PROYECTO FINAL DE CARRERA

TÍTULO DEL PFC: Redes mesh basadas en puntos de acceso inteligentes 802.11 open source (III)

TITULACIÓN: Ingeniería de Telecomunicaciones (segundo ciclo)

AUTORES: Luis Daniel Ruiz López
Marc Oliveras Pla

DIRECTOR: Eduard García Villegas

CODIRECTOR: Rafael Vidal Ferré

FECHA: 22 de Febrero de 2006

Título del PFC: Redes mesh basadas en puntos de acceso inteligentes 802.11 open source (III)

Autores: Daniel Ruiz López y Marc Oliveras Pla

Director: Eduard García Villegas

Codirector: Rafael Vidal Ferré

Fecha: 22 de Febrero de 2006

Resumen

El número de redes inalámbricas continúa creciendo sin control debido a que éstas funcionan sobre una banda de frecuencias libre, son de fácil instalación, atractivas al eliminar el cableado además de ser asequibles económicamente. En entornos muy poblados se puede observar la coexistencia de redes inalámbricas de empresas, redes domésticas, *hot spots* públicos, etc. que comparten la misma banda de frecuencias, que es de hecho un recurso limitado. En muchos casos el número de canales libres para no producir solapamiento con otras redes no es suficiente, por lo que la coexistencia resulta cada vez más problemática debido a la aparición de interferencias entre redes vecinas. La presencia de interferencias afecta al funcionamiento de las redes produciendo errores y colisiones, y como consecuencia se reduce el caudal además de aumentar el retardo de los paquetes.

La estructura de este proyecto se basa principalmente en dos grandes bloques que se detallan a continuación:

El primero pretende estudiar el impacto que conlleva la coexistencia de diferentes redes WLAN IEEE 802.11 sin una gestión inteligente que permita una óptima distribución de los recursos. Para ello se ha analizado el comportamiento de puntos de acceso meshcubes mediante pruebas en entornos reales. Posteriormente, se han corroborado los resultados obtenidos con dos simuladores de redes: NS-2 y OPNET Modeler. De esta manera se ha realizado una comparación exhaustiva de los resultados obtenidos.

A partir de los resultados obtenidos se planteó la necesidad de introducir en las redes inalámbricas mecanismos distribuidos que optimizaran los recursos disponibles con el fin de reducir los efectos negativos de las interferencias.

En el segundo bloque se presentan dos alternativas para solucionar la problemática mediante el diseño de protocolos de señalización entre nodos de una red mesh que permiten llevar a cabo diferentes estrategias para le gestión de recursos radio en una red WLAN IEEE 802.11. Uno está elaborado mediante Raw Ethernet Sockets y el otro se ha desarrollado para funcionar sobre el protocolo de encaminamiento OLSR.

Title of PFC: Mesh Networks based on open source intelligent access points 802.11 (III)

Authors: Daniel Ruiz López y Marc Oliveras Pla

Director: Eduard García Villegas

Codirector: Rafael Vidal Ferré

Date: Febraury, 22th 2006

Overview

The number of wireless networks keeps growing without control due to the fact that they use unlicensed frequency bands, are easily configured, and reduce to the minimum the wired distribution, apart from being not very expensive. In densely populated areas, we can observe the coexistence of enterprise WLANs, public hot spots, wireless domestic networks, etc. sharing the same frequency spectrum which is in fact a scarce resource. In many cases, the number of non overlapping frequency channels available is not enough to ensure an innocuous coexistence, so the coexistence is becoming more problematic due to the appearance of interferences between neighbouring cells. The presence of interference affects the performance of the network by adding errors and collisions, and hence, reducing the effective throughput and increasing packet delays.

This project is structured in two main building blocks that are detailed next:

The first one is aimed to study the impact that entails the coexistence of different WLAN IEEE 802.11 networks without an intelligent management that allows an optimal distribution of the resources. In this way, the behaviour of meshcube access points has been analyzed by means of tests in real environments. Later, the obtained results have been used to evaluate two network simulators: NS-2 and OPNET Modeler.

From this comparison, it was considered to introduce distributed mechanisms in wireless networks in order to optimize the resources and reduce the negative effects of interferences.

In the second block two alternatives are introduced to solve the mentioned problem by means of the design of signalling protocols between nodes of a mesh network that allow to carry out different resource management solutions in a WLAN IEEE 802.11 network. One is developed with Raw Ethernet Sockets and the other one works over the OLSR routing protocol.

ÍNDICE

GLOSARIO

AP	Access Point
AODV	Ad-Hoc On-Demand Distance Vector
BER	Bit Error Rate
BLER	Block Error Rate
CSMA/CA	Carrier Sense Multiple Access with Collision Avoidance
CTS	Clear To Send
DCM	Default Channel Match
DLC	Default Link Closure
DLL	Dynamically Loadable Library
DSR	Dynamic Source Routing Protocol for Ad-Hoc Networks
ESSID	Extended Service Set Identifier
ETSI	European Telecommunications Standards Institute
FSM	Finite State Machine
FTP	File Transfer Protocol
HTTP	Hypertext Transfer Protocol
IANA	Internet Assigned Numbers Authority
IP	Internet Protocol
IPC	Interprocess Communication
MANET	Mobile Ad-hoc Network
MPR	Multipoint Relay
NAM	Network Animator
NS	Network Simulator
OLSR	Optimized Link State Routing Protocol
OPNET	Optimized Network Engineering Tools
PER	Packet Error Rate
RNRT	Réseau Nacional de Reserche en Télécommunications
RTS	Ready To Send
SNR	Signal Noise Ratio
TCP	Transmission Control Protocol
TeNS	The Enhanced Network Simulator
TMM	Terrain Modeling Module
UDP	User Datagram Protocol
VoIP	Voice over Internet Protocol
WAN	Wide Area Network
WLAN	Wireless Local Area Network
WNRA	Wireless Network Roadmap Algorithm

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

INTRODUCCIÓN

El número de redes inalámbricas continúa creciendo sin control debido a que éstas funcionan sobre una banda de frecuencias libre, son de fácil instalación, atractivas al eliminar el cableado además de ser asequibles económicamente. En entornos muy poblados se puede observar la coexistencia de redes inalámbricas de empresas, redes domésticas, *hot spots* públicos, etc. que comparten la misma banda de frecuencias. En muchos casos el número de canales libres para no producir solapamiento con otras redes no es suficiente, por lo que la coexistencia resulta cada vez más problemática debido a la aparición de interferencias entre redes vecinas. La presencia de interferencias afecta al funcionamiento de las redes produciendo errores y colisiones, y como consecuencia se reduce el caudal además de aumentar el retardo de los paquetes. El estándar IEEE 802.11 (b, g) usa la banda de 2.4GHz. En el caso de Europa, la ETSI (European Telecommunication Standards Institute) define 13 canales en esta banda, de los cuales, debido al ancho de banda de la señal 802.11, sólo pueden utilizarse 3 simultáneamente si se quieren evitar situaciones donde las interferencias entre dispositivos produzcan una pérdida de la calidad del canal. Por lo tanto, se puede observar que la banda de frecuencias del estándar 802.11 es un recurso limitado.

Por un lado, en este proyecto se estudia el impacto que conlleva la coexistencia de diferentes redes WLAN IEEE 802.11 sin una gestión inteligente que permita una óptima distribución de los recursos. Además se corroboran los efectos negativos que supone el utilizar los recursos limitados bajo diferentes situaciones. Para ello, se han extraído resultados a partir de la realización de pruebas en entornos reales que posteriormente se han contrastado con diferentes simuladores de redes. Una comparación exhaustiva de los resultados prácticos con los obtenidos mediante simulación nos permiten evaluar el grado de exactitud que éstos proporcionan. El uso de estas herramientas es fundamental para medir a gran escala el rendimiento de las redes, permitiendo detectar qué configuraciones proporcionan mejores prestaciones antes de su instalación y despliegue.

A partir de los resultados obtenidos se planteó la necesidad de introducir en las redes inalámbricas mecanismos que optimizaran los recursos disponibles con el fin de reducir los efectos negativos de las interferencias. Estos efectos negativos se suelen minimizar con un buen diseño previo de la red: una buena colocación de los APs aparte de fijar de manera óptima las potencias transmitidas y los canales asociados para asegurar la cobertura de la zona y minimizar las interferencias con otras redes. La planificación es una buena técnica para paliar este problema en una red multicelular, pero en la realidad, los administradores de red no saben cuándo va a aparecer un nuevo nodo interferente; además, no pueden administrar todos los componentes involucrados ya que pueden pertenecer a diferentes redes y de hecho, a diferentes propietarios.

Es por eso que un mecanismo centralizado de control no resulta apropiado, por lo que la solución más óptima seria un mecanismo de control distribuido. Pero para que un mecanismo de control sea distribuido es necesaria la comunicación entre todos los componentes de la red, aún siendo de diferentes dominios y propietarios. Para introducir señalización en una red inalámbrica se han utilizado APs con dos interfaces radio: una realizando las funciones normales de un AP mientras que la otra funciona en modo Ad-hoc para permitir la comunicación entre APs dentro de una red mesh. El intercambio de información entre APs no debe ser solamente entre nodos cercanos, se debe hacer llegar tal información al nodo más lejano si fuera necesario. En consecuencia, los APs deben ser capaces de retransmitir o encaminar la información para así poder llegar a cualquier nodo de la red.

Para ello, la segunda parte de este proyecto ha estado dedicada a la implementación de dos mecanismos para solucionar la problemática comentada anteriormente. El primero de ellos se ha diseñado para trabajar en la capa 2 de la pila OSI mientras que el segundo trabaja en la capa 3.

El objetivo del mecanismo de capa 2 se basa en la creación de un sistema eficiente capaz de analizar la red en cada momento, reuniendo la información que cada nodo tiene de su entorno para dibujar un mapa completo de las interferencias entre los diferentes APs sobre el cual poder llevar a cabo estrategias de planificación frecuencial.

El segundo mecanismo implementa un sistema de reparto de carga entre celdas vecinas conocido como *Cell Breathing*. Dicha técnica consiste en disminuir o aumentar el radio de una celda, dependiendo de su utilización. En redes WLAN 802.11 son los dispositivos cliente los que deciden la asociación y el traspaso basándose únicamente en nivel de señal. Si una celda sobre-utilizada reduce su cobertura (*breath out*), parte de sus clientes recibirán mejor señal de otras celdas vecinas, liberando al AP congestionado de parte de su carga. A su vez, una celda con poca utilización, puede aumentar su cobertura (*breath in*), atrayendo así a clientes lejanos, de manera que se puede ofrecer servicio a los clientes excluidos de una celda sobre-cargada. Al reducir una celda se consigue disminuir su utilización además de reducir la probabilidad de que nuevos clientes demanden servicio, y así vayan a celdas menos ocupadas.

La estructura de este proyecto se basa principalmente en dos grandes bloques que se detallan a continuación:

El primero pretende analizar el comportamiento de puntos de acceso meshcubes en situaciones reales y comparar los resultados con dos simuladores de redes: NS-2 y OPNET Modeler.

En el segundo bloque se presentan dos alternativas para solucionar la problemática mediante el diseño de protocolos de señalización entre nodos de una red mesh que permiten llevar a cabo diferentes estrategias para le gestión de recursos radio en una red WLAN IEEE 802.11. Uno está elaborado mediante Raw Ethernet Sockets y el otro se ha desarrollado para funcionar sobre el protocolo de encaminamiento OLSR.

En el primer capítulo se muestran las pruebas realizadas en entornos reales: en la primera se analiza qué modelo de propagación se sigue en nuestro escenario de pruebas (SNR vs Distancia), en la segunda se mide el caudal máximo que se alcanza dado un nivel de SNR y según la modulación utilizada, y en la tercera se comprueba el efecto que suponen las interferencias entre canales adyacentes. Los resultados obtenidos se comparan con modelos teóricos para verificar el correcto funcionamiento.

En el capítulo 2 se reproducen las mismas pruebas del primer capítulo pero en un entorno de simulación de redes como es NS-2 con el fin de comparar los resultados con los de las pruebas reales. En el tercer capítulo se realizan las mismas pruebas pero esta vez en el simulador de redes OPNET Modeler para conocer el funcionamiento de este simulador y así también tener otros resultados para poder realizar una comparación más exhaustiva.

Tal comparación entre resultados de pruebas reales y resultados extraídos de simuladores se realiza en el cuarto capítulo, donde se ven reflejadas las características de los simuladores utilizados con sus ventajas y desventajas.

En el capítulo 5 se expone la problemática existente y se presentan dos soluciones a nivel teórico. En la primera se realiza un estudio de las diferentes alternativas que existen para el envío de paquetes de información a nivel 2 entre diferentes APs que participan de la red mesh gracias a una segunda interfaz radio y se escoge una como la más apropiada para este proyecto. En la segunda se presenta OLSR como protocolo de encaminamiento entre redes MANETs y la posibilidad que ofrece para añadir nuevas funcionalidades mediante plugins.

En el capítulo 6 se presentan las implementaciones que se han desarrollado para este proyecto. Se explica su funcionamiento y las pruebas realizadas para verificar las mejoras aportadas a la gestión de recursos en redes inalámbricas con APs dotados de cierta inteligencia.

Finalmente, en el capítulo 7, se pueden observar las conclusiones extraídas además de las consideraciones medioambientales que se han tenido en cuenta en la realización de este PFC. También este capítulo va acompañado de una propuesta sobre futuras líneas de investigación que se pueden seguir a partir de este proyecto.

Como anexos se incluyen los manuales elaborados en este proyecto para los dos simuladores evaluados con el fin de facilitar su aprendizaje en futuros proyectos. En el anexo I se puede encontrar el manual para NS-2 donde se explica detalladamente la creación de una simulación y el funcionamiento del simulador. En el anexo II se puede encontrar el segundo manual elaborado, pero esta vez para el simulador OPNET Modeler. Este último también incluye una guía para la elaboración de una simulación en redes inalámbricas.

CAPÍTULO 1. PRUEBAS REALES

En este capítulo se analizará el comportamiento de equipos IEEE 802.11 ante variaciones en la calidad de la señal, debidas a pérdidas de propagación e interferencias en entornos reales, con el fin de validar los resultados obtenidos mediante simulación. El hardware utilizado se compone de APs meshcube [1], utilizados como plataforma de desarrollo en el resto de la memoria y que podemos observar en **Fig. 1.1**.

Fig. 1.1 Punto de acceso Meshcube

Se han realizado tres pruebas que representan tres situaciones diferentes y que se explicarán detalladamente a continuación.

Se han extraído resultados de cada prueba que posteriormente se han contrastado con modelos teóricos para poder verificar el comportamiento de estos equipos.

1.1. Prueba 1. Potencia recibida vs Distancia

Con esta prueba se ha conseguido analizar el alcance del radio de cobertura en un entorno abierto como es la playa de Castelldefels. El objetivo era obtener una gráfica que representara la evolución de la SNR en el punto de medida respecto la distancia.

Esta prueba se ha realizado en dos entornos diferentes. En el primero *(Escenario 1a)* se ubicaba el meshcube en la arena de la playa con ciertas precauciones. Éste se configuró correctamente como punto de acceso y un ordenador portátil se configuró como cliente inalámbrico. Una vez montado el escenario, con el portátil se obtenía la señal recibida cada diez metros durante una distancia de 200 metros. La obtención de dicha potencia recibida se realizó mediante el comando *iwconfig* [2].

En el segundo entorno *(Escenario 1b)* se realizó el mismo procedimiento en el mismo entorno abierto, pero esta vez se ubicó el punto de acceso en el paseo marítimo en vez de en la arena de la playa. De esta forma hemos podido contrastar qué efectos causa ésta en la propagación de la señal.

A continuación se pasan a describir los resultados obtenidos en cada escenario:

1.1.1. Escenario 1a

En este primer escenario (**Fig. 1.2**) podemos observar el meshcube en modo de AP (Access Point) al cual se conecta el cliente (el portátil).

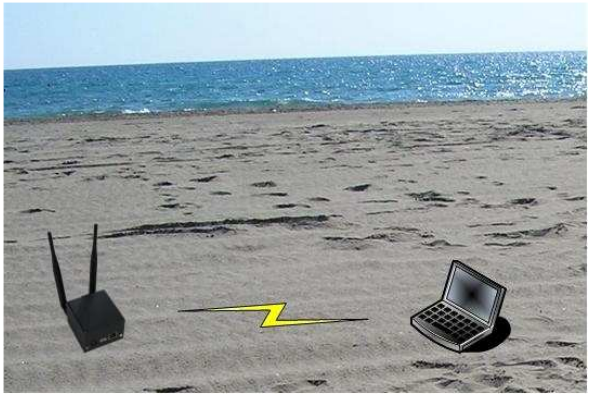

Fig. 1.2 Escenario 1a

En la **Fig. 1.3** podemos ver la gráfica de la evolución de la SNR respecto a la distancia. Su trayectoria tiene muchas variaciones sobretodo a distancias cercanas. Conforme el cliente se aleja del punto de acceso, la trayectoria resultante es más equilibrada y los cambios no son tan notables.

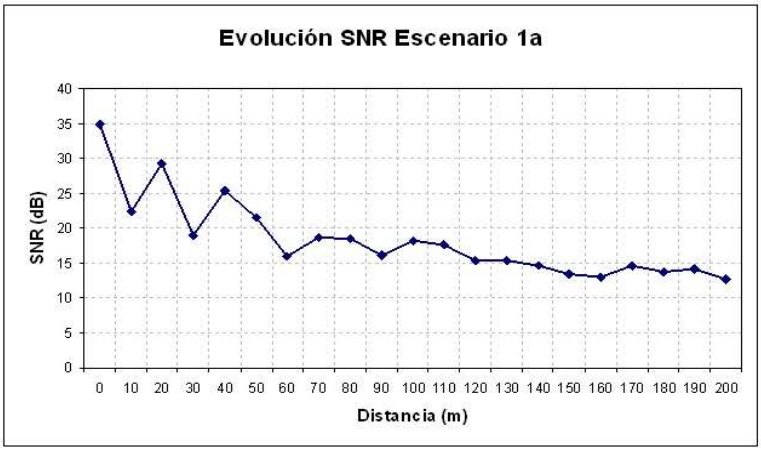

Fig. 1.3 Gráfica resultante escenario 1a

Teniendo en cuenta que la potencia de ruido es siempre constante, unos -95 dBm, podemos ver que a distancia 0 obtenemos una potencia recibida de unos -60 dBm. Es un valor bajo teniendo en cuenta la proximidad de los elementos. Después de realizar el escenario 1b y seguidamente compararlo con modelos teóricos se ha llegado a la conclusión que el terreno (arena de playa) no es

adecuado para la propagación de señal, debido a su alto nivel de atenuación por su alta humedad.

1.1.2. Escenario 1b

A partir de los resultados inesperados del *escenario 1a*, se decidió realizar la misma prueba en el mismo entorno abierto pero en un terreno diferente (**Fig. 1.4**) para contrastar resultados y averiguar la causa de las variaciones de SNR a distancias cercanas.

Fig. 1.4 Escenario 1b

La **Fig. 1.5** muestra los resultados obtenidos en este escenario. Podemos ver que el gráfico resultante de este escenario no presenta tantas variaciones inesperadas debido a la menor atenuación del terreno. Cabe decir que en los dos escenarios se han utilizado los mismos componentes y configuraciones. Además, en los dos escenarios se ha verificado previamente que no existiera ningún tipo de interferencia de otras redes inalámbricas. De hecho los dos escenarios están realizados en la misma zona del paseo de Castelldefels.

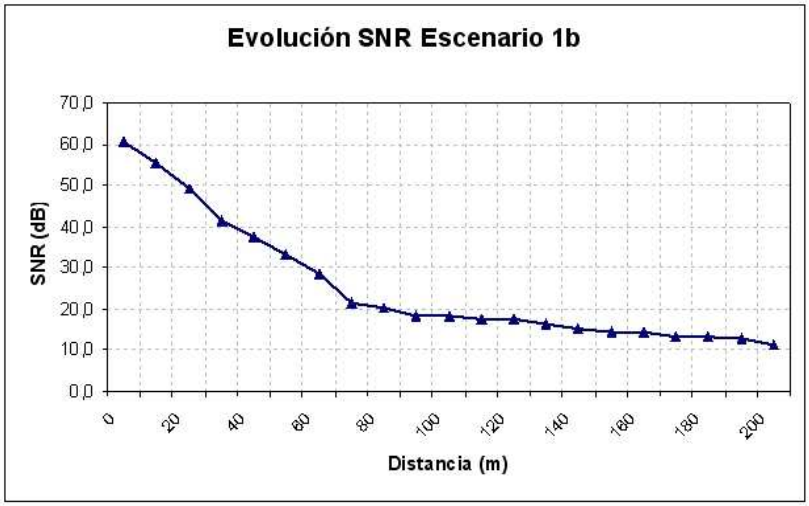

Fig. 1.5 Gráfica resultante escenario 1b

En este caso, obtenemos una potencia recibida a distancia 0 de -35 dBm aproximadamente, valor más cercano a los modelos teóricos.

1.1.3. Comparación con modelos teóricos

Para validar los resultados de los escenarios probados compararemos la evolución de la potencia recibida respecto la distancia de los dos escenarios con unos modelos teóricos de propagación de la señal como por ejemplo propagación en espacio libre, modelo de Lee y los modelos de Rappaport en entornos abiertos, cerrados y semi-cerrados [3].

En la **Fig. 1.6** se muestra la comparativa. Podemos observar que a distancias lejanas los dos escenarios tienen un comportamiento aproximado al modelo de Lee. A corta distancia, el *escenario 1a* se comporta como modelos de entornos cerrados, como son los modelos de Rappaport en entornos semi-cerrados y cerrados. Esto es debido a la gran atenuación que supone la arena de la playa. A distancia cero podemos observar que el *escenario 1a* sufre mucha más atenuación que ningún otro modelo.

El *escenario 1b* en cortas distancias se puede aproximar como una media de los modelos en entornos abiertos como es el caso del modelo de propagación en espacio libre, el modelo de Lee y el modelo de Rappaport en entornos abiertos.

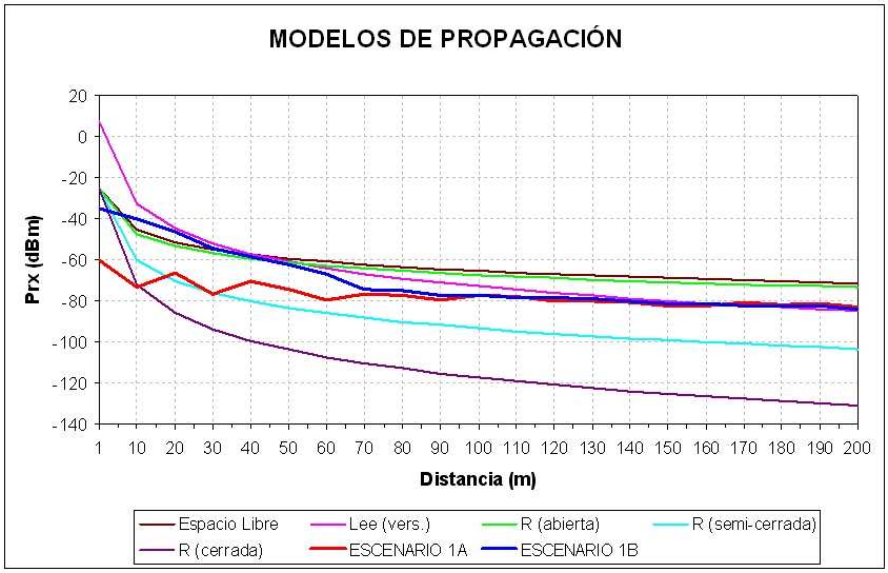

Fig. 1.6 Comparativa de modelos de propagación

Dado el entorno en el que estamos trabajando podemos asegurar que el escenario 1b se corresponde más al comportamiento esperado, debido a que estamos trabajando en un espacio abierto. Así pues, tomaremos este escenario como referencia para otras pruebas.

1.2. Prueba 2. Throughput vs SNR

Con esta prueba se pretende observar cómo afecta el nivel de SNR en la tasa de transferencia entre los nodos móviles. Se realizará la misma prueba con las diferentes tasas de transmisión estandarizadas en el 802.11b (Auto, 11Mbps, 5,5 Mbps, 2 Mbps y 1 Mbps). De esta forma se quiere ver hasta qué valor de SNR son capaces de seguir transmitiendo a la tasa definida.

El entorno en el que se desarrolla esta prueba es el mismo que el presentado como *Escenario 1b* (entorno abierto) (**Fig. 1.4**). En un lado el *meshcube* se configura como punto de acceso transmisor y en el otro lado el terminal móvil (portátil) como receptor.

A continuación se limitará la tasa de transferencia en el punto de acceso con el siguiente comando: *iwconfig wlanX rate N fixed*, donde "N" representa el valor de la tasa a fijar.

Para cada tasa fijada se comprobará, para cada nivel de SNR, qué throughput se puede alcanzar con la ayuda de la herramienta *iperf* [4].

Para saber qué SNR recibimos en cada punto, en todo momento de la prueba se obtendrá la potencia recibida (dBm) en el terminal móvil y la potencia del ruido calculada con la Formula del ruido (N = KTB en dBm). La diferencia de las dos nos permitirá saber el nivel de SNR.

Para el cálculo del Ruido se han utilizado la **Fórmula 1.1** y la **Fórmula 1.2**.

Fórmula 1: N = KTB, K=1,38*10^{-23}, T=298ºK, B=22 MHz
N= 9,05*10^{-11} mW → **N = -100,43 dBm** **(1.1)**

Fórmula 2: N ' = N + Factor de ruido = -100,43 dBm + 5 dB
N ' = -95,43 dBm **(1.2)**

El nivel de ruido en nuestro caso, lo consideraremos igual para todas las pruebas y con un nivel de -95,43 dBm.

Como referencia de esta prueba tenemos una gráfica ideal (**Fig. 1.7**) que nos muestra la tasa máxima que se puede obtener según el nivel de SNR para cada una las tasas estandarizadas. Cada tasa estandarizada utiliza una modulación diferente para poder adaptarse mejor a las condiciones del canal.

