tiene la dirección en el bloque.
- El número de saltos que ha pasado este bloque.
- Un bit indicando si actúa como gateway.

El nodo que crea el RREQ se tiene que esperar RREQ_WAIT_TIME para poder enviar otro mensaje de RREQ. Para reducir una posible congestión en la red, éste tiene que seguir un tiempo de backoff binario exponencial, es decir, el primer intento de RREQ tiene que esperar el tiempo RREQ_WAIT_TIME por dos, el segundo por cuatro y así sucesivamente. Se pueden hacer hasta RREQ_TRIES intentos antes de notificar que el nodo no es accesible. En el descubrimiento de la ruta el nodo emisor guarda los paquetes en un buffer, del que se borra la información si se agotan los intentos.

En el caso que le llegue un RREQ a un nodo intermedio y éste no disponga de la dirección del nodo destino, o el número de secuencia del RREQ (RBNodeSeqNum) sea superior al de su tabla de encaminamiento (Route.SeqNum), este paquete deberá actualizar la tabla de encaminamiento del nodo para realizar la ruta inversa. El nodo tiene que actualizar su OwnSeqNum sumándole uno, introducir un nuevo RBlock con sus propios valores, disminuir el valor del TTL y sumarle un salto al campo THopCnt. Una vez introducidos los cambios en el paquete se reenvía en modo broadcast.

Si un nodo recibe un paquete RREQ que contenga la dirección destino en su tabla de encaminamiento, entonces compara si el Route.SeqNum es superior

al TargetSeqNum del paquete. En caso que no lo sea, se reenvía, ya que el nodo tiene caduca su tabla y actualiza los campos. Cuando es superior el nodo tiene que actualizar su OwnSeqNum sumándose uno y crear un paquete RE indicando que es RREP de respuesta al descubrimiento. Se introducen los saltos que le falta para llegar a dicho nodo, el número de secuencia del nodo destino, la dirección del siguiente salto y el RBlock del nodo que lo emite. Este nodo no debe enviar ningún paquete de RREQ al destino, por lo que el nodo destino tiene que hacer también un descubrimiento de ruta si desea realizar un enlace bidireccional.

Por último tenemos el caso que el nodo destino sea el que procesa el RREQ, en este caso el nodo compara si el OwnNumSeq es superior al TargetSeqNum, en el caso que no sea así se descarta el paquete. Contrariamente, si es superior, el nodo guarda los datos del RREQ en la tabla de encaminamiento para un posible enlace bidireccional. El nodo debe crear un paquete RE indicando que es una repuesta a un descubrimiento (*RREP*). Aumenta en uno el OwnSeqNum e introduce los datos en el paquete, este paquete se transmite al último nodo que ha transmitido el RREQ.

Una vez se envía el mensaje RREP por la ruta inversa, los nodos tienen que transmitirlo de forma unicast[6] por la ruta inversa que le ha llegado el RREQ, estos nodos tienen que sumarle uno al OwnSeqNum y actualizar su tabla de enrutamiento. Cuando es recibido por el creador del mensaje RREQ, éste está preparado para transmitir los datos, estos son enviados vía el buffer.

3.2.2 Mantenimiento de las rutas

Este apartado es similar al descrito en el protocolo DSR. Cada nodo es el encargado de mantener el enlace del siguiente nodo tal y como se puede ver en la Fig.2.2. Cuando un nodo detecta la pérdida de un enlace, éste crea un RERR y lo transmite a los nodos anteriores de la ruta. Para el descubrimiento de una ruptura en una ruta, el draft propone cuatro posibles alternativas:

- Reconocimientos en la capa de enlace (*similar al usado en DSR*).
- Mensajes de Hellos: Este mecanismo no está descrito por el draft en la última versión.
- Descubrimientos de vecinos.
- Timeout de ruta: Este mecanismo no transmite el RERR. Una vez pasado un tiempo ROUTE_TIMEOUT la ruta queda inhabilitada, este tiempo va actualizándose cada vez que pasa un paquete por el nodo que utiliza dicha ruta.

Cuando un nodo descubre una ruptura, éste tiene que crear un paquete RERR, el cual tiene que introducir en el campo UNodeAddress1 la dirección del nodo inalcanzable, en el caso que se sepa el número de secuencia del nodo se introduce en el campo UNodeSeqNum. En caso que no se conozca, este campo toma valor cero. En el campo TTL se introduce el NET_DIAMETER y se

[6] Es la comunicación establecida entre un solo emisor y un solo receptor en una red.

envía en modo broadcast, si hay otra ruta que es perjudicada por la ruptura del enlace, se introduce también en el paquete RERR, en el cual se añaden nuevos campos de dirección y de números de secuencia.

Cuando un nodo recibe un RERR tiene que invalidar la ruta si:

- La ruta que invalida tiene como siguiente salto la misma dirección IP del paquete que ha transmitido el RERR.
- La ruta que invalida tiene como siguiente salto la interfaz del paquete que ha transmitido el RERR.
- El número de secuencia del nodo que nos ha llegado es cero o el resultado de restar el Route.SeqNum del nodo destino y el número de secuencia de la ruta inalcanzable del paquete es menor o igual a cero.

Si no pasa por algún filtro de éstos, la ruta inválida se retira del paquete y si el RERR contiene alguna ruta más se reenvía en modo broadcast. En caso que el paquete ya no tenga ninguna ruta inválida, éste ya no se transmitirá más. Si el paquete pasa todos los filtros, el nodo tiene que reenviarlo en modo broadcast bajando el campo TTL.

3.3 Parámetros y valores por defecto para DYMO

Los valores representados en la tabla 3.1 pueden ser configurados dependiendo de la amplitud de la red o el dinamismo de topología de una red. Por ejemplo, en una red donde los nodos no sufran cambios en su ubicación y sea estable, el ROUTE_TIMEOUT tiene que ser mayor que el descrito en el draft. Todos los nodos deben llevar los mismos ajustes de parámetros, ya que un desajuste en parámetros en el ROUTE_TIMEOUT o en el ROUTE_DELETE_TIMEOUT pueden producir bucles en la red o roturas de un enlace, debido a retrasos arbitrarios de paquetes.

Los principales valores que se tienen que seguir para realizar una correcta implementación del protocolo DYMO son los descritos en la Tabla 3.1

Tabla 3.1 Valores por defecto del protocolo DYMO

Nombre de la variable	Cantidad	Unidades
NET_DIAMETER	10	Saltos
RATE_LIMIT	10	Mbps
ROUTE_TIMEOUT	3000	Milisegundos
ROUTE_DELETE_TIMEOUT	5*ROUTE_TIMEOUT	Milisegundos
RREQ_WAIT_TIME	1000	Milisegundos
RREQ_TRIES	3	Intentos
ROUTE_DELETE_PERIOD	-	-
UNICAST_MESSAGE_SENT_TIMEOUT	-	-
TTL	NET_DIAMETER	Saltos

3.4 Implementaciones del DYMO

Son muy pocas las implementaciones disponibles para este protocolo ya que es bastante reciente. Para este trabajo se han utilizado las dos implementaciones disponibles. Todas ellas necesitan el sistema operativo linux.

- NIST-DYMO (*S.O:Linux*)
  - ✓ Creado por el National Institute of Standards and Technology, funciona sobre Linux con kernels 2.4 y 2.6.
  - ✓ Es necesario el modulo NETFILTER para el correcto funcionamiento de la implementación.
  - ✓ Da soporte para múltiples interfaces, puede hacer funciones de Gateway y puede hacer el enrutado de subredes locales.
  - ✓ El descubrimiento para que sea bidireccional se tiene que realizar en los dos extremos.
  - ✓ Homepage:http://www-x.antd.nist.gov/twiki/bin/view/ANTDProjects/NistDymo

- DYMO-UM (S.O:Linux)
  - ✓ Creado por Pedro M. Ruiz y Francisco Ros, de la Universidad de Murcia.
  - ✓ Funciona en plataformas linux con kernels 2.4, 2.6 y sobre el simulador de redes NS-2
  - ✓ Escrito en lenguaje de programación C y C++.
  - ✓ Es necesario el modulo integrado en Linux NETFILTER.
  - ✓ La versión 6 del protocolo IP no está disponible.
  - ✓ Homepage: http://masimum.dif.um.es/?Software:DYMOUM:Introduction

Se han probado las dos implementaciones sobre una distribución Linux Gentoo, con el kernel 2.6.9 y el modulo NETFILTER activo. Como se verá en el apartado 4.6, la implementación NIST-DYMO no es compatible con el uso de las iptables, la cual cosa no ha permitido realizar pruebas de GAPS y de ancho de banda. La implementación DYMOUM no realiza mantenimiento de rutas, por lo que tampoco se han podido realizar tales pruebas.

Tal y como se puede observar en las implementaciones y en el capítulo, el protocolo Dymo es un protocolo reciente que se está en continuo desarrollo para que en un futuro cercano se realice el estándar y pueda implantarse en las redes Ad-Hoc del futuro.

CAPÍTULO 4. INSTALACIÓN Y CONFIGURACIÓN DE LOS DISPOSITIVOS DE LAS REDES DSR Y DYMO

En este capitulo se explica la instalación y configuración de la red Ad-Hoc para evaluar los protocolos DSR y DYMO. El software necesaro cambia dependiendo del protocolo. Para el protocolo DSR se utiliza un sistema operativo Linux con la distribución Fedora Core 3 y el kernel 2.6.9-1.667. Mientras que para el protocolo DYMO se utiliza la distribución Gentoo de Linux, esta distribución utiliza el kernel 2.6.9. En el capítulo también se explica la configuración de los escenarios que se utilizan para realizar las pruebas.

4.1 Dispositivos y software utilizado

Para poder realizar la red Ad-Hoc con los dos protocolos se han necesitado los siguientes elementos Hardware.

- 4 portátiles Acer Aspire 1692 WLMi.
- 4 tarjetas wireless 3Com 3CRUBSB10075 que cumple el estándar 802.11b/g (*54Mbps*).

En los portátiles están instaladas las dos distribuciones que se utilizan. Para el protocolo DSR es preciso utilizar la distribución Fedora Core 3 (*véase [20]*), se utiliza el kernel 2.6.9-1.667 ya que funciona perfectamente con dicho protocolo. La distribución Fedora tiene un entorno amigable basado en ventanas, la instalación también es muy sencilla y únicamente hay que seguir los pasos descritos. Para el protocolo DYMO es necesario una distribución que contenga el modulo NETFILTER, como la distribución Fedora no tiene el kernel modular, se ha optado por instalar la distribución Gentoo (*véase [21]*), que dicho kernel está basado en un kernel modular, ésta es el 2.6.9, en el anexo C se puede ver como se instala una distribución Gentoo. Los dos kernels disponen del paquete Wireless Tools (*véase [22]*) que contienen herramientas como por ejemplo el iwconfig.

4.2 Configuración de los dispositivos de la red ad-hoc

4.2.1 Instalación de los drivers

Los kernels utilizados no dan soporte para las tarjetas de red 3Com, ya que dichas tarjetas son recientes, mientras que el kernel es un poco antiguo. Para las distribuciones Fedora y Gentoo utilizamos unos drivers realizados por la empresa ZyDAS, que desarrolla software para este tipo de productos, éste utiliza un modulo llamado ZD1211 (*véase [23]*).

Los drivers de la tarjeta se pueden cargar en el kernel al iniciarse el sistema operativo, o pueden cargarse una vez el sistema está en marcha. En la distribución Gentoo cargamos el kernel al iniciarse, mientras que en Fedora se carga una vez es iniciado.

Para llegar a este paso, previamente se ha tenido que descargar el driver, una vez tenemos el driver, se abre una ventana de terminal y se debe ir al directorio donde se encuentra el archivo y descomprimirlo, ya que está en formato “tgz”. Para realizar este cometido se tiene que ejecutar:

 # tar –zxvf ZD1211LnxDrv_2_2_0_0.tgz

Una vez que se descomprime el paquete se tiene que ir a la carpeta, compilar el código fuente (es preciso que este instalado el compilador gcc) y guardarlo en el directorio correspondiente. Una vez lo tenemos instalado solo queda cargarlo. Estos pasos se tienen que realizar con las siguientes órdenes:

 # make
 # make install
 # modprobe ZD1211

En el caso que se esté utilizando la distribución Gentoo no se debe cargar el módulo ya que ha sido cargado con anterioridad.

4.2.2 Instalación de las tarjetas wireless

La instalación en las distribuciones Fedora de las tarjetas 3Com se pueden realizar de dos maneras diferentes; con el entorno gráfico con un terminal. El primero, es muy intuitivo, aunque no detecta del todo bien los dispositivos. El segundo, es el modo que se seguirá en este trabajo, ya que así nos sirve para explicar la instalación en las dos distribuciones.

Inicialmente, se tiene que cargar el módulo del driver si no está cargado previamente. Después se debe utilizar el comando *ifconfig* para dar de alta a una interfaz, asignarle una dirección IP a la tarjeta y una máscara de red. Para realizar estos pasos se tienen que ejecutar las siguientes órdenes:

 # modprobe ZD1211
 # ifconfig eth1 up
 # ifconfig eth1 192.168.10.x netmask 255.255.255.0 (*donde la x depende del portátil*)

Con el comando *iwconfig* se configura la red wireless. Con este comando se indica que el terminal quiere crear o acceder a una red Ad-Hoc, el nombre de la misma y el canal de frecuencia que utiliza la red para transmitir. Estas acciones se ejecutan de la siguiente forma:

 # iwconfig eth1 mode Ad-Hoc

 # iwconfig eth1 essid dsr_net[7]
 # iwconfig eth1 channel 7

En este caso se escoge este canal, ya que estaban ocupados los canales 1 y 14 por otras redes y se decide escoger una intermedia para no tener interferencias con las otras comunicaciones (*lo recomendado es tener una separación de 4 canales)*. Para observar este caso se tiene que introducir el siguiente comando:

 # iwconfig eth1 scanning

El cual nos devuelve el nombre de los canales que detecta la tarjeta y el canal que utiliza. La interfaz eth1 es el nombre de la tarjeta cuando se está utilizando la distribución Fedora, en cambio cuando se utiliza la distribución Gentoo se denomina wlan0.

Una vez realizados los pasos anteriores, se tiene que comprobar que los módulos de las tarjetas están bien cargados y también la configuración de las mismas. Para revisar la correcta carga del módulo se introduce el comando *lsmod*, el cual nos enseña todos los módulos cargados por el sistema operativo. Para revisar la configuración de la tarjeta se utiliza el comando *ifconfig* el cual nos enseña la dirección IP, la máscara de red y la dirección MAC del dispositivo. Por último se revisa la configuración de la red con el comando *iwconfig,* que nos facilita datos de la red creada. Para hacer otra prueba es recomendable realizar *pings* entre todos los nodos de la red. Estos pasos se pueden englobar en *scripts* para una rápida configuración de la red. De esta forma se asegura una buena configuración de la misma. Los *scripts* utilizados para las pruebas se pueden ver en el anexo D.

4.3 Escenarios de pruebas

En este apartado se describen los escenarios que se han utilizado para realizar las pruebas de este trabajo. Definimos dos tipos de escenarios: los escenarios estáticos, en los que los terminales no sufren cambios en la red, y los escenarios dinámicos, que son aquellos que tienen un cambio en la topología de la red.

4.3.1 Escenarios estáticos

En el momento que se incorporan más de dos nodos a la red, el ancho de banda total del canal baja, ya que tienen que compartir éste y competir por el medio. La principal desventaja que se tiene es la proximidad de los nodos.

[7] dsr_net es el nombre empleado en este TFC para la red cuando se utiliza el protocolo DSR. Para el protocolo DYMO esta red se denomina dymo_net.

Éstos, al estar en un mismo rango de cobertura siempre tendrán visibilidad directa con los otros.

Las pruebas de la implementación NIST-DYMO se han realizado alejando los nodos para que no tengan el mismo rango de cobertura (*véase en el anexo E*). Sin embargo este sistema no es del todo fiable. Para contrarrestarlo se ha intentado modificar la potencia de transmisión de las tarjetas 3Com sin exito, debido a que los sistemas operativos utilizados no soportan esta opción para las tarjetas 3Com. Finalmente se optó por la solución empleada en otros trabajos similares a éste. Esta solución es utilizar la herramienta iptables implementada en los sistemas Linux, para poder utilizar correctamente las implementaciones DSR-UU y DYMOUM.

Iptables (*véase en el anexo F*) es una herramienta que es incluida por las distribuciones de Linux utilizadas, ésta nos sirve para emular los escenarios que se quieren realizar, ya que puede filtrar paquetes para que el nodo no los procese, la cual cosa nos permite introducir un número de nodos N y un número de saltos N-1. Se remarca que los nodos están en el mismo rango de cobertura, por lo que el ancho de banda se reduce a medida que se introducen nuevos nodos.