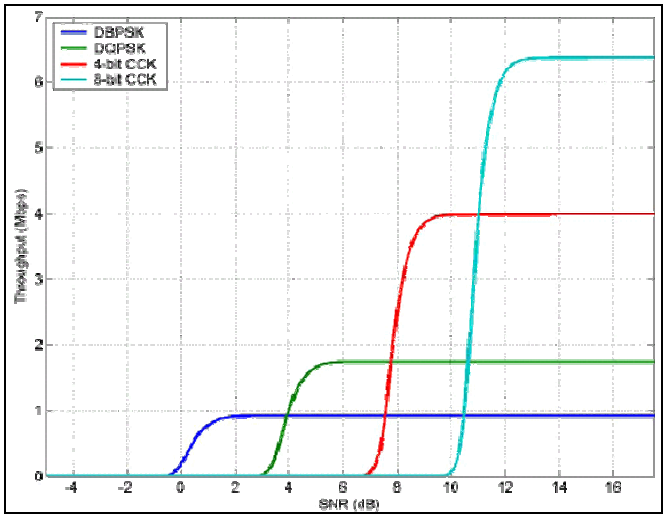

Fig. 1.7 Gráfica ideal de la tasa respecto SNR según las codificaciones

En la **Fig. 1.7** se observa que, dependiendo de la modulación utilizada en la transmisión, tendremos tasas máximas diferentes. La codificación que nos permite obtener la tasa de transferencia más elevada (6 Mbps máx. teórico) es más sensible a las interferencias ya que a SNR menor que 10, la tasa de error en el bit (BER) es demasiado elevada como para decodificar correctamente las tramas. En cambio, la modulación más lenta (1 Mbps máx.) es más tolerante a interferencias y soporta niveles de SNR mucho más bajos.

Cabe decir que la elección de poner el punto de acceso como transmisor y el cliente como receptor no se ha debido a una elección arbitraria. En un principio se probó de realizar una comunicación full-duplex, es decir, los dos dispositivos haciendo de transmisor y receptor en el mismo tiempo. Se comprobó que los resultados obtenidos no eran del todo lógicos. A partir de aquí pudimos saber que el hecho de fijar una tasa mediante el comando *iwconfig* únicamente fijaba la tasa máxima de transmisión, es decir, únicamente fijaba la tasa en el sentido *downstream* (Punto de acceso → Cliente). Por ello se obtenían resultados inesperados y por ello se eligió de esta manera la distribución del escenario.

Una vez visto el objetivo de la prueba pasaremos a ver los resultados obtenidos en cada una de las tasas estandarizadas.

Cabe comentar que debido a la baja sensibilidad que los equipos utilizados llevan por defecto, no se podían alcanzar niveles de señal inferiores a -85 dBm (lo que equivale a una SNR de 10dB). La **Fig. 1.8** representa la evolución de la potencia recibida en el receptor a medida que nos íbamos alejando del emisor. Se puede observar que a partir de una potencia recibida de -85 dBm, la señal se pierde y no se pueden realizar medidas a menores potencias. La siguiente aparición de señal es debido a que nos volvimos a acercar al emisor. La representación de la potencia recibida se ha realizado con el software Network Stumbler [5] para entornos Windows.

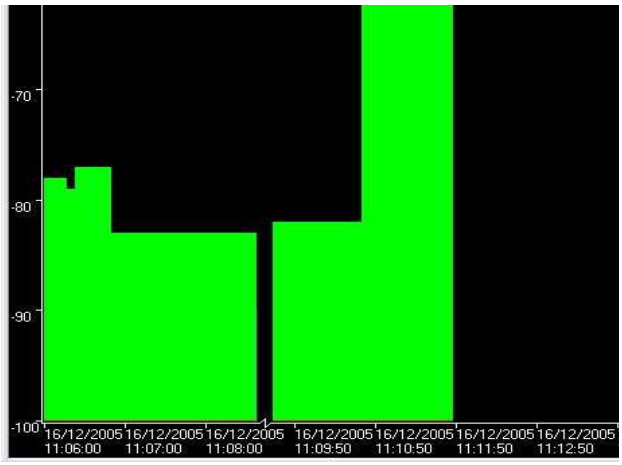

Fig. 1.8 Representación de la potencia recibida en el receptor

Para mejorar esta sensibilidad, se tuvo que trabajar en Linux y con tarjetas que incorporaban el chipset Prism 2, que estaban controladas con el driver Hostap [6]. De esta manera se pudo configurar tanto la sensibilidad del punto de acceso como la del ordenador portátil mediante el comando *iwconfig wlanX sens 3/3* para que así pudiera trabajar con la máxima sensibilidad posible.

Para cada tasa de transmisión se extrajeron las respectivas gráficas, las cuáles se han resumido en una sola gráfica (**Fig. 1.9**) para poder analizar mejor los resultados.

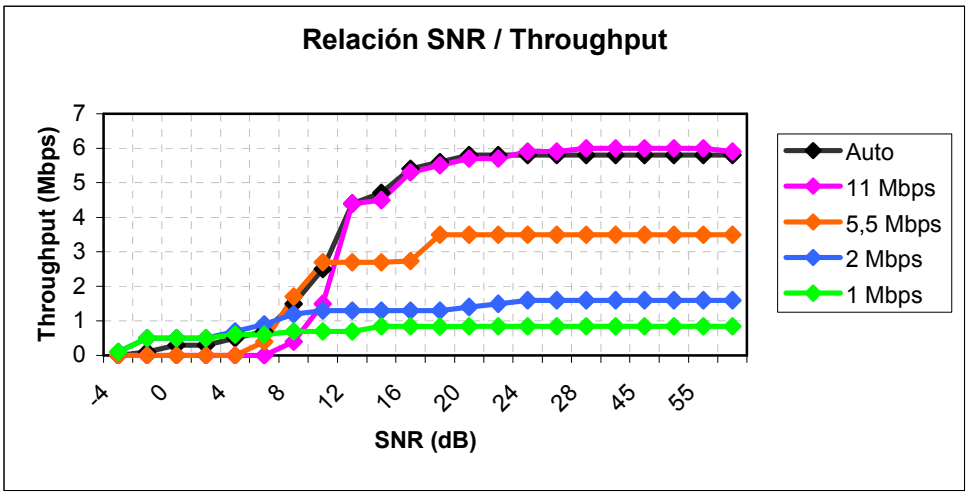

Fig. 1.9 Resultados obtenidos prueba 2

La gráfica nos muestra las tasas máximas obtenidas para cada una de las tasas estandarizadas (modulaciones) según el nivel de SNR.

En la **Fig. 1.9** se puede observar como la tasa estándar Auto es la que nos permite tener una tasa de transferencia máxima en general en cada nivel de

SNR. Debido a su automatización, configura la tasa más adecuada en cada momento optimizando así los recursos del canal.

A la tasa estándar 11Mbps, la tasa de transferencia cae a niveles de 1 Mbps a partir de una SNR de 10dB, en cambio, a tasa estándar de 5,5 Mbps para el mismo nivel de SNR, consigue mantener la tasa de transferencia a 3 Mbps. Esto nos muestra que la modulación utilizada en 5,5 Mbps es más tolerante a interferencias que la utilizada por la tasa de 11 Mbps.

Para las tasas estándar de 2Mbps y 1Mbps se consigue mantener una tasa de transferencia más o menos constante a lo largo de todos los niveles de SNR mostrados en la gráfica. Podemos ver que a niveles menores a -4 dB, la señal ya se pierde por lo que el throughput es nulo. Podemos ver que estas tasas son las más robustas a interferencias, sobretodo la de 1 Mbps.

1.3. Prueba 3. Interferencias entre canales

El estándar IEEE 802.11b permite configurar hasta 13 canales (1 - 13) diferentes, los cuales ofrecen acceso a los clientes. Se opera en la banda de 2,4 GHz con una separación entre canales de 5 MHz entre portadoras. En la práctica solo se configuran tres canales en la misma zona debido a las interferencias entre canales adyacentes (normalmente se utilizan los canales 1, 6 y 11), tal y como se puede observar en la **Fig. 1.10**.

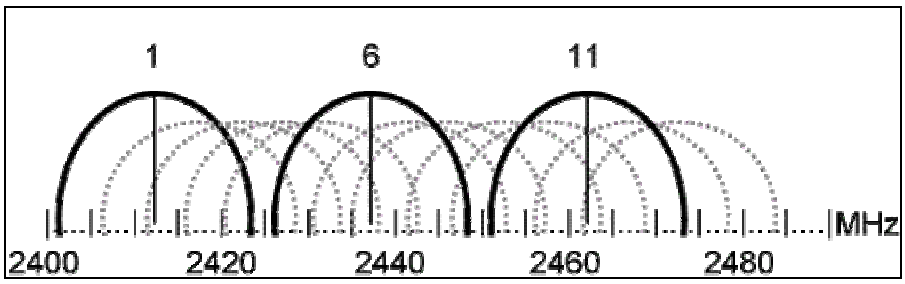

Fig. 1.10 Canales ISM a 2,4 GHz

Esta prueba tiene como objetivo observar el impacto de las interferencias entre canales adyacentes en las tasas de transmisión. Se pretende analizar el comportamiento que tienen las tasas de transferencia cuando existen otras redes inalámbricas cercanas en frecuencia que interfieren (en forma de ruido) a la transmisión.

Como en la prueba anterior, se realizará la misma prueba para las diferentes tasas estandarizadas en el 802.11b (Auto, 11Mbps, 5,5 Mbps, 2 Mbps y 1 Mbps).

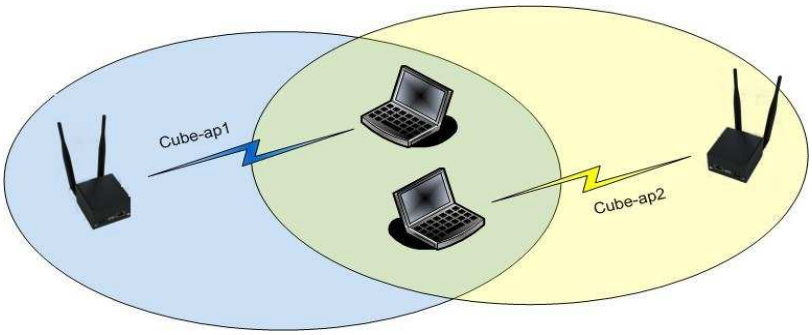

Fig. 1.11 Escenario 3

Tal y como se muestra en la **Fig. 1.11**, en esta prueba intervienen 4 entidades (2 cubos mesh y 2 clientes), los cuales forman dos redes inalámbricas independientes. Es decir, tenemos en la misma área dos redes con identificador de sesión (ESSID) diferentes: *cube-ap1* y *cube-ap2*. El entorno donde se realiza esta prueba es un entorno cerrado, un laboratorio del Campus de Castelldefels.

La primera red (*cube-ap1*) se utilizará como red interferente, con una transmisión constante que utilizará todo el ancho de banda del canal (aproximadamente una transmisión de 6 Mbps). Esta red operará en el canal 1. La segunda red (*cube-ap2*) será nuestra red de pruebas en la cual observaremos el impacto de la interferencia en la tasa de transmisión. Esta red la iremos cambiando de canal, y en cada canal se realizarán las pruebas de throughput correspondientes para cada una de las tasas soportadas por los equipos. Los resultados esperados siguen un patrón como el que se muestra en **Fig. 1.12**.

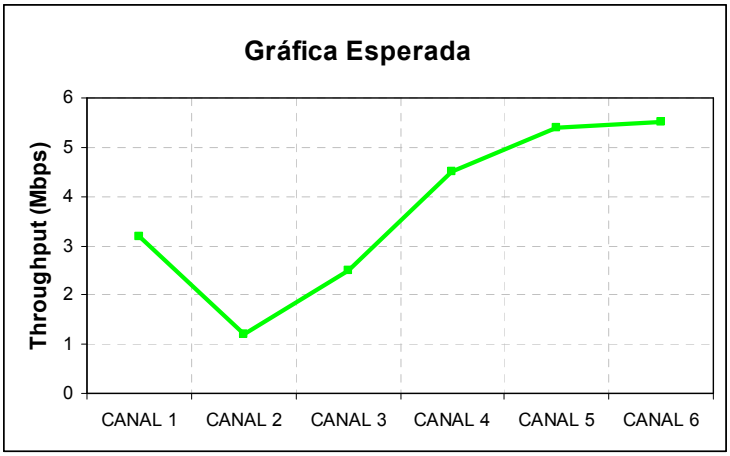

Fig. 1.12 Gráfica esperada prueba 3

En la gráfica vemos como en el canal 1, el mismo en el que opera la red inalámbrica interferente, el throughput no se ve tan afectado como en el canal adyacente (canal 2). En el canal 2 es cuando la señal se ve más afectada por

la red interferente y como consecuencia, el throughput disminuye hasta su valor mínimo (aprox. 1 Mbps).

Esto es debido al funcionamiento del propio protocolo de acceso al medio CSMA/CA [7] (Carrier Sense Multiple Access/Collision Avoidance). Una estación antes de transmitir escucha el canal y si está libre, transmite. En el caso que el medio esté ocupado, la estación se espera un tiempo aleatorio (backoff) antes de volver a intentar transmitir. De esta manera, se intenta evitar colisiones con otras estaciones que estén usando el mismo canal.

Así pues, si dos estaciones están trabajando en el mismo canal, aún siendo de redes diferentes, se escucharán entre ellas y podrán evitar colisiones, lo que se traduce en que se repartirán el ancho de banda por igual.

En cambio, si dos estaciones están en redes diferentes y canales adyacentes, no podrán evitar las colisiones porque no se podrán escuchar entre ellas. Por lo tanto, la transmisión de una estación, si la otra no dispone de buenos filtros o los canales están solapados (por ejemplo una distancia menor a 5 canales), se traducirá en un nivel de ruido en el canal adyacente que provocará un mayor número de paquetes perdidos y un descenso del throughput. A medida que se aleje el canal, el filtro podrá seleccionar mejor y eliminar esta potencia interferente, mejorando el rendimiento.

En la **Fig. 1.12**, podemos ver que conforme la red inalámbrica opera en canales más alejados del canal 2, cada vez las interferencias son más pequeñas. A partir del canal 4 la tasa es prácticamente la máxima (aprox. 6 Mbps).

Así pues, podemos afirmar que la interferencia de canal adyacente en ocasiones es peor que la interferencia co-canal.

Esta forma de gráfica es la que esperamos obtener en nuestras pruebas, sobretodo para las de mayor tasa de transmisión ya que son más sensibles a interferencias. Otra situación esperada es el hecho que se reparta la tasa de transmisión entre todos los usuarios que comparten el mismo canal, es decir, si dos usuarios de redes diferentes o de la misma red utilizan el mismo canal, deberían repartirse el ancho de banda de una manera más o menos equitativa.

A continuación se muestra en **Fig. 1.13** los resultados obtenidos en esta prueba. Como podremos ver, la mayoría de las formas no se corresponde con la gráfica esperada. Esto es un hecho que nos llevó a repetir varias veces las mismas pruebas dado su comportamiento no esperado, pero viendo que los resultados no eran nada estables además de no semejarse a los de **Fig. 1.12**, se concluye que el comportamiento de los meshcubes no es el esperado.

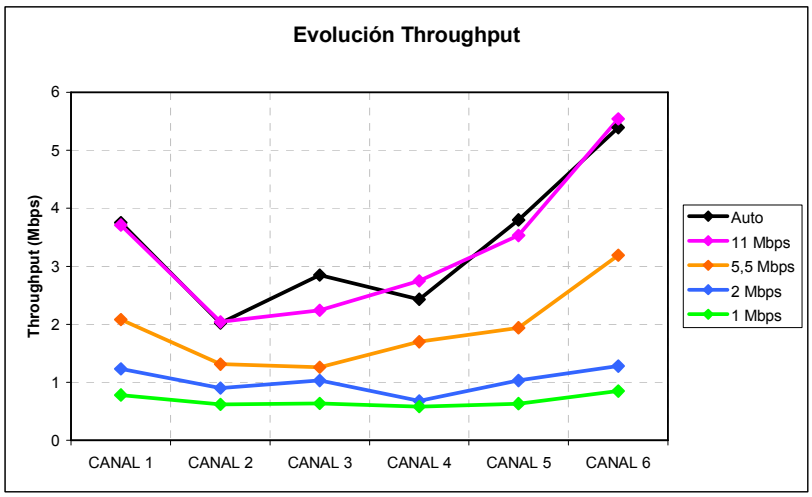

Fig. 1.13 Resultados obtenidos prueba 3

A pesar de este comportamiento, en algunos casos podemos ver que los resultados obtenidos se corresponden con los resultados esperados. Por ejemplo, en la forma de la tasa Auto, podemos ver que en el canal 2 obtenemos menor throughput que en el canal 1, tal y como se esperaba. También podemos observar como va creciendo el throughput a medida que nos alejamos del canal 2. El resultado inesperado lo encontramos en el canal 4, donde se obtiene un throughput menor que en canal 3 y 5. La causa de este hecho la podemos encontrar en el propio algoritmo de adaptación de tasa que incorpora la tasa "Auto", que en este canal seguramente no ha elegido la modulación más óptima. Si no fuese por este resultado inesperado, la forma de la gráfica se parecería bastante a la gráfica esperada.

En la forma de la tasa de 11 Mbps hemos podido comprobar que la repartición de throughput entre los dos clientes se ha efectuado correctamente, dado que en el canal 1 cada uno obtenía una tasa media de 3,5 Mbps. En esta gráfica vemos que la evolución es bastante lineal y el throughput asciende correctamente a medida que hay un mayor número de canales en medio de las dos redes. Tan sólo el throughput de los canales 4 y 5 es ligeramente inferior al esperado.

En la forma de la tasa de 5,5 Mbps observamos un comportamiento no deseado en el canal 3 y 5. Vemos que en el canal 3 obtenemos menor throughput que en su anterior, hecho que no coincide con la teoría. Comportamientos indeseados en el canal 3 los hemos encontrado en todas las repeticiones de esta prueba.

En la forma de la tasa de 2 Mbps, el comportamiento inesperado lo encontramos sobretodo en el canal 4.

El comportamiento de la tasa de 1 Mbps sí que corresponde más o menos con el esperado. Podemos ver que en general la gráfica se mantiene constante. La teoría en este caso se cumple ya que a tasas menores, los dispositivos son menos sensibles a interferencias y prácticamente en su totalidad cumplen con la tasa especificada ya que sin problemas alcanzan de 700 a 800 Kbps.

CAPÍTULO 2. SIMULACIONES CON NS-2

Este capítulo tiene como objetivo realizar las simulaciones de las pruebas efectuadas anteriormente con los meshcubes y comparar los resultados obtenidos en la simulación con los que se obtuvieron en las pruebas reales.

Para este proyecto se ha elaborado un manual para poder crear un escenario con NS-2 y realizar una simulación. Este manual se encuentra en el Anexo I.

2.1. Prueba 1. Potencia recibida vs Distancia

Para la simulación de esta prueba se necesita saber qué potencia recibe el receptor a una distancia determinada. Para ello no se ha requerido el uso de la herramienta de simulación, sino más bien la fórmula del modelo de propagación que utiliza el simulador para calcularla.

Existen tres modelos de propagación que vienen modelados en el simulador NS-2 [8]. Son los siguientes:

- FreeSpace (espacio libre)
- Tworayground
- Shadowing

El modelo de propagación de FreeSpace (espacio libre) viene programado en el simulador NS-2, concretamente en el archivo *propagation.cc* en el directorio /mobile/. La fórmula **2.1** es la que utiliza el simulador para calcular la potencia recibida en espacio libre:

$$\Pr(d) = \frac{PtGtGr\lambda^2}{(4\pi)^2 d^2 L} \qquad \textbf{(2.1)}$$

donde,

Pt = 15 dBm = 0,031623 W
Gt = 5 dB = 3,16227
Gr = 3 dB = 2
λ = c / f = 3*10^8 (m/s) / 2,412*10^9 (Hz) = 0,1244 m
D = [0 – 200] m
L = 1

El otro modelo también utilizado es el de Tworayground que a distancias largas es más realista que a distancias cortas. Simula que en el receptor existen dos señales que provienen del emisor: el rayo directo y el rayo reflejado. Mediante

la interferencia que simula el rayo reflejado se consigue una simulación más cercana a la realidad que en el modelo de Freespace (espacio libre).

El modelo que se acerca más al caso real es el de Shadowing que intenta simular los desvanecimientos provocados por obstáculos y por la propagación multicamino que se tienen en recepción. De esta forma se consigue que los resultados de la simulación se acerquen más a la realidad.

En este modelo se define un factor exponente β que puede tomar diferentes valores según el tipo de entorno donde se realizan las pruebas. La siguiente tabla muestra algunos valores típicos del factor β.

Tabla 2.1. Valores típicos del factor β

Environment		β
Outdoor	Free space	2
	Shadowed urban area	2.7 to 5
In building	Line-of-sight	1.6 to 1.8
	Obstructed	4 to 6

Para ver el modelo que se acerca más al entorno de nuestras pruebas se ha realizado una comparación de los resultados obtenidos en la prueba con valores teóricos de cada modelo.

A partir de la **Fórmula 2.1** se extrae la siguiente gráfica comparativa con los resultados reales:

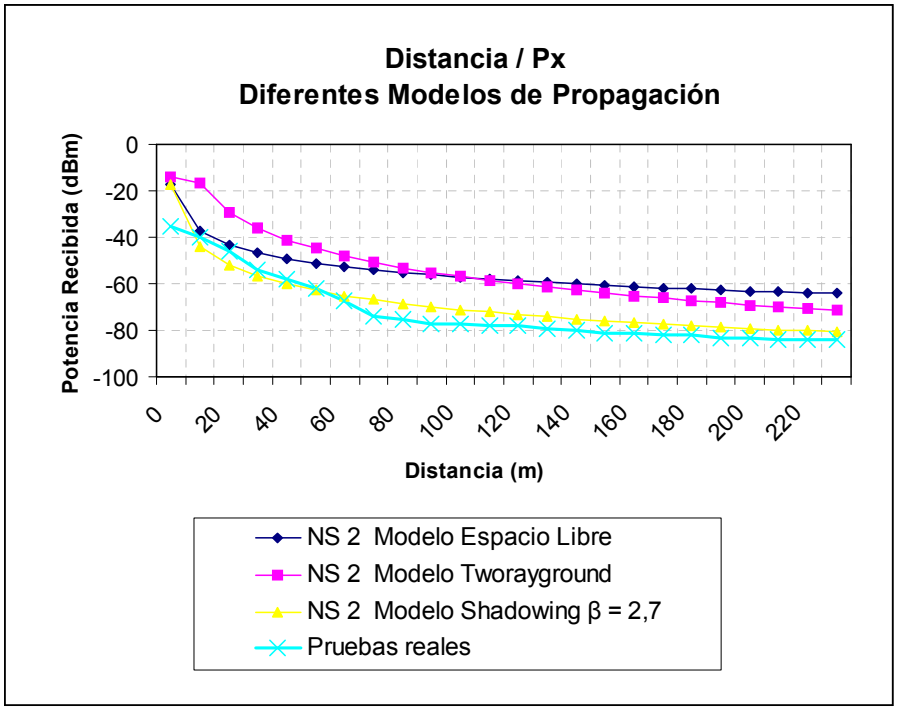

Fig. 2.1 Comparativa entre los resultados teóricos de cada modelo de propagación con los resultados de las pruebas reales

Se puede observar que el modelo de Shadowing con β = 2,7 es el que se acerca más a los resultados obtenidos en la prueba real. Para asegurarnos que tomamos la decisión correcta realizaremos el cálculo del error relativo medio:

Tabla 2.2. Error relativo medio

	NS 2 Modelo Espacio Libre	NS 2 Modelo Tworayground	NS 2 Modelo Shadowing β = 2,7
Error relativo medio	23%	27%	8%

Como se puede observar en la **Tabla 2.2**, el modelo de propagación de shadowing con B = 2,7 es el que más se aproxima a nuestras pruebas reales dado que es el que tiene un error más pequeño. Por ello se configurarán las siguientes pruebas con este modelo de propagación.

La configuración del modelo de propagación Shadowing en el script tcl es de la siguiente forma.

```
# first set values of shadowing model

Propagation/Shadowing set pathlossExp_ 2.7 ;#  factor β
Propagation/Shadowing set std_db_      4.0 ;# desviación (dB)
Propagation/Shadowing set dist0_       1.0 ;# distancia de referencia(m)
Propagation/Shadowing set seed_        0 ;# seed for RNG

$ns_ node-config -propType Propagation/Shadowing
```

2.2. Prueba 2. Throughput vs SNR

En esta prueba se pretende observar el comportamiento de la red a diferentes niveles de SNR en el receptor para las tasas estandarizadas. A partir de la prueba anterior se configura la simulación con el modelo de propagación Shadowing y el factor β = 2,7. Con ello se conseguirá una buena aproximación al entorno donde se realizaron las pruebas reales.

Para determinar qué nivel de SNR se tiene en cada una de las simulaciones, nos basaremos en los cálculos teóricos de potencia recibida representados en la **Fig. 2.1**. Para configurar la simulación con el fin de obtener en el receptor una SNR deseada se introducirá la distancia entre nodos en el escenario de la simulación. De esta forma podremos observar qué throughput es capaz de

transmitir el emisor al receptor dado un nivel de SNR. Posteriormente, se realizará la comparación entre los resultados obtenidos en la simulación y los resultados obtenidos en las pruebas reales.