Para realizar pruebas de latencia de paquetes y de ancho de banda, se utiliza el escenario de la Fig.4.2 previamente configurado con las iptables, es una configuración en línea de 4 portatiles y 3 saltos, en la que los nodos sólo ven a sus nodos adyacentes. Estas pruebas se han realizado con los protocolos DYMO, DSR y con rutas estáticas. La comunicación se hace extremo a extremo y saltando por cada nodo. Los nodos tienen que pertenecer a la misma red, el mismo nombre de red y el mismo canal como hemos indicado en el apartado anterior.

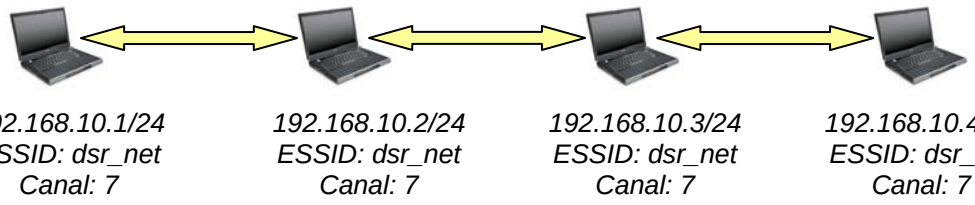


Fig.4.2 Escenario de 4 nodos y 3 saltos en línea

Para configurar la red como en la Fig.4.2 es necesario disponer de la dirección IP y la dirección MAC, ya que siempre se utiliza la misma tarjeta en los terminales para una rápida configuración de los mismos utilizando *scripts*. En la Tabla 4.1 se puede ver la relación entre portátiles, dirección IP y dirección MAC.

Tabla 4.1 Vinculo entre terminales, direcciones IP y direcciones MAC

Terminal	Dirección MAC	Dirección IP
Terminal 1	00:14:7c:5b:08:1f	192.168.10.1
Terminal 2	00:14:7c:5b:08:13	192.168.10.2
Terminal 3	00:14:7c:5b:08:07	192.168.10.3
Terminal 4	00:14:7c:5b:08:08	192.168.10.4

Para que los nodos reenvíen los paquetes se tiene que activar esta opción ya que por defecto no está activada, también se tiene que impedir que los nodos no envíen paquetes *ICMP redirect*. Para realizar estas dos condiciones se tiene que introducir dentro del registro de Linux para cambiar los parámetros. Nos indica 1 activado o 0 desactivado. Se quiere activar el *forwarding* que está en el directorio /proc/sys/net/ipv4 y desactivar los parámetros para generar y recibir *ICMP redirects* que están en el directorio /proc/sys/net/ipv4/conf, estos parámetros se activan con las órdenes:

 # echo 1 > /proc/sys/net/ipv4/ip_forward
 # echo 0 > /proc/sys/net/ipv4/conf/all/ accept_redirects
 # echo 0 > /proc/sys/net/ipv4/conf/all/send_redirects

Con estos comandos cargamos dicha configuración en todas las interfaces. Una vez tenemos instalados los parámetros tenemos que cargar las iptables para que únicamente se vean los nodos adyacentes, esto se realizacon el script que se encuentra en el anexo D.1:

Con estos pasos logramos una correcta configuración de la red estática para que la puedan utilizar los protocolos DSR y DYMO, ya que encaminan la información entre el nodo origen y el destino. En cambio, para realizar el encaminamiento estático se tiene que configurar por comandos. Las rutas estáticas no pueden descubrir el destino si no está en su tabla de encaminamiento, por lo que se tienen que introducir manualmente todas las rutas, para ello se utiliza el comando *route*, el cual nos permite crear rutas estáticas extremo a extremo. A cada terminal se le tiene que indicar la interfaz por la que se envía la información y el siguiente salto que tiene que hacer para cada ruta. Para realizar esto, tenemos que introducir las órdenes que nos encontraremos en los scripts de configuración en el anexo D.3.

Una vez se han realizado todos los pasos, los nodos extremos únicamente pueden enviar información a los nodos opuestos y a sus propios vecinos. Para comprobar una correcta configuración de la red se tiene que utilizar el comando *traceroute* para ver por donde pasan los paquetes o ralizando una captura con la aplicación *ethereal*.

4.3.2 Escenarios dinámicos

El escenario dinámico que se configura en este apartado pretende ser más real, se introduce un corte de conectividad debido a la pérdida de un nodo y

observar sus consecuencias. Estas pruebas se tienen que realizar con protocolos de encaminamiento funcionando, ya que con rutas estáticas no pueden realizar un cambio de ruta para llegar al destino.

Este escenario se utiliza para hacer pruebas con el protocolo DSR, de cambio de pérdida de un enlace y cambio de ruta para llegar a un destino. El escenario dinámico esta compuesto por cuatro portátiles, en el que los dos nodos intermedios tienen visibilidad con todos y los de los extremos únicamente tienen acceso directo con uno, tal y como se puede ver en la fig.4.3.

Para este caso también es necesario la utilización de iptables ya que no se desea que los nodos extremos se detecten. Para ello se tienen que aplicar los comandos incluidos en el anexo D.2.
Con la configuración indicada se consigue que el terminal 1 solo detecte el 2, el 2 detecta el nodo 1 y 4, el nodo 4 se da cuenta que tiene a un salto el nodo 2 y por último el nodo 3 se percata que tiene todos los nodos a un salto, ya que se han borrado todas las iptables de este nodo. Tal y como se puede ver en la Fig.4.3.

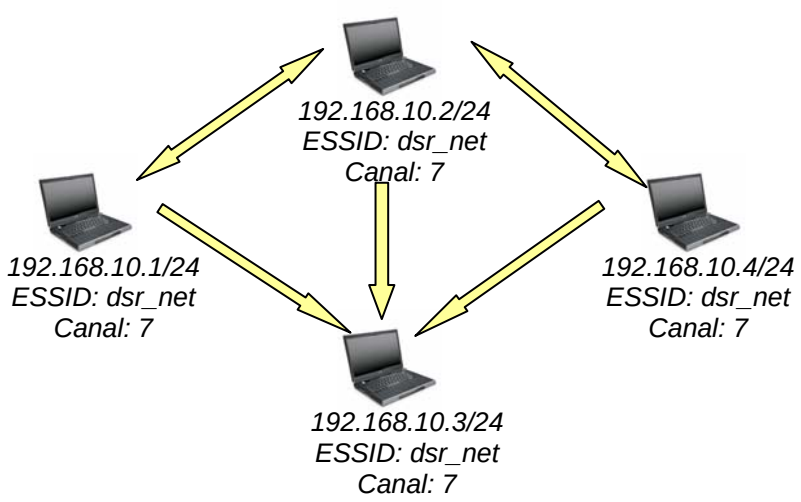


Fig.4.3 Escenario de 4 nodos, 2 saltos y 1 camino.

Al dar de alta una interfaz virtual en la implementación, esta envía 17 mensajes indicando su presencia. Una vez pasado ese tiempo que utiliza la implementación para construir las rutas, se libera el terminal 3, borrando todas las iptables de los otros terminales, y únicamente se pondrán iptables en los nodos extremos, con lo que se consigue 2 caminos, uno que pasa por el terminal 2 y otra que pasa por el 3, este último tiene que ser descubierto por los otros terminales. Se realizan estos cambios, ya que el echo request y el echo reply pueden coger caminos distintos y por lo tanto sería muy dificil medir un gap dandose esa circunstancia, siguiendo estos pasos logramos que sigan la misma ruta los dos. Para que la acción se lleve a cabo correctamente se deben introducir los comandos que estan en el anexo D.2. Una vez introducidos los comandos, se pueden observar en la Fig.4.4

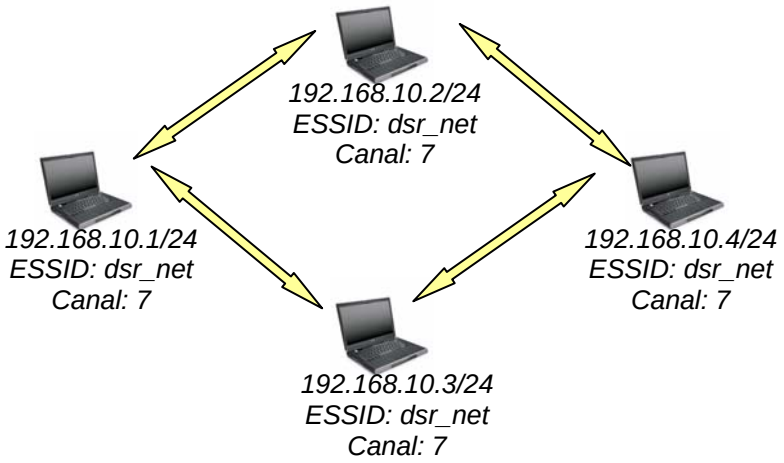


Fig.4.4 Escenario de 4 nodos, 2 saltos y 2 caminos

4.4 Instalación y funcionamiento de DSR-UU

En el caso de la implementación DSR-UU nos descargamos la aplicación comprimida de la web de Uppsala University indicada en el apartado 2.4. Una vez esté en el directorio que se desee instalar, se debe descomprimir, como se puede ver en el siguiente comando:

 # tar –zxvf dsr-uu-0.1.tar.gz

Cuando se crea la carpeta, hay que introducirse dentro de ella y leer el archivo de instalación *README*, el cual nos indica la forma de instalar el protocolo y también la forma de utilizarlo. Para instalar el protocolo se tiene que estar en dicho directorio y ejecutar las órdenes:

 # make
 # make install

La primera instrucción instala el programa en la versión del kernel que está en proceso, mientras la segunda orden instala los módulos que utiliza el protocolo. Para hacer que la implementación entre en funcionamiento, se tienen que cargar los siguientes módulos en el kernel:

 # insmod linkcache.ko && insmod dsr.ko ifname=eth1

La implementación DSR de Uppsala crea interfaces virtuales (*dsr0*) para cada nodo, por lo que pueden coexistir éstas con las que se han creado sin la implementación. Las implementaciones no utilizan el protocolo *ARP*[8] como se podrá ver posteriormente en las capturas de las pruebas. La implementación crea su propia tabla MAC-IP durante el descubrimiento de la ruta. Las tablas de encaminamiento se cargan en un módulo separado de Linux, por lo que cuando

[8] Es un protocolo de nivel de red responsable de encontrar la dirección hardware

la implementación necesite de él, va donde esta e introduce, lee o cambia datos de la tabla.

Para crear las cabeceras extras en los paquetes de datos de DSR, la implementación reserva bytes DSR_OPTS_MAX_SIZE. Con lo que reduce el MTU del dispositivo de red virtual, y en consecuencia origina un problema que se tiene que tener en cuenta a la hora de medir el ancho de banda. Para crear la interfaz se tiene que introducir la siguiente orden:

 # ifconfig dsr0 192.168.100.x up (*donde la x es el número de portátil*)

También es posible utilizar la implementación con un *script* que viene en la implementación. Para inicializar y parar la implementación se tiene que introducir:

 # dsr-uu.sh start eth1
 # dsr-uu.sh stop

La implementación sólo generar la interfaz virtual, envía 17 paquetes anunciándose a toda la red Ad-Hoc, tal y como podemos ver en el valor MaxRequestRexmt. Si hay otro nodo y recibe la información, se introduce los datos en la tabla de encaminamiento, por lo que así al principio no se tiene que realizar ningún descubrimiento de ruta, ya que se asegura que al iniciar la interfaz se realiza. En el caso que se quiera enviar información a una ruta que no esté accesible, el emisor envía un descubrimiento de ruta, si no obtiene respuesta en el tiempo NonpropRequestTimeout éste reenvía el RREQ, este valor se va doblando si no recibe un RREP, hasta un tiempo MaxRequestPeriod. La gestión de mantenimiento de ruta con reconocimiemtos en la capa de red, se realiza con el tiempo MaintHoldoffTime. Estos valores y otros más del protocolo se pueden modificar durante el transcurso de la implementación, están ubicados en el directorio /proc/net/. En este directorio están los archivos:

- dsr_config: Esta toda la información de los parámetros.
- dsr_dbg: Nos enseña la información del protocolo.
- dsr_lc: Se puede ver en este fichero la cache de enlace del protocolo.
- dsr_neigh_tbl: Indica la tabla de vecinos disponibles de un nodo.
- dsr_rreq_tbl: Se puede observar la tabla de descubrimiento.
- maint_buf: Se pueden ver los paquetes del buffer de mantenimiento.
- send_buf: Indica los paquetes que están en el buffer de salida.

Para visualizar el contenido de un fichero de este directorio se tiene que introducir la instrucción siguiente:

 # cat /proc/net/dsr_config

También es posible cambiar los parámetros que nos encontramos en la Tabla 4.2 con el comando:

echo "PrintDebug=0" > /proc/net/dsr_config

Tabla 4.2 Parámetros del DSR-UU

Nombre de la variable	Valor	Unidades
BroadcastJitter	20	Milisegundos
RouteCacheTimeout	300	Segundos
SendBufferTimeout	30	Segundos
RequestTableSize	64	Nodos
RequestTableIds	16	identificadores
MaxRequestRexmt	16	Retransmisiones
MaxRequestPeriod	10	Segundos
RequestPeriod	500	Milisegundos
NonpropRequestTimeout	30	Milisegundos
RexmtBufferSize	50	Segundos
MaintHoldoffTime	250	Milisegundos
MaxMaintRexmt	2	Retransmisiones
TryPassiveAcks	1	Intento
PassiveAckTimeout	100	Milisegundos
GratReplyHoldoff	1	Segundo
PromiscOperation	1	Binario
SendBufferSize	100	Quanta
UseNetworkLayerAck	1	Binario
MAX_SALVAGE_COUNT	15	Salvar

Tal y como se puede ver la implementación cumple las especificaciones del draft, sólo duplica el BroadCastJitter para que se produzcan menos colisiones en el medio.

4.5 Instalación y funcionamiento de DYMOUM

Para poder utilizar la implementación de la Universidad de Murcia, se tiene que descargar de la web indicada en el apartado 2.4. Una vez descargada la implementación comprimida, se procede a la descompresión, con el comando (*se substituye el "0.1" por la versión descargada*):

 # tar zxvf dymoum-0.1.tgz

Una vez descomprimido la implementación se procede a acceder al directorio e instalarlo, esta actúa como un demonio y se tiene que instalar en el root, tal y como se ve puede ver:

 # cd dymoum-0.1
 # make
 # make install # as root

Una vez instalado se tiene que ejecutar la implementación, para realizarlo se tiene que poner el siguiente comando:

 # dymod –v –i wlan0

La implementación no envía paquetes de hellos, ni realiza ningún mantenimiento de ruta, por lo que en el momento que se rompa una ruta los nodos no se dan cuenta y siguen enviando sus datos. En próximas versiones del DYMOUM se arreglará este problema, ya que el autor no ha querido entrar en esta problemática, debido a que no esta especificada en el draft.

A la implementación se le pueden introducir otros parámetros para que actúe de otra forma, como se puede ver:

- -h o --help: Muestra todas las opciones que soporta DYMOUM.
- -d: Pone la implementación en modo deamon.
- -v: Te deja ver el DYMOUM en tiempo real las funciones que realiza.
- -n: Desactiva el modo Path Acumulation.
- -s: Activa el bit S de la cabecera del paquete DYMO.
- -r: Crea más de un RREQ si no obtiene respuesta, después del algoritmo de backoff exponencial.

Los principales parámetros de la implementación se pueden variar en los ficheros kdymo_main.c, dymo_generic.h y rtable.h. Cuando se varien los parámetros, la implementación se tiene que reinstalar para que los éstos se actualicen. Los principales valores de esta implementación son los de la Tabla 4.4:

Tabla 4.4 Parámetros del DYMOUM

Nombre de la variable	Valor	Unidades
Dymo_RateLimit	10	Mbps
Net_Diameter	10	Nodos
Route_Timeout	3	Segundos
Route_Delete_Timeout	5 * Route_Timeout	Segundos
Dymo_Max_NR_Interfaces	10	Interfaces

4.6 Instalación y funcionamiento de NIST-DYMO

Tal y como se ha visto en los dos apartados anteriores, lo primero que se debe hacer es bajar el código de la implementación y una vez bajado y dispongamos de él, se descomprime tal y como se muestra a continuación (*donde “0.5” es la versión. Es la primera versión y por lo tanto la que se ha bajado*):

 # tar zxvf nist-dymo_0.5.tgz

El siguiente paso que se debe realizar es, introducirse en el directorio que se ha descomprimido e instalar el programa, como se ve a continuación:

 # cd nist-dymo
 # cp Makefile-2.6 Makefile
 # make

Ahora la aplicación ya está instalada, para arrancarla se debe llamar al módulo que ha hecho:

 # insmod nist-dymo.ko use_dev=wlan0

Otras posibles opciones que nos deja utilizar la aplicación, son:

- Posibilidad de utilizar subnets: local_subnet=[network address]/[subnet prefix].
- Ignorar ciertas subredes: ignore_subnet=[network address]/[subnet prefix].
- Crear un nodo que actúe como gateway: is_gateway=1.

Los principales parámetros de la aplicación se pueden retocar en el archivo dymo_def.h. Cada vez que se toquen los valores de dicho archivo, la aplicación se tiene que reinstalar para que los parámetros se actualicen. Se observa que cuando la aplicación se inicia, la interfaz no para en ningún momento de enviar hellos, la cual cosa hace que malgaste un ancho de banda y un poco de batería innecesariamente, ya que no tiene que mantener ninguna ruta. La implementación recurre al mantenimiento basado en hellos, basándose en el protocolo AODV. Los principales valores de esta implementación son los de la Tabla 4.4:

Tabla 4.4 Parámetros del NIST-DYMO

Nombre de la variable	Valor	Unidades
NET_DIAMETER	10	Nodos
HELLO_LOSS	4	Paquetes
HELLO_GAIN	7	Paquetes
HELLO_WINDOW	10	Paquetes
HELLO_INTERVAL	1	Segundo
RREQ_WAIT_TIME	1	Segundo
ROUTE_TIMEOUT	3	Segundos
ROUTE_DELETE_TIMEOUT	5 * ROUTE_TIMEOUT	Segundos

Los parámetros de la aplicación se ciñen al draft del protocolo, para realizar el mantenimiento podemos ver los valores utilizados de los hellos, para descubrir la pérdida de una ruta, realiza un método utilizando un intervalo entre un cierto número de paquetes y otro. Manteniendo una histéresis entre los dos.

CAPÍTULO 5. DEFINICIÓN DE LAS PRUEBAS Y ANÁLISIS DE LOS RESULTADOS

En este capítulo se explican los resultados obtenidos de los escenarios que se han mentado anteriormente. El capítulo está formado por tres bloques en los que se describen los resultados de cada implementación. Para el protocolo DSR se han realizado pruebas de latencia de descubrimiento de ruta, ancho de banda, gaps de conectividad y consumo de batería. Mientras que para el protocolo DYMO sólo se ha hecho pruebas de latencia de descubrimiento de ruta, por limitaciones derivadas de las propias implementaciones.

5.1 Pruebas con DSR-UU

En este apartado se representan los resultados de las pruebas con el protocolo DSR en funcionamiento. Hay cuatro tipos de pruebas; de retardo extremo a extremo, aspectos de ancho de banda, gaps de conectividad y consumos de batería. Los dos primeros se realizan con escenarios estáticos, mientras que el tercero se realiza con el escenario dinámico descrito en el capítulo 4.

5.1.1 Retardo extremo extremo

Para realizar dichas pruebas se han escogido el escenario del apartado 4.3.1. Las rutas las tiene que descubrir el protocolo DSR al enviar información a un extremo. Entonces se ve la latencia introducida por el protocolo al iniciar una transmisión de datos.

Para realizar las pruebas se ha tenido que bloquear la entrada de paquetes que no sean nodos adyacentes, para que no se vean entre sí, ya que sino al iniciar la interfaz virtual obtendrían la ruta sin necesidad de utilizar RREQ explícitos para una ruta. Una vez logrado que los portátiles no realicen lo indicado se procede a quitar las reglas de las iptables y a ponerlas tal y como se han indicado en el apartado de escenarios estáticos, para forzar las peticiones de ruta.

Para realizar la prueba se han dejado los valores por defecto del protocolo, para una simulación más real.

Se utiliza el comando ping para realizar el envío de información y así ver el retardo de ida y vuelta de un paquete. Para empezar, se realiza el descubrimiento de ruta con un solo salto y así, hasta llegar al máximo emulado (*3 saltos*). Para realizar dichas pruebas se tienen que introducir los siguientes comandos (*siempre basándose en que el emisor es el terminal 1*):

 # Ping 192.168.100.2 –c 10 (1 salto)

 # Ping 192.168.100.3 –c 10 (2 saltos)
 # Ping 192.168.100.4 –c 10 (3 saltos)

Se han realizado 15 pruebas para cada salto. Las pruebas son de 10 pings cada uno. En éstas se pueden ver la latencia de descubrimiento de ruta. Para cada intento se tiene que dar de baja la interfaz y crear otra con una nueva IP. Ya que sino el protocolo tiene en la caché información de encaminamiento del destino.

En la Fig.5.1 se ve una media del RTT del primer paquete y de los nueve restantes, se puede ver que a medida que se van sumando nodos este retraso va aumentando linealmente. En el primer ping se le suma el tiempo de descubrimiento de ruta y el retardo de ida y vuelta del paquete.

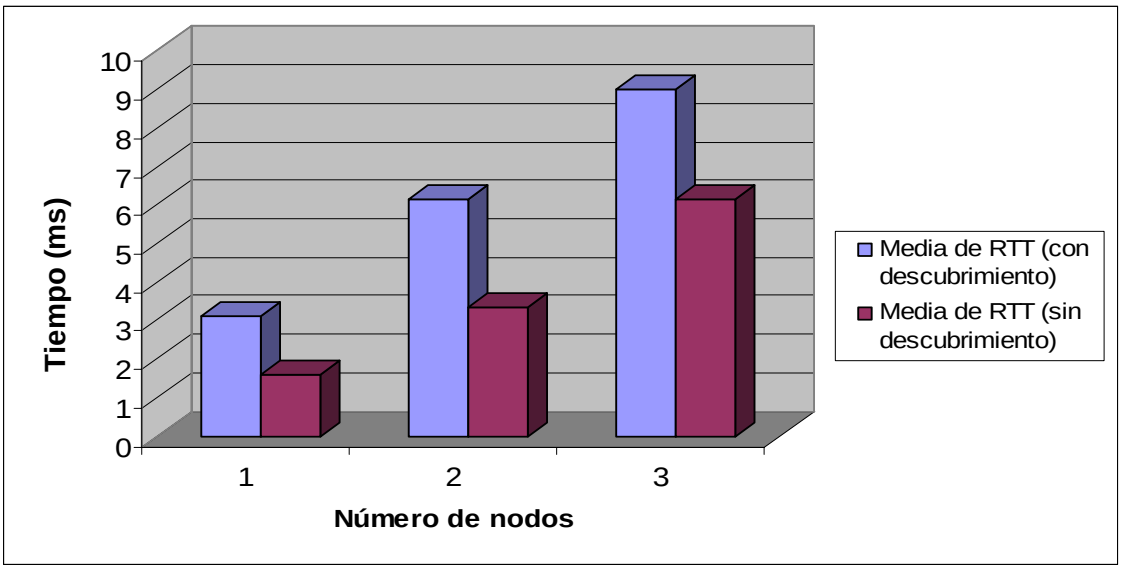


Fig.5.1 Comparativa de RTTs, con descubrimiento y sin descubrimiento

Se ve en la Fig.5.1 que el primer ping es superior a los otros ya que tiene que hacer el descubrimiento de la ruta. Por lo que primero envía un paquete RREQ y espera respuesta de un RREP, cuando recibe la respuesta el emisor envía el primer ping (*echo request*) y espera hasta que le llega el echo reply. Por lo que el tiempo RTT tiene que ser aproximadamente el doble que el tiempo de RTT sin descubrimiento de rutas. Cabe destacar que este tiempo es inferior cuando se utilizan rutas estáticas ya que en este caso no tienen que descubrir rutas y no introducen información de encaminamiento en los paquetes.

5.1.2 Ancho de banda

En este apartado se realizan pruebas de ancho de banda para medir como repercute el protocolo en el ancho de banda extremo a extremo. Se utiliza el mismo escenario que en el apartado anterior, es decir, todos los nodos en cadena. En este caso se utiliza la herramienta iperf (*véase [24]*). Para ello se

debe abrir un terminal y asignar un nodo cliente y un nodo servidor. El nodo cliente es el que envía el tráfico y al que se le asigna las opciones que debe seguir el tráfico. El nodo servidor es el que recibe el tráfico y el que muestra los resultados. El rendimiento de ancho de banda es posible medirlo vía TCP o UDP. En el lado del cliente se le asigna un tiempo, un ancho de banda y una cantidad de datos a transmitir y poder así medir el ancho de banda.

Tal y como se ha hecho anteriormente el nodo emisor (*cliente*) es el terminal 1 y, dependiendo del número de saltos, se varía el receptor. Para realizar estas pruebas se ha enviado tráfico durante 30 segundos a una velocidad de 9 Mbps y con un tamaño en el campo de datos de 1389 bytes. Para realizar medias fiables se han realizado 10 pruebas de flujos TCP y otras 10 de flujo UDP. Los comandos a seguir son los siguientes.

 # iperf –s –u (servidor utilizando flujo UDP)
#iperf –c 192.168.100.4 –u –b 9M –l 1389 –t 30 (Cliente utilizando flujo UDP, la ip varia dependiendo el número de saltos)
 # iperf –s (servidor utilizando flujo tcp)
 # iperf –c 192.168.100.4 –b 9M –l 1389 –t 30 (Cliente utilizando flujo TCP, la ip varia dependiendo el número de saltos)

Se han escogido estos valores en los parámetros ya que; con dicho tiempo tenemos una muestra fiable de ancho de banda, con este ancho de banda enviamos a máxima velocidad y por lo tanto no se trabaja a menos velocidad de lo que puede ofrecer, la medida de los paquetes es la máxima para que no los fragmente, ya que la implementación reserva unos bits en la cabecera como hemos comentado anteriormente.

A medida que se van incorporando nodos el ancho de banda va cayendo, esto es debido a que todos los nodos tienen que ir compartiendo el ancho de banda del canal.

En este punto nos interesa ver cómo repercuten los mensajes de control enviados por la aplicación en el ancho de banda. Para ello se ha variado el parámetro MaintHoldoffTime, de 50 ms a 5000 ms. Se observa que con 50 ms aún envía periódicamente los mensajes en los instantes esperados, ya que con otras aplicaciones se ha demostrado que variando mucho los mensajes de control no se envían estos periódicamente.

Se tiene en cuenta que, a más tráfico de control, menor será el ancho de banda disponible, y a su vez a menos tráfico de control mayor será el ancho de banda, pero mayor es el tiempo de detección de rotura de una ruta.

En este caso se ven los resultados de forma separada según sea el número de saltos. En la Fig.5.2 se puede ver el ancho de banda de 1 salto con flujos TCP y UDP.

Se puede ver que en TCP se tiene un ancho de banda disponible menor que en UDP debido a que el protocolo de transporte incorpora un mecanismo de establecimiento de enlace y un mecanismo de reconocimiento. En cambio UDP

es un protocolo que no está orientado a la conexión, permite una transmisión continua y por lo tanto es el protocolo más adecuado para proporcionar el ancho de banda disponible para estas pruebas.

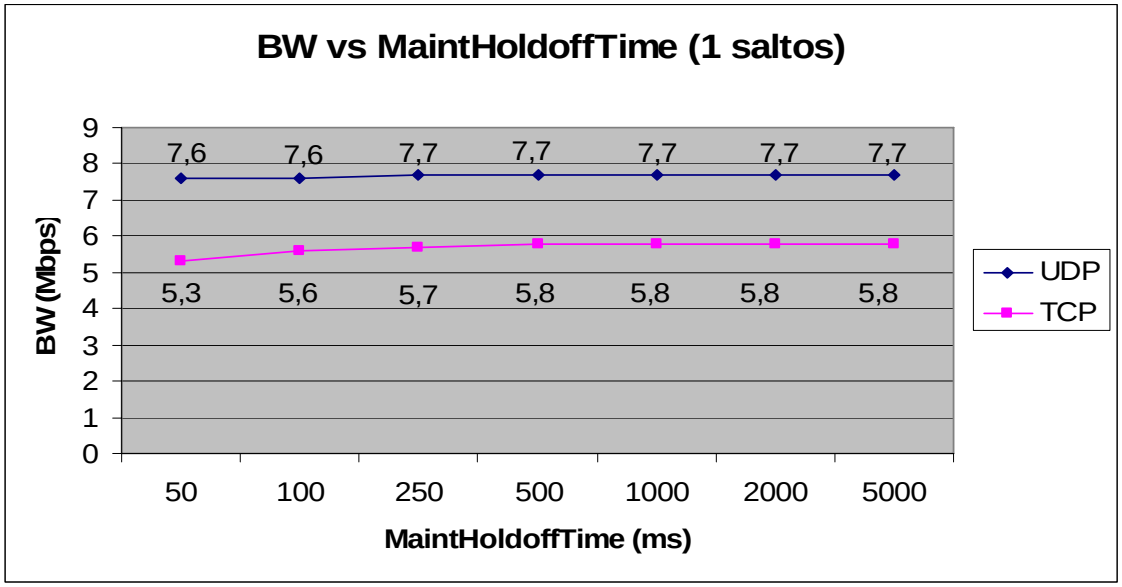


Fig.5.2 Ancho de bando con 1 salto

Si la implementación del DSR trabaja con el valor más pequeño de MaintHoldoffTime (*50 ms*) el ancho de banda de control es un 1.509% del ancho total disponible, mientras que a partir de un valor de 250 ms el ancho de banda consumido por los datos de control es prácticamente cero.

Tal y como se puede ver en la Fig.5.2 los valores a partir de 100 ms para la información de control, ya los podemos dar como óptimos ya que no consumen prácticamente ancho de banda con 1 salto. Se escogerán estos valores dependiendo de la movilidad de los terminales, es decir, para una red Ad-Hoc que los nodos estén en continuo movimiento, se escogerá un MaintHoldoffTime de 100 ms ya que de esta forma se detectan más rápidamente los cortes en las rutas, como se verá en el apartado 5.1.3. En casos que los nodos tengan una movilidad menos acentuada el parámetro MaintHoldoffTime subirá, para así poder utilizar una mayor velocidad a la hora de enviar datos.

En la Fig.5.3 se pueden ver los resultados obtenidos utilizando 3 nodos y 2 saltos. En esta prueba se puede ver que el valor del MaintHoldoffTime entre 250 ms y 500 ms va estabilizando el ancho de banda al máximo posible dando una rápida detección en caso de pérdida de enlace con un nodo. El ancho de banda se reduce aproximadamente a la mitad, debido a que ahora existen dos nodos que intentan acceder al medio a la vez y por tanto que compiten por él. Por lo que estos verán reducida a la mitad la capacidad de transmisión. El ancho de banda de control utilizando 50 ms (*este es el más grande de todos*) es de 14.098% del total, por lo que es más elevado que en el punto anterior, esto es debido a que el ancho de banda total ha caído y hay un nodo más transmitiendo mensajes de control.

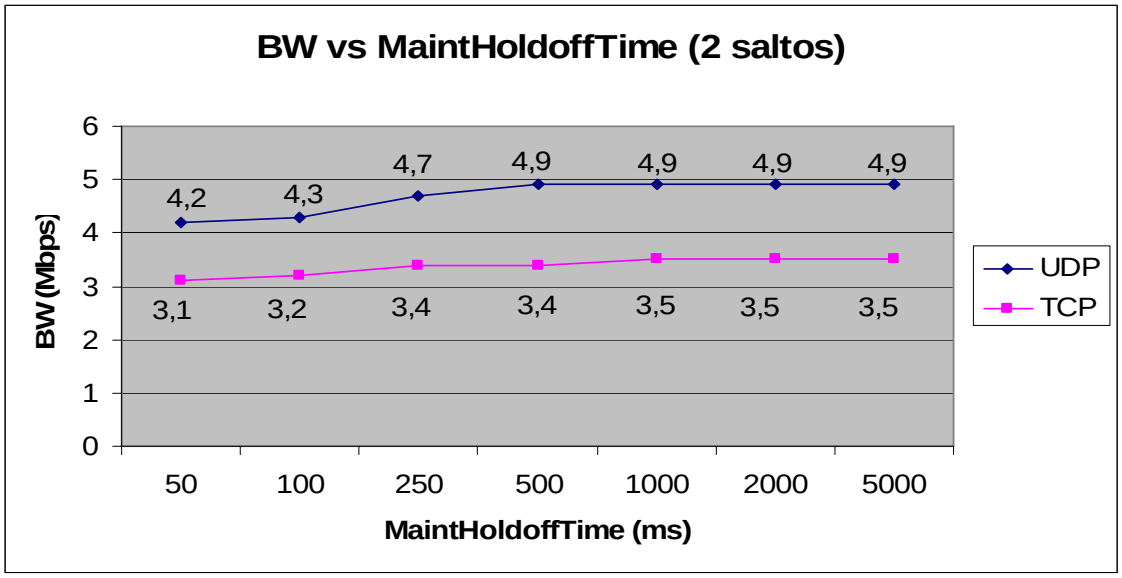


Fig. 5.3 Ancho de banda con 2 saltos

En la Fig.5.4 se puede ver que para tener un mayor ancho de banda disponible se precisa de un valor MaintHoldoffTime grande, de aproximadamente 1000 ms, a partir de este valor el ancho de banda disponible se estabiliza ya que no es tan grande el porcentaje de datos de control enviado al nodo siguiente. El problema de enviar las peticiones de reconocimiento tan espaciadas es que si una ruta se rompe se detecta mucho más tarde que en los otros casos, por lo tanto el ancho de banda obtenido con valores entre 250 y 500 ms se puede considerar adecuado, debido a su rápida respuesta en roturas de rutas. El ancho de banda utilizado por datos de control en el peor caso en este escenario, es de un 27.63%.