Para ver mejor la comparativa la representaremos gráficamente (**Fig. 2.2**):

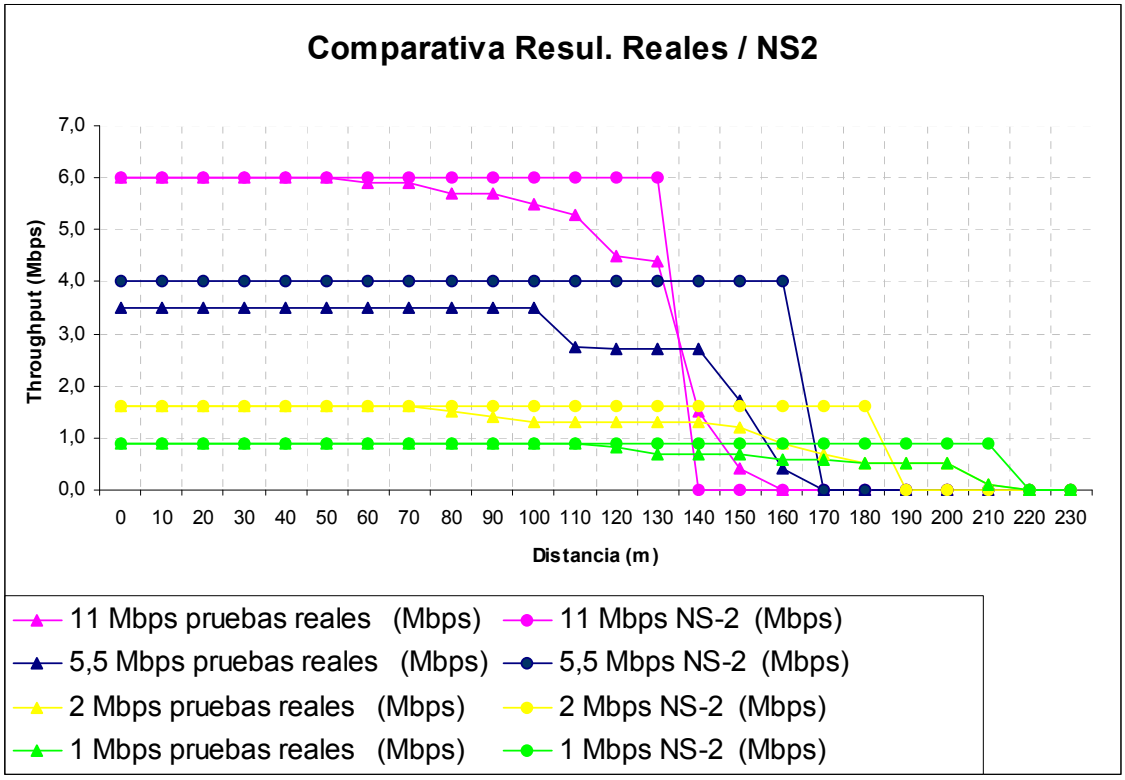

Fig. 2.2 Comparativa entre resultados reales y simulaciones con NS-2

La **Fig. 2.2** nos muestra las tasas máximas de las transmisiones para cada una de las tasas estándar según la distancia. Se puede observar que los resultados son parecidos hasta que la distancia es suficientemente grande como para perder la señal de transmisión. Es en esta distancia donde la tasa para las simulaciones cae de su máximo a tasa nula en pocos metros de diferencia. Esto es debido a que el valor de desviación std_db del modelo shadowing de propagación esta fijado a 0dB, con lo cual no existe una tolerancia entre los valores de medida.

Una vez realizadas las pruebas se ha detectado una de las características que tiene NS-2 en su versión estándar. Debido a la forma en que se ha programado el simulador, un paquete de una transmisión simulada será correcto dependiendo de la potencia con la que éste se reciba en el receptor siempre y cuando el valor de desviación std_db no permita una tolerancia del valor de la potencia recibida. Por ello a partir de un cierto umbral de potencia recibida no se interpretarán paquetes correctos con lo cual la tasa de transferencia será nula. En realidad lo que sucede es que a mayor distancia menor nivel de SNR, lo que provoca que el **BER** (Bit Error Ratio) y el **PER** (Packet Error Ratio)

aumenten con la distancia. Todo ello provoca una pérdida de la eficiencia del canal y por lo tanto una caída en el throughput de transferencia.

Por ello las tasas obtenidas en los resultados de las simulaciones son o la tasa máxima de transferencia o tasa nula (0 Mbps). Las variables que marcan cuándo se dejan de recibir paquetes son la distancia y el umbral de potencia recibida (RXThresh_) que varía según la tasa con la que se configure el receptor de la transferencia.

Los resultados muestran como a medida que la tasa física estándar utilizada es menor, el throughput de datos se mantiene durante una distancia cada vez mayor. Esto es debido a las codificaciones que utilizan las tasas estándar de capa física. Cuanto más baja la tasa, más robustez ante ruido e interferencias.

A pesar que los resultados de las simulaciones no se aproximan a la realidad debido a las características del simulador, los resultados demuestran que el simulador se comporta como en la realidad en términos generales ya que a menor tasa física más distancia se consigue en la transferencia.

2.3. Prueba 3. Interferencias entre canales

La última prueba que se simula intenta ver las pérdidas de datos a causa de las interferencias entre canales adyacentes en frecuencia. En las pruebas reales se han realizado todas las combinaciones posibles variando el canal de la red inalámbrica y su tasa estándar de comunicación (que hace variar la codificación utilizada en la transferencia).

El simulador de redes NS-2 no fue creado para simular redes inalámbricas. Debido a esto se han programado extensiones para poder simular de forma más real las interferencias entre nodos móviles. Por ello mediante el paquete NS-2 estándar sólo se puede realizar la simulación de redes inalámbricas que operen en la misma frecuencia, es decir, en el mismo canal de frecuencias 802.11 (canal 1).

Los resultados obtenidos para esta simulación de la prueba 3 son los presentados en la **Fig. 2.3**.

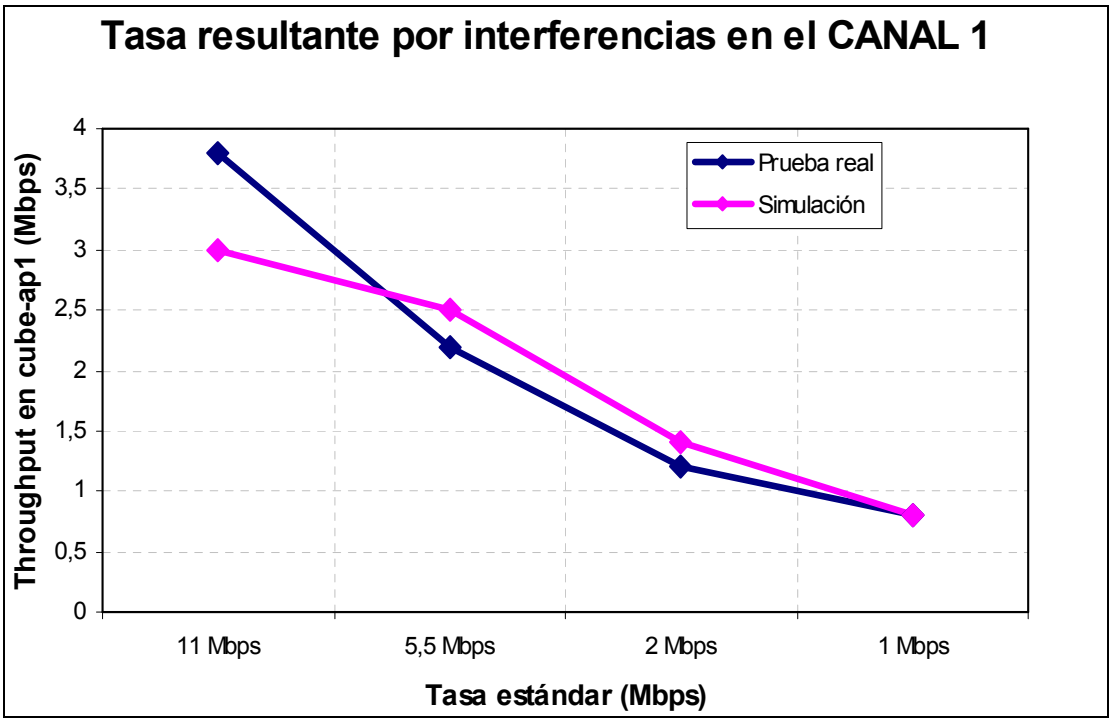

Fig. 2.3 Comparativa entre la tasa real obtenida y la tasa simulada

En la **Fig. 2.3** se muestran los resultados de las pruebas reales en el Canal 1 para las diferentes tasas estándar de transmisión y los de las simulaciones. Podemos ver que los resultados son bastante parecidos con lo que se puede concluir que el simulador nos permite simular el comportamiento de la red de una forma correcta.

Para el desarrollo de las simulaciones de canales adyacentes al Canal 1 es necesario la instalación de una extensión para redes wireless llamada **TeNS** (The Enhanced Network Simulator) [9]. Esta herramienta permite cubrir parte de las deficiencias que tiene el simulador NS-2 en el protocolo de acceso al medio 802.11, dado que éste está programado de forma muy simplificada en el NS-2 estándar. Aparte de implementar el protocolo de una manera más realista también aporta nuevas funcionalidades como la simulación de múltiples interfaces radio en un mismo nodo móvil, un protocolo de encaminamiento estático para redes inalámbricas, simulación de los 13 canales del protocolo 802.11 o la implementación del comportamiento de antenas unidireccionales.

Así pues, la herramienta TeNS nos permitiría simular las interferencias entre canales adyacentes en frecuencia ya que está implementado para ello.

CAPÍTULO 3. SIMULACIONES CON OPNET MODELER

En este apartado se mostrarán los resultados de las simulaciones en OPNET Modeler [10]. Se han simulado las tres pruebas realizadas con los *meshcubes* para así poder comparar los resultados con los que se obtuvieron en las pruebas reales.

Para este proyecto se ha elaborado un manual para poder crear un escenario con OPNET y realizar una simulación. Este manual se encuentra en el Anexo II.

3.1. Prueba 1. Potencia recibida vs Distancia

En esta prueba podremos comprobar qué potencia recibe el receptor a una cierta distancia. Esto no es trivial ya que depende de varios parámetros como es potencia recibida, transmitida o lo más importante en este caso, el modelo de propagación y de canal que utiliza el simulador.

Por defecto, OPNET Modeler utiliza el modelo de propagación “Default Link Closure” (DLC), tal y como se menciona en el programa. Este modelo de propagación se parece bastante al de espacio libre (FreeSpace), pero con alguna deficiencia. Si quisieras utilizar otro modelo de propagación, puedes realizar dos cosas:

- Adquirir la licencia de OPNET Terrain Modeling Module (TMM). A partir de este módulo se podría elegir el modelo de propagación a utilizar, aparte de poder simular un escenario real y poder definir obstáculos, variables meteorológicas, etc.

- Modificar manualmente los ficheros de programación que incorpora el simulador. En la **Fig. 3.1** se puede observar todo el proceso y todos los ficheros que son necesarios para calcular los resultados derivados de una transmisión. Los ficheros más fundamentales son el *dra_closure.ps.c* y el *dra_chanmatch.ps.c* que se pueden encontrar en el directorio <reldir>/models/std/wireless.

  Como se puede ver en la **Fig. 3.1**, hay multitud de ficheros que dependen unos de los otros, así que esta opción de modificar manualmente los ficheros de programación es una tarea que conlleva mucho tiempo y que se salía de los objetivos del proyecto.

  Así que, para las simulaciones descritas a continuación se ha utilizado el algoritmo DLC como modelo de propagación y el algoritmo “Default Channel Match” (DCM) como modelo de canal.

La explicación de dichos ficheros se puede encontrar en el "Product Documentation" del "menú Help", en el apartado "Wireless Module User Guide → Radio Transceiver Pipeline".

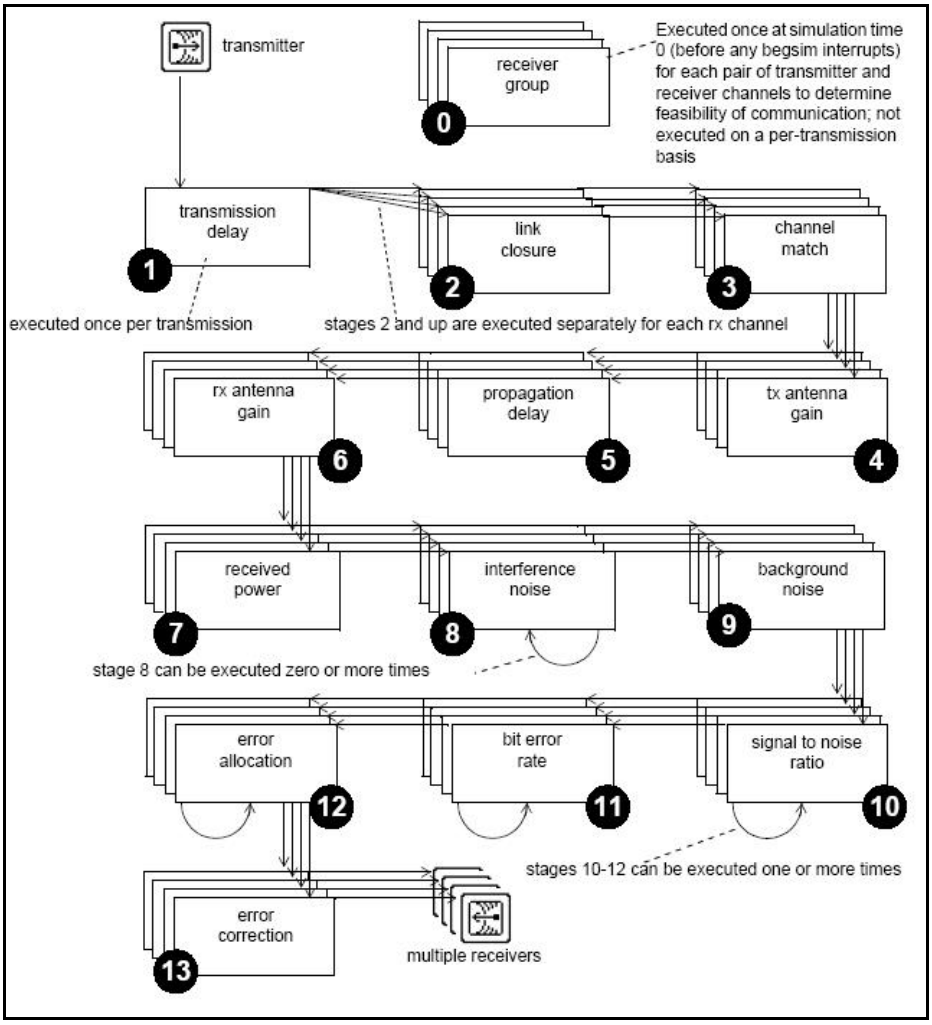

Fig. 3.1 Secuencia de cálculo de potencia para una transmisión

La fórmula que utiliza DLC (**Fig. 3.2**) es la misma que utiliza NS-2 por defecto, la de espacio libre.

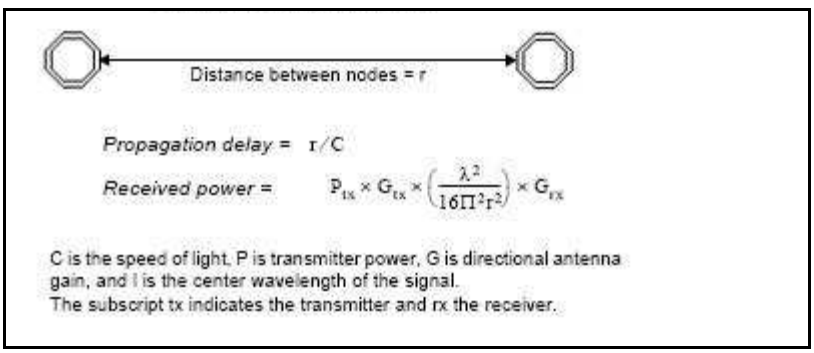

Fig. 3.2 Fórmulas para cálculo de potencia según algoritmo por defecto

donde,

Pt = 15 dBm = 0,031623 W
Gt = 3 dB = 2
Gr = 3 dB = 2
λ = c / f = 3*10^8 (m/s) / 2,412*10^9 (Hz) = 0,1244 m
D = [0 – 250] m
L = 1

Para simular esta prueba se definió un escenario (**Fig. 3.3**) con un nodo inalámbrico y un punto de acceso. Además, se definió una trayectoria para el nodo inalámbrico para que se fuera alejando del punto de acceso a una velocidad de 1,2 Km/h. Durante toda su trayectoria el simulador medía la potencia y la SNR recibida.

Fig. 3.3 Escenario prueba 1 con OPNET

En la **Fig. 3.4** podemos ver que los resultados obtenidos con OPNET están en medio de los resultados teóricos del modelo de espacio libre y los resultados obtenidos en las pruebas reales.

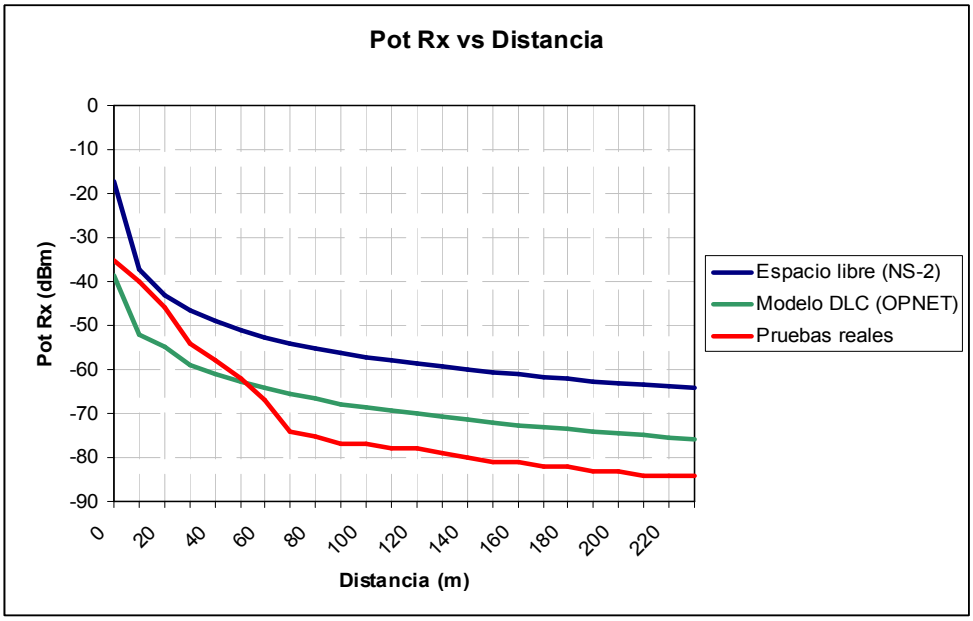

Fig. 3.4 Comparativa resultados prueba 1

3.2. Prueba 2. Throughput vs SNR

En esta prueba podremos comprobar qué throughput son capaces de alcanzar los nodos inalámbricos a medida que se alejan y disminuye la SNR. El escenario será el mismo que el de la **Fig. 3.3**. Se realizará una simulación por cada tasa estandarizada. En cada simulación deberemos cambiar las siguientes variables (**Tabla 3.1**) en los dos nodos que componen el escenario.

Tabla 3.1. Variables a configurar para cada tasa de transmisión

Tasa	Modulación	Sensibilidad
11 Mbps	cck_11	- 81 dBm
5,5 Mbps	cck_55	- 84 dBm
2 Mbps	dpsk	- 85 dBm
1 Mbps	dpsk	- 87 dBm

Para poder medir el throughput con OPNET a lo largo de toda la simulación, se ha definido una aplicación FTP donde el nodo inalámbrico se descarga un fichero de gran tamaño a lo largo de toda la simulación para así tener siempre el canal ocupado a su máximo rendimiento.

A partir de los resultados obtenidos en esta prueba se ha elaborado la **Fig. 3.5**, donde se comparan los resultados obtenidos en OPNET con los de las pruebas reales.

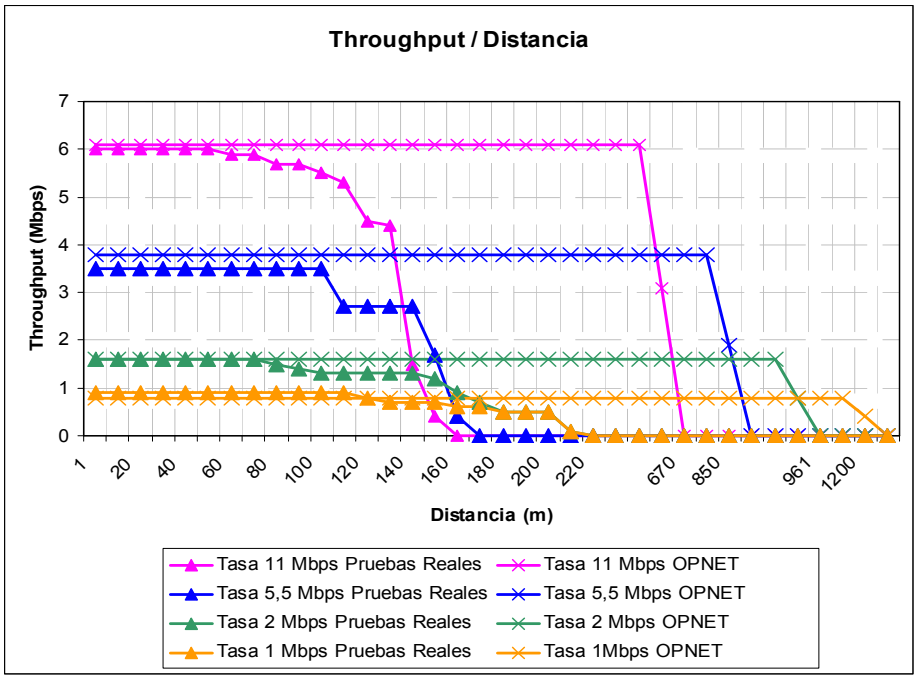

Fig. 3.5 Comparativa resultados prueba 2

Como podemos ver en la **Fig. 3.5**, a cortas distancias se alcanzan valores máximos de throughput en todas las tasas, tanto en las pruebas reales como en las simuladas en OPNET.

A medida que se aleja el cliente del servidor, podemos ver que las gráficas correspondientes a las simulaciones en OPNET alcanzan valores de throughput máximo a distancias mucho más lejanas que las de las pruebas reales. Aparte de esto, podemos ver que en OPNET siempre se mantiene el valor máximo de throughput mientras llega algo de señal. En el momento que la potencia recibida es inferior al umbral, se pierde la señal y la conexión con el servidor. El funcionamiento del DLC de OPNET se basa en un modelo booleano de “True” o “False” donde si la potencia recibida es mayor al umbral, el throughput es máximo y viceversa.

En la **Fig. 3.6** se pueden observar dos gráficas realizadas por el propio simulador: una de SNR y la otra de throughput. Cuando la estación no recibe señal, se observa como automáticamente el throughput es nulo.

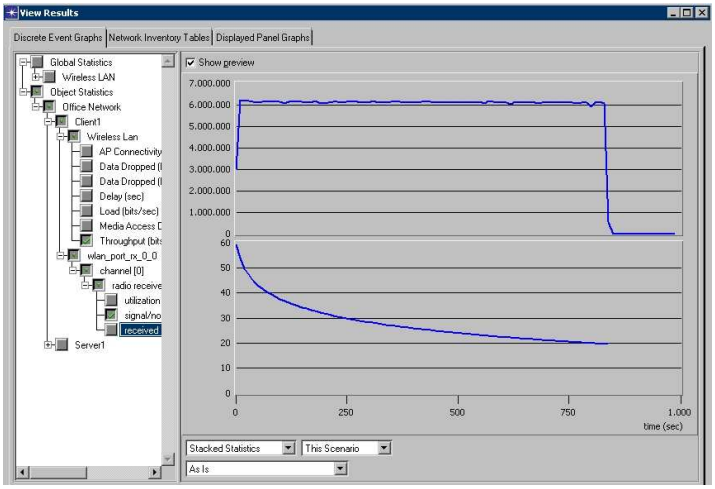

Fig. 3.6 Pantalla resultados OPNET

Podemos afirmar pues, que el simulador OPNET en estos momentos no está aplicando ningún algoritmo de error ya que las distancias a las que se llega son extremadamente grandes, además de la no existente pérdida de throughput a medida que nos alejamos, tal y como pasa en las pruebas reales.

3.3. Prueba 3. Interferencias entre canales

Esta prueba tiene como objetivo observar el impacto de las interferencias entre canales adyacentes además de las interferencias co-canal en las diferentes tasas de transmisión.

Se pretende analizar el comportamiento que tienen las tasas de transferencia cuando existen otras redes inalámbricas cercanas en frecuencia que interfieren (en forma de ruido) a la transmisión.

El escenario de la simulación es el que se muestra en la **Fig. 3.7**. Está compuesto por dos servidores y por dos clientes inalámbricos. Un cliente y un servidor estarán siempre transmitiendo al máximo throughput posible en el canal 1. Los otros dos nodos se cambiarán de canal además de tasa de transmisión para ver qué repercusiones tiene en el sistema. Estos dos últimos también intentarán siempre transmitir a throughput máximo. Se realizará una simulación para cada tasa estandarizada y para cada canal en la que operen.

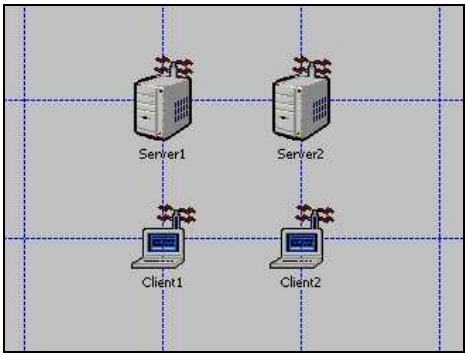

Fig. 3.7 Escenario prueba 3 con OPNET

En la **Tabla 3.2** podemos ver la repercusión que tienen todos los nodos del escenario en su Throughput cuando los nodos "variantes" cambian de canal y cuando cambian de tasa de transmisión.

Tabla 3.2. Resultados obtenidos prueba 3

	Tasa 11 Mbps		Tasa 5,5 Mbps		Tasa 2 Mbps		Tasa 1 Mbps	
	Through. Nodos Fijos (Mbps)	Through. Nodos Variantes (Mbps)	Through. Nodos Fijos (Mbps)	Through. Nodos Variantes (Mbps)	Through. Nodos Fijos (Mbps)	Through. Nodos Variantes (Mbps)	Through. Nodos Fijos (Mbps)	Through. Nodos Variantes (Mbps)
CANAL 1	3	3	2,2	2,2	1,2	1,2	0,73	0,73
CANAL 2	3	3	2,2	2,2	1,2	1,2	0,73	0,73
CANAL 3	3	3	2,2	2,2	1,2	1,2	0,73	0,73
CANAL 4	3	3	2,2	2,2	1,2	1,2	0,73	0,73
CANAL 5	3	3	2,2	2,2	1,2	1,2	0,73	0,73
CANAL 6	6,1	6,1	6,1	3,8	6,1	1,7	6,1	0,84

En la **Fig. 3.8** podremos ver reflejados estos resultados.

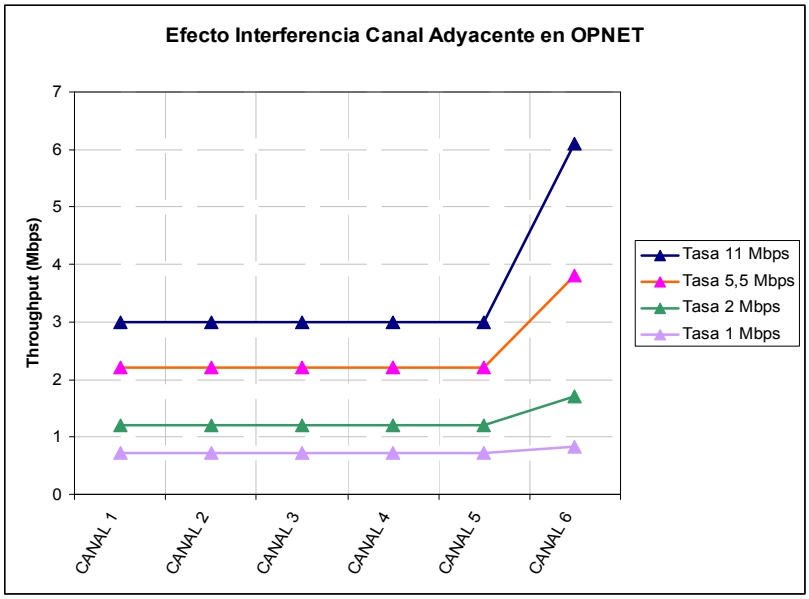

Fig. 3.8 Efecto Interferencia canal adyacente en Nodos variantes

Podemos ver que hasta el canal 5, los dos clientes inalámbricos se reparten el throughput equitativamente. No es hasta el canal 6 cuando los nodos recuperan el throughput máximo a cada tasa, debido a que ya existen 5 canales de separación entre las dos ESSID y por lo tanto no generan interferencias la una a la otra.

Estos resultados nos hacen pensar que OPNET no está utilizando un buen filtro para eliminar la potencia interferente de las estaciones vecinas, ya que a partir del canal 4 se debería mejorar el throughput, tal y como se ha comentado en el capítulo de las pruebas reales.

CAPÍTULO 4. COMPARATIVA SIMULADORES CON PRUEBAS REALES

Este capítulo tiene como objetivo comparar los resultados obtenidos de diferentes simuladores (NS-2 y OPNET) con las medidas realizadas en los escenarios reales del Capítulo 1. Además se realizará una cierta comparativa entre OPNET y NS-2 relativa a su aspecto funcional.

OPNET y NS-2 son dos de los simuladores de redes más conocidos dentro del entorno académico y empresarial.

NS-2 como ya se ha comentado anteriormente es un simulador “open source”, por lo que no conlleva ningún coste, un aspecto atractivo para el entorno académico. Esto conlleva a que no dispone de soporte técnico, pero se pueden encontrar multitud de documentos y foros en la red. Para la creación de escenarios sencillos mediante NS-2 es necesario el conocimiento del lenguaje de script TCL. Para visualizar los resultados se requiere una herramienta adicional, el NAM.

OPNET Modeler, en cambio, es un simulador por el que hay que adquirir licencias. Para entornos universitarios existen licencias gratuitas limitadas, pero si se requiere soporte técnico o algún módulo adicional en especial para la simulación, se debe adquirir la correspondiente licencia de pago. Los modelos que utiliza OPNET son de código abierto ya que tienes acceso al código para implementar nuevas funciones. Las fuentes del kernel del simulador, en cambio, no son públicas.

El entorno de simulación de OPNET es mucho más atractivo que el de NS-2. Además, tiene multitud de parámetros modificables, por lo se puede realizar una simulación mucho más detallada, hecho que a su vez comporta mayor dificultad de manejo, y en consecuencia la necesidad de adquirir la licencia de soporte técnico.

A continuación se comparan los resultados de los simuladores con los de las pruebas reales. En la prueba 1 hemos podido comprobar que NS-2 se acerca más a los resultados de las pruebas reales utilizando el modelo de propagación de shadowing ajustando el parámetro β a 2,7. Con OPNET Modeler, como ya se ha comentado anteriormente, se pueden utilizar otros modelos de propagación aparte del DLC para ajustarse más a las pruebas reales, pero se debería adquirir la licencia de TMM. El obstáculo de esta prueba para NS-2 es que no permite medir la potencia recibida en cada punto, por lo que se ha tenido que realizar de forma teórica. En OPNET sí que se permite.

En la prueba 2, si comparamos la **Fig. 2.2** y la **Fig. 3.5**, podemos ver que NS-2 vuelve a semejarse más a las pruebas reales. El problema de OPNET en este caso es el mismo que el de la prueba 1. El modelo de propagación por defecto atenúa muy poco la señal a largas distancias, por lo que se conseguían

caudales máximos a distancias mucho más grandes que en las pruebas reales. Un hecho común para los dos simuladores es que implementan un modelo binario, por lo que si la potencia recibida en un punto es mayor que un umbral, el caudal es máximo. Si es menor, el caudal es nulo en el caso de OPNET. En el caso de NS-2, pasado el umbral se puede establecer una probabilidad de pérdida variando el valor de *std_db*.

Por último en la prueba 3, podemos ver en la **Fig. 2.3** y **Fig. 3.8** que tanto OPNET como NS-2 no proporcionan resultados correctos y los clientes se han repartido el ancho de banda hasta que la separación entre canales era la recomendada, 5 canales.

En conclusión podemos decir que NS-2 está más orientado para redes cableadas y en su distribución básica no implementa muchas posibilidades para las redes inalámbricas. Por ejemplo, para la simulación de interferencias entre canales se deberían instalar módulos adicionales.

OPNET Modeler sí que está pensado tanto para redes cableadas como inalámbricas, pero el problema es que si se quieren realizar simulaciones para compararlas con la realidad, se deben adquirir licencias adicionales y no estaría de más la licencia de soporte técnico dada la cantidad de variables a configurar.

Otra comparativa más detallada de los dos simuladores y de otras pruebas reales se puede encontrar en [11] y [12]. En estas referencias se compara también resultados obtenidos en escenarios reales con los obtenidos en simulaciones con NS-2 y OPNET. Describen las limitaciones que se han encontrado con cada simulador y algún que otro comportamiento inesperado. En general concluyen que NS-2 proporciona resultados simulares a los de OPNET, pero la distribución gratuita de NS-2 resulta más atractiva para los investigadores. Por otro lado se comenta que OPNET al tener muchas más funcionalidades que NS-2 resulta más atractivo para las empresas telemáticas.

CAPÍTULO 5. NECESIDAD DE SEÑALIZACIÓN

5.1 Introducción

Hoy en día, se puede decir que la tecnología WLAN ha alcanzado un importante grado de madurez en el mercado y no se puede predecir cuando va a llegar su fin.

El número de redes inalámbricas continúa creciendo sin control debido a que éstas funcionan sobre una banda de frecuencias libre. En entornos muy poblados se puede observar la coexistencia de redes inalámbricas de empresas, redes domésticas, *hot spots* públicos, etc. que comparten la misma banda de frecuencias. En muchos casos el número de canales libres para no producir solapamiento con otras redes no son suficientes, por lo que la coexistencia resulta cada vez más problemática debido a la aparición de interferencias entre redes vecinas. La presencia de interferencias afecta al funcionamiento de las redes produciendo errores y colisiones, y como consecuencia se reduce el caudal además de aumentar el retardo de los paquetes.

Estos efectos se suelen minimizar con un buen diseño previo de la red: una buena colocación de los APs aparte de fijar de manera óptima las potencias transmitidas y los canales asociados para asegurar la cobertura de la zona y minimizar las interferencias con otras redes. La planificación es una buena técnica para paliar este problema en una red multicelular, pero en la realidad, los administradores de red no saben cuando va a aparecer un nuevo nodo interferente; además, no pueden administrar todos los componentes involucrados ya que pueden pertenecer a diferentes redes y de hecho, a diferentes propietarios.

Es por eso que un mecanismo centralizado de control no resulta apropiado, por lo que la solución más óptima sería un mecanismo de gestión distribuida. Pero para que un mecanismo de gestión sea distribuido, es necesaria la comunicación entre todos los componentes de la red, aún siendo de diferentes dominios y propietarios. Para introducir señalización en una red inalámbrica se propone la utilización de APs con dos interfaces inalámbricas: una realizando las funciones normales de un AP mientras la otra funciona en modo Ad-hoc para permitir la comunicación entre APs.

El intercambio de información entre APs no debe ser solamente entre nodos cercanos, se debe hacer llegar tal información al nodo más lejano si fuera necesario. En consecuencia, los APs deben ser capaces de retransmitir o encaminar la información para así poder llegar a cualquier nodo de la red.

En los apartados 5.2 y 5.3 se presentan dos mecanismos para realizar el transporte de la información por toda la red. En el primer apartado se presenta un mecanismo de transmisión a nivel 2 mediante packet sockets. En el

apartado 5.3 se plantea como solución a nivel 3 el uso de OLSR [13], un protocolo de encaminamiento proactivo.

5.2 Packet Sockets

5.2.1 Estudio Teórico

En este apartado del proyecto se realizará un estudio de las diferentes alternativas a la hora de implementar el intercambio de mensajes entre los APs de una infraestructura WLAN para enviar la información que se desee. En concreto, este estudio pretende escoger una técnica para posteriormente implementarla en el algoritmo del proyecto [14] y de esta forma optimizar el envío de información necesaria entre los diferentes AP de la infraestructura WLAN.

En los siguientes subapartados se muestran las diferentes alternativas disponibles y posteriormente se analizan las ventajas y desventajas de cada una de ellas.

5.2.1.1 Packet Sockets

Actualmente el algoritmo WNRA implementado utiliza sockets convencionales para el envió de información entre APs. En este caso pretendemos ver las características de los sockets a más bajo nivel, en concreto los sockets de la capa de enlace. La documentación de packet sockets se ha referenciado únicamente del manual de programador de Linux “Packet” [15].

Los conectores de paquetes (packet sockets) se usan para recibir o enviar paquetes directos (raw) en el nivel del driver de dispositivo (Nivel 2 de OSI). Permiten al usuario implementar módulos de protocolo en el espacio de usuario por encima de la capa física.

```
#include <sys/socket.h>
#include <features.h>   /* para el número de versión de glibc */

#if __GLIBC__ >= 2 && __GLIBC_MINOR >= 1

#include <netpacket/packet.h>
#include <net/ethernet.h>       /* los protocolos de nivel 2 */

#else

#include <asm/types.h>
#include <linux/if_packet.h>
#include <linux/if_ether.h>     /* los protocolos de nivel 2 */

#endif

packet_socket = socket(PF_PACKET, int socket_type, int protocol);
```

- *Socket_type* es o bien **SOCK_RAW** para paquetes directos incluyendo la cabecera del nivel de enlace o bien **SOCK_DGRAM** para paquetes preparados con la cabecera del nivel de enlace eliminada. La información de la cabecera del nivel de enlace está disponible en un formato común en una estructura sockaddr_ll.

- *Protocol* es el protocolo IEEE 802.3 con los bytes en orden de red. Cuando se asigna a *protocol* el valor **htons (ETH_P_ALL)**, se reciben todos los protocolos. Todos los paquetes de entrada con el tipo de protocolo indicado se pasarán al conector de paquetes antes de ser pasados a los protocolos implementados dentro del núcleo

Los paquetes **SOCK_RAW** se pasan a y desde el driver del dispositivo sin ningún cambio en los datos del paquete. Cuando se recibe un paquete, la dirección todavía se analiza y se pasa en una estructura de dirección *sockaddr_ll* estándar. Cuando se transmite un paquete, el buffer proporcionado por el usuario debería contener la cabecera de la capa física. A continuación, ese paquete se encola sin modificar en la tarjeta de red de la interfaz definida por la dirección de destino. SOCK_RAW es similar pero no compatible con el obsoleto SOCK_PACKET de la versión 2.0 de Linux.

Los paquetes **SOCK_DGRAM** operan en un nivel ligeramente superior. Se elimina la cabecera física antes de que el paquete se pase al usuario. Los paquetes enviados a través de un conector de paquetes SOCK_DGRAM obtienen una cabecera adecuada de la capa física según la información de la dirección de destino *sockaddr_ll*, antes de ser encolados.

Por defecto, todos los paquetes del tipo de protocolo especificado se pasan a un conector de paquetes. Para obtener sólo los paquetes de una interfaz específica, se usa bind especificando una dirección en una estructura struct sockaddr_ll para enlazar el conector de paquetes a una interfaz. Sólo se usan para propósitos de enlace los campos de dirección **sll_protocol** y **sll_ifindex**.

5.2.1.1.1 Tipos de direcciones

sockaddr_ll es una dirección de la capa física independiente del dispositivo.

```
struct sockaddr_ll
{
   unsigned short   sll_family;    /* Siempre es AF_PACKET */
   unsigned short   sll_protocol;  /* Protocolo de la capa física */
   int              sll_ifindex;   /* Número de la interfaz */
   unsigned short   sll_hatype;    /* Tipo de cabecera */
   unsigned char    sll_pkttype;   /* Tipo de paquete */
   unsigned char    sll_halen;     /* Longitud de la dirección */
   unsigned char    sll_addr[8];   /* Dirección de la capa física */
};
```

- **sll_protocol** es el tipo del protocolo ethernet estándar dado en orden de red definido en el fichero cabecera **linux/if_ether.h**.

- **sll_ifindex** es el índice de la interfaz. Un 0 concuerda con cualquier interfaz.

- **sll_hatype** es un tipo ARP de los definidos en el fichero cabecera **linux/if_arp.h**.

- **sll_pkttype** contiene el tipo del paquete. Los tipos válidos son:

    - **PACKET_HOST** para un paquete aplicado al anfitrión (host) local.

    - **PACKET_BROADCAST** para un paquete de difusión de la capa física.

    - **PACKET_MULTICAST** para un paquete enviado a una dirección multidestino de la capa física.

    - **PACKET_OTHERHOST** para un paquete destinado a otros anfitriones que ha sido capturado por el manejador del dispositivo en modo promiscuo.

    - **PACKET_OUTGOING** para un paquete originado desde el anfitrión local que es devuelto de regreso a un conector de paquetes. Estos tipos sólo tienen sentido para recibir.

- **sll_addr** y **sll_halen** contienen la dirección de la capa física (por ejemplo, IEEE 802.3) y su longitud. La interpretación exacta depende del dispositivo.

5.2.1.1.2 Opciones de los conectores

Los conectores de paquetes sólo se pueden usar para configurar el envío multidestino de la capa física y el modo promiscuo. Esto funciona llamando a **setsockopt** con SOL_PACKET, para un conector de paquetes, y una de las opciones PACKET_ADD_MEMBERSHIP para añadir un enlace o PACKET_DROP_MEMBERSHIP para eliminarlo. Ambas esperan una estructura packet_mreq como argumento:

```
struct packet_mreq
{
   int           mr_ifindex;    /* índice de la interfaz */
   unsigned short    mr_type;       /* acción */
   unsigned short    mr_alen;       /* longitud de la dirección */
   unsigned char     mr_address[8]; /* dirección de la capa física */
};
```

mr_ifindex contiene el índice de la interfaz cuyo estado debe cambiarse.

mr_type indica la acción a realizar.

- PACKET_MR_PROMISC habilita la recepción de todos los paquetes sobre un medio compartido.
- PACKET_MR_MULTICAST enlaza el conector al grupo multidestino de la capa física indicado en **mr_address** y **mr_alen**.
- PACKET_MR_ALLMULTI configura el conector para recibir todos los paquetes multidestino que lleguen a la interfaz.

Además, se pueden usar las ioctls tradicionales, **SIOCSIFFLAGS**, **SIOCADDMULTI** y **SIOCDELMULTI**, para el mismo propósito.

5.2.1.2 Libpcap

Libpcap es una librería open source escrita en C que ofrece al programador una interfaz desde la que capturar paquetes en la capa de red, además Libpcap es perfectamente portable entre un gran número de Sistemas Operativos.

Antes de empezar a programar con las librerías Libpcap, se debe tener instalada una copia en la máquina de desarrollo, las fuentes oficiales de libpcap están en *tcpdump* [16]. La instalación es rápida, basta con seguir los tres pasos habituales: configurar (*./configure*), construir (*make*) e instalar como root (*make install*).

5.2.1.2.1 Esquematización de un programa

No importa lo complicado que sea el programa que vayamos a construir con Libpcap, este siempre seguirá un esquema básico (**Fig. 5.1**).

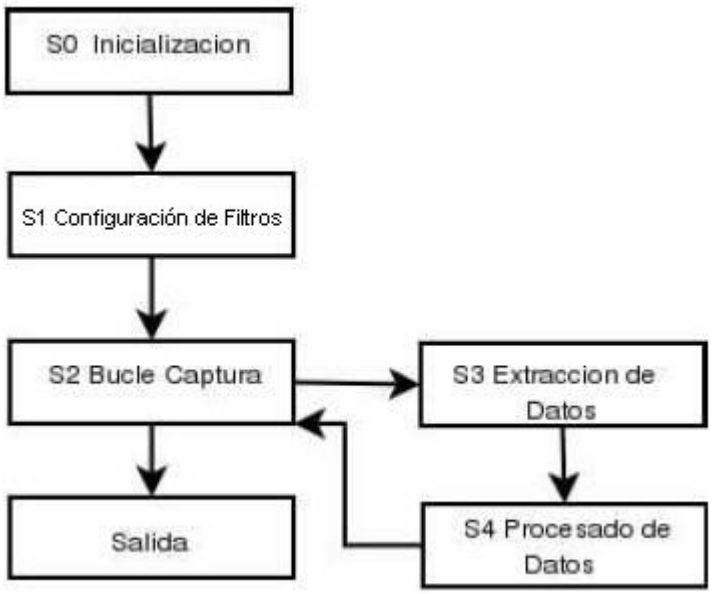

Fig. 5.1 Esquematización de un programa con Libpcap

En las siguientes secciones se desarrollan con ejemplos cada una de las fases mostradas en la **Fig. 5.1** numeradas como S1, S2, S3 y S4.

(S0) Obteniendo información del sistema

Como puede verse en el gráfico, la primera fase de nuestro programa es la Inicialización. Esta engloba las funciones capaces de obtener información del sistema: Las interfaces de red instaladas, los configuración de estas interfaces (Máscara de Red, Dirección de Red) etc.

(S1) Filtrar sólo lo que nos interesa

¿Qué es un Packet/Socket Filter?

Ya se ha comentado que Libpcap es una librería de funciones y que los procesos que ejecutan estas funciones, lo hacen a nivel de usuario (user space). Sin embargo la captura real de datos tiene lugar en las capas más bajas del Sistema operativo, en la llamada zona Kernel (Kernel area), debe por lo tanto existir un mecanismo capaz de traspasar esta frontera, y hacerlo además de un modo eficiente y seguro, ya que cualquier fallo en capas tan profundas degradaría el rendimiento de todo el sistema.

Supongamos que nos interesa programar una aplicación capaz de monitorizar la red en tiempo real en busca de paquetes cuyo puerto TCP destino sea el 80. Si no existiese ningún mecanismo de filtrado, el Kernel no sabría cuales son los paquetes en los que está interesada nuestra aplicación, por lo que tendría que traspasar la frontera Kernel - User space por cada paquete que transite por la red.

Para evitar esto, la solución pasa por que la aplicación establezca un filtro en la zona Kernel que sólo deje pasar los paquetes que nos interesen, por ejemplo, aquellos cuyo puerto TCP destino sea el 80, si queremos capturar paquetes del protocolo HTTP. Esta es la principal labor de un Packet/Socket Filter.

No existe un sistema único de filtrado, por el contrario prácticamente cada SO rescribe su propia solución: NIT para SunOS, Ultrix Packet Filter para DEC Ultrix, BPF para sistemas BSD (originalmente) y más recientemente LSF (Linux Socket Filter) la implementación para Linux.

(S2) Capturando Paquetes

Una vez sabemos cómo obtener el listado de las interfaces instaladas en nuestro sistema y sus configuraciones, ya estamos preparados para comenzar con la captura en sí. Existen varias funciones para capturar paquetes, las principales diferencias entre ellas son:

- Número de paquetes que queremos capturar.
- Modo de captura: normal o promiscuo.
- Manera en que se definen sus funciones de llamada o Callbacks (la función invocada cada vez que se captura un paquete).

(S3) Interpretando los datos

Organización de un Paquete.

Cuando una aplicación quiere enviar datos a través de una red, antes tiene que añadir las cabeceras de los protocolos que vaya a emplear en la transmisión.

Con el conjunto de los datos en bruto, también llamados datos RAW, para poder obtener información inteligible, tenemos que hacer la labor que la pila de protocolos hubiera hecho por nosotros, es decir extraer e interpretar las cabeceras añadidas por el remitente.

(S4) Volcando los datos a un fichero

Libpcap nos ofrece también la posibilidad de guardar los datos en un fichero para procesarlos más adelante.

5.2.1.3 Libnet

Libnet [17] es una librería en C que proporciona una interfaz de alto nivel a las capas de inyección de paquetes en múltiples plataformas. Libnet permite de una forma fácil y cómoda agrupar todas las funciones para escribir paquetes de datos en la red de cualquier protocolo basado en la pila TCP/IP.

Libnet fue escrita entre otros motivos para que los programadores tuvieran una interfaz más amigable que el lenguaje de bajo nivel a la hora de crear paquetes para su inyección por la red.

Libnet es el inyector de paquetes más utilizado. Trabaja junto a la librería libpcap para el control total de la red a nivel de paquete o datagrama.

5.2.1.3.1 Herramienta Némesis

Némesis [18] es una aplicación capaz de enviar la información que se quiera en una red utilizando TCP/IP. También trabaja a nivel de enlace (Ethernet) y permite la construcción de este tipo de paquetes.

El Proyecto Némesis está diseñado para ser una pila de IP (`IP stack') humana, portable y basada en línea de comandos para UNIX/Linux. El set está separado por protocolos, y debería permitir crear scripts útiles desde un(a) shell.

Se utiliza para probar y depurar una red, probar y depurar servicios de red específicos. Es una herramienta habitual a la hora de auditar servicios.

Existe una versión para Windows y otra para linux que se puede descargar en Packet Factory [18].

La versión de windows requiere además la instalación previa de Winpcap. La versión de linux requiere tener instalado libnet 1.0.2a.

En general para obtener ayuda basta con poner 'nemesis protocolo help' donde protocolo es el protocolo que pretendemos inyectar.

Utilizar un inyector de paquetes correctamente requiere tener conocimientos avanzados de TCP/IP porque sino, difícilmente podrán interpretarse los resultados.

5.2.1.4 Libdnet

Libdnet [19] es una librería que proporciona una interfaz simple y portable a las rutinas de los niveles bajos de la red, incluyendo la manipulación de las direcciones de red, los kernel arp cache, la tabla de encaminamiento, los cortafuegos de red y la transmisión de tramas Ethernet.

Esta librería es utilizada junto con la librería Libnet para la inserción de paquetes en la red.

5.2.1.5 Conclusiones del estudio teórico

Raw Sockets

Ventajas

- La principal ventaja con la que cuentan los raw sockets para ser la mejor opción es que la versión anterior del *algoritmo WNRA implementa packet sockets* convencionales para el envío de los paquetes de información entre APs. Es decir, el programa que implementa el algoritmo WNRA esta basado en esta técnica y la utilización de esta opción facilitaría la actualización del tipo de sockets, dado que el grado de impacto en el código original seria menor que si se utilizaran librerías libnet o libpcap.

- A partir del ejemplo de la utilización de los raw ethernet sockets [6] permite la fácil comprensión de los raw sockets y una rápida implementación a partir del código de ejemplo. Es decir, *fácil implementación*.

Desventajas

- Los RAW sockets no están estandarizados por la mayoría de plataformas en las cuales se utilizan lo que hace difícil una compatibilidad de funcionamiento en las diferentes capas del sistema.

- Linux requiere SO_BROADCAST para ser configurado para la inyección paquetes IP broadcast de RAW sockets. Esta función ya esta implementada en las librerías libnet.

- No son portables a otros sistemas operativos.

Librerías: Libpcap, Libnet y Libdnet

Ventajas

- Están programadas en lenguaje C el cual tenemos conocimiento.
- Son *portables* a cualquier SO con el cual se trabaje.
- Existe *mayor documentación* sobre ellas con respecto a los Packet Sockets.

Desventajas

- El código original del algoritmo WNRA implementa packet sockets para el envío de información entre APs. La opción por esta alternativa

provocaría un impacto mayor en la estructura del código original que los packet sockets.

- El estudio de las diferentes librerías comporta demasiado tiempo.

- En nuestro caso de implementación no se requiere tanta precisión en la creación de paquetes para el transporte de la información entre los APs. Las librerías Libpcap, Libnet y Libdnet permiten la creación de todo tipo de paquetes preparados para viajar por la red y los desarrollan con multitud de parámetros que en nuestro caso no son necesarios.

Comparando las varias soluciones a nivel teórico y de documentación la opción de utilizar Packet Sockets, del tipo *Raw Ethernet Sockets*, se cree que es la más adecuada para el proyecto que llevamos a cabo.

5.2.2 Ejemplo Implementación Packet Sockets

En este apartado se expone la forma en que se utilizan los packet sockets y un ejemplo de su utilización.

5.2.2.1 Utilización de Raw Ethernet Sockets

Actualmente el código que utiliza el algoritmo desarrollado en el TFC anterior [14] utiliza sockets convencionales a partir del protocolo de transporte UDP. Con ello la eficiencia en el transporte disminuye mucho ya que al utilizar protocolos de capas superiores como UDP/IP en la pila OSI se envían bytes innecesarios.

Los Raw Sockets sobre Ethernet eliminan esas cabeceras de protocolos de capas superiores para optimizar los bytes útiles respecto al total de bytes de la trama enviada.

Los Raw Ethernet Sockets nos permiten mediante la programación la construcción de paquetes Ethernet. Desde el punto de vista de programación son bastante parecidos a los sockets convencionales. Las principales diferencias son las siguientes:

- Los parámetros para la función de creación de sockets son diferentes.

- En lugar de direcciones IP (p ej. 192.255.255.255) se utilizan direcciones MAC (p ej. 00:00:00:5A:65:11).

- Una trama ethernet tiene que ser creada manualmente.

La **Fig. 5.2** nos muestra los campos en una trama ethernet convencional.