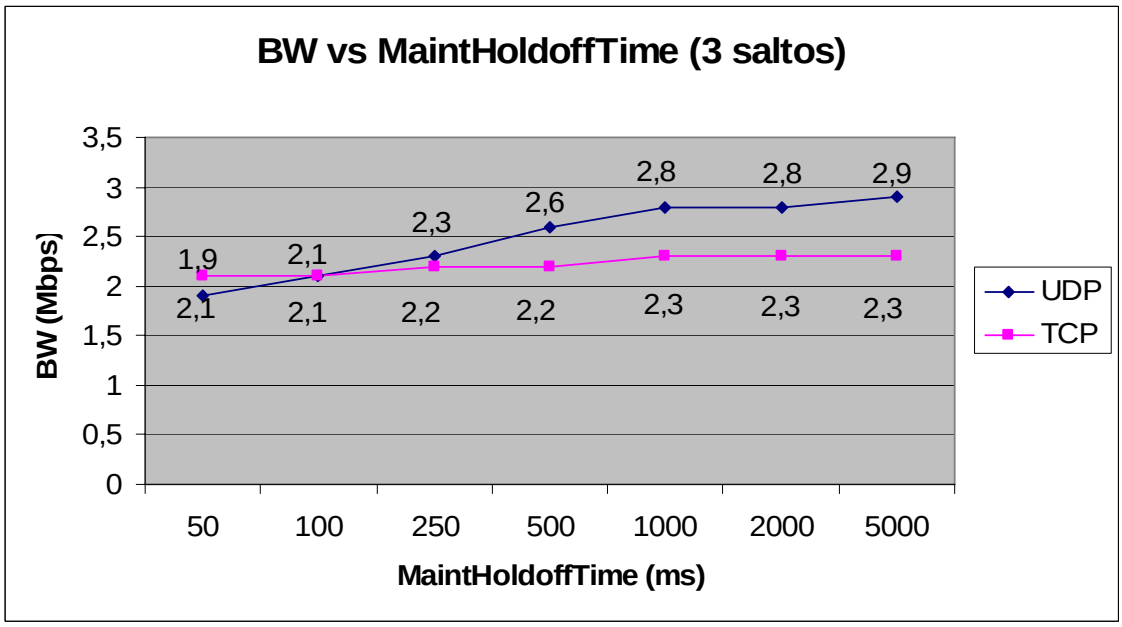


Fig.5.4 Ancho de banda con 3 saltos

Para concluir este apartado, se muestra en la Fig.5.5 el ancho de banda disponible en todos los casos. Esta figura es un gráfico resumen de todos los anteriores.

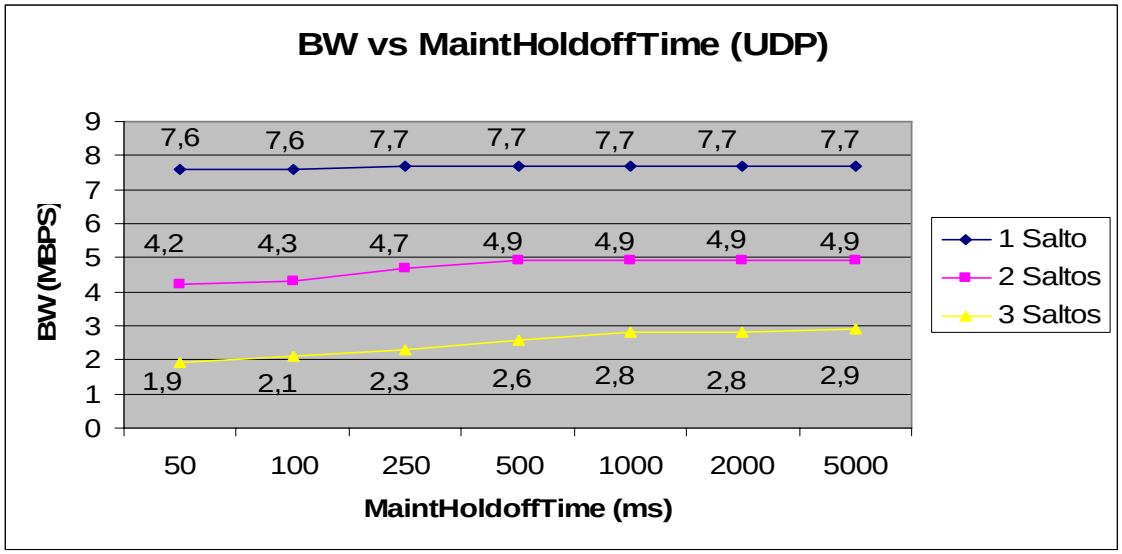


Fig.5.5 Resumen de ancho de banda en UDP

En la Fig.5.5 se puede ver que a mayor número de saltos menor es el ancho de banda disponible debido a que los terminales tienen que partirse el ancho de banda disponible del canal. También se puede observar que a mayor tiempo entre peticiones de reconocimientos el ancho de banda se tiende a estabilizarse al máximo posible. Por último indicar que el tanto por ciento de ancho de banda utilizados por mensajes de control, crece con el número de saltos.

5.1.3 Gaps de conectividad

Para realizar estas pruebas se ha utilizado el escenario dinámico descrito en el apartado 4.3.2. Éstas se realizan con la herramienta ping, y con valores desde 10 ms hasta 750 ms dependiendo del tiempo empleado para las peticiones de reconocimiento. Los timepos entre los pings deben ser mayores a medida que crece el MaintHoldoffTime ya que sino el buffer de los nodos satura, debido a que no saben si está el enlace activo.

Se define el tiempo de gap como el tiempo que tarda un nodo en cambiar de ruta por causas de una ruptura de la misma, este valor se mide en el receptor y es el tiempo transcurrido desde que recibe un paquete por un nodo que posteriormente se da de baja y el primer paquete que recibe reencaminado por otro nodo. En este tiempo el emisor ha tenido que percatarse de que la ruta estaba rota, borrar dicha ruta y buscar otra nueva para poder seguir con la comunicación.

La configuración del escenario es la descrita en el apartado 4.3.2, cuando todos los nodos estén preparados para poder trasmitir, en el terminal 1 se debe introducir el comando:

 # ping 192.168.100.4 –i 0.01 (*dependiendo del intervalo utilizado*)

Anteriormente se ha encendido en todos los nodos la aplicación ethereal para cerciorarse de que sigue la ruta que se ha forzado. Una vez se introduce el comando en el terminal, se tiene que provocar la rotura del enlace. Este punto se realiza dando de baja la interfaz wireless del nodo 2. Una vez realizada esta acción el emisor tiene que reencaminar la información a otro terminal para poder llevar la información al destino. Para dar de baja la interfaz, se introduce el comando:

 # ifconfig eth1 192.168.10.2 down

En el nodo receptor se tiene que parar la captura de la aplicación ethereal cuando ya encamine los datos por la nueva ruta. Una vez se tenga esta información se analiza la captura del ethereal para poder obtener el gap de esta ruta. En la Fig.5.5 se puede ver un gap obtenido con un MaintHoldoffTime de 250 ms en el lado del receptor.

No.	Time	Source	Destination	Protocol	Info
845	283.268929	192.168.103.4	192.168.103.1	IP	Unknown (0xa8)
846	*REF*	192.168.103.1	192.168.103.4	IP	Unknown (0xa8)
847	0.000056	192.168.103.4	192.168.103.1	IP	Unknown (0xa8)
848	0.204918	192.168.103.1	255.255.255.255	IP	Unknown (0xa8)
849	0.205713	192.168.103.1	255.255.255.255	IP	Unknown (0xa8)
850	0.206994	192.168.103.1	255.255.255.255	IP	Unknown (0xa8)
851	0.210027	192.168.103.1	192.168.103.4	IP	Unknown (0xa8)
852	0.210062	192.168.103.4	192.168.103.3	IP	Unknown (0xa8)
853	0.210777	192.168.103.4	192.168.103.1	IP	Unknown (0xa8)
854	0.212922	192.168.103.1	192.168.103.4	IP	Unknown (0xa8)
855	0.212984	192.168.103.4	192.168.103.1	IP	Unknown (0xa8)
856	0.218021	192.168.103.1	192.168.103.4	IP	Unknown (0xa8)
857	0.218086	192.168.103.4	192.168.103.1	IP	Unknown (0xa8)
858	0.221146	192.168.103.1	192.168.103.4	IP	Unknown (0xa8)
859	0.221728	192.168.103.4	192.168.103.1	IP	Unknown (0xa8)
860	0.222131	192.168.103.1	192.168.103.4	IP	Unknown (0xa8)
861	0.222190	192.168.103.4	192.168.103.1	IP	Unknown (0xa8)
862	0.222918	192.168.103.1	192.168.103.4	IP	Unknown (0xa8)
863	0.222977	192.168.103.4	192.168.103.1	IP	Unknown (0xa8)
864	0.223716	192.168.103.4	192.168.103.2	IP	Unknown (0xa8)
865	0.224872	192.168.103.1	192.168.103.4	IP	Unknown (0xa8)

Fig.5.5 Gap obtenido en el receptor

Se puede ver que el paquete número 846 es el último paquete antes de la ruptura en la ruta que pasa por el nodo 2. Desde el paquete 848 hasta el 850 son paquetes RREQ, entonces el nodo 3 tiene que responder con un RREP y el emisor puede reanudar la transmisión pasando por el nodo 3. El paquete número 851 es el primer paquete que recibe pasando por el nodo 3, por lo tanto el tiempo de gap es de 209.971 ms tal y como se puede ver. Por último el paquete que envía el nodo 4 hacia el nodo 2 es de petición de reconocimiento, ya que los echos reply son enviados a este nodo, debido a que el receptor no

se ha percatado aun que esa ruta no está rota, por lo que las respuestas no llegan al emisor.

En la Tabla 5.1 se puede ver los tiempos de gaps de conectividad que se han conseguido variando el MaintHoldoffTime, los intervalos de flujo dependen según el valor de las peticiones de reconocimientos, desde 50 ms hasta 500 ms se utiliza un intervalo de 10 ms, para los valores de 1000 ms y 2000 ms se utiliza un valor de 50 ms, mientras que en el valor más alto el intervalo es de 750 ms.

Tabla 5.1 Gaps obtenidos

MaintHoldoffTime (ms)	50	100	250	500	1000	2000	5000
Gap #1 (ms)	67.8	19.0	227.1	350.9	235.9	1322.9	6036.0
Gap #2 (ms)	121.2	107.9	347.7	534.1	175.6	1446.1	4510.4
Gap #3 (ms)	80.9	108.1	264.0	506.6	829.9	530.7	2274.6
Gap #4 (ms)	105.9	72.3	210.0	428.2	236.1	895.9	4513.1
Gap #5 (ms)	81.0	54.0	308.8	189.4	536.5	1056.8	3760.3
Media de gaps (ms)	**87.5**	**72.3**	**271.5**	**401.8**	**402.8**	**1050.5**	**4218.9**

Estos tiempos de gaps se pueden ver gráficamente en la Fig.5.6.

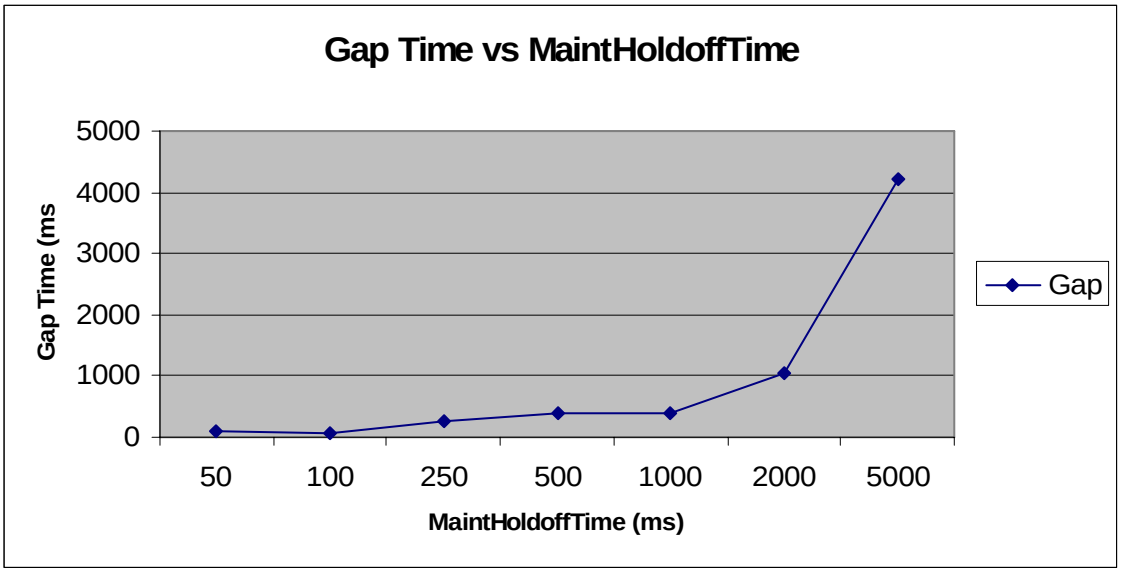


Fig.5.6 Gaps obtenidos

En la Fig.5.6 se puede ver que a medida que se amplia el MaintHoldoffTime el tiempo de gap crece, debido a que los reconocimientos son enviados con intervalos más grandes.

En el lado del emisor se pueden calcular los gaps tal y como está descrito en la Fig.5.7, donde se pueden ver la petición de reconocimiento, la petición de nueva ruta y la recepción de una nueva ruta.

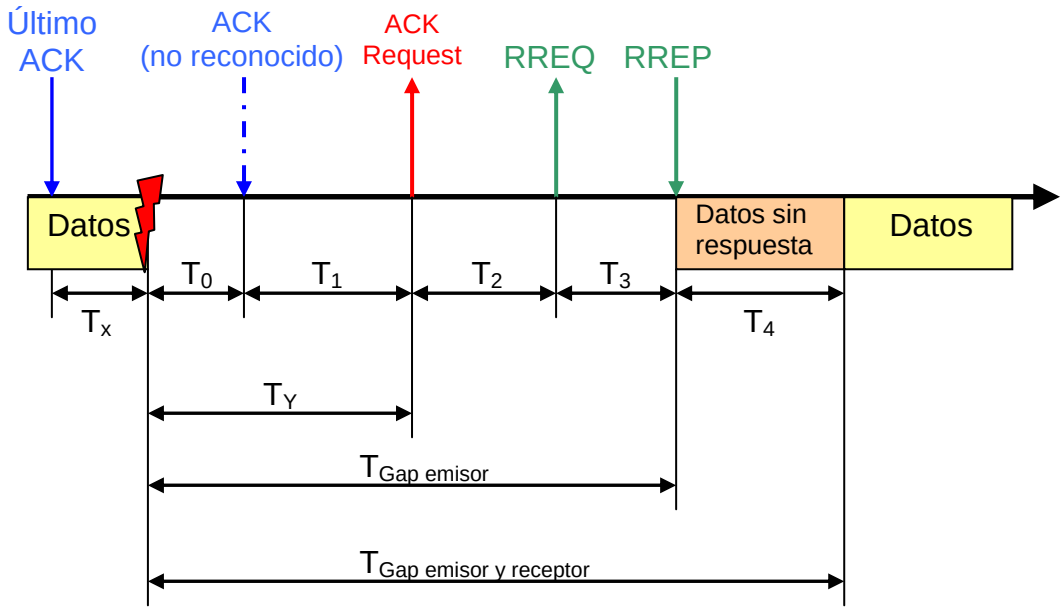


Fig.5.7 Cronograma de un gap de conectividad en el emisor

En el cronograma se ve la asignación de tiempos en que se ha dividido un tiempo de gap. A continuación se describe todos los tiempos:

- T_x: Es el tiempo transcurrido desde el último ACK[9] recibido por un nodo hasta la perdida de conectividad del nodo siguiente.
- T_Y: Tiempo que dista entre la pérdida de conectividad y la petición de reconocimiento del emisor.
- T_0: Este periodo es el que pasa entre la perdida de conectividad hasta que el emisor debería recibir el ACK del nodo posterior.
- T_1: Es el tiempo entre el reconocimiento que no recibe el emisor y la petición de reconocimiento.
- T_2: Es el periodo transcurrido entre la petición de reconocimiento y el nuevo descubrimiento de ruta.
- T_3: Tiempo que pasa desde la petición de ruta y la respuesta de una nueva ruta.
- T_4: Este tiempo es el que dista entre que el emisor empieza a enviar datos y obtiene la primera respuesta del receptor.
- $T_{gap\ emisor}$: Es el tiempo que pasa desde que el emisor pierde la conectividad con el receptor y la encuentra gracias a una nueva ruta.
- $T_{Gap\ emisor\ y\ receptor}$: Es el tiempo total de un gap considerando el sentido de emisión y recepción de forma conjunta.

En la Tabla 5.2 se puede ver las medidas de estos tiempos. Se han obtenido realizando unas pruebas con la herramienta ping e intervalos de 10 ms en el caso que utilice una petición de reconocimiento cada 100 ms, y con intervalos de 50 ms utilizando peticiones de reconocimientos cada 500 ms. Gracias a los valores de la Tabla 5.2, se puede explicar que el valor de tiempo entre gaps

[9] Se refiere al ACK de nivel 3 que emplea DSR-UU

medido en el receptor es inferior del que en realidad tendría que salir. Cabe remarcar que las pruebas de la Tabla 5.2 no son las mismas que las de la Tabla 5.1. En el caso que fueran iguales, tendrían que salir tiempos de gaps de conectividad similares.

Tabla 5.2 Resultados del Cronograma de gaps en el emisor

	MaintHoldoffTime (100 ms)	MaintHoldoffTime (500 ms)
T_X	69.2	223.7
T_Y	45.1	332.1
T_0	30.8	276.3
T_1	13.8	55.8
T_2	13.4	53.8
T_3	4.7	1.2
T_4	39.0	59.9
$T_{gap\ emisor}$	63.1	387.1
$T_{Gap\ emisor\ y\ receptor}$	102.1	446.9

Según la Tabla 5.2 se puede decir que el T_1 y el T_2 es el tiempo que se introduce entre los intervalos de envios de pings, ya que practicamente es el mismo tiempo. Es decir cuando se introduce un intervalo entre pings de 10 ms, T_1 y T_2 son aproximadamente 10 ms cada uno.

Un cálculo teórico de medio de un gap de conectividad en el receptor se realiza con la Fórmula 5.1:

$$T_{Gap} = T_{detección\ de\ ruptura} + T_{descubrimiento\ de\ nueva\ ruta} + 0.5 * RTT \quad \textbf{(5.1)}$$

Donde el tiempo de detección de ruptura depende del tiempo introducido en el MaintHoldoffTime y del intervalo introducido en el flujo del emisor. Es decir en el cronograma de la Fig.5.7 el tiempo de detección es el de la Fórmula 5.2 y el tiempo de T0 esta indicado en la Fórmula 5.3:

$$T_{detección\ de\ ruptura} = T_0 + T_1 + T_2 \quad \textbf{(5.2)}$$

$$T_0 = \text{v.a. } [0, \text{MaintHoldoffTime}] \quad \textbf{(5.3)}$$

Durante todo el tiempo transcurrido en el gap, el nodo destino no obtiene los paquetes que envía el emisor, por lo que estos paquetes son perdidos. Para saber exactamente cuántos de éstos son perdidos, se ha observado mediante en la aplicación ethereal, el último paquete recibido por el emisor, antes del gap, se abre éste y se coge el campo *identification* del paquete IP y se le resta al mismo campo del primer paquete que recibe después del gap. Los resultados de estas pruebas están representados en la Tabla 5.3. En esta tabla también están los resultados teóricos de perdidas de paquetes, para poder comparar éstos con las medias obtenidas en los casos anteriores.

Tabla 5.3 comparativa de paquetes perdidos teóricos y reales

MaintHoldoffTime (ms)	50	100	250	500	1000	2000	5000
Media de gaps (ms)	87.5	72.3	271.5	401.8	402.8	1050.5	4218.9
Gap #1 (paquetes perdidos)	4	1	18	30	20	105	525
Gap #2 (paquetes perdidos)	2	7	16	45	20	115	375
Gap #3 (paquetes perdidos)	5	7	9	37	65	40	150
Gap #4 (paquetes perdidos)	4	5	14	16	20	70	375
Gap #5 (paquetes perdidos)	3	3	12	47	50	85	300
Media de tasa (ms/paq)	0.075	0.0809	0.065	0.089	0.01937	0.01959	0.00133
$T_{\text{detección de ruptura}}$ (ms)	51.66	74.7	155.8	272.5	603.2	1102	4003.7
Media de paquetes real	**3.6**	**4.6**	**13.8**	**35**	**35**	**83**	**345**
Media de paquetes teóricos	**3.9**	**6**	**10.1**	**24.2**	**58.4**	**107.95**	**399.4**

Para realizar los valores teóricos se tiene que tener en cuenta la tasa de envío de paquetes que se utiliza, para ello se hace una media entre los 10 paquetes enviados. En la Fórmula 5.4 se realiza un cálculo de paquetes perdidos, utilizando un MaintHoldoffTime de 250 ms, tal y como está lo definido en el draft.

$$\text{Paquetes perdidos} = T_{\text{detección de ruptura}} * \text{Tasa de envio} \quad \textbf{(5.4)}$$

$$\text{Paquetes perdidos} = 155.8 \text{ ms} * 0.065 \text{ ms/paq} = 10.1 \text{ paquetes} \quad \textbf{(5.5)}$$

La tasa de envío se puede dar como buena, ya que no distan mucho entre sí los paquetes. Se tiene una diferencia sustancial de pérdidas de paquetes a medida que crece el MaintHoldoffTime. Los valores teóricos son superiores o inferiores a los medidos dependiendo del MaintHoldoffTime, ya que el tiempo T_0 es variable, como se puede ver en la Fórmula 5.3, por lo que se ha escogido la mitad de éste.

5.1.4 Aspectos de consumo de batería

Tal y como se ha descrito anteriormente el consumo de la batería en los nodos es muy importante, ya que ésta es limitada en los nodos de una red Ad-Hoc. Utilizar protocolos de encaminamiento que mantengan la ruta, gastan este recurso, ya que cada paquete de control enviado supone consumo energético, si el mantenimiento se realiza a nivel 3.

El valor de MaintHoldoffTime se varía en este apartado para ver cómo repercute el protocolo en el gasto de batería. En el caso que éste valor sea muy pequeño, detectara las rupturas de rutas antes, pero enviaría más cantidad de peticiones de reconocimiento. En cambio si este valor crece, la detección de las rupturas de rutas es más grande, pero la cantidad de peticiones de reconocimientos baja notablemente.

Para realizar esta prueba, se han puesto 3 ordenadores en línea, y se configura la red Ad-Hoc tal y como se ha podido ver en el apartado 4.3.1. Una vez configurado el escenario se ha escogido utilizar la herramienta ping con intervalos de 10 ms. Los paquetes ICMP tienen que realizar echo request y echo reply, por lo que el tráfico de datos es grande. Se ha escogido medir el consumo de batería en el nodo 2 ya que es el que mayor tráfico recibe en el escenario.

La batería del nodo 2 se carga al máximo, en el nodo 3 se enciende la aplicación *ethereal* para medir el tiempo desde el primer paquete hasta el último que reciba. Una vez que el nodo emisor empieza a enviar datos el nodo intermedio se desconecta de la alimentación y se cierra la pantalla para lograr un consumo de batería más real del protocolo. Una vez que se consuma toda la batería del nodo enrutador se apaga y el terminal receptor deja de recibir información. En este momento se apaga la aplicación *ethereal* y se calcula el tiempo de duración de batería.

En la Tabla 5.4 se puede ver las pruebas que se han realizado y en la Fig.5.8 se puede ver la tendencia a bajar el consumo cuando se utilizan más tiempo en las peticiones de reconocimientos o incluso cuando no existe protocolo de encaminamiento, y por lo tanto, no existen datos de control.

Tabla 5.4 Duración de batería en función del MaintHoldoffTime y sin él

MaintHoldoffTime (ms)	50	100	250	-
Duración de batería (min)	168	171	176	179
Duración de batería (h:min)	2:48	2:51	2:56	2:59

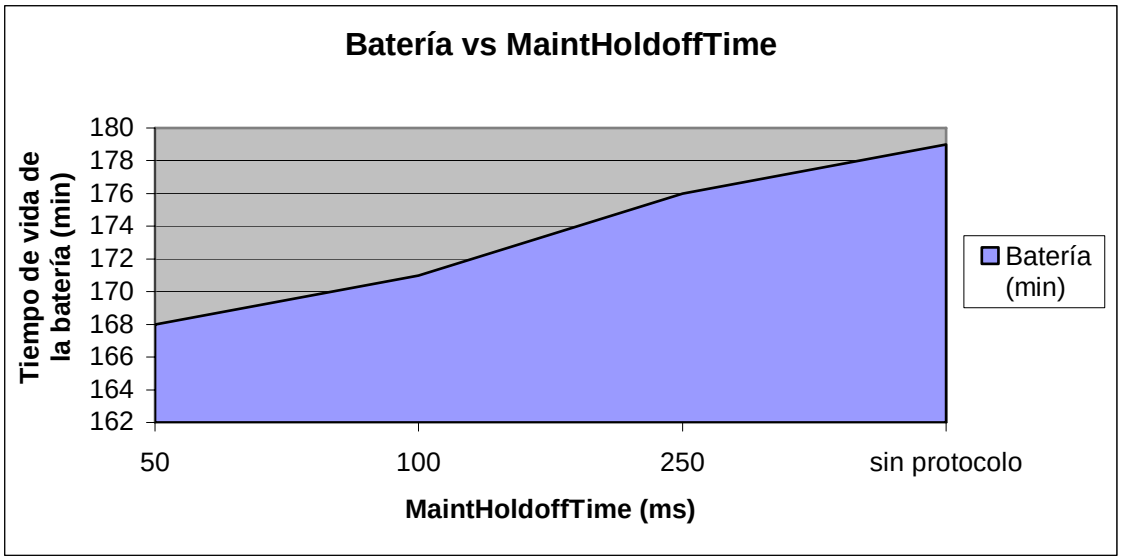


Fig.5.8 Duración de batería con DSR y sin él

Tal y como se puede ver en la Fig.5.8, cuando se introduce un valor de 50 ms en el MaintHoldoffTime, la duración de batería baja 11 minutos respecto al valor que se obtiene en el caso que no se utilice protocolo de encaminamiento.

Por lo tanto se utiliza un 6.14% menos la batería en el peor de los casos. La cual cosa para una red de sensores o una red en el que los terminales sean semi-fijos es un gasto grande.

5.1.5 Problemas de la implementación

La implementación del DSR no funciona del todo correctamente, es la primera versión que se ha publicado. Está no es posible utilizarla en la distribución Gentoo, por lo que se optó por la distribución Fedora.

La implementación se bloquea después de un tiempo funcionando y se tiene que reiniciar el sistema operativo, ya que también bloquea este. Otro problema presente en la implementación es que si se envía un flujo grande de información en un espacio corto, los nodos cuando utilizan una petición de reconocimiento grande se bloquean, ya que guardan toda esta información en un buffer y la saturan.

Muchas veces cuando un nodo intermedio está reencaminado información esté se paraliza y no deja reenviar más información, esta información es recibida por este nodo pero no reenviada. Cuando se intenta enviar información de este nodo, aparece un mensaje que indica que la operación no está permitida.

El principal problema en cambio es, que la ruta que caduca debido al RouteCacheTimeout no se borra nunca de la caché. Una vez pasado esté tiempo comienza en un valor muy grande, por lo que no se borran del todo bien las rutas cuando no son utilizadas.

Por último cabe destacar que cuando un nodo no recibe la respuesta a la petición de reconocimiento, esté debería hacer un par de peticiones más tal y como indica en sus parámetros, pero en cambio únicamente realiza una, tal y como se ha visto en el apartado de los gaps de conectividad.

Esta implementación sufrirá cambios a lo largo del tiempo, debido a que es reciente y sobretodo a que el protocolo aún no está estandarizado. Cuando el protocolo se estandarice la implementación se estabilizará y se solventarán dichos fallos y otros posibles.

5.2 Pruebas con DYMOUM

En este apartado se realizan las pruebas con la implementación DYMOUM del protocolo DYMO. En esta sección se realizan pruebas de latencia de extremo a extremo, como se ha hecho en el apartado anterior. El escenario utilizado es el estático. No se han podido realizar más pruebas debido a que esta implementación aun no dispone de mecanismos de detección de rutas.

5.2.1 Retardo extremo extremo

Tal y como se explica en el apartado 5.1.1, se tiene que utilizar el escenario con los portátiles en forma de cadena. En este caso se toman dos tipos de medidas: con rutas estáticas y con rutas dinámicas. En el primer caso las rutas son introducidas anteriormente, mientras en el segundo caso el protocolo es el que descubre las rutas.

Cuando se introducen las rutas de forma estática, el protocolo DYMO no está en funcionamiento y los nodos no saben la dirección MAC a la cual van dirigidos los paquetes al principio de una comunicación, por lo que tiene que utilizar el protocolo ARP.

Tal y como se ha hecho anteriormente se han realizado 15 pruebas de 10 pings cada una. La Tabla 5.5 recoge un resumen de los valores de estás pruebas.

Tabla 5.5 Resultados obtenidos midiendo el retardo de los pings con rutas estáticas.

	1 Salto	2 Saltos	3 Saltos
RTT mínimo (ms)	0.9	2.2	2.6
RTT máximo (ms)	4.5	6.5	8.6
Media de RTT (ms)	1.5	2.9	3.7
Desviación estándar (ms)	0.5	0.7	0.6

Se puede ver que el crecimiento de estos valores es lineal a medida que se van introduciendo nodos. El valor de RTT se duplica con cada nodo introducido.

En la siguiente prueba el protocolo DYMO tiene que estar en funcionamiento, una vez este en ejecución el nodo emisor tiene que realizar pruebas con 1, 2 y 3 saltos, para ver el retardo al descubrir una ruta producido por el protocolo. en la Fig.5.9 se puede ver una gráfica resumen de los dos casos realizados, se ve la diferencia de tiempo al descubrir una ruta.

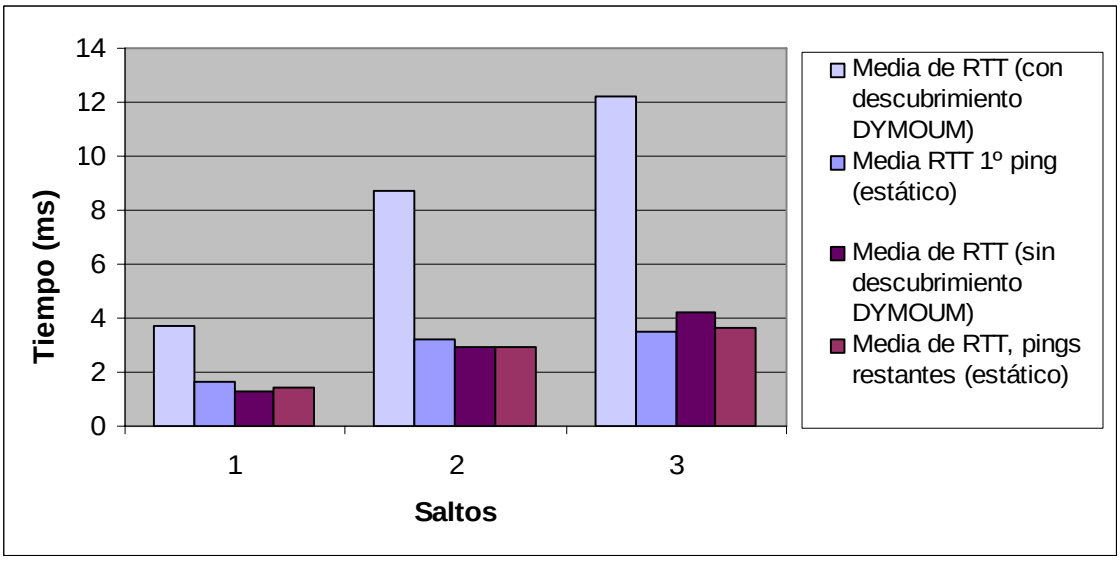


Fig.5.9 Comparativa de RTT

la diferencia de tiempo al descubrir una ruta, es debido a que el emisor tiene que crear un paquete RE para solicitar una ruta y así a su vez los nodos intermedios también tienen que realizar este paso, hasta que el receptor le llegue el mensaje y cree otro RE de respuesta.

En la Fig.5.9 se ve la diferencia que existe al utilizar rutas estáticas y rutas dinámicas. El primer paquete al utilizar rutas dinámicas es más que el doble, debido al descubrimiento de ruta. Tal y como se puede ver cuando se envían datos con rutas estáticas y con dinámicas después del descubrimiento de la ruta es similar, ya que el protocolo no introduce ningún tipo de demora en el paquete.

5.2.2 Problemas de la implementación

Debido a que el draft no especifica del todo la forma de detectar la ruptura de rutas, esta implementación no realiza ningún mecanismo de detección. Se le ha preguntado al autor de la implementación y éste ha indicado que realizará el mecanismo de hellos, cuando el draft especifique el formato de los mismos y parámetros que indican el tiempo entre ellos, reintentos después de una pérdida, etc.