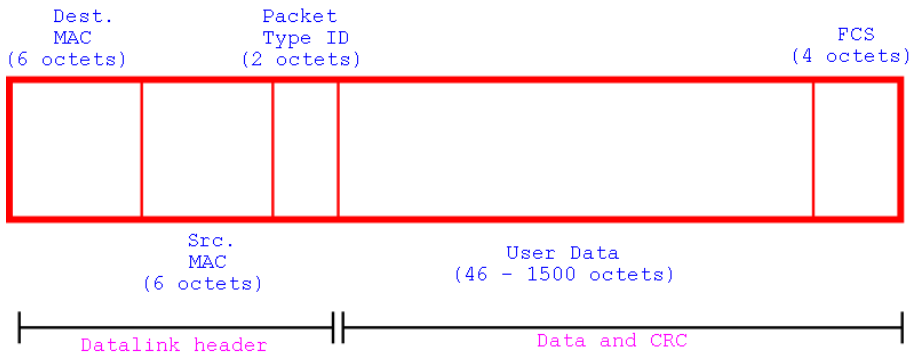

Fig. 5.2 Campos de una trama ethernet

Como se puede observar la trama consta de los siguientes campos:

- Dirección MAC Destino 6 bytes
- Dirección MAC Origen 6 bytes
- Tipo de Paquete 2 bytes
- Datos útiles 46 - 1500 bytes
- FCS 4 bytes

Con lo cual hacen un tamaño total máximo de 1518 bytes cuando el campo de datos útiles esta lleno.

Una vez visto los campos que tienen las tramas ethernet se procede a la utilización de los Raw Ethernet Sockets para el transporte de la información que nos interesa. Para ello nos hemos basado en el código fuente de ejemplo que se puede consultar en [20]. Este código será nuestra base para la modificación del código del algoritmo WNRA del cual queremos mejorar su rendimiento.

Los puntos más importantes que se han utilizado del código fuente de ejemplo son los presentados a continuación:

- Creación de un RAW ethernet socket

```
#include <sys/socket.h>
#include <linux/if_packet.h>
#include <linux/if_ether.h>
#include <linux/if_arp.h>
…
int s; /*descriptor del socket*/

/*creación del socket*/
s = socket (AF_PACKET, SOCK_RAW, htons(ETH_P_ALL));
if (s == -1) { errorhandling ... }
```

- Envío de una trama Raw ethernet

```
#define ETH_FRAME_LEN 1518
...
struct sockaddr_ll socket_address;                  /*Estructura de la dirección*/
void* buffer = (void*)malloc(ETH_FRAME_LEN); /*buffer para la trama ethernet*/
unsigned char* etherhead = buffer;                  /*Puntero a la cabecera ethernet*/
unsigned char* data = buffer + 14;                  /*Datos del usuario en la trama ethernet*/
struct ethhdr *eh = (struct ethhdr *)etherhead;     /*Otro puntero a la cabecera de la trama*/

unsigned char src_mac[6] = {0x00, 0x01, 0x02, 0xFA, 0x70, 0xAA};        /* @ MAC origen*/
unsigned char dest_mac[6] = {0x00, 0x04, 0x75, 0xC8, 0x28, 0xE5};       /* @ MAC destino*/

/*preparando sockaddr_ll*/

/*Comunicación RAW*/
socket_address.sll_family   = PF_PACKET;
socket_address.sll_protocol  = htons(ETH_P_IP); /*ya que no se utilizan protocolos por
encima de la capa de enlace (ethernet) se introduce cualquier valor en este campo*/
socket_address.sll_ifindex  = 2; /*índice de la interfaz de comunicación/
socket_address.sll_hatype   = ARPHRD_ETHER; /*identificador hardware ARP ethernet*/
socket_address.sll_pkttype  = PACKET_OTHERHOST; /*target is another host*/
socket_address.sll_halen    = ETH_ALEN; /*longitud de la dirección*/
/*MAC - comienzo*/
socket_address.sll_addr[0]  = dest_mac[0];
socket_address.sll_addr[1]  = dest_mac[1];
socket_address.sll_addr[2]  = dest_mac[2];
socket_address.sll_addr[3]  = dest_mac[3];
socket_address.sll_addr[4]  = dest_mac[4];
socket_address.sll_addr[5]  = dest_mac[5];
/*MAC - final*/
socket_address.sll_addr[6]  = dest_mac[6];  /*no utilizado*/
socket_address.sll_addr[7]  = dest_mac[7];  /*no utilizado*/

/*configuración de la cabecera de la trama ethernet*/
memcpy((void*)buffer, (void*)dest_mac, ETH_ALEN);
memcpy((void*)(buffer+ETH_ALEN), (void*)src_mac, ETH_ALEN);
eh->h_proto = 0x00;

/*Llenamos el campo de datos con información (en este caso cualquiera)*/
for (j = 0; j < 1500; j++) {
        data[j] = (unsigned char)((int) (255.0*rand()/(RAND_MAX+1.0)));
}

/*Envío del paquete*/
send_result = sendto(s, buffer, ETH_FRAME_LEN, 0,
            (struct sockaddr*)&socket_address, sizeof(socket_address));
if (send_result == -1) { errorhandling... }
```

- Recepción de una trama Raw ethernet

```
void* buffer = (void*)malloc(ETH_FRAME_LEN); /*Buffer para la trama ethernet*/
unsigned char* data = buffer + 14;                    /*Datos del usuario en la trama ethernet*/
int length = 0; /*longitud de la trama recibida/
...
length = recvfrom(s, buffer, ETH_FRAME_LEN, 0, NULL, NULL);
if (length == -1) { errorhandling .... }

printf("Longitud de la trama recibida: %i\n",length);  /*longitud de la trama ethernet recibida*/
printf("Datos recibidos: %s\n",data);                  /*datos del usuario en la trama ethernet*/
```

Mediante la utilización de los Raw Ethernet Sockets como se ve en las tres funciones básicas expuestas justo antes se consigue enviar y recibir paquetes ethernet a todos los APs que formen parte de la infraestructura WLAN, consiguiendo así que el algoritmo WNRA optimice la utilización del canal de comunicación de la red mesh.

5.2.2.2 Ejemplo Práctico

En este ejemplo se pretende observar el contenido de los paquetes Raw Ethernet que viajan por la red mediante la herramienta ethereal. El escenario consiste en dos maquinas: un cliente y un servidor.

La maquina cliente envía un paquete ethernet con el string "MARC: Hola!". El servidor la recibe y contesta con el string "DANI: Hola! Como estas?". En la siguiente captura (**Fig. 5.3**) podemos ver la conversación entre las dos maquinas.

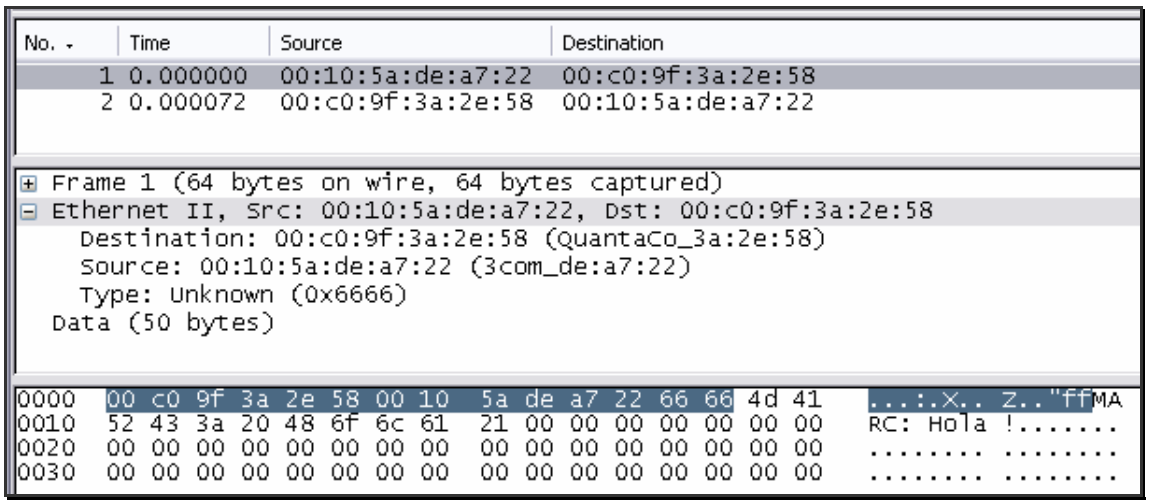

Fig. 5.3 Captura del Ethereal del Mensaje 1

Tabla 5.1. Campos del paquete Mensaje 1

Dirección MAC Destino	00:c0:9f:3a:2e:58
Dirección MAC Origen	00:10:5a:de:a7:22
Tipo paquete	0x6666[1]
Payload	"MARC: Hola!"
Longitud del paquete	64 bytes

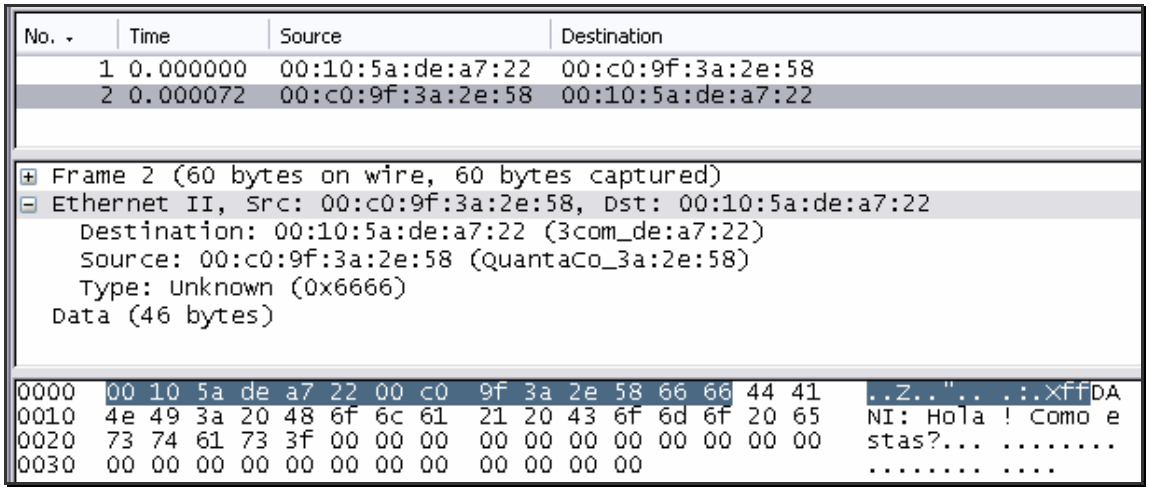

Fig. 5.4 Captura del Ethereal del Mensaje 2

Tabla 5.2. Campos del paquete Mensaje 2

Dirección MAC Destino	00:10:5a:de:a7:22
Dirección MAC Origen	00:c0:9f:3a:2e:58
Tipo paquete	0x6666
Payload	"DANI: Hola! Como estas?"
Longitud del paquete	60 bytes

Como se puede observar en las dos capturas los paquetes están creados por los campos de un paquete ethernet como se ha visto en la **Fig. 5.2**. Cabe decir que la longitud mínima requerida para una trama ethernet es de 60 bytes por lo cual el paquete es rellenado el campo de payload hasta 46 bytes que junto los 14 de cabecera suman el mínimo requerido.

[1] El tipo de protocolo 0x6666 es inventado y esta definido como ETH_P_PFC en el código fuente.

5.3 OLSR

5.3.1 Introducción OLSR

OLSR (Optimized Link State Routing Protocol) es un protocolo pensado para MANETs (Mobile Ad hoc Networks) cuyo estándar es el RFC 3626 [13].

Este protocolo se basa en tablas de enrutado. Una de sus características más importantes es que es proactivo [21], lo que quiere decir que cada nodo dispone en cada momento de toda la información del estado y disposición de los nodos de la red. Esto conlleva a una velocidad de enrutado mayor, además de la ventaja de conocer todos los nodos e IPs de la red. En contra, también aporta algunos inconvenientes como por ejemplo el hecho de que los nodos requieran un gran uso de memoria, además de ocupar la red con paquetes de información de enrutado. De esta manera el factor escalabilidad queda un poco en duda debido a que contra más nodos compongan una red, mayor memoria e información de enrutado se deberá transmitir por esta.

Lo contrario de proactivo sería el encaminamiento reactivo, que comportaría que cada vez que un nodo desee enviar un paquete a otro nodo, debe preguntar qué ruta debe seguir. Así pues, vemos que con este tipo de encaminamiento existe un mayor retardo, pero no ocupa la red con tantos mensajes de señalización.

OLSR se ha desarrollado para que pueda trabajar de manera independiente con otros protocolos, con lo que aporta gran versatilidad a la hora de implantarlo sobre un escenario, El concepto más importante que aporta este protocolo es el de MPR (Multipoint Relay), que como veremos más adelante son nodos cuya función principal es la de optimizar el número de mensajes de señalización por la red.

OLSR utiliza el concepto de retransmitir de HIPERLAN [22] (un protocolo de la capa de enlace) estandarizado por la ETSI. El protocolo se ha desarrollado en el proyecto IPANEMA (parte del programa Euclid) y en el proyecto PRIMA (parte del programa RNRT).

5.3.2 Mensajes OLSR

OLSR utiliza un único formato para todos los mensajes relacionados con este protocolo, de esta forma se facilita la extensión del protocolo sin problemas de compatibilidad con versiones anteriores.

Los paquetes se transmiten por la red, encapsulados en paquetes UDP utilizando el puerto 698 asignado por IANA [4].

Cada paquete puede contener uno o más mensajes. Una característica importante de estos mensajes es que incluyen un campo de número de secuencia, lo que permite a los nodos averiguar qué información es más actual.

Los tres tipos de mensajes que hacen posible el funcionamiento del protocolo son los siguientes:

- **Mensajes HELLO:** Realizan las funciones de descubrimiento de nodos, de detección de estado de enlaces y de señalización y elección de MPRs.

- **Mensajes TC (Topology Control):** Realizan la función de declaración de la topología de la red. Cada nodo guarda información de la topología de la red para poder realizar cálculos de tablas de encaminamiento.

- **Mensajes MID:** Realizan la función de descubrimiento de la presencia de múltiples interfaces en un nodo.

Para más información, consultar el RFC 3626 [13].

5.3.3 Funcionamiento

Mediante mensajes HELLO cada nodo puede detectar nodos vecinos y el estado de los enlaces con estos (enlace simétrico o asimétrico).

En la **Fig. 5.5** se puede ver un ejemplo sencillo de la detección de un vecino mediante mensajes HELLO. Tales paquetes se emiten periódicamente y en su interior se publican además los nodos vecinos de cada nodo.

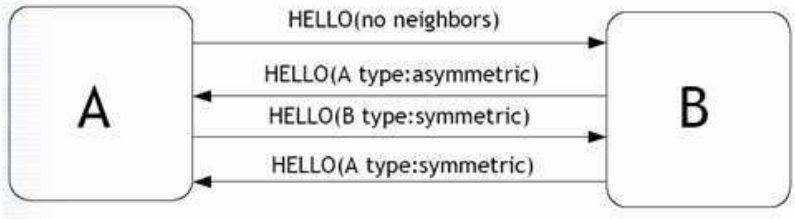

Fig. 5.5 Intercambio de mensajes OLSR para detectar vecinos

También es con este tipo de mensaje con los que cada nodo escoge sus MPRs. El concepto de MPR se ha introducido para minimizar el número de mensajes de control por la red. Cada nodo de la red escoge sus nodos MPR entre sus nodos vecinos, que como mucho distan un salto. Los nodos MPRs serán los encargados de retransmitir los mensajes, mientras los no escogidos como MPR recibirán y procesarán los mensajes pero no los retransmitirán.

Una condición básica para elegir a los nodos MPR es que el nodo inicial pueda llegar a todos los nodos situados a dos saltos a través de tales nodos MPR. Así pues, un mensaje broadcast emitido por el nodo inicial podrá llegar a todos los nodos situados a dos saltos gracias al proceso de retransmisión de los MPR. Tal explicación se comprenderá mejor con la **Fig. 5.6**, donde los nodos “m”

serian los MPR y los nodos “n” serían los nodos situados a dos saltos respecto el nodo inicial.

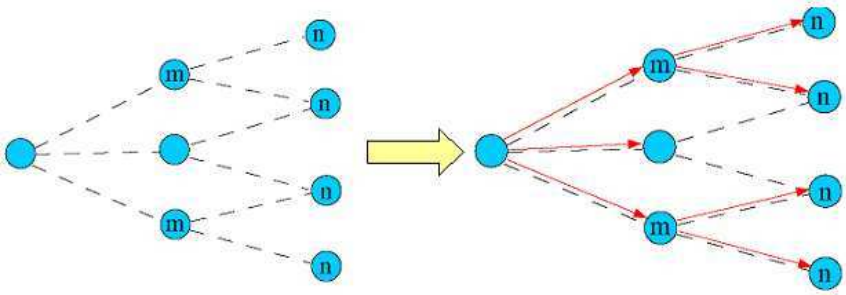

Fig. 5.6 Elección de nodos MPR

Los nodos MPR proporcionan un mecanismo eficiente para transmitir por toda la red los mensajes de control reduciendo el número de transmisiones necesarias. Para visualizar mejor la eficiencia del mecanismo se muestran a continuación dos imágenes (**Fig. 5.7** y **Fig. 5.8**) donde se ven dos procesos de retransmisión de mensajes: el primero utiliza el típico broadcast y el segundo utiliza inundación por MPRs.

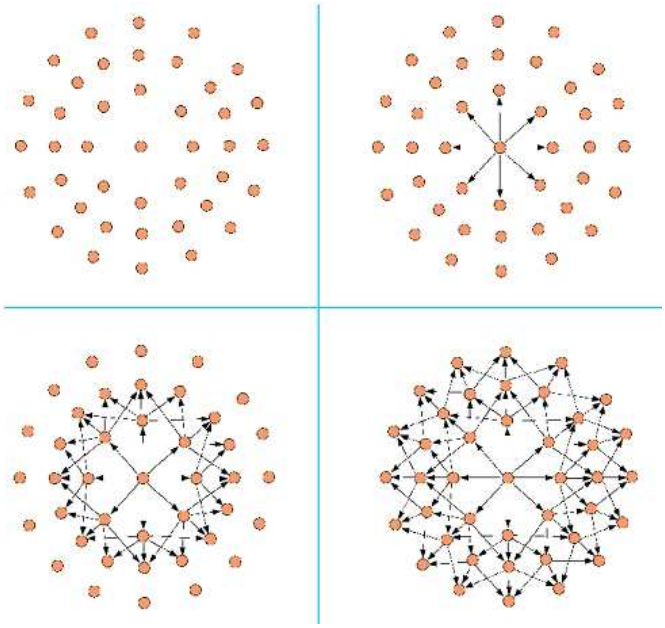

Fig. 5.7 Típico proceso de inundación por la red

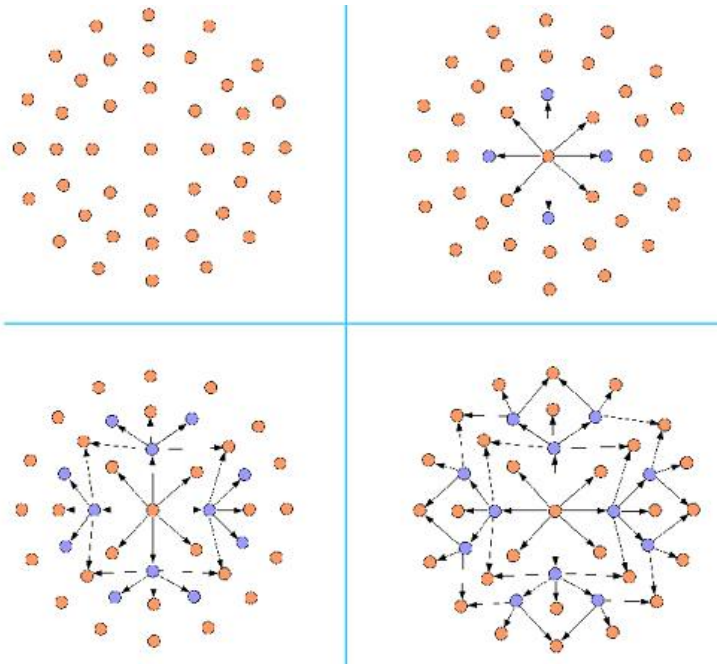

Fig. 5.8 Inundación por la red mediante nodos MPR

Así pues, los nodos escogidos como MPR, basándose en este tipo de inundación, difunden y retransmiten por toda la red información sobre la topología con los mensajes TC (Topology Control).

Un mensaje TC es creado y enviado por la red por un nodo MPR para declarar un grupo de enlaces, denominado “advertised link set”, entre los cuáles se ha de incluir como mínimo los enlaces a todos los nodos que lo han escogido como MPR.

De esta forma, cada nodo almacena una tabla de encaminamiento que se va formando a partir de los mensajes recibidos periódicamente con la información del estado de los enlaces. Esta tabla se actualiza cuando se detecta algún cambio en el enlace, en el vecino, en el vecino a dos saltos o en la topología.

Normalmente la ruta se realiza a través de los MPRs. De esta manera evitas problemas como por ejemplo la transmisión de paquetes de datos sobre enlaces uni-direccionales a través de los cuales no recibirías contestación por parte del nodo destino.

Para mayor fiabilidad se utilizan MPRs redundantes, es decir, se escoge un número más grande de MPRs que los que propiamente harían falta. Este hecho inserta más tráfico en la red, pero es muy útil en redes con mucha movilidad para poder asegurar que los paquetes llegan a su destino.

5.3.4 Demostración

A continuación se muestran las capturas de pantalla (**Fig. 5.9**) de una famosa simulación [24] en flash del protocolo OLSR, donde se puede ver el proceso de descubrimiento de vecinos y elección de MPRs.

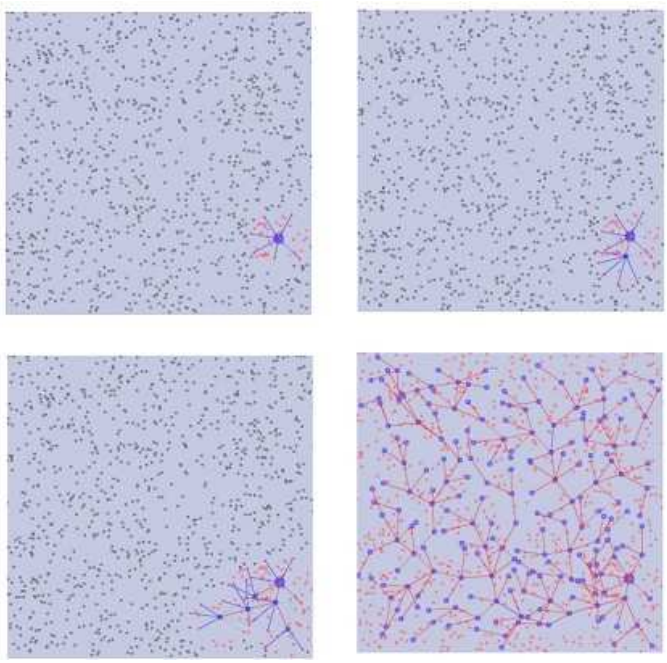

Fig. 5.9 Demostración en flash del funcionamiento de OLSR

5.3.5 Funcionalidades adicionales mediante plugins

El protocolo OLSR tiene la peculiaridad de ser fácilmente extensible para poder añadirle diversas funcionalidades [25] no contempladas en el RFC. Como se ha explicado anteriormente, los mensajes siempre siguen el mismo formato sea cuál sea el tipo.

El campo de “tipo de mensaje” está formado por un byte (256 posibilidades), de las cuáles las primeras 127 están reservadas para mensajes propios del RFC y las demás se pueden utilizar para otros fines.

Debido a que el área de las MANETs está todavía en desarrollo [26], la posibilidad de añadir o modificar funciones en sus protocolos de encaminamiento, ofrece una gran posibilidad para mejorar o implementar nuevas soluciones.

Protocolos pensados para MANETs, como por ejemplo AODV, DSR, etc., están limitados al no disponer de un método eficiente de retransmisión. El método de inundación por MPR junto con su algoritmo de retransmisión hacen de OLSR un protocolo muy interesante para su extensión.

Estas dos funcionalidades son las que permiten añadir, utilizando plugins, nuevas aplicaciones que requieran inundar la red como por ejemplo el servicio DNS. Esto significa que cualquier aplicación podría enviar tráfico por la red ad-hoc utilizando el método de inundación de OLSR. De esta manera el demonio olsrd funcionaría como agente de retransmisión para aplicaciones locales y así se reduciría la carga de la red.

OLSR está diseñado para retransmitir paquetes con cabecera OLSR, a pesar de ser de un tipo desconocido [8]. Puede ocurrir que sólo algunos nodos de la red entiendan este tipo de mensajes, pero esto no influiría en el proceso de retransmisión. Es un aspecto muy útil ya que si un usuario desea utilizar la técnica de inundación de OLSR para transmitir cierta información que puedan procesar algunos nodos de la red, solamente tendrá que encapsular estos datos dentro de mensajes con formato OLSR y con algún identificador de tipo de mensaje desconocido para utilizarlo en su plugin.

Otro motivo para crear plugins sería el poder cambiar funcionalidades del propio OLSR. Por ejemplo, mediante un plugin se podría cambiar el tratamiento que se da cuando llega un mensaje HELLO. Esto implica que mediante un plugin se puede manipular cualquier parte del demonio olsrd.

Los plugins se pueden escribir en cualquier lenguaje que se pueda compilar como una DLL (Dynamically Loadable Library) [28]. Una DLL es una parte de código que contiene funciones y datos y que se pueden ejecutar en tiempo real. Los plugins proporcionan nuevas funciones a una aplicación sin alterar su funcionamiento normal. Las DLL en Linux se conocen como ficheros *.so* mientras que en Windows se conocen como ficheros *.dll*

Para que los plugins y el demonio olsrd se comuniquen perfectamente, hace falta un interfaz tal como el que aparece en la **Fig. 5.10**:

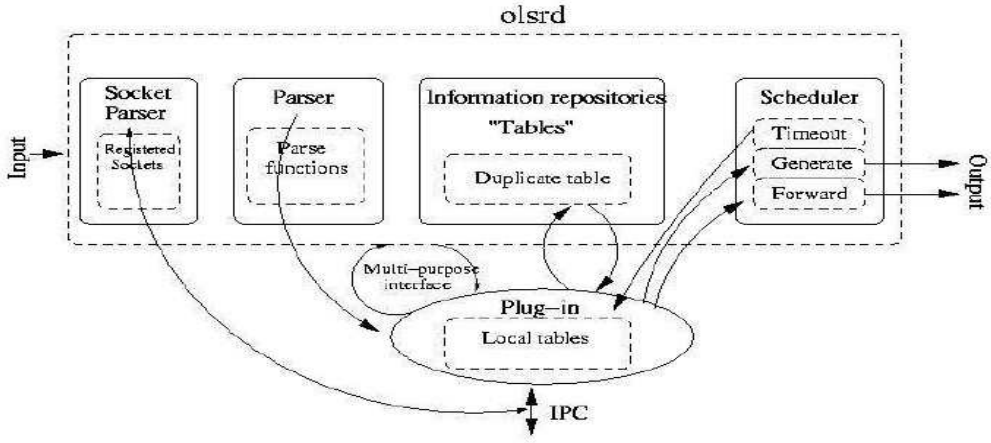

Fig. 5.10 Interfaz entre olsrd y plugin

Con esta interfaz, el demonio sabe qué puede recibir y enviar al plugin y viceversa. Como se puede ver, un plugin puede interactuar con cada parte del demonio, lo que aporta grandes posibilidades a la hora de modificarlo.

A continuación se detallan las distintas partes que componen el demonio olsrd:

- **Socket Parser:** Es el encargado de escuchar el tráfico que llega al nodo y llamar a las funciones asociadas a los datos a tratar.

- **Parser:** Recibe todo el tráfico entrante OLSR y dependiendo del mensaje decide descartarlo si no es válido, retransmitirlo según el algoritmo de retransmisión por defecto o enviar el mensaje a la función asociada a ese tipo de mensaje.

- **Information Repositories:** Es donde se guarda la información obtenida de los mensajes OLSR y a partir de la cuál se realizarán todos los cálculos de rutas.

- **Scheduler:** Se encarga de realizar funciones en su correspondiente instante de tiempo. Es decir, si un nodo ha de transmitir un mensaje en un momento determinado se debería registrar la generación del paquete en este componente.

La relación del demonio olsrd con los plugins permite interacciones como las siguientes:

- El plugin puede añadir y quitar sockets además de sus correspondientes funciones con el Socket Parser, muy útil para el IPC.

- El plugin puede registrar funciones en el Parser para un tipo de mensaje determinado.

- El plugin puede establecer en el scheduler un timeout para el refresco de las tablas.

- El plugin puede configurar el scheduler para realizar una función en un instante determinado.

- El plugin puede acceder a las tablas duplicadas de OLSR para comprobar qué mensajes han sido procesados o retransmitidos anteriormente.

- El plugin puede acceder a la función de transmitir mensajes de OLSR.

A través de esta interfaz podemos ver que olsrd permite el acceso de todas sus funciones y datos a los diferentes plugins para que puedan hacer uso de ellos.

CAPÍTULO 6. IMPLEMENTACIONES

En este capítulo se presentan dos implementaciones que intentan satisfacer las necesidades de señalización en la red mesh. La primera de ellas se presenta en el apartado 6.1 y plantea una solución en la capa 2 mediante Raw Ethernet Sockets. La solución planteada en el capítulo 6.2 es una solución de capa 3 y se basa en la creación de un plugin para trabajar sobre el protocolo de encaminamiento OLSR.

6.1 Capa 2 – WNRA con Raw Ethernet Sockets

En un proyecto anterior [14] se desarrolló un algoritmo (WNRA – Wireless Network Roadmap Algorithm), [29] capaz de construir en cada nodo de la red mesh un mapa completo de la red que reúne los datos necesarios para efectuar la distribución frecuencial entre los diferentes APs de manera automática y dinámica. Este algoritmo realiza el envío de información entre APs, los cuales, mediante el uso de una segunda interfaz radio, participan en una red mesh dedicada a la señalización. Dicho mecanismo se basa en la creación de sockets UDP convencionales, los cuales son ineficientes debido a que en realidad no es necesario operar con todos los protocolos de comunicación que estos proporcionan ya que la información puede viajar en tramas de nivel 2. Así pues, esta parte del proyecto tiene como objetivo mejorar la eficiencia en el transporte de los datos en la red de señalización, con la utilización de *raw sockets* para permitir la eliminación de las cabeceras de protocolos que no se utilizan y efectuar así un transporte de la información directamente sobre la capa de enlace de 802.11.

Las mejoras del algoritmo son las presentadas a continuación:

1. Sustitución de los sockets convencionales que utilizan UDP/IP (capa IP) actualmente en el algoritmo por Raw Ethernet Sockets de nivel 2 (capa MAC) mejorando así su eficiencia en la formación de los paquetes que utiliza el algoritmo para el transporte de información entre los nodos de la red.

2. La segunda mejora es a partir de la anterior, ya que una vez se ha mejorado la eficiencia en la cabecera del paquete la optimización se ha de realizar también en el campo de datos (payload). Actualmente el algoritmo envía la información en una cadena de carácteres de codificación ASCII. Esta forma de envío es poco eficiente ya que se necesitan mucho bytes del campo de datos para almacenar todos los carácteres de información. El objetivo de esta mejora es optimizar al máximo el campo de datos del paquete. Para ello se ha de diseñar una estructura de los datos óptima para que la información se envíe con el mínimo número de bytes posibles.

Se puede observar que las mejoras pretenden que el algoritmo WNRA sea más eficiente en el transporte de sus datos entre los nodos de la red.

6.1.1 Mejora 1: Paso de sockets convencionales a packet sockets

El algoritmo WNRA en su actual versión utiliza los sockets convencionales con una estructura UDP/IP en la creación de sus conexiones. Al no ser necesaria la capa IP de nivel 3, sus cabeceras junto con las de transporte UDP son innecesarias para el paso de la información. Por ello se trabajará con la capa MAC de nivel 2, la cual es suficiente para el algoritmo WNRA. Basándonos en el apartado 5.2.2, donde se muestra la utilización de los packet sockets como raw ethernet sockets, se ha procedido a la implementación de los mismos en el código del algoritmo.

No se ha querido reestructurar por completo el algoritmo y se mantenido la estructura general del algoritmo. En concreto se han modificado solo dos funciones del código: la función de envío sendMsg() y la función de recepción recvMsg(). Se ha considerado oportuno dejar una versión del código **wnrp_mejora1.c** con esta mejora implementada para posteriores modificaciones que se puede encontrar en [30].

Para ver más claramente la mejora, se han realizado una serie de capturas de los mensajes que envía el algoritmo para ver en detalle los campos del mensaje enviado.

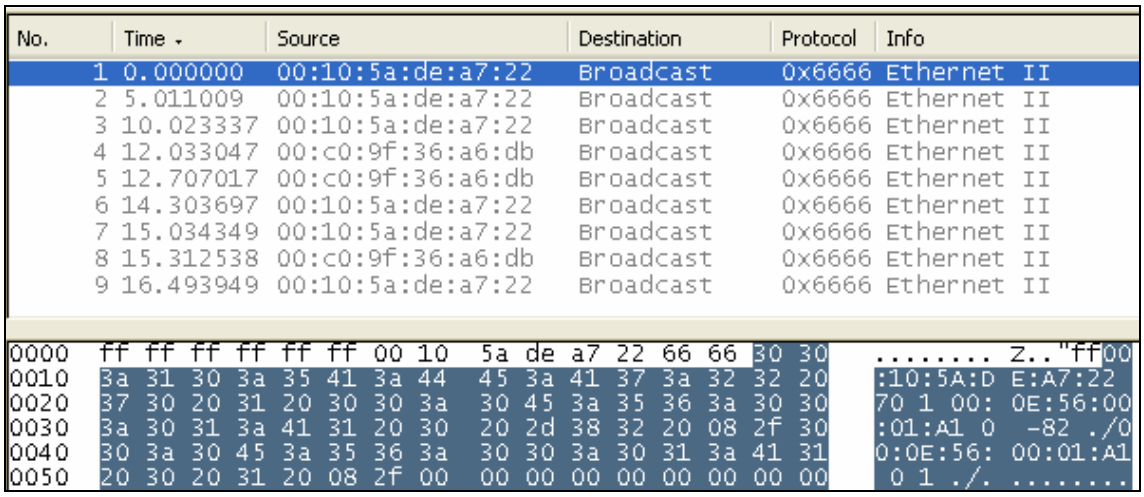

Fig. 6.1 Captura Ethereal del Mensaje 1 de la secuencia

Se puede observar como la mejora elimina las cabeceras de las capas UDP e IP, dejando solamente los campos del protocolo Ethernet. En la **Tabla 6.1** se muestran los siguientes campos:

Tabla 6.1 Campos del mensaje 1 de la secuencia

MAC Destino	FF : FF : FF : FF : FF : FF
MAC Origen	00 : 10 : 5A : DE : A7 : 22
Tipo Protocolo	0x6666 (inventado por nosotros)
Datos	00:10:5A:DE:A7:22 70 1 00:0E:56:00:01:A1 0 –82 / 00:0E:56:00:01:A1 0 1
Longitud	6+6+2+70+4 = 88 bytes

El mensaje nos dice que el Nodo1 *00:10:5A:DE:A7:22* tiene una utilización de un 70%, que opera en el canal 1 y que tiene un enlace con el Nodo2 *00:0E:56:00:01:A1* del cual recibe -82 dBm de potencia. Por otro lado, el Nodo1 sabe que el Nodo2 opera en el canal 1 pero no sabe su utilización ni tampoco si este tiene enlaces con otros nodos.

Pero lo importante aquí no es explicar el funcionamiento del algoritmo sino observar la reducción de tamaño que ha experimentado el paquete respecto el algoritmo original. Vamos a ver en la **Tabla 6.2** la comparativa con el mensaje original:

Tabla 6.2 Comparativa entre tamaños de paquete (Mejora 1)

	Longitud con Algoritmo Original	**Lontigud con Algoritmo Mejora 1 (raw ethernet sockets)**
Mensaje 1 de la secuencia	Cabecera UDP (8 bytes) + Cabecera IP (20 bytes) + Cabecera Ethernet (14 bytes) + Campo de datos (70 bytes) + FCS (4 bytes) **TOTAL = 116 bytes**	Cabecera Ethernet (14 bytes) + Campo de datos (70 bytes) + FCS (4 bytes) **TOTAL = 88 bytes**
Reducción tamaño = 116 bytes – 88 bytes = 28 bytes **% Reducción Mejora 1** = (28 / 116) * 100 = **24,1 %**		

Como se puede observar, eliminando las capas superiores a Ethernet se ha conseguido reducir un 24% el tamaño del paquete que envía el algoritmo WNRA. Ésta es solo la primera mejora para su reducción ya que en el siguiente paso se pretende reducir considerablemente el tamaño, hasta más de un 50%.

6.1.2 Mejora 2: Optimización del campo de datos del mensaje

Una vez realizada la primera mejora del algoritmo, esta mejora pretende reducir aún más el tamaño del paquete que se envía por la red. Para ello se cogerá la cadena de caracteres que se envía del campo de datos y se procesará de forma inteligente para codificarla a bytes de información. En este paso se ha requerido diseñar una estructura de los datos que viajan junto con el mensaje para que se envíe la misma información en el mínimo número de bytes posibles. La estructura diseñada es la que se muestra en la **Fig. 6.2**:

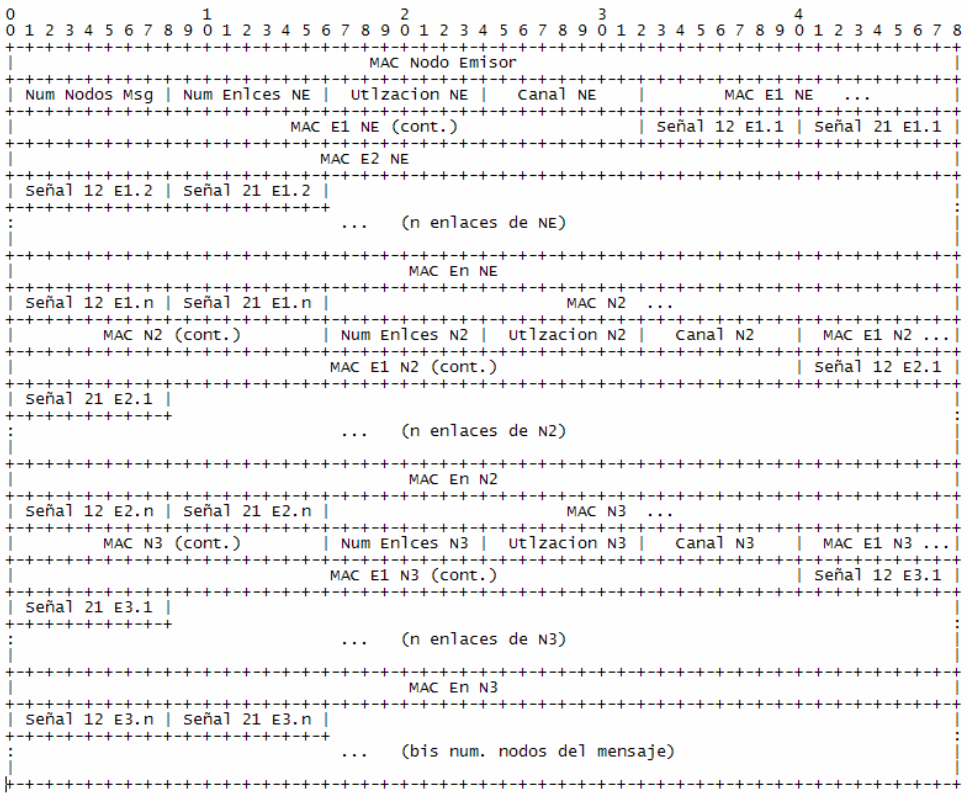

Fig. 6.2 Estructura de datos diseñada

El objetivo es implementar la estructura de mensajes diseñada en el algoritmo WNRA mejorado. Para ello se han de modificar otra vez las funciones de envío sendMsg() y de recepción recvMsg() del algoritmo.

Esta vez la estructura de mensaje se ha implementado en el archivo de cabecera **stringw.h** el cual almacena la función *converter( )* que convierte la cadena de caracteres del campo de datos en la estructura de datos que se ha diseñado (ver **Fig. 6.2**). Por otro lado, el archivo **stringreceptor.h** almacena la función *deconverter( )* que decodifica el mensaje en bytes a la cadena de carácteres original. Se ha decidido separar las funciones en ficheros diferentes para facilitar el proceso de desarrollo. Todos los archivos que se referencian se pueden encontrar en [30].

A continuación se presenta una captura (**Fig. 6.3**) para observar la correcta codificación del mensaje 1 de la **Fig. 6.1**.

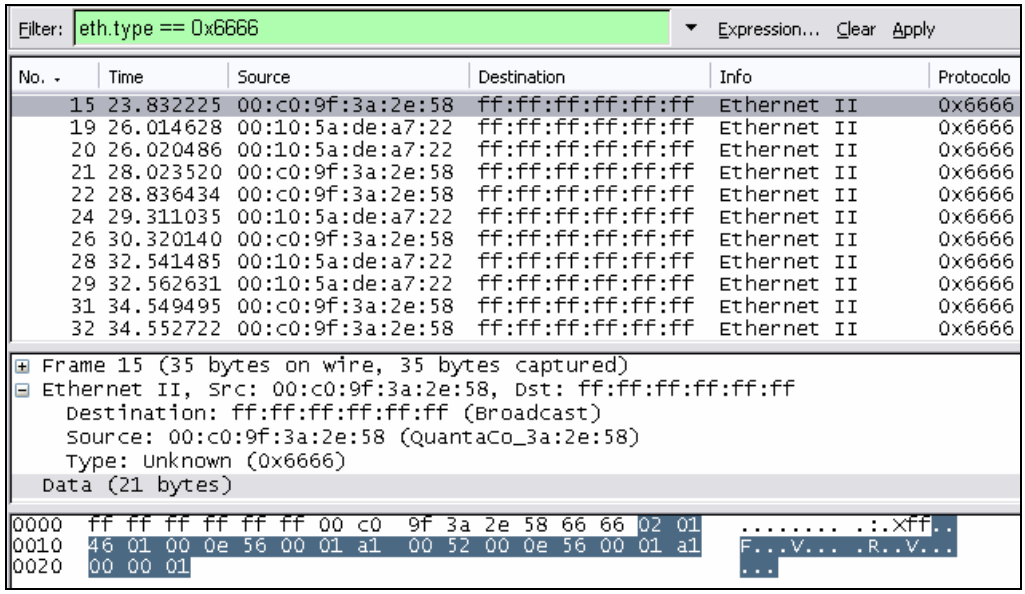

Fig. 6.3 Captura del campo de datos del mensaje 1 de la secuencia

Como se puede observar en la **Fig. 6.3**, el mensaje 1 de la secuencia presentada en el apartado anterior se ha codificado acorde con la estructura diseñada y expuesta en la **Fig. 6.2**. A continuación, se decodificarán los bytes según la estructura de datos diseñada para compararlos con la cadena de caracteres original.

Cadena Original: 00:10:5A:DE:A7:22 70 1 00:0E:56:00:01:A1 0 –82 / 00:0E:56:00:01:A1 0 1

Significado: El Nodo1 *00:10:5A:DE:A7:22* tiene una utilización de un 70%, que opera en el canal 1 y que tiene un enlace con el Nodo2 *00:0E:56:00:01:A1* del cual recibe -82 dBm de potencia. Por otro lado, el Nodo1 sabe que el Nodo2 opera en el canal 1 pero no sabe su utilización ni tampoco si este tiene enlaces con otros nodos.

Tabla 6.3 Decodificación del mensaje 1 de la **Fig. 6.1**

Numero de byte	Campo	Hexadecimal	Valor
15	Número de nodos del mensaje	0x02	2
16	Número de enlaces del emisor	0x01	1
17	Utilización emisor	0x46	70
18	Canal	0x01	1
19 - 24	MAC enlace1 del emisor	0x00,0x0e,0x56,0x00,0x01,0xa1	00:0E:56:00:01:A1[2]
25	Señal 1 2	0x00	0
26	Señal 2 1	0x52	- 82[3]

[2] Transformado de hexadecimal a cadena de caracteres con el carácter " : " entre valores como en el mensaje original.

27 - 33	MAC Nodo2	0x00,0x0e,0x56,0x00,0x01,0xa1	00:0E:56:00:01:A1
34	Utilización Nodo2	0x00	0
35	Canal Nodo2	0x01	1

Una vez decodificados los bytes se observa que los datos coinciden con el mensaje original y están acorde con la estructura de datos diseñada según la **Fig. 6.2**. Lo cual nos permite afirmar que la codificación se realiza correctamente.

Una vez hemos conseguido enviar y recibir correctamente los mensajes en formato de bytes según la estructura diseñada, vamos a ver cuanto se ha ganado en eficiencia. Para ello realizaremos en la **Tabla 6.4** el cálculo del % de reducción de igual forma que en el apartado anterior.

Tabla 6.4 Comparativa entre tamaños de paquete (Mejoras 1-2)

	Longitud con Algoritmo Original	**Lontigud con Algoritmo Mejora 1 (raw ethernet sockets)**	**Lontigud con Algoritmo Mejora 2 (optimización del campo de datos)**
Mensaje 1 de la secuencia	Cabecera UDP (8 bytes) + Cabecera IP (20 bytes) + Cabecera Ethernet (14 bytes) + Campo de Datos (70 bytes) + FCS (4 bytes) **TOTAL = 116 bytes**	Cabecera Ethernet (14 bytes) + Campo de datos (70 bytes) + FCS (4 bytes) **TOTAL = 88 bytes**	Cabecera Ethernet (14 bytes) + Campo de datos (27 bytes) + FCS (4 bytes) **TOTAL = 41 bytes**
Reducción tamaño = 116 bytes - 41 bytes = 75 bytes **% Reducción Mejoras 1-2** = (75 / 116) * 100 = **65 %**			

Como se muestra en la **Tabla 6.4**, la mejora 2 consigue reducir hasta un 65% el tamaño del mensaje de información que envía el algoritmo. Esta reducción de tamaño mejora mucho la eficiencia del algoritmo y permite que en escenarios futuros donde la cantidad de APs sea elevada, el algoritmo sea ligero y la información que viaje por la red sea lo más óptima posible.

El código de esta mejora es **wnrp_mejora2.c** y se puede encontrar en [30].

Cabe comentar que todas las pruebas se han realizado en un escenario estático, es decir, a partir de un archivo *(scan1)* con el resultado de un scan

[3] El valor es enviado en valor positivo. Al decodificarlo se le añade el signo negativo como en la cadena de caracteres original

fictício del medio, en lugar de obtener información a partir de las redes disponibles realmente. Ver TFC anterior [14] para el funcionamiento escenarios con scans dinámicos.

6.2 Capa 3 – Mecanismo de reparto de carga mediante plugin OLSR

Para este proyecto se ha desarrollado un plugin que funciona sobre la estructura básica de OLSR y que utiliza para ello un nuevo tipo de mensaje (mensaje tipo 130), que coexistirá con los mensajes por defecto de OLSR. Se trata de un mecanismo de reparto de carga entre celdas vecinas conocido como *Cell Breathing*. Dicha técnica consiste en disminuir o aumentar el radio de una celda, dependiendo de su utilización. En redes WLAN 802.11 son los dispositivos cliente los que deciden la asociación y el traspaso basándose únicamente en nivel de señal. Si una celda sobre-utilizada reduce su cobertura (*breath out*), parte de sus clientes recibirán mejor señal de otras celdas vecinas, liberando al AP congestionado de parte de su carga. A su vez, una celda con poca utilización, puede aumentar su cobertura (*breath in*), atrayendo así a clientes lejanos, de manera que se puede ofrecer servicio a los clientes excluidos de una celda sobre-cargada. Al reducir una celda se consigue disminuir su utilización además de reducir la probabilidad de que nuevos clientes demanden servicio, y así vayan a celdas menos ocupadas.

6.2.1 Funcionamiento plugin OLSR

La idea del algoritmo es que cuando un AP está sobrecargado, es decir, su utilización supera cierto umbral, pide ayuda a los APs de las celdas adyacentes a través de un mensaje SOS. Si alguno de los nodos vecinos tiene una utilización baja, podrá prestar su ayuda y contestará con un mensaje RELIEF. Cuando un nodo deja de estar sobrecargado, enviará un mensaje DISCHARGE a sus nodos vecinos para así volver todos al estado normal.

El funcionamiento concreto del plugin se puede ver representado en el diagrama de actividad de la **Fig. 6.4.** La utilización de un punto de acceso representa el número de clientes que está soportando combinado con la carga que estos suponen para el AP. Esta variable la conseguimos a partir de un script de un proyecto anterior [28]. Su valor puede oscilar, por lo que para la implementación del plugin se han fijado dos umbrales (LLINDAR_MIN_UTILITZACIO y LLINDAR_MAX_UTILITZACIO) con el fin de proporcionar cierta histéresis y evitar así un indeseado efecto ping-pong.

Tabla 6.5. Resumen funcionamiento plugin OLSR

ESTADOS	MENSAJES	ACCIONES	VARIABLES A GUARDAR
ST_DEFAULT	SOS	Breath_out	Utilización
ST_SOS	RELIEF	Breath_in	Estado
ST_ALTRUIST	DISCHARGE	Breath_default	Timer (Normal ó Infinito)
			@ IP del nodo que ayudas

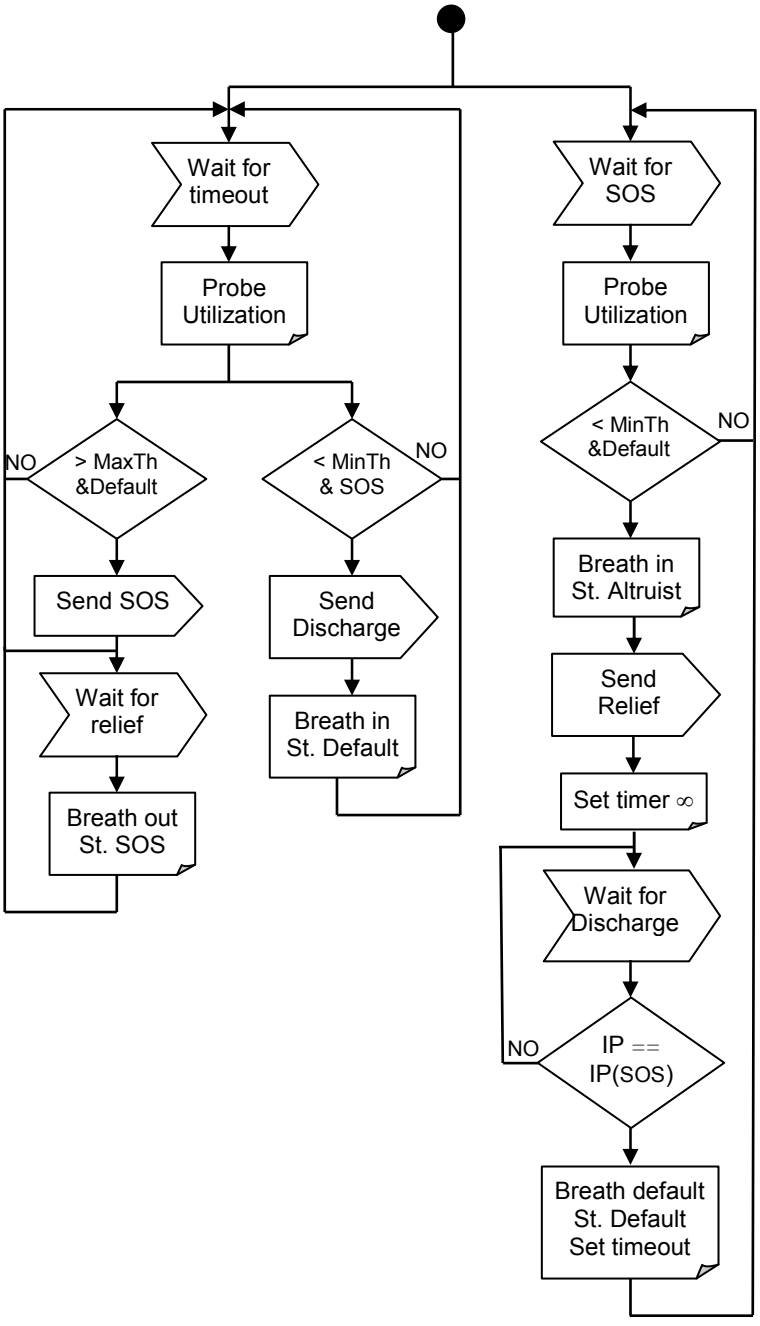

Fig. 6.4 Diagrama de actividad del plugin OLSR

Un nodo solo podrá ayudar a un único vecino. En cambio, un nodo puede ser ayudado por tantos nodos puedan ayudar y estén situados a un salto de distancia como máximo.