5.3 Pruebas con NIST-DYMO

Por último se explican los resultados de las pruebas con el protocolo DYMO y la implementación NIST-DYMO. Para este caso únicamente existe una prueba realizada, debido a que esta implementación no acepta la herramienta iptables. Esta prueba se realiza con el escenario estático y se centra en la latencia extremo a extremo que tarda el protocolo en recibir información.

5.3.1 Retardo extremo extremo

Para realizar esta prueba se ha escogido el escenario estático, tal y como se ha hecho en los otros apartados. La configuración es la misma que en el apartado de la implementación DYMOUM. Para crear el escenario se han tenido que alejar los terminales, ya que no aceptaba la función iptables, como se puede ver en el anexo E.

En este caso también se han hecho medidas del RTT utilizando rutas estáticas, esto se realiza para tener en cuenta las pérdidas de tiempo al tener obstáculos como paredes entre los terminales.

Las pruebas se realizan enviando 10 ping y realizando la media de 15 pruebas. Los resultados de las mismas se pueden ver en la Tabla 5.3.

Tabla 5.3 Resultados obtenidos utilizando rutas estáticas

	1 Salto	2 Saltos	3 Saltos
RTT mínimo (ms)	13.9	12.1	19.2
RTT máximo (ms)	17.9	17.9	29.8
Media de RTT (ms)	15.3	14.3	24.1
Desviación estándar (ms)	0.9	1.1	2.1

Vemos que los valores son muy elevados en comparación de los del apartado anterior. Esto es debido a los obstáculos que tiene cada nodo entre sí. Se deduce que en el primer salto había más obstáculos que en el segundo, ya que el valor es prácticamente el mismo y esto no es posible en un escenario ideal. Se ve también que la desviación estándar es muy grande por lo que se puede deducir que el escenario es muy inestable, ya que los resultados son muy dispares entre sí, como se ve en el escenario de 3 saltos que hay 10 ms de diferencia entre el RTT más grande y el más pequeño.

En la Fig.5.10 se pueden ver los resultados obtenidos con rutas estáticas y con la aplicación NIST-DYMO en marcha.

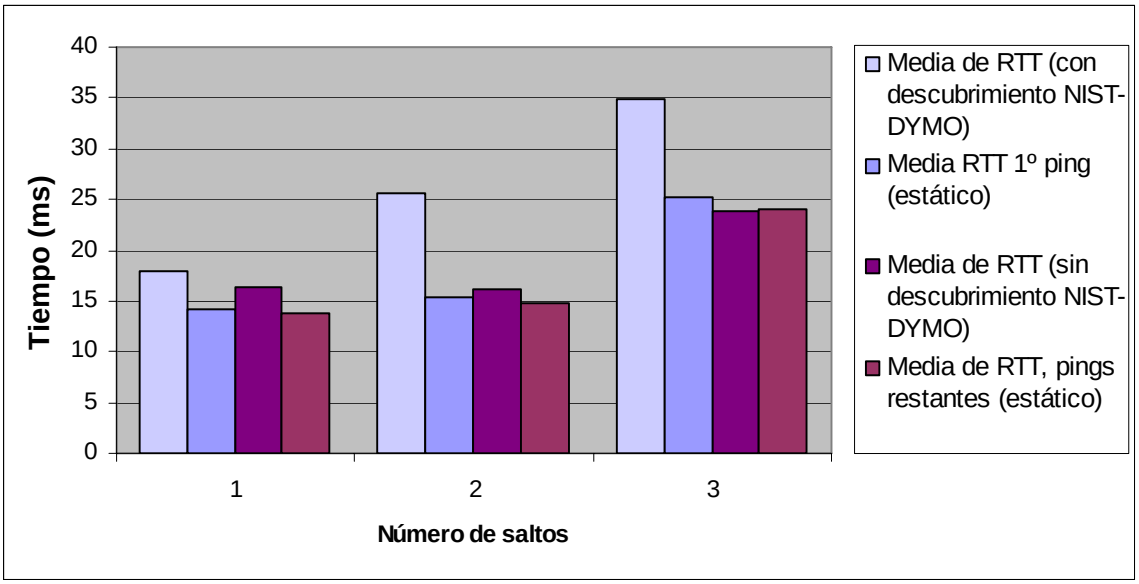


Fig.5.10 Comparativa de RTT

Se puede ver que estos valores son más elevados de lo esperado, debido a que el canal sufre cambios, al estar muy espaciados los terminales. La conclusión que se puede obtener, es que el descubrimiento tarda en este caso menos que en los otros, debido a que no todos los paquetes RE tienen que viajar por toda la red, ya que la ruta te la suministra un nodo antes de llegar al último, debido a que el anterior ya sabe la ruta de éste, esto es gracias al mecanismo de mantenimiento de rutas de hellos, que con ellos ya se puede

saber la dirección IP de los terminales adyacentes. Cuando se está enviando información no se ve ampliada la latencia de los paquetes, ya que ya tiene descubierta la ruta.

5.3.2 Problemas de la implementación

El principal problema que se ha detectado en la implementación es el inconveniente de no poder utilizar iptables. Ésta parece que coge los paquetes antes de filtrarlos por la MAC, por lo tanto los paquetes se procesan y no pasan el filtro de iptables.

No es posible coger gaps de conectividad fiables, debido a que los terminales se han puesto lejos para que no estén en el mismo rango de cobertura y poder simular el escenario con varios saltos. Debido a esta separación entre terminales es más probable que interfieran otros elementos o ellos mismos, ya que en algunos momentos tienen visibilidad directa entre sí. Por lo que no es posible disponer de un escenario controlable.

A la hora de instalar la implementación, se tiene que tener instalado el módulo NETFILTER, ya que sino, no es posible instalarla. Un problema que apareció instalando la aplicación, es que las librerías que utiliza el kernel escogido son demasiado modernas para utilizarlas con esta implementación, por lo que se han tenido que modificar los siguientes ficheros:

- Packet_in.c: Este fichero está ubicado donde se quiere instalar la implementación, se deben borrar las instrucciones que contengan la variable ethernet (ya que no perjudica a la implementación), ya que ésta no la detecta en la librería que está asignada.
- Dymo_dev.c: El cual se encuentra en la carpeta que se quiere instalar el protocolo. En este caso la variable que no detecta es la inetdevice que esta en la librería inetdevice.h. Al intentar cambiar esta librería, el kernel de Linux no deja compilar, debido a que no es una librería de éste. Por lo que se debe borrar la variable, ya que no perjudica en el desarrollo de la aplicación.
- Dymo_work_handler.c: En éste fichero tenemos el mismo problema que en el primero, por lo que se opta por borrar el mismo, antes asegurándose que no afecta a la implementación.

Por último, otro problema que se presenta en la implementación es que al intentar enviar datos con la herramienta ping, los tres primeros se pierden, esto puede ser debido a la distancia de los terminales y de los obstáculos que existen en el escenario.

CAPÍTULO 6. CONCLUSIONES Y LINEAS FUTURAS

6.1 Conclusiones

En este trabajo de final de carrera se ha pretendido evaluar los protocolos de encaminamiento reactivos de redes Ad-Hoc DSR y DYMO, en unos escenarios reales. Para ello se han escogido tres implementaciones existentes, la DSR-UU, la NIST-DYMO y la DYMOUM. En las cuales la principal ambición era descubrir la latencia en el descubrimiento de las rutas y obtener el ancho de banda, tiempos de gaps de conectividad y del consumo de batería, variando parámetros del mecanismo de mantenimiento de las rutas. Para poder alcanzar estos objetivos se ha hecho un estudio depurado de los protocolos y de las implementaciones utilizadas.

La implementación DYMOUM es una implementación inacabada, ya que no tiene ningún mecanismo para realizar la detección de rupturas de rutas. Debido a esto, únicamente se han podido realizar pruebas de latencia de descubrimiento de rutas. Estas pruebas no han resultado muy satisfactorias debido a que el descubrimiento, a medida que va creciendo el número de nodos, es muy elevado, porque necesita más tiempo para procesar los paquetes.

La implementación NIST-DYMO es la primera versión que se publica debido a que el protocolo es muy reciente. Ésta también lo es. Sólo se han podido realizar pruebas de descubrimiento de ruta, ya que esta aplicación no admite la utilización de iptables, la cual cosa no nos ha facilitado dichas pruebas. Los resultados obtenidos son muy inestables por causa de interferencias y obstáculos entre nodos.

Por último la implementación DSR-UU es la más experimentada de todas las utilizadas. En ésta, las pruebas de descubrimiento de ruta que se han obtenido son satisfactorias. Cabe remarcar que a partir de un número grande de saltos, este tiempo será muy elevado. Las pruebas de ancho de banda realizadas son satisfactorias, ya que el protocolo no utiliza mucho ancho de banda, debido al valor del tiempo entre reconocimientos del draft y el siguiente que se ha probado. Estos dos tiempos son aptos para una buena detección de ruta. El consumo del ancho de banda y batería en el caso de que no exista comunicación es nulo, ya que no hay comunicación de control entre los nodos. En el apartado de emulación de gaps de conectividad, se puede indicar, que los resultados de esta implementación no son muy esperanzadores, ya que no realiza explícitamente lo indicado en el draft, y por lo tanto estos datos son engañosos, aunque resultan unos buenos tiempos, sobre todo, el implementado por defecto.

Dado que el protocolo DYMO es joven, y sufrirá numerosas actualizaciones hasta llegar a una estabilidad, que será cuando se apruebe el estándar, las implementaciones también se irán actualizando hasta llegar éste a la

aprobación comentada. El protocolo DYMO se impondrá en las redes MANET ya que será un RFC Standards Track, mientras el protocolo DSR será un RFC experimental.

6.2 Líneas futuras

Debido a que las redes Ad-Hoc están en continua alza, los protocolos, las aplicaciones, dispositivos y otros elementos también sufren esta evolución. Por lo tanto el desarrollo de este trabajo augura un futuro muy enriquecido en este sector.

Hoy en día existen ya diferentes métodos para encaminar el tráfico en este tipo de redes. Los dos comentados en el trabajo están aun inacabados, por lo que una vez que se estandaricen será de gran utilidad abarcar estos temas:

- Introducir en la implementación NIST-DYMO un mecanismo para poder utilizar la herramienta iptables y así realizar más pruebas con esta implementación.
- Incluir mantenimiento de rutas en la implementación de la Universidad de Murcia o en su caso esperar a una versión, para experimentar las pruebas que no se han podido realizar en este documento.
- Resultaría de gran valor realizar en una implementación, pruebas con el mecanismo de detección de enlace, debido a que éste es el mejor que puede tener los protocolos estudiados.
- Debido a que estos protocolos no tienen ningún sistema fiable de seguridad, se sugiere un estudio intensivo para cubrir la necesidad de seguridad de encaminamiento en la red. Actualmente existe un draft en para el protocolo AODV.
- Es posible redefinir una implementación para que elija la ruta a un posible destino según batería existente en el nodo, número de nodos hasta llegar al nodo final, capacidad de transferencia que tiene el enlace para poder enviar datos, etc.
- Por último se podría evaluar la selección de un gateway de entre los existentes entre una red fija y una red Ad-hoc y seleccionar el mejor en cada condición.

IMPLICACIONES MEDIOAMBIENTALES

El principal impacto medioambiental que sufragan estas redes, es el medio que están hechas las redes fijas, es decir, el cable que se utilizan en ellas. Este cable se realiza de cobre, que tiene un coste grande debido a que este material empieza a ser escaso. Gracias a las redes wireless podemos prescindir de éste material.

Otro impacto que puede sufrir el medioambiente, es debido a la utilización en los terminales de pilas o baterías. Éstas ofrecen una fuente de energía cómoda y portátil, no obstante suponen una carga para el medioambiente, tanto en su fabricación como en su eliminación. Algunos de los componentes que forman estas baterías (cadmio, mercurio, zinc) pueden ser gravemente perjudiciales para el ecosistema y ocasionar enfermedades a los seres humanos sino reciben el tratamiento adecuado.

Por lo contrario, los protocolos que se estudian intentan realizar un ahorro de la red, con lo que conlleva ahorro de la batería. Únicamente envían datos de control cuando quieren enviar, recibir o encaminar datos.

Cabe destacar también que las batería que se fabrican hoy en día son mucho más duraderas y mucho menos contaminantes que antiguamente.

Por último, otra ventaja que se encuentra, es que gracias a aplicaciones de redes Ad-Hoc se pueden crear grandes redes de sensores de manera rápida y sin una infraestructura previa, para prevenir desastres naturales como huracanes, incendios, inundaciones, etc. De esta forma se pueden salvar numerosos seres vivos o incluso vidas humanas.

REFERENCIAS

[1] Draft del protocolo DSR (The Dynamic Source Routing Protocol)
http://www.ietf.org/internet-drafts/draft-ietf-manet-dsr-10.txt

[2] Draft del protocolo DYMO (Dynamic MANET On-demand Routing Protocol)
http://www.ietf.org/internet-drafts/draft-ietf-manet-dymo-03.txt

[3] RFC de New Remo, SACKs y ELN
http://rfclibrary.hosting.com/rfc/Printable/RFC3782.PDF
http://www.faqs.org/rfcs/rfc2018.html
http://www.packetizer.com/rfc/rfc.cgi?num=3135

[4] Xavi Mantecón Ferrer *Avaluació d'una xarxa ad-hoc real amb protocol d'encaminament AODV* 25 de febrero del 2005

[5] Páginas oficial de Bloutooth
http://www.bluetooth.com/bluetooth/

[6] Tomas García Sotelo *Emulación de RFID activo mediante la plataforma MICAz* 25 de febrero de 2006

[7] Definición de CSMA/CA de wikipedia
http://es.wikipedia.org/wiki/Carrier_sense_multiple_access_with_collision_avoidance

[8] Draft del protocolo SAODV (Secure Ad-Hoc On-Demand Vector)
http://tools.ietf.org/wg/manet/draft-guerrero-manet-saodv-04.txt

[9] Algoritmo de vector distancia
http://es.wikipedia.org/wiki/Vector_de_distancias

[10] Algoritmo de estado de enlace
http://bioinfo.uib.es/~joemiro/aenui/ProcWeb/actas2001/saalg223.pdf

[11] IETF MANET working group
http://www.ietf.org/html.charters/manet-charter.html

[12] RFC del protocolo OLSR (Optimized Link State Routing Protocol)
http://www.ietf.org/rfc/rfc3626.txt

[13] RFC del protocolo TBRF (Topology Dissemination Based on Reversed–Path)
http://www.ietf.org/rfc/rfc3684.txt

[14] Draft del protocolo OLSRv2 (Optimized Link State Routing Protocol version 2)

http://www3.ietf.org/proceedings/05nov/IDs/draft-ietf-manet-olsrv2-00.txt

[15] RFC del protocolo AODV (Ad-hoc On Demand Distance Vector)
http://www.ietf.org/rfc/rfc3561.txt

[16] Draft del protocolo ZRP (The Zone Routing Protocol)
http://www3.ietf.org/proceedings/02nov/I-D/draft-ietf-manet-zone-zrp-04.txt

[17] Draft del protocolo OSPF-MANET (Open Shortest Path First)
http://www.watersprings.org/pub/id/draft-chandra-ospf-manet-ext-01.txt

[18] Draft del protocolo DYMO-low
http://tools.ietf.org/wg/6lowpan/draft-montenegro-6lowpan-dymo-low-routing-00.txt

[19] Draft del protocolo LOAD (The 6LoWPAN Ad hoc Routing Protocol)
http://ietfreport.isoc.org/cgi-bin/id2pdf?f1=draft-daniel-6lowpan-load-adhoc-routing-00.txt

[20] Página oficial de Fedora Core en Castellano
http://www.fedora-es.com/

[21] Página oficial de Gentoo en Castellano
http://www.gentoo-es.org/

[22] Web oficial de las herramientas de configuración de wireless tools
http://www.hpl.hp.com/personal/Jean_Tourrilhes/Linux/Tools.html

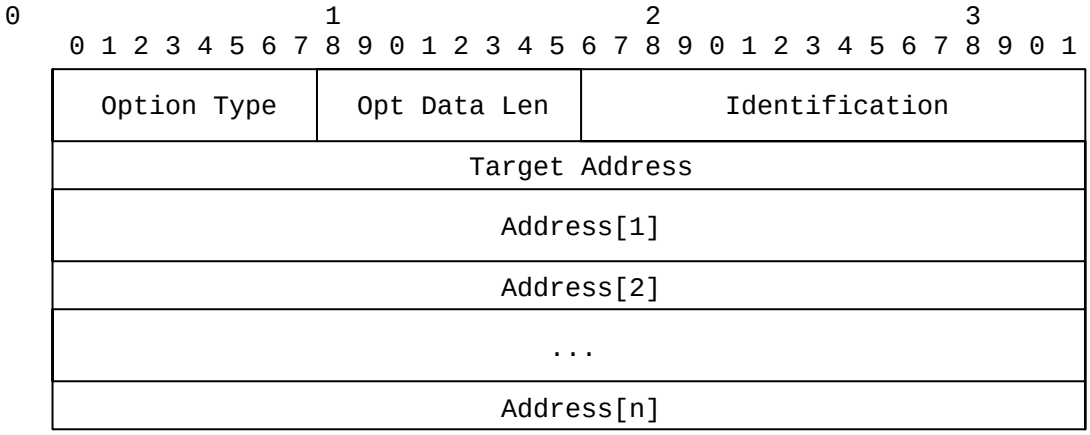
[23] página del ZD1211
http://sourceforge.net/projects/zd1211/

[24] Iperf. Herramienta para medir el rendimiento de la red. Web oficial.
http://dast.nlanr.net/Projects/Iperf/

ANEXOS

A. Formato de los paquetes del protocolo DSR

A.1 Route Request



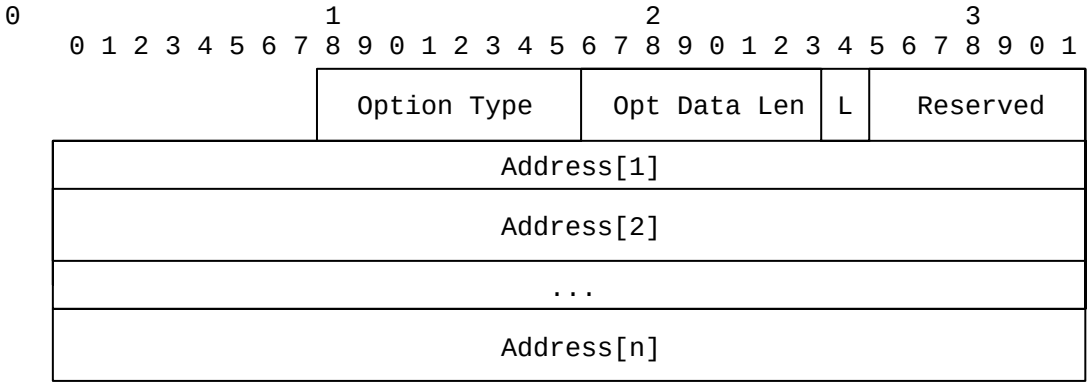
Campos IP:

- Source address: Debe introducir la dirección del nodo que origina el mensaje.
- Destination address: Debe introducir la broadcast (255.255.255.255)
- Hop Limit: Puede ponerse de 1 a 255, dependiendo si quiere propagar o no el paquete request.