6.2.2 Desarrollo del plugin

La aplicación se ha desarrollado en código C a partir del ejemplo *"powerstatus plugin"* [27], de donde se han extraído las funciones básicas para implementar un plugin. Este plugin es parte de la distribución OLSR de Unik, y se encuentra dentro de la carpeta */lib/powerinfo.*

Allí podemos encontrar una serie de ficheros .c y las correspondientes cabeceras .h. A continuación se explican los principales parámetros a modificar de cada fichero para implementar un plugin:

- ***olsr_plugin_io.h***

  Este fichero contiene todas las definiciones de los comandos para interactuar con el demonio olsrd desde el plugin. No es necesario modificar este fichero para crear un nuevo plugin.

- ***olsrd_plugin.c***

  Este fichero contiene las funciones para inicializar el plugin y algunas funciones básicas para su funcionamiento. También se recomienda no modificar este fichero.

- ***olsrd_plugin.h***

  Este fichero contiene todas las definiciones de variables y estructuras necesarias para crear el plugin. Este fichero sí que se debe modificar según el plugin a crear. En este caso, se han definido los umbrales de utilización, los estados de los nodos, los tipos de mensajes a enviar y los valores del timer.

  En este fichero se define el tipo de mensaje que se enviará mediante este plugin; en este caso mensajes OLSR tipo 130. Otros datos a modificar son los que componen el nombre del plugin, la versión de éste y el nombre del creador del plugin.

  También, en este fichero se crea la estructura de los mensajes a enviar. En este caso, aparte de la cabecera básica de OLSR (**Fig. 6.5**), se añade un campo más (**Fig. 6.6**) compuesto de 1 byte que indicará el tipo de mensaje que se envían los nodos {SOS, RELIEF, DISCHARGE}. No será necesario enviar nada más. El campo TTL siempre se fijará a 1 porque no tiene sentido que el mensaje viaje más allá de los nodos vecinos.

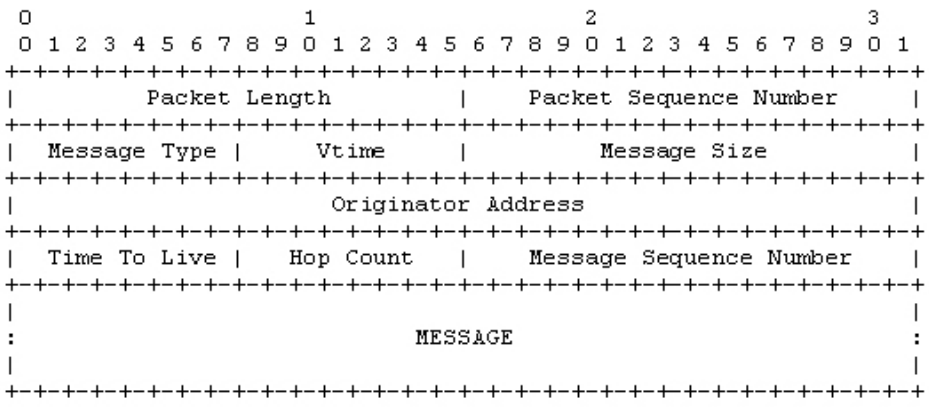

Fig. 6.5 Estructura mensaje OLSR

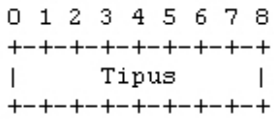

Fig. 6.6 Estructura del campo “MESSAGE” propio del plugin

- ***olsrd_power.h***

  En este fichero se encuentran y se deberán crear todas las definiciones de las funciones existentes en *olsrd_power.c.* Aparte de esto, aquí se define el EMISSION_INTERVAL que se comenta más adelante, y que en este caso se ha fijado a 5 segundos.

- ***olsrd_power.c***

  Este fichero contiene la implementación del plugin. A continuación se comenta a grandes rasgos la estructura de este fichero y su funcionamiento.

La función *olsr_plugin_init()* es la primera que se llama al cargar el plugin y es la que organiza todo el funcionamiento del programa. Dentro de ella se llaman a estas tres funciones:

```
/* Register functions with olsrd */

olsr_parser_add_function(&olsr_parser, PARSER_TYPE, 1);

olsr_register_timeout_function(&olsr_timeout);

olsr_register_scheduler_event(&olsr_event, NULL, EMISSION_INTERVAL, 0, NULL);
```

En la primera se registran todas las funciones que se deben ejecutar cuando llega un mensaje del tipo definido en el plugin. Es decir, cada vez que llegue un mensaje OLSR de tipo 130, se ejecutará la función *olsr_parser* para tratar el nuevo mensaje.

La segunda registra una función en el *olsrd scheduler.* Por tanto *olsr_timeout* se ejecutará cada vez que el *scheduler* haga un "polling". Esto significa diez veces cada segundo. Esta función no se ha modificado, ya que se utiliza para actualizar la información de los nodos.

Por último, la tercera función registra las funciones que se van a ejecutar cada "EMISSION_INTERVAL" segundos. Así pues, *olsr_event* se ejecutará cada 5 segundos en nuestro caso para comprobar el estado de la utilización. Dentro de esta última función también se construyen los mensajes y se envían a los nodos vecinos.

Una vez construido el plugin, se ha de compilar para crear la librería dinámica. Para ello, editaremos el *Makefile*, donde se puede cambiar entre otras cosas el compilador y los flags a utilizar. Se ha dejado todo por defecto menos el nombre del plugin que se ha cambiado por *olsrd_llindar.so.0.3*.

A continuación se debe crear la librería, copiarla a uno de los directorios por defecto para cargar librerías en Linux y actualizar los registros de librerías dinámicas con los siguientes comandos:

```
# make OS=linux

# cp olsrd_llindar.so.0.3 /usr/lib

# ldconfig
```

Por último se debe editar el fichero */etc/olsrd.conf* y añadir la siguiente línea para que pueda cargar el plugin:

```
LoadPlugin "olsrd_llindar.so.0.3"
{
}
```

Y ya se puede arrancar el demonio *olsrd* indicándole la interfaz donde se quiere arrancar: *olsrd -i wlan0* por ejemplo.

6.2.3 Pruebas del plugin

En este apartado se explica el escenario de pruebas realizado para comprobar el correcto funcionamiento del plugin, cuyo código se puede encontrar en el CD anexo [30]. El escenario de pruebas es el que se presenta en la **Fig. 6.7**. Está compuesto por 3 meshcubes y por 3 portátiles que realizarán el rol de clientes.

Cada meshcube dispone de dos interfaces inalámbricas: 1 se configura como AP para ofrecer cobertura a los clientes, en este caso se identifica el ESSID como *hostap*; la otra interfaz se configura en modo Ad-Hoc para que se pueda comunicar con los otros meshcubes y se puedan enviar información de señalización. Estas interfaces Ad-Hoc formarán la red mesh. En esta última

interfaz es donde se arrancará *olsrd* y por donde se enviarán los mensajes propios del plugin y los de OLSR. En cada meshcube se estará ejecutando continuamente el script realizado para cuantificar en cada momento la utilización existente en la interfaz que realiza las funciones de AP. Por último, en cada meshcube se arrancará un servidor *iperf* [4] para que los clientes puedan generar tráfico contra los puntos de acceso y así poder trabajar con valores de utilización reales.

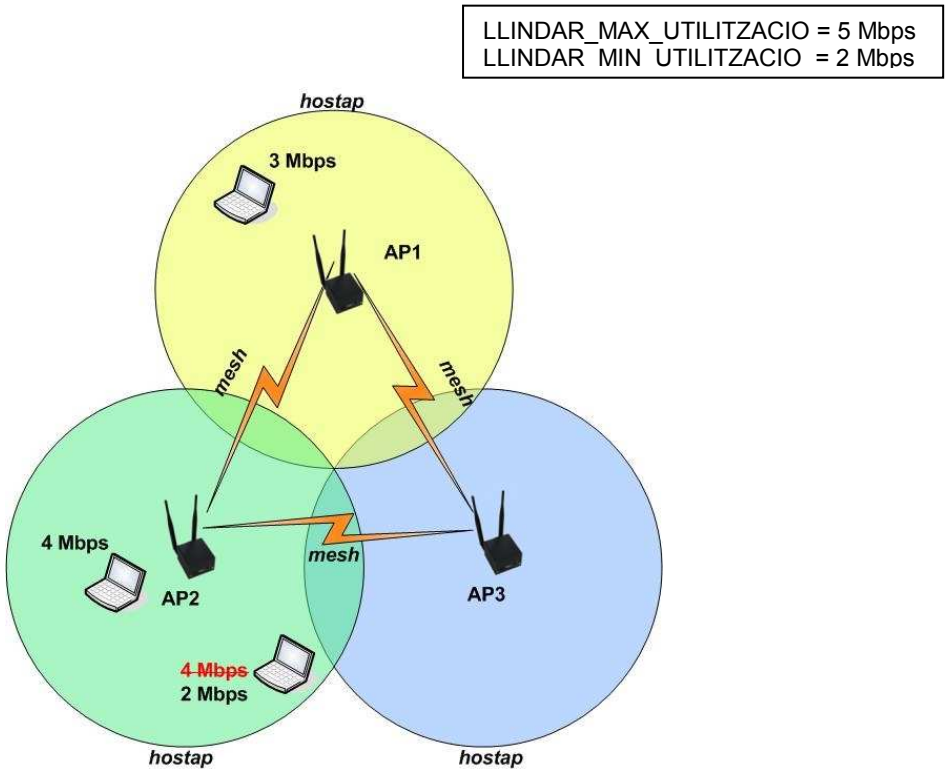

Fig. 6.7 Escenario de pruebas del plugin (Estado Default)

Cada portátil realizará la función de cliente. Para ello se ejecutará el generador de tráfico *iperf* como cliente dirigiendo el tráfico al AP al cuál están asociados según la **Fig. 6.7** y con las tasas de transmisión que se contemplan. Para configurar un cliente, únicamente se debe especificar a qué ESSID se debe conectar y él sólo se asociará a un AP, normalmente al AP del cuál recibe mayor potencia. Para que el cliente estuviera correctamente configurado durante toda la prueba, se ha implementado un script que según el AP al cuál se asocie, se le asigna al cliente una dirección IP dentro del rango del AP y la dirección de la puerta de enlace correspondiente para que no tenga problemas de conectividad.

En la **Fig. 6.7** se puede observar que según los umbrales definidos, el AP2 estaba sobrecargado, por lo que se lo hizo saber a sus vecinos mediante un mensaje SOS tal y como se puede ver en la secuencia de mensajes de la **Fig. 6.8**. El único AP que le podía ayudar era el AP3, por lo que fue el único que le envió un mensaje RELIEF, tal y como se puede ver en la captura de Ethereal de la **Fig. 6.9**. En el mensaje seleccionado se puede ver como el tipo de mensaje es el 130 y en el campo de datos se envía un 02, lo que corresponde

al mensaje RELIEF. Para que el mensaje ocupe el mínimo tamaño de paquete establecido, el propio MAC realiza un *zero-padding*.

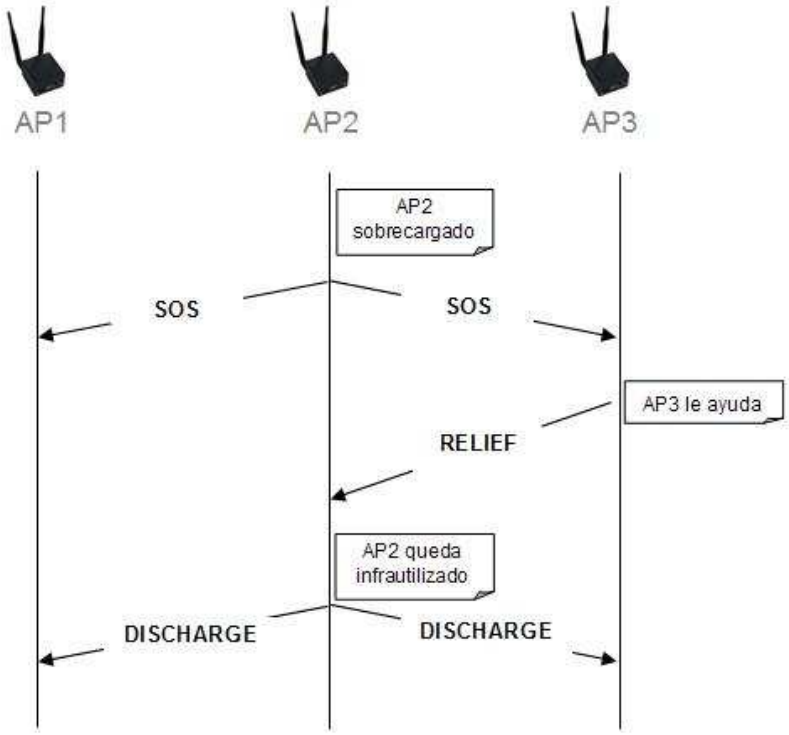

Fig. 6.8 Secuencia de mensajes del plugin OLSR

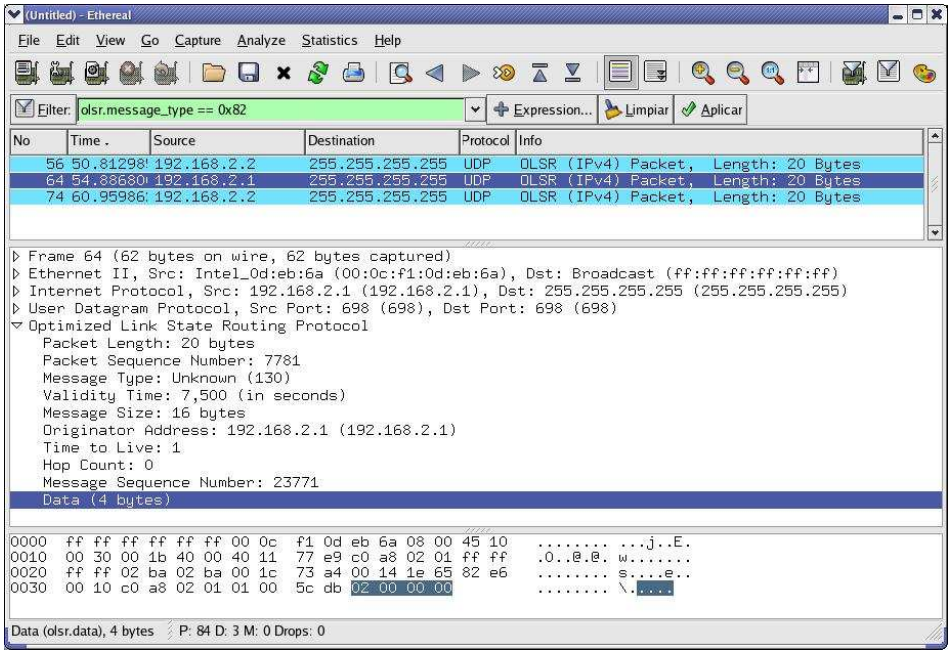

Fig. 6.9 Captura de mensaje RELIEF

AP2 tenía asociados dos clientes que intentaban transmitir un flujo de 4 Mbps cada uno. Por limitaciones del propio 802.11, la celda de AP2 sólo tenía

capacidad para 6 Mbps, por lo que se pudo observar que el nodo más lejano sólo alcanzaba una tasa de 2 Mbps.

En el momento que AP2 recibe el mensaje RELIEF, reduce la potencia transmitida y su sensibilidad mientras que AP3 realiza justamente lo contrario, tal y como se puede ver en la **Fig. 6.10**. De esta manera el cliente más lejos de AP2 recibe mayor potencia de AP3 y se asocia a él realizando el cambio de direcciones IP como se ha comentado anteriormente.

El flujo del cliente que ha realizado el traspaso está destinado a AP2, por lo que ahora el flujo pasará a través de AP3 y de la red mesh para llegar a AP2. De esta manera podemos ver que hay una mejora en todo el escenario ya que este último cliente consigue transmitir a 4 Mbps.

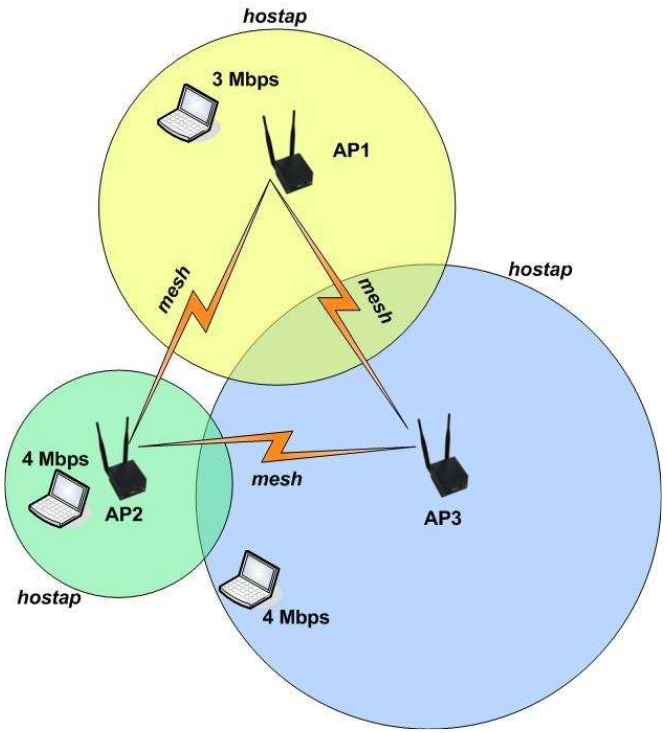

Fig. 6.10 Escenario de pruebas del plugin (Estado SOS)

En el momento que el cliente asociado a AP2 bajó su utilización, se comprobó que AP2 envió correctamente el mensaje DISCHARGE y tanto AP2 como AP3 volvieron a estado normal.

Un inconveniente que se ha encontrado es que en el momento que un cliente realiza el traspaso de AP con el consecuente cambio de IPs, todas las conexiones abiertas se pierden, por lo que se tuvo que volver a arrancar el *iperf* en el cliente. La solución de este hecho no entra dentro de los objetivos de este proyecto y se podría solucionar con algún protocolo que permitiera la movilidad de nodos a nivel 3, como por ejemplo Mobile IP [31].

También se ha observado que la elección de los umbrales y la carga que ofrece cada cliente es un hecho clave para el correcto funcionamiento de este plugin, pudiendo llegar a ser contraproducente.

Imaginemos el caso que en AP2 hubieran dos clientes. El más cercano con un caudal de 1 Mbps y el más alejado con uno de 6 Mbps. AP2 pediría ayuda por estar sobrecargado, por lo que AP3 acogería el de 6 Mbps. En este momento AP2 quedaría infrautilizado y AP3 estaría sobrecargado. De esta manera se produciría continuamente un efecto “ping-pong” con el nodo de 6 Mbps y tanto AP2, como AP3 y tal nodo entrarían en un bucle donde el nodo sufriría desconexiones continuas. Este problema se solucionaría combinando el mecanismo de *Cell Breathing*, con otras técnicas de reparto de carga como el mecanismo de expulsión de clientes diseñado en un proyecto anterior [32], con el que se decide de manera eficiente qué cliente expulsar para que el AP no esté sobrecargado pero que tampoco se quede infrautilizado.

CAPÍTULO 7. CONCLUSIONES

Este proyecto ha seguido las líneas marcadas por proyectos anteriores con el fin de dar respuesta a la problemática de las redes inalámbricas IEEE 802.11 en cuanto a la limitación de los recursos disponibles que padecen. Estos recursos deben ser gestionados de forma inteligente para poder ofrecer un buen servicio a los usuarios finales.

Para extraer unas conclusiones del trabajo que se ha desarrollado en este PFC sería conveniente recordar los objetivos iniciales. Estos se dividían en dos grandes bloques.

El primer bloque nos ha ocupado buena parte del tiempo de este proyecto debido a que la preparación de las pruebas llevó consigo un minucioso proceso de familiarización con componentes del escenario, como por ejemplo los APs meshcube. Esta primera parte tenia como objetivo analizar el impacto de la gestión de los recursos radio sobre el rendimiento de la red, además de evaluar el comportamiento del hardware utilizado, realizando escenarios en entornos reales de funcionamiento y comparando posteriormente los resultados obtenidos con simulaciones de las mismas pruebas en dos simuladores de redes, NS-2 y OPNET.

Como se ha visto en la memoria, estos objetivos han sido alcanzados de forma satisfactoria aun sin olvidar los problemas a los cuales hemos tenido que hacer frente. La realización de las pruebas en escenarios en espacio libre como la playa de Castelldefels, tuvo que ser repetida debido en gran parte a la humedad existente en el terreno, factor que degradaba la señal y provocaba resultados indeseados. Estas pruebas se volvieron a repetir en el paseo de Castelldefels donde las condiciones fueron mejores y los resultados se semejaban a los esperados.

En cuanto a las pruebas realizadas en un entorno de interior como es el laboratorio, los resultados nos mostraron un comportamiento irregular de los APs. Por este motivo, las pruebas se repitieron varias veces para verificar que los resultados obtenidos no se ajustaban a los esperados cosa que nos ocupó un mayor tiempo del que en un principio se había planteado.

Una vez realizadas las pruebas en entornos reales se simularon los escenarios en los simuladores de red NS2 y OPNET con el fin de comparar los resultados obtenidos. Los resultados del simulador de redes NS-2 fueron bastante parecidos a los de las pruebas reales. Con esta conclusión podemos afirmar que los meshcubes se ajustan al estándar en entornos simples de funcionamiento. También cabe decir, que NS-2 fue diseñado para simular redes cableadas, por ello muchas de las funcionalidades de las redes inalámbricas no están presentes en la versión estándar del simulador.

En cuanto al simulador OPNET se refiere, el hecho que se necesite tener licencia para utilizar esta herramienta junto con la particularidad de que sólo se

contaba con un modelo de propagación, el DLC, no nos han permitido realizar una buena comparación de los resultados. Aun así se ha aprendido el funcionamiento del simulador, documentando todos los pasos realizados. Aprender el funcionamiento de este simulador con redes inalámbricas fue complicado debido a la poca información existente al no ser un simulador de código abierto. Los resultados obtenidos se compararon con los de NS-2 y se pudo evaluar el funcionamiento de los dos simuladores y sus características.

Cabe comentar, que la realización de manuales de los simuladores NS2 y OPNET expuestos en el anexo y publicados en Internet fue encaminada a facilitar el aprendizaje en el uso de estas herramientas en futuros proyectos.

Con las pruebas en entornos reales realizadas y los resultados comparados con las simulaciones podemos concluir que los objetivos relacionados con el primer bloque fueron alcanzados y cumplidos de forma satisfactoria.

El segundo bloque ha ocupado la recta final del proyecto y nos ha permitido desarrollar dos soluciones para la problemática de la limitación de recursos en las redes 802.11. En concreto, siguiendo las pautas de proyectos anteriores se han desarrollado dos sistemas que permiten el intercambio de información entre los APs de una red mesh, de manera que se permite la aplicación de diferentes mecanismos distribuidos para la gestión de recursos.

En una primera fase se realizaron los capítulos referidos a la discusión de las diferentes soluciones de implementación y a sus ventajas y desventajas, como también el análisis funcional de las soluciones desarrolladas en los proyectos anteriores. Una vez nos decidimos por un mecanismo respecto a los demás se pasó a la fase de implementación.

La primera solución consistió en la realización del algoritmo WNRA, desarrollado en el proyecto anterior a éste y donde se propuso como líneas futuras la realización de mejoras en el algoritmo, a fin de obtener un sistema más eficiente. Las mejoras se debían realizar en el transporte de información entre los nodos que utilizaban dicho algoritmo.

La primera mejora ha sido la de sustituir los sockets UDP convencionales que incluyen cabeceras de protocolos que no se utilizan, por raw ethernet sockets. De esta forma es posible efectuar un transporte de la información directamente sobre la capa de enlace de 802.11 mejorando notablemente la eficiencia del algoritmo. A partir del ejemplo práctico de raw sockets del cual se disponía, la implementación de esta mejora ha sido más plácida de lo que se creía en un principio debido a la poca documentación que existe sobre raw sockets.

La segunda mejora que se ha introducido ha sido más laboriosa. Su desarrollo permite al algoritmo optimizar la información a enviar, de forma que se transmitan el mínimo número de bytes posibles. Para ello se ha diseñado una nueva estructura de datos. Desarrollar esta mejora ha ocupado la mayor parte del tiempo en la recta final del proyecto. Debido a esto no ha quedado tiempo disponible para probar el algoritmo en escenarios más complejos lo cual debe ser una línea futura en la que trabajar.

Esta mejora junto a la anterior permiten al algoritmo ser mucho más ligero en el paso de información, funcionalidad que añade escalabilidad y que permitirá el correcto funcionamiento en una gran infraestructura Wi-Fi donde el número de nodos que dan servicio sea elevado.

La segunda solución ha ido encaminada a la implementación de un plugin para que funcionara sobre OLSR y realizara un mecanismo de reparto de carga para mejorar la eficiencia de la red en conjunto. Este desarrollo no supuso demasiados problemas debido a que se contaba con una mayor documentación. Para ir probando el funcionamiento del plugin durante su desarrollo fue necesario el uso de ordenadores con tarjetas PCMCIA con el chipset Prism2.5 para que funcionaran sobre el driver hostap, lo que hizo necesario la recompilación del Kernel y configuración de interrupciones IRQ.

También es necesario comentar que dentro de la segunda solución fue necesario el estudio de la viabilidad de utilizar una infraestructura IPv6. Finalmente, esta opción fue desestimada por la falta de madurez en la implementación de mecanismos de *tunneling*. En concreto, se intentó establecer un túnel "IPv4 over IPv6" sin llegar a buen fin debido a la poca documentación existente. Se encontró un proyecto que implementaba este tipo de túnel que se puede encontrar en [51] pero no se pudo probar por falta de tiempo.

Con las implementaciones de las soluciones realizadas y a partir de los escenarios probados se puede concluir que el funcionamiento de los mecanismos diseñados cumple con su propósito y así se cubren los objetivos de este segundo bloque.

Este proyecto, aparte de ser continuación de otros abre las puertas a nuevas implementaciones y líneas a seguir en un futuro.