Campos del Route Request:

- Option Type: Los nodos que no entienden esta opción no le hacen caso.
- Opt Data Len: Longitud de la opción excluyendo este campo y el anterior.
- Identification: Un valor único generado por el iniciador de la Petición de Ruta. Los nodos que inician una Petición de Ruta generan un nuevo valor de Identificación para cada Petición de Ruta. Si un nodo ya tiene este valor se desprecia.
- Target Address: La dirección del nodo que es el objetivo de la Petición de Ruta.
- Address [1..n]: Indica la dirección por la que pasa el nodo.

A.2 Route Reply



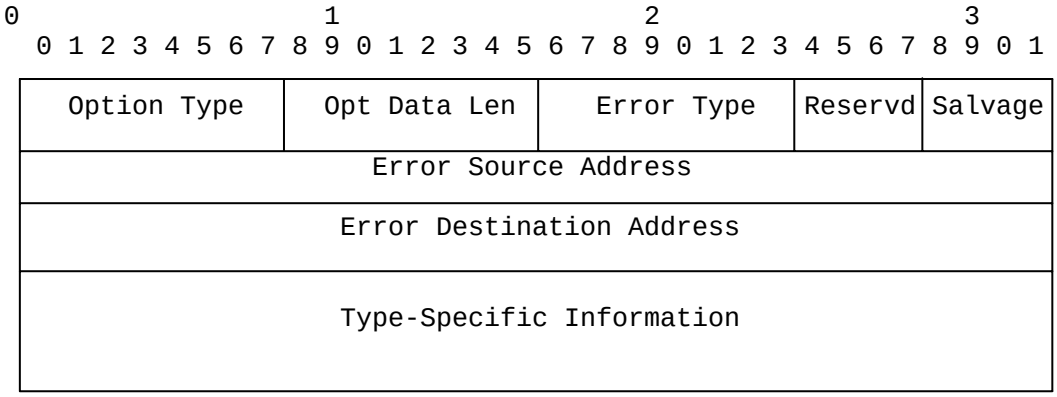
Campos IP:

- Source address: Debe introducir la dirección del nodo que origina el mensaje si es el destino, sino se introduce al que le iba el RREQ.
- Destination address: Introduce la dirección destino del emisor del RREP.

Campos del Route Reply:

- Option Type: Los nodos que no entienden esta opción no le hacen caso.
- Opt Data Len: Longitud de la opción excluyendo este campo y el anterior.
- Last Hop External (L): indicar que el último salto dado por la Respuesta de Ruta es en realidad un camino arbitrario en una red externa a la red de DSR
- Reserved: Si esta a 0 se ignora.
- Address [1..n]: Indica las direcciones por la que pasa el nodo.

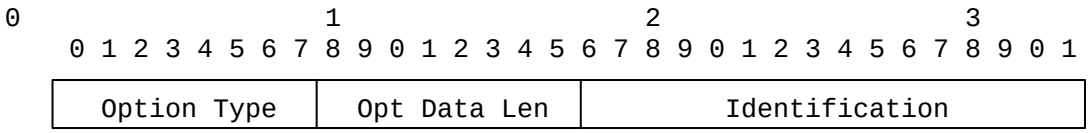
A.3 Route Error



Campos del Route Error:

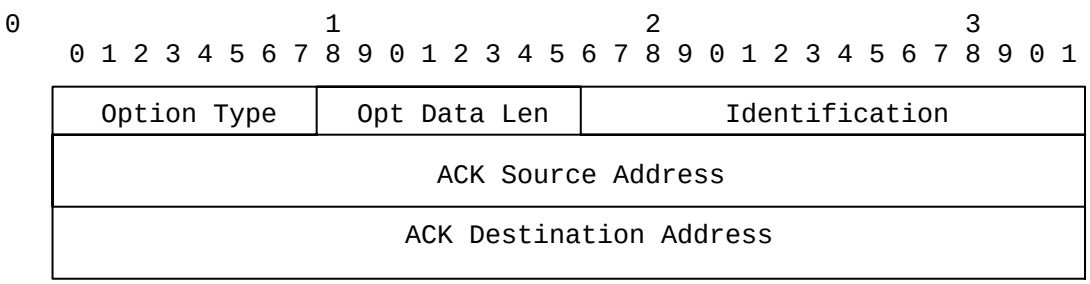
- Option Type: Los nodos que no entienden esta opción no le hacen caso.
- Opt Data Len: Longitud de la opción excluyendo este campo y el anterior.
- Error Type: El tipo de error encontrado. los valores son de tipo:
  - 1 = NODE_UNREACHABLE
  - 2 = FLOW_STATE_NOT_SUPPORTED
  - 3 = OPTION_NOT_SUPPORTED
- Reservd: Si esta a 0 se ignora.
- Salvage: Copiado del campo de salvamento
- Error Source address: La dirección del nodo que origina el Route Error
- Error Destination address: La dirección del nodo al cual el Error de Ruta debe ser entregado.
- Type-Specific Information: Información específica al Tipo de Error de este mensaje Route Error.

A.1.4 Acknowledgement Request



- Option Type: Los nodos que no entienden esta opción no le hacen caso.
- Opt Data Len: Longitud de la opción excluyendo este campo y el anterior.
- Identification: El campo de Identificación es puesto a un valor único y es copiado en el campo de identificación de la opción de Reconocimiento cuando es devuelto por el nodo que recibe el paquete sobre este salto.

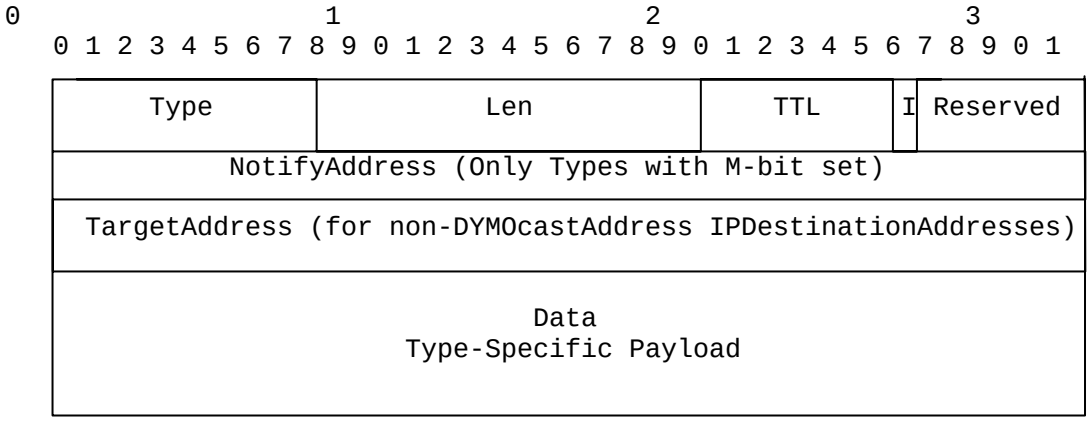
A.1.3 Acknowledgement



- Option Type: Los nodos que no entienden esta opción no le hacen caso.
- Opt Data Len: Longitud de la opción excluyendo este campo y el anterior.
- Identification: Copia la identificación, del campo de Identification de la opción de Petición de Reconocimiento del paquete.
- ACK Source Address: Dirección del reconocimiento que origina el paquete.
- ACK Destination Address: Dirección a la que va el reconocimiento.

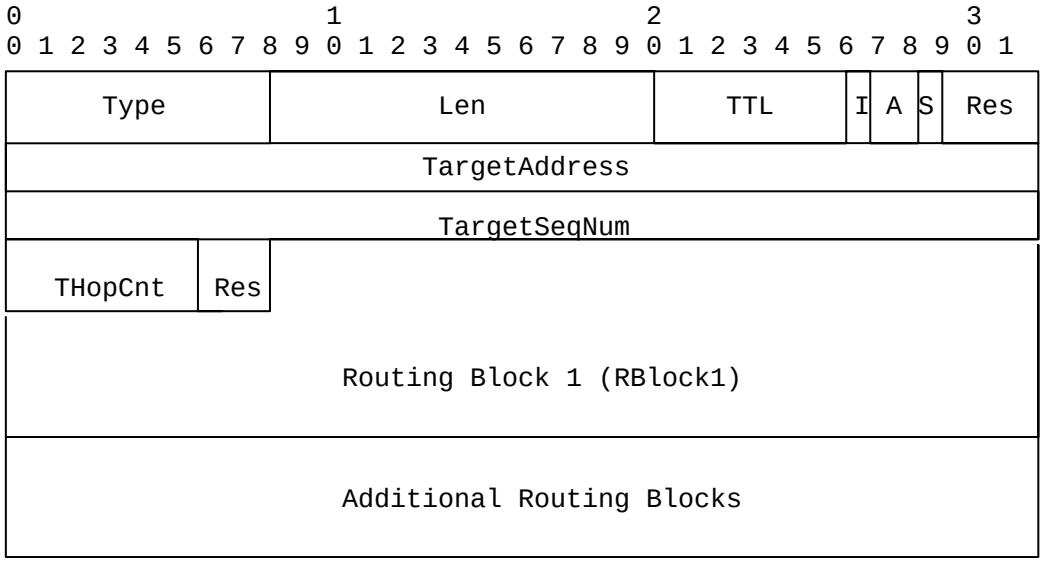
B. Formato de los paquetes del protocolo DYMO

B.1 Generic Dymo Element Structure



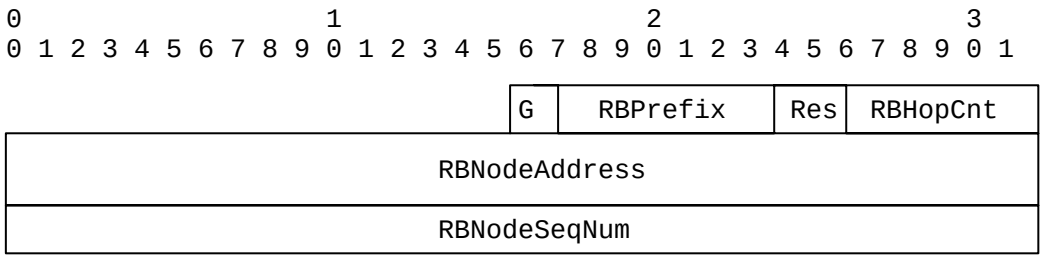
- Type: El campo de Tipo identifica el elemento.
- Len: el campo que indica el tamaño del elemento en bytes, incluyendo la parte fija.
- TTL: el campo que identifica el número máximo de veces el elemento debe ser transmitido de nuevo.
- I bit: Si esta activo algún nodo no ha hecho caso a este paquete.
- Reserver: Estos bits están reservados, en caso normal irán a 0.
- NotifyAddress: Se introduce el nodo para enviar un UERR si el tipo de elemento no es soportado.
- TargetAddres: la dirección de nodo que esta destinada el elemento.
- Data:datos.

B.2 Routing Element



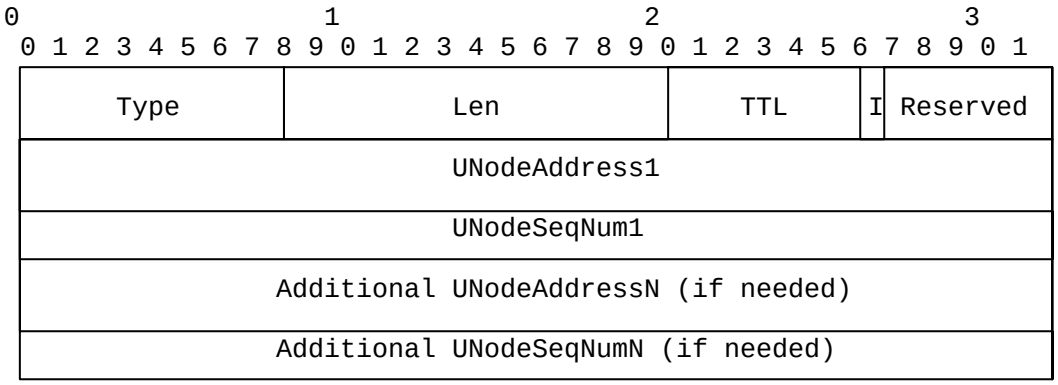
- A bit: El selector que indica si este RE requiere un RREP por el TargetAddress.
- S bit: El selector que indica si este RE requiere enviar un mensaje unicast a la dirección de salto anterior.
- Target Address: El nodo que es la última destinación del RE.
- TargetSeqNum: El número de secuencia de la última destinación de este RE. Si el Número de Secuencia es desconocido para esta Ruta entonces se pone a cero.
- THopCnt: el campo que identifica el número de nodos intermedios por el que un paquete es atravesado a lo largo de una ruta.
- RBlock: La estructura de datos que describe la información de encaminamiento relacionada con una dirección IP particular, RBNodeAddress.

B.2 Routing Block



- G bit: el selector para indicar si el RBNODEADDRESS es un gateway. Si G=1 RBNodeAddress es un gateway.
- RBPrfix: El campo que especifica el tamaño de la subred accesible por el nodo asociado.
- RBHopCnt: El campo que identifica el número de nodos intermedios por el que ha pasado el RBLOCK asociado.
- RBNodeAddress: La dirección IP asociada a este bloque.
- RBNodeSeqNum: El número de secuencia del nodo asociado a este RBLOCK.

B.3 Route Error



- UNodeAddress1: La dirección de IP del nodo inalcanzable.
- UNodeSeqNum1: El número de secuencia del nodo inalcanzable, si no se sabe se pone a cero.

C. Instalación de Gentoo

Para llevar a cabo este proyecto hemos escogido una distribución de Linux, concretamente una Gentoo. Comprobamos que dicha distribución nos permite instalar y por tanto configurar el NETFILTER, siendo este modulo indispensable para la utilización del protocolo DYMO.

Como se ha comentado anteriormente nos decidimos por la distribución de Gentoo ya que es una distribución muy flexible, moderna y rápida y el motivo principal es que nos permite crear un kernel compacto.

C.1 Configuración de Gentoo

En este anexo se muestra de manera detallada la configuración de la distribución Gentoo y su kernel para un correcto funcionamiento.

Los pasos utilizados para configurar Gentoo son los siguientes:

- Arrancar PC con ISO de Gentoo: Para iniciar una correcta configuración de la gentoo, lo primero que se debe que hacer es arrancar el PC con el CD de la imagen de la distribución Gentoo.

- Configuración de red: Una vez que se arranca el programa de instalación de gentoo, se deberá realizar la configuración de la red para poder tener acceso a nuevas actualizaciones compatibles con la arquitectura del portatil, esta se tiene que hacer con el siguiente comando:

  # net-setup ethX

  Dicho comando nos permite configurar la dirección IP, la mascara, los DNS y la puerta de enlace por la que saldremos al exterior.
  Todo esto lo comprobamos con la herramienta ping que nos indica si tenemos conectividad con el exterior.

- Administrar particiones: Todo sistema operativo requiere de varias particiones, ya sean virtuales o físicas con las que almacenar memoria temporal, arrancar el sistema o almacenar el propio sistema operativo. En la tabla C.1 se ve el esquema de nuestras particiones.

  Tabla C.1 Esquema de particiones

Partición	Sistema de ficheros	Tamaño	Descripción
/dev/hda1	ext2	32M	Partición de arranque
/dev/hda2	(swap)	512M	Partición de intercambio
/dev/hda3	Reiserfs	35G	Partición de raíz

  Para llevar a cabo estas particiones se han realizado con la herramienta fdisk, dicha herramienta sirve para partir discos duros (para una mejor implementación es aconsejable formatear los discos duros).

  En primer lugar debemos crear una pequeña partición de arranque. En primer lugar indicamos donde está ubicada la partición, después debemos crear la partición (n) y elegir el tipo de partición a utilizar, en este caso se escoge la primaria y se da un tamaño de 32M.

  #fdisk /dev/had
  Command (m for help): n
  Command action
  e extended
  p primary partition (1-4) p
  Partition number (1-4): 1
  First cylinder (1-3876, default 1): (Hit Enter)

Using default value 1
Last cylinder or +size or +sizeM or +sizeK (1-3876, default 3876): +32M

En segundo lugar se tiene que crear la partición de intercambio, de la misma que la anterior con la salvedad del tamaño (512M en este caso) e indicando con la letra t que esta partición será swap.

Por último tendremos que crear la partición de raíz del sistema operativo, esta partición se hará igual que las anteriores, con la extensión restante.

Una vez creadas las particiones, se darán formato:

- En el caso de la partición boot, este utilizará un formato reiserfs

  #mkreiserfs /dev/hda1

- En el caso de la partición raíz, este utilizará un formato ext2

  #mke2fs /dev/hda3

- En el caso de la partición swap, únicamente se deberá activar ya que anteriormente hemos definido su formato.

  #mkswap /dev/hda2
  #swapon /dev/hda2

  Ahora que las particiones están inicializadas y albergan sistemas de ficheros, es la hora de montarlas. Para ello crearemos ficheros donde montaremos dichas particiones.

  # mount /dev/hda3 /mnt/gentoo
  # mkdir /mnt/gentoo/boot
  # mount /dev/hda1 /mnt/gentoo/boot

- Utilizar un stage desde Internet: Dependiendo del medio de instalación, tendremos un par de herramientas disponibles para descargar el stage. disponemos de links2, por lo que podremos navegar por la lista de servidores de Gentoo y escoger el más cercano a nosotros.

  # links2 http://www.gentoo.org/main/en/mirrors.xml

  Una vez se ha descargado el stage correspondiente se procederá a la descompresión del mismo, utilizando el comando *tar* tal como se puede ver a continuación.

  # tar -xvjpf stage3-.tar.bz2*

- Instalando imagen de Portage: Ahora se tiene que instalar una imagen de Portage, que es un conjunto de archivos que informan a Portage sobre los programas que puede instalar en gentoo (se utilizará con el programa

emerge). Se tiene que bajar de la lista de servidores de gentoo igual que hemos hecho con el stage y después descomprimirlo, todo esto se realizará en el punto de montage donde esta situado el sistema de ficheros, tal y como podemos ver a continuación.

 # cd /mnt/gentoo
 # links2 http://www.gentoo.org/main/en/mirrors.xml
 # tar -xvjpf /mnt/gentoo/portage-<date>.tar.bz2 –C /mnt/gentoo/usr.

- Instalando el sistema base de gentoo: Se monta el sistema de ficheros /proc en /mnt/gentoo/proc para permitir a la instalación utilizar el entorno chroot para poder trabajar en el directamente, en lugar de trabajar sobre la imagen del CD. Esto se realiza con los comandos siguientes:

  # mount -t proc none /mnt/gentoo/proc
  # chroot /mnt/gentoo /bin/bash
  # env-update
  # source /etc/profile

- Configuración del kernel: El kernel es el punto alrededor del cual se construyen todas las distribuciones de Linux. Es la capa entre los programas de usuario y el hardware del sistema. En este caso se ha creído conveniente no utilizar la última versión del kernel, ya que se considera que para el proyecto se necesita una versión del kernel estable. Para instalar las fuentes de gentoo utilizamos el programa emerge, del que descargaremos la versión 2.6.9 del kernel.

  # emerge gentoo-sources

  Una vez descargado, se realiza un enlace con un puntero que apunte a las fuentes y que estará en Linux.

  # ls -l /usr/src/Linux

  Ahora se deberá compilar el kernel con las opciones que se crean necesarias, como por ejemplo: arquitectura, netfilter, tarjetas de red, etc.

  # cd /usr/src/linux
  # make menuconfig

  Cuando se acaba de configurar el kernel tal y como se puede ver en el anexo XX se tendrá que compilar e instalarlo (bzImage será el nombre que le hemos asignado a la imagen de linux), de igual forma que los módulos.

  #make bzImage
  #make modules
  #make modules_install

Cuando el kernel ha terminado de compilar, se copia la imagen a /boot. De aquí en adelante se asume que el kernel que está instalando es bzImage.

 # cp arch/i386/boot/bzImage /boot/bzImage

- Configurando el sistema: En Linux, todas las particiones usadas por el sistema deben estar reflejadas en /etc/fstab. Este fichero contiene los puntos de montaje de esas particiones, marcando su tipo, etc. Se ejecuta para asegurar todos los cambios realizados anteriormente.

  # nano –w /etc/fstab

- Instalando Grub: La grub, es un gestor de arranque que nos permite escoger el sistema operativo con el que se inicia en cada momento. Para instalar GRUB, primero lo instalamos con emerge:

  # emerge grub

  Aunque GRUB esté instalado, todavía necesitamos crear un archivo de configuración para él. Se realiza de la siguiente manera:

  #nano –W /boot/grub/grub.conf
  # cp /proc/mounts /etc/mtab
  # grub-install /dev/hda

  - Reinicio del sistema y ya esta listo para usar Gentoo.

Cabe destacar que la configuración explicada en este anexo es personalizada.

D. Scripts de configuración

D.1 Escenarios en línea

Terminal x 192.168.10.x

- Configurar red ad-hoc:

 # modprobe ZD1211
 # ifconfig eth1 up
 # ifconfig eth1 192.168.10.x netmask 255.255.255.0
 # iwconfig eth1 mode Ad-Hoc
 # iwconfig eth1 essid dsr_net
 # iwconfig eth1 channel 7

- Configurar iptables para emular áreas de cobertura deseadas:

Terminal 1 = 192.168.10.1
 # iptables -t filter -A INPUT -m mac --mac-source 00:14:7c:5b:08:07 -j DROP
 # iptables -t filter -A INPUT -m mac --mac-source 00:14:7c:5b:08:08 -j DROP
Terminal 2 = 192.168.10.2
 # iptables -t filter -A INPUT -m mac --mac-source 00:14:7c:5b:08:08 -j DROP
Terminal 3 = 192.168.10.3
 # iptables -t filter -A INPUT -m mac --mac-source 00:14:7c:5b:08:1f -j DROP
Terminal 4 = 192.168.10.4
 # iptables -t filter -A INPUT -m mac --mac-source 00:14:7c:5b:08:1f -j DROP
 # iptables -t filter -A INPUT -m mac --mac-source 00:14:7c:5b:08:13 -j DROP

D.2 Escenarios con más de un camino al destino

- Configurar red ad-hoc:

 # modprobe ZD1211
 # ifconfig eth1 up
 # ifconfig eth1 192.168.10.x netmask 255.255.255.0
 # iwconfig eth1 mode Ad-Hoc
 # iwconfig eth1 essid dsr_net
 # iwconfig eth1 channel 7

- Configurar iptables para emular áreas de cobertura deseadas sin que el terminal 3 interfiera:

Terminal 1 = 192.168.10.1
 # iptables -t filter -A INPUT -m mac --mac-source 00:14:7c:5b:08:07 -j DROP
 # iptables -t filter -A INPUT -m mac --mac-source 00:14:7c:5b:08:08 -j DROP
Terminal 2 = 192.168.10.2
 # iptables -t filter -A INPUT -m mac --mac-source 00:14:7c:5b:08:07 -j DROP
Terminal 3 = 192.168.10.3
 # iptables -F
Terminal 4 = 192.168.10.4
 # iptables -t filter -A INPUT -m mac --mac-source 00:14:7c:5b:08:1f -j DROP
 # iptables -t filter -A INPUT -m mac --mac-source 00:14:7c:5b:08:07 -j DROP

- Configurar iptables para emular áreas de cobertura deseadas para que el terminal 3 interfiera:

Terminal 1 = 192.168.10.1
 # iptables -F
 # iptables -t filter -A INPUT -m mac --mac-source 00:14:7c:5b:08:08 -j DROP
Terminal 2 = 192.168.10.2
 # iptables -F
Terminal 4 = 192.168.10.4
 # iptables -F
 # iptables -t filter -A INPUT -m mac --mac-source 00:14:7c:5b:08:1f -j DROP

D.3 Realizar rutas estáticas.

Terminal 1 = 192.168.10.1
 # route add –host 192.168.10.4 gw 192.168.10.2
Terminal 2 = 192.168.10.2
 # route add –host 192.168.10.4 gw 192.168.10.3
Terminal 3 = 192.168.10.3
 # route add –host 192.168.10.1 gw 192.168.10.2
Terminal 4 = 192.168.10.4
 # route add –host 192.168.10.1 gw 192.168.10.3

E. Escenario de la implementación NIST-DYMO

En este anexo se puede ver el escenario que se ha utilizado para realizar las pruebas de la implementación NIST-DYMO en la Fig.E.1.

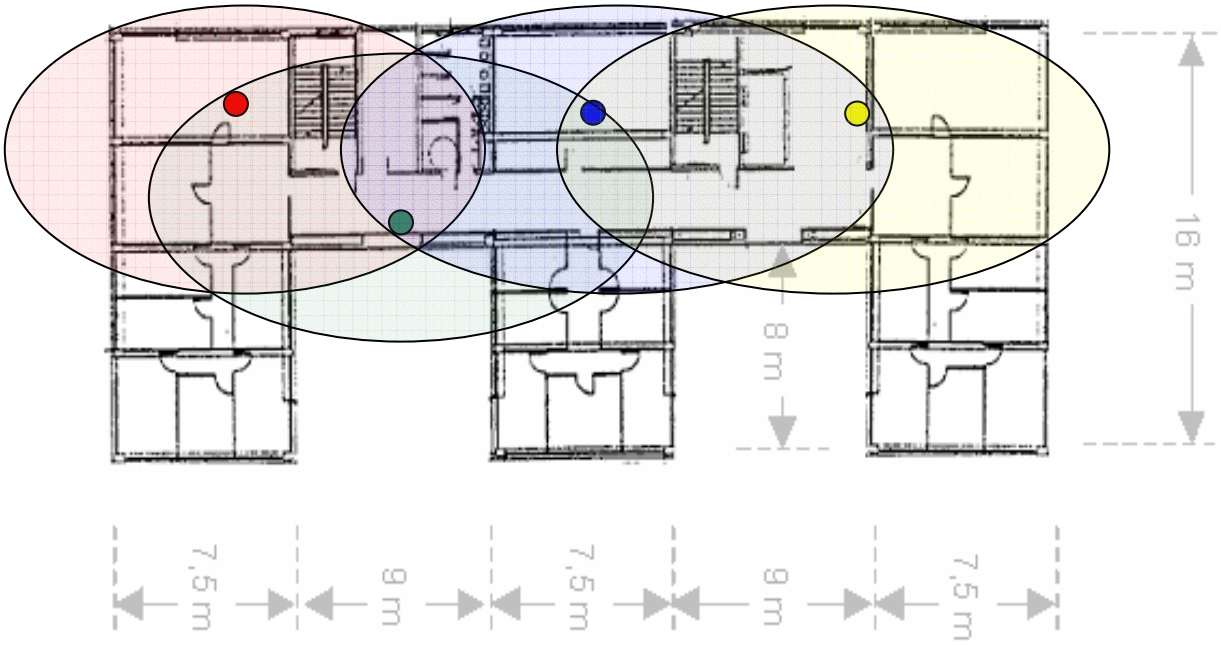


Fig E.1 Escenario de cadena utilizado en Nist-DYMO

Vemos que el nodo rojo es el terminal 1, el nodo verde es el terminal 2, el nodo azul es el terminal 3 y el nodo amarillo el terminal 4. También se puede ver sombreada el área de cobertura de cada nodo según su color.

F. Iptables

Iptables es una herramienta de configuración de reglas de filtrado de un dispositivo, creando un cortafuegos o firewall. Se puede aceptar o desechar un paquete según su dirección MAC, IP, puerto entrante, etc. Otra característica es que pueden configurar un Network Addrss Translation (*NAT*).

Se tienen que definir tres bloques:

- Tablas. Una tabla ofrece una cierta funcionalidad. Las que existen por defecto son filter, nat y mangle.
- Cadenas. Es el camino que tiene que atravesar un paquete. Tablas diferentes contienen cadenas diferentes. Si un paquete atraviesa una cadena construida por defecto, puede ser aceptado o rechazado dependiendo de la política de dicha cadena.
- Reglas. Son lo que ponemos en las cadenas para alcanzar una coincidencia deseada.

La tabla por defecto es la filter. Las cadenas de esta tabla son INPUT, FORWARD y OUTPUT. INPUT solo se aplica a datagramas destinados al host local. La cadena FORWARD solo afecta a datagramas con origen y destino diferente al host local. Por último la cadena OUTPUT aplica sólo a datagramas generados en el host local y encaminados hacia cualquier otro.

La tabla nat es consultada cuando se encuentra un datagrama que crea una nueva conexión, y por descontado cuando se quiere configurar una NAT. Las cadenas por defecto son PREROUTING, POSTROUTING y OUTPUT. La cadena PREROUTING altera la información tan buen punto entran en el host local, la cadena POSTROUTING altera los datagramas cuando estos están listos para salir del host local, y OUTPUT es la encargada de modificar los datagramas generados localmente en el host donde se aloja el cortafuegos.

Por último la tabla mangle, se encarga de la alteración especializada de los datagramas. Sus cadenas por defecto son PREROUTING y OUTPUT.

La sintaxis de la mayoría de comandos iptables es la siguiente:

```
# iptables comando especificación-regla extensiones.
```

El comando iptables afecta a diferentes tablas (-t nombre_tabla). Si se omite la opción –t, la configuración afectará por defecto a la tabla filter.

En general, un comando puede ser del tipo:

- Crear una cadena nueva: iptables –N nombre_cadena.
- Borrar una cadena vacía: iptables –X nombre_cadena (sólo para cadenas del usuario; si se omite el nombre de la cadena, se borran todas las del usuario).
- Cambiar la política por defecto de una cadena: iptables –P nombre_cadena nombre_política (que puede ser DROP o ACCEPT).
- Ver las reglas de una cadena: iptables –L nombre_cadena.
- Elimina las reglas de una cadena: iptables –F nombre_:cadena (si se omite el nombre, elimina todas las cadenas).
- Poner a cero los contenedores de paquetes y bytes en todas las reglas de una cadena: iptables –Z nombre_cadena.
- Añadir (-A), borrar (-D) o sustituir (-R) o insertar (-I) seguido del nombre de una cadena en esa tabla.

Las especificaciones de las reglas pueden ser:

- -p (protocolo: TCP, UDP, ICMP, etc).
- -s (dirección fuente).
- -d (dirección fuente).
- -i (interfaz de entrada).
- -o (interfaz de salida).
- --dport (puerto destino).
- --sport (puerto fuente).
- --port (los puertos fuente y destino son iguales).
- -j (especifica la acción a seguir).
- --mac-source (dirección MAC origen).

Dónde mandar el paquete si existe coincidencia. Objetivos disponibles:

- ACCEPT (aceptar paquete).
- DROP (eliminar paquete sin decir nada).
- REJECT (eliminar el paquete y avisar al emisor).
- LOG (hacer un log del paquete vía syslog y continuar su travesía).

Opciones disponibles:

- -v, iptables enseñará sus salidas en versión texto.
- -n, iptables mostrará direcciones IP y puertos sin intentar resolver sus correspondientes nombres.
- -x, iptables no redondeará las salidas , mostrando los valores numéricos exactos.
- -line-number, iptables muestra números de líneas cuando lista reglas que corresponderán a la posición de la regla en la cadena correspondiente.