Para empezar, en la parte de simulación se deberían realizar algunos cambios para que el comportamiento de los simuladores se asemeje más al observado en escenarios reales. Para ello, en NS-2 se debería incorporar el módulo comentado en el capítulo 2 para poder simular las interferencias entre canales adyacentes. En lo que respecta a OPNET Modeler, se debería adquirir la licencia TMM, o en su defecto, implementar en el propio código del simulador un modelo de propagación más semejante al de las pruebas reales, con lo que resultará más económico que la primera opción.
En cuanto al segundo bloque de la memoria, sería interesante probar las dos implementaciones en escenarios reales a gran escala, es decir, con gran cantidad de APs y clientes asociados. En la parte de la solución a nivel 3, sería conveniente combinar el mecanismo implementado con otros sistemas diseñados en proyectos anteriores para así ofrecer mayor eficacia a tal mecanismo.

Se puede decir que este proyecto no introduce consecuencias de carácter ético ya que las redes inalámbricas en los últimos años se han convertido en una tecnología muy común y no despiertan ningún aspecto ético negativo. Además,

se puede afirmar que provoca un mínimo impacto medioambiental. Estamos utilizando tecnología inalámbrica que cumple con todas las normas de medio ambiente. La banda utilizada es un rango de frecuencias libres (2,4 GHz), por lo que cumple con la normativa IEEE 802.11 y con las leyes vigentes (Ley General de Telecomunicaciones). Además, la potencia a la que transmitimos para realizar las pruebas está dentro del margen de las normativas estatales y autonómicas.

Los ordenadores utilizados no son de última tecnología, por lo que los estamos reaprovechando para este proyecto. En el caso que nos tuviéramos que deshacer de ellos, al ser equipos que contienen componentes electrónicos, no se pueden dejar en cualquier contenedor. Según la Generalitat, para deshacernos de este tipo de material deberíamos devolverlo al fabricante para que se hicieran cargo del equipo. En caso contrario, bastaría con llevarlo a un vertedero oficial de materias selectivas (el vertedero más cercano sería el de Castelldefels [33]).

BIBLIOGRAFÍA

[1] Meshcube.org. *The meshing community website.* [En línea] [Consulta: 21 de enero de 2006] Disponible en: <http://www.meshcube.org/>

[2] LinuxCommand.org. *Linux Programmer's Manual – IWCONFIG* [En línea] [Consulta: 21 de enero de 2006] Disponible en: <http://linuxcommand.org/man_pages/iwconfig8.html>

[3] T. S. Rappaport, *Wireless Communications Principles and Practices*. Prentice Hall PTR, second ed., December 2001.

[4] Debian. *Debian – IPERF. Package: iperf (2.0.2-1, 1.7.0-1)* [En línea] [Consulta: 21 de enero de 2006] Disponible en: <http://packages.debian.org/unstable/net/iperf>

[5] Netstumber.com. *Netstumbler* [En línea] [Consulta: 21 de enero de 2006] Disponible en: <http://www.netstumbler.com/>

[6] Malinen, J. *Host AP driver for Intersil Prism2/2.5/3, hostapd, and WPA Supplicant* [En línea] [Consulta: 21 de enero de 2006] Disponible en: <http://hostap.epitest.fi/>

[7] Wikipedia. *Carrier Sense Multiple Access with Collision Avoidance* [En línea] [Consulta: 21 de enero de 2006] Disponible en: <http://es.wikipedia.org/wiki/CSMA/CA>

[8] Ye, W. *Radio Propagation Models.* [En línea] [Consulta: 31 de enero de 2006] Disponible en: <http://www.isi.edu/~weiye/pub/>

[9] Kumar, A. & Roy, S. *The Enhanced Network Simulator TeNS*. Department of Computer Science & Engineering, Indian Institute of Technology, Kanpur, India. [En línea] [Consulta: 31 de enero de 2006] Disponible en: <http://www.cse.iitk.ac.in/~bhaskar/tens/>

[10] OPNET Technologies, Inc. [En línea] [Consulta: 21 noviembre 2005] Disponible en: < http://www.opnet.com >

[11] Flores, G.; Paredes, M.; Jammeh, E.; Fleury, M. y Reed, M. *OPNET Modeler and Ns-2: Comparing the accuracy of Network Simulators for Packet-Level Análisis using a Network Testbed.* University of Essex, United Kingdom. [En línea] [Consulta: 5 de febrero de 2006] Disponible en: <http://privatewww.essex.ac.uk/~fleum/weas.pdf>

[12] Fleury, M.; Flores, G. y Reed, M. *Clarification of the "OPNET NS-2 Comparison" Paper with regards to OPNET Modeler.* [En línea] [Consulta: 5 de febrero de 2006] Disponible en: <http://privatewww.essex.ac.uk/~fleum/OPNET-NS2_Comparison.pdf>

[13] T. Clausen, Ed., P. Jacquet, Ed., “Optimized Link State Routing Protocol (OLSR)”. IETF RFC 3626. October 2003 [En línea] [Consulta: 20 de febrero de 2006] Disponible en: <http://www.ietf.org/rfc/rfc3626.txt>

[14] García, V. *Redes mesh basadas en puntos de acceso inteligentes 802.11 open source (II)*. Trabajo Final de Carrera, EPSC (UPC). Departamento Ingeniería Telemática, Septiembre 2005. [Biblioteca Escola Politécnica Superior de Castelldefels]

[15] ThePacketFactory . *The PacketFactory* [En línea] [Consulta: 19 de enero de 2006] Disponible en:<http://www.packetfactory.net>

[16] Sourgeforce. *Tcpdump Public Repository.* [En línea] [Consulta: 15 de enero de 2006] Disponible en: <www.tcpdump.org>

[17] D. Schiffman,M. *The Libnet Reference Manual*. [En línea] [Consulta: 12 de enero de 2006] Disponible en: <www.packetfactory.net/libnet/dist/deprecated/manual/lrm.html>

[18] Grimes, M. *Nemesis 1.4beta3 (Build 22) released June 29, 2003* . [En línea] [Consulta: 19 de enero de 2006] Disponible en:<http://www.packetfactory.net/projects/nemesis/

[19] Song, Dug. *Libdnet.* Sourceforge [En línea] [Consulta: 25 de enero de 2006] Disponible en: <http://l.ibdnet.sourceforge.net/>

[20] Schaufler,A. *Chapter 1. RAW ethernet programming*. Linux Network Performance RAW ethernet vs. UDP.. [En línea] [Consulta: 28 de enero de 2006] Disponible en: <http://www.landshut.org/bnla01/members/Faustus/fh/linux/udp_vs_raw/index.html>

[21] MeshCube Wiki. *Proactive Protocol* [En línea] [Consulta: 20 de febrero de 2006] Disponible en: <http://www.meshcube.org/meshwiki/ProactiveProtocol>

[22] Tonnesen, A. *OLSR.ORG* [En línea] [Consulta: 20 de febrero de 2006] Disponible en: <http://www.olsr.org>

[23] IANA. *Internet Assigned Numbers Authority (IANA).* [En línea] [Consulta: 20 de febrero de 2006] Disponible en: <http://www.iana.org/>

[24] INRIA – Unité de Recherche de Rocquencourt. *OLSR – Demonstration of MPR flooding*. [En línea] [Consulta: 20 de febrero de 2006] Disponible en: <http://hipercom.inria.fr/olsr/mpr-flooding.html>

[25] The Linux Documentation Project. *Linux Optimized Link State Routing Protocol (OLSR) IPv6 HOWTO.* [En línea] [Consulta: 20 de febrero de 2006] Disponible en: <http://www.tldp.org/HOWTO/OLSR-IPv6-HOWTO/>

[26] Tonnesen, A.; Hafslund, A; Kure, A. *The Unik – OLSR plugin library.* [En línea] OLSR Interop and Workshop, 2004. [Consulta: 8 octubre 2005]. Disponible en: <http://www.olsr.org/docs/olsr_plugin_paper.pdf>

[27] Tonnesen, A. *Unik olsrd plugin implementation HOWTO.* [En línea] 2004 [Consulta: 8 octubre 2005]. Disponible en: <http://www.olsr.org/docs/olsrd-plugin-howto.html>

[28] Faixó, L. *Redes mesh basadas en puntos de acceso inteligentes 802.11 open source (I).* Proyecto Fin de Carrera, EPSC (UPC). Departamento Ingeniería Telemática, Septiembre 2005. [Biblioteca Escola Politécnica Superior de Castelldefels]

[29] Andi Kleen, Matthew Wilcox. "Packet", en *Linux Programmer's Manual*. 1999

[30] Oliveras, M y Ruiz, D. Redes mesh basadas en puntos de acceso inteligentes 802.11 open source (III). Proyecto Fin de Carrera, EPSC (UPC). Departamento Ingeniería Telemática, Febrero 2006. [CD-ROM] [Biblioteca Escola Politécnica Superior de Castelldefels]

[31] C. Perkins, Ed. "IP Mobility Support for IPv4" August 2004 (RFC 3344) [En línea] [Consulta: 21 febrero 2006]. Disponible en: <http://www.faqs.org/rfc/rfc3344.txt>

[32] Beltrán, A. *Puntos de acceso inteligentes basados en Linux (I).* Trabajo Fin de Carrera, EPSC (UPC). Departamento Ingeniería Telemática, Febrero 2005. [Biblioteca Escola Politécnica Superior de Castelldefels]

[33] Ajuntament de Castelldefels. *Web de Castelldefels – La Deixalleria de Castelldefels.* [En línea] [Consulta: 5 de febrero de 2006] Disponible en: <http://www.castelldefels.org/serveisurbansfitxa.asp?accid=10>

[34] Gris, M. *Tutorial for the UCB/LBNL/VINT Network Simulator "ns"* [En línea] [Consulta: 21 de diciembre de 2005] Disponible en: <http://www.isi.edu/nsnam/ns/tutorial/index.html>

[35] Harrison, D. *Xgraph* [En línea] [Consulta: 5 de enero de 2006] Disponible en: <http://www.isi.edu/nsnam/xgraph/>

[36] Turegano Molina, J. *LinuxAlbacete (Articulos) Simulador NS-2*. [En línea] [Consulta: 21 de enero de 2006] Disponible en:< http://linux.ideando.net/articulos/ns/ns.htm>

[37] Malek, J. *Tracegraph, Trace graph - network simulator ns2 trace files analyser* [En línea] [Consulta: 14 de enero de 2006] Disponible en: <http://www.tracegraph.com/>

[38] OPNET Technologies, Inc. [En línea] [Consulta: 21 noviembre 2005] Disponible en: < http://www.opnet.com >

[39] Área de Ingeniería Telemática, Universidad de Oviedo. *OPNET Modeler.* [En línea] [Consulta: 21 noviembre 2005] Disponible en: <http://www.it.uniovi.es/contenidos/opnet.html>

[40] Banús, J. *Primers passos amb OPNET IT Guru* [En línea] [Flash] [Consulta: 21 noviembre 2005] Disponible en: <http://www.etse.urv.es/EngInf/assig/xcii/itguru.html>

[41] Departament Enginyeria Telemàtica EPSEVG. *Academic OPNET Research and Educational Projects.* [En línea] [Consulta: 21 noviembre 2005] Disponible en: < http://www.candle.eu.org/opnet/ >

[42] Di Lorenzo, P. *OPNET mobility simulation models* [En línea] [Consulta: 21 noviembre 2005] Disponible en: <http://labreti.ing.uniroma1.it/wine/papers/mms.pdf >

[43] Ergen, M. UC Berkeley. *OPNET.* [En línea] [Consulta: 21 noviembre 2005] Disponible en: < http://www.eecs.berkeley.edu/~ergen/docs/OPNET.pdf >

[44] Griffiths, A. *OPNET Tutorials* [En línea] [Consulta: 21 noviembre 2005] Disponible en: <http://www.soc.staffs.ac.uk/alg1/2004_5/Semester_1/Network%20Systems%20&%20Technologies,%20NSTs1%20(CE00227-M)/NST/msc_tutorial.html>

[45] Katholieke Universiteit Leuven. *OPNET Starting Session* [En línea] [Consulta: 21 noviembre 2005] Disponible en: <http://homes.esat.kuleuven.be/~h239/opnetstart.htm>

[46] Katholieke Universiteit Leuven. *Wireless LAN: Simulation.* [En línea] [Consulta: 21 noviembre 2005] Disponible en: <http://homes.esat.kuleuven.be/~h239/reports/2001/wlan/simulation.php>

[47] Svigelj, A.; Bostic, J.; Mohorcic, M. y Kandus, G. *Wireless IP Simulation using OPNET Modeler.* Jozef Stefan Institute [En línea] [Consulta: 21 noviembre 2005] Disponible en: <http://www.opnet.com>, requiere autenticación.

[48] Milner, S; Chandrashekar, K; Thakkar, S; Chen, Ch. *Routing and Mobility Performance in Wireless Base-Station Networks*. Clark School of Engineering, University of Maryland [En línea] [Consulta: 17 enero 2006] Disponible en: <http://www.isr.umd.edu/CSHCN/people/faculty/milner/publications/opnetfinal.pdf>

[49] Chang, X. *Network Simulations with Opnet*. Network Technology Research Center School of EEE Nanyang Technological University SINGAPORE 639798 [En línea] [Consulta: 17 enero 2006] Disponible en: <www.informs-cs.org/wsc99papers/043.PDF>

[50] Chi-Kit Chow, J. *Performance of TCP Protocol Running over WLAN 802.11 with the Snoop Protocol*. [En línea] [Consulta: 17 enero 2006] Disponible en: <www.ensc.sfu.ca/~ljilja/ENSC833/Spring01/Projects/chow_ng/snoop_report.pdf >

[51] Medina, O. *DSTM –Dual Stack Transition Mechanism*. [En línea] [Consulta: 21 febrero 2006] Disponible en: <www.ipv6.rennes.enst-bretagne.fr/dstm/>

Escola Politècnica Superior de Castelldefels
UNIVERSITAT POLITÈCNICA DE CATALUNYA

ANEXOS

TÍTULO DEL PFC: Redes mesh basadas en puntos de acceso inteligentes 802.11 open source (III)

TITULACIÓN: Ingeniería de Telecomunicaciones (segundo ciclo)

AUTORES: Luis Daniel Ruiz López
Marc Oliveras Pla

DIRECTOR: Eduard García Villegas

CODIRECTOR: Rafael Vidal Ferré

FECHA: 22 de Febrero de 2006

ANEXO I. MANUAL NS-2

I.I. INTRODUCCIÓN

Ns-2 [34] es un simulador de redes que está escrito en el lenguaje C++.

La simulación se escribe en lenguaje Otcl, que es un lenguaje orientado a objetos de tipo intérprete, es decir, que las instrucciones del código se van traduciendo una a una conforme se van ejecutando, dándole una flexibilidad durante el desarrollo del código para hacer la simulación.

De esta forma los enlaces entre los nodos (links), son objetos OTcl que influyen y que se pueden programar situaciones como:

- Retrasos
- Gestión de colas
- Módulos de pérdidas
- Errores
- Etc.

Si se quiere modificar alguno de estos parámetros o incluir uno propio, se emplea C++.

El resto de las funciones se implementan en estos leguajes:

- El encaminamiento está casi todo implementado en OTcl.
- Los algoritmos de Dijkstra están implementados en C++.

NS puede trabaja con intérprete de comandos de OTcl, que es por donde se introducen individualmente los distintos comandos, este es un método lento y complicado. Por ello se utiliza el procedimiento de los programas, donde se escribe todo el código de la simulación a realizar y después se ejecuta.

Una vez hecha la simulación podemos realizar dos cosas para conocer los resultados:

- Ver el funcionamiento de la red simulada en forma de gráfico animado con un programa llamado NAM.

- También se puede analizar la carga de la red a través de un programa que da a conocer de forma gráfica la relación tiempo carga de la red a través de Xgraph [35].

I.II. ESTRUCTURA BÁSICA EN UN SCRIPT TCL

Para empezar un script en lenguaje Tcl (que es el que utiliza el usuario para que sea interpretado posteriormente por el simulador) siempre se empieza definiendo un nuevo simulador de la forma siguiente:

```
set ns [new Simulator]
```

Esta orden crea una instancia del objeto simulador, diciéndole que se va a realizar una nueva simulación.

Posteriormente se abre el fichero donde se escribirán todos los datos obtenidos por el simulador, para después leerlos con NAM y los represente de forma gráfica con el Xgraph.

```
set nf [open out.nam w]
$ns namtrace-all $nf
```

En la línea siguiente, los datos creados por NS se guardan en el out.nam que los visualizará.

En el programa se introduce un procedimiento "finish", que deberá ser definido al principio del programa, el cual cerrará el fichero de valores de trazado (out.nam) y pondrá en marcha a NAM, este deberá tener la siguiente estructura:

```
proc finish{} {
                global ns nf
                $ns flush-trace
                close $nf
                exec nam out.nam &
        exit=
}
```

La siguiente línea que escribiremos será el tiempo que vamos a simular la red

```
$ns at <tiempo><elemento>
```

Donde <tiempo> será el valor en segundos y <elemento> será en que procedimiento se cierra la simulación, por ejemplo:

```
$ns at 5.0 "finish"
```

Así se le dice a NS que ejecute la simulación durante 5.0 seg y después ejecute el procedimiento "finish"

En la última línea del programa será para que arranque la simulación de la siguiente forma:

```
$ns run
```

Como en todo programa si se desea poner comentarios, para poderlo entender en lecturas posteriores, estos se hacen de la siguiente forma:

```
# Comentario
```

I.III. ESCENARIO BÁSICO: Dos nodos, una conexión

En este escenario básico [36] se van a unir dos nodos por una línea dúplex tal como aparecen en la **Fig. I.I**.

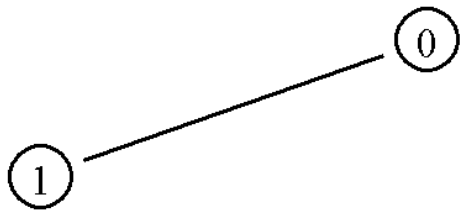

Fig. I.I Enlace entre los dos nodos en NAM

Estos dos nodos están conectados por una línea dúplex de 1Mbps y la transmisión tarda en ir de uno a otro 10ms, es decir, la distancia que se encuentra uno del otro es de 10ms por la velocidad de propagación de la señal en el medio en que se realiza dicha transmisión. Por ejemplo: si es una fibra donde v=0.7*c la distancia seria L=0.01*0.7*3 exp 8=2100Km.

Los comandos que se muestran continuación deben ser añadidos al script anterior (estructura básica) para realizar la simulación.

Los nodos en NS se definen con la instrucción:

```
set <nombre nodo>[$ns node]
```

El comando set en OTcl significa que se crea una instancia del objeto a simular. Siempre que se creen nodos se debe poner el comando "$ns node".

Se tienen dos nodos, n0 y n1, se definen en el script de la siguiente forma:

```
ns n0[$ns node]
ns n1[$ns node]
```

El siguiente paso es unir ambos nodos por una línea, en este caso una línea dúplex de la forma:

```
$ns duplex-link $n0 $n1 1Mb 10ms Droptail
```

En el apartado anterior solamente se ha definido su topología, pero en una red se deben enviar datos del nodo n0 al nodo n1.

En NS los datos se envían desde un agente a otro.

Por tanto el primer paso será crear el agente que envía los datos del nodo 0 (n0), al agente que recibirá los datos en el nodo 1 (n1).

```
#Crear un agente UDP y unirlo al nodo n0
set udp0 [new Agent/UDP]
$ns attach-agent $n0 $udp0
```

Después se debe crear un generador de tráfico CBR que se unirá al agente UDP.

Se debe definir el tamaño de los paquetes a enviar, en este caso se define de 500 bytes y cada paquete será enviado cada 0.005 segundos, es decir, se enviarán 200 paquetes por segundo.

```
#Se crea una fuente de tráfico CBR que se une a udp0
set cbr0 [new Application/Traffic/CBR]
$cbr0 set paquetSize_500
$cbr0 set interval_0.005
$cbr attach-agent $udp0
```

Después se debe crear el agente que se asociara al nodo 1, en este caso se crea un agente nulo.

```
#Se crea un agente de tipo NULL asociados al nodo 1
set null0 [new Agent/Null]
$ns attach-agent $ns1 $null0
```

Ahora se debe conectar ambos agentes:

```
#Se conecta el tráfico de la fuente con el del receptor
$ns connect $udp0 $null0
```

También se debe decir cuándo el agente CBR enviará datos y cuándo dejará de enviarlos y se deberá hacerlo antes de la línea *'ns at 5.0 "finish" '*.

```
#Se fija el funcionamiento de eventos de CBR
#Arrancará a los 0.5 seg de empezar la simulación
$ns at 0.5 "$cbr0 start"
#Y hará la parada a los 4.5 segundos
$ns at 4.5 "$cbr0 stop"
```

Este ultimo código hace que cuando se pulse el botón de arranque de NAM después de 0.5 seg empezará a enviar paquetes del nodo0 al nodo 1.

En la **Fig. I.II** se puede observar la simulación que muestra NAM:

Fig. I.II Imagen del envío de datos entre los nodos 0 y 1

En el apartado I.VI se puede ver el script final que une los diferentes apartados que se han comentado.

I.IV. SIMULACIÓN EN REDES WIRELESS

En este apartado, se mostrarán los pasos a seguir para simular un escenario con dos nodos inalámbricos utilizando la tecnología Wi-Fi con el estándar (IEEE 802.11).

La topología consiste en dos nodos móviles, nodo 0 y nodo 1. Durante la simulación, los nodos pueden quedarse en una posición estática o pueden moverse a lo largo del plano que se define en un área de 500mx500m.

En NS-2 un nodo móvil se define a partir de una serie de componentes de red, como son: la capa de enlace (LL - Link Layer), interfaz de colas (IfQ), la capa MAC, el canal donde los nodos envían y reciben los datos, etc.

Para empezar la simulación, se deben definir el tipo para cada uno de estos componentes de red. Adicionalmente, se debe introducir otros parámetros como el tipo de antena de los nodos, el modelo de propagación, el tipo de protocolo de encaminamiento que usan los nodos de la red, etc.

Los componentes básicos que debe tener cualquier script tcl para las simulaciones wireless son los que se muestran a continuación: (para información sobre los diferentes valores de cada componente, mirar el Capitulo 15 (redes móviles en NS-2) de la documentación de NS-2 [34])

```
# ======================================================================
# Definición de opciones
# ======================================================================
set val(chan)        Channel/WirelessChannel  ;# tipo de canal
set val(prop)        Propagation/TwoRayGround ;# modelo de radio-propagación
set val(ant)         Antenna/OmniAntenna      ;# tipo de antena
set val(ll)          LL                       ;# tipo de capa de enlace
set val(ifq)         Queue/DropTail/PriQueue  ;# tipo de disciplina de cola
set val(ifqlen)      50                       ;# tamaño max de paquetes en la cola
```

```
set val(netif)      Phy/WirelessPhy         ;# tipo de interficie de red
set val(mac)        Mac/802_11              ;# tipo de MAC
set val(rp)         DSDV                ;# protocolo de rutado ad-hoc
set val(nn)         2                   ;# numero de nodos moviles
```

Antes de crear un nodo en la simulación se deben configurar las características como se muestra a continuación:

```
    # Configuración de nodos
$ns_ node-config -adhocRouting $val(rp) \
            -llType $val(ll) \
            -macType $val(mac) \
            -ifqType $val(ifq) \
            -ifqLen $val(ifqlen) \
            -antType $val(ant) \
            -propType $val(prop) \
            -phyType $val(netif) \
            -topoInstance $topo \
            -channelType $val(chan) \
            -agentTrace ON \
            -routerTrace ON \
            -macTrace OFF \
            -movementTrace OFF
```

Una vez configurado el tipo de nodo se crean los nodos del escenario con la instrucción siguiente:

```
for {set i 0} {$i < $val(nn) } {incr i} {
    set node_($i) [$ns_ node ]
     $node_($i) random-motion 0       ;# deshabilita el movimiento aleatorio
}
```

La función random se utiliza cuando no importa el movimiento (velocidad y dirección) de los nodos dentro del escenario (random-motion 1). Si, por el contrario, el movimiento desea ser definido este valor tiene que ser 0 (random-motion 0).

Una vez se han creado los nodos, es momento de definir la posición inicial dentro del escenario. Mediante coordenadas se define de la siguiente forma:

```
$node_(0) set X_ 5.0
$node_(0) set Y_ 2.0

$node_(1) set X_ 390.0
$node_(1) set Y_ 385.0
```

El nodo 1 empieza en la posición (5,2), mientras que el nodo 2 lo hace en la posición (390,385). La unidad en la cual están definidos los valores son metros.

Dado que los nodos son móviles, para programar un movimiento del nodo a través del escenario intervienen tres valores: el tiempo de inicio del movimiento t(s), la posición de destino (x,y) y la velocidad el nodo v(m/s). Se puede observar en el siguiente ejemplo:

```
# El Node_(1) inicia el movimiento en dirección al node_(0)
```

```
$ns_ at 50.0 "$node_(1) setdest 25.0 20.0 15.0"

#Ahora el Node_(1) se aleja del node_(0)
$ns_ at 100.0 "$node_(1) setdest 490.0 480.0 15.0"
```

En el primer movimiento, el nodo 1 inicia el movimiento en el segundo 50 acercándose al nodo 0 con una velocidad de 15 m/s hasta la posición (25,20). En el instante 100, el nodo 1 se aleja del nodo 0 a una velocidad de 15 m/s hasta la posición (490,480).

El paso siguiente es configurar el tráfico que habrá entre los nodos del escenario. En este caso se configurará una aplicación FTP a un agente emisor TCP (tcp) al nodo node_(0) y un agente receptor (sink) al nodo node_(1)

```
# Conexiones TCP entre el node_(0) y el node_(1)

set tcp [new Agent/TCP]
$tcp set class_ 2
set sink [new Agent/TCPSink]
$ns_ attach-agent $node_(0) $tcp
$ns_ attach-agent $node_(1) $sink
$ns_ connect $tcp $sink
set ftp [new Application/FTP]
$ftp attach-agent $tcp
$ns_ at 10.0 "$ftp start"
```

Una vez se determina el tráfico entre los nodos de la simulación solo falta cerrar los ficheros de traza abiertos de la siguiente forma:

```
#
# Definimos en que instante se acaba la simulación
#
for {set i 0} {$i < $val(nn) } {incr i} {
        $ns_ at 150.0 "$node_($i) reset";
}
$ns_ at 150.0001 "stop"
$ns_ at 150.0002 "puts \"NS EXITING...\" ; $ns_ halt"
proc stop {} {
        global ns_ tracefd
        close $tracefd
}
```

En el segundo 150 de la simulación se da la orden de parar el simulador. Se resetean los parámetros de los nodos y se cierran los archivos de trazas instantes después.
Para terminar el fichero tcl, se introduce la ejecución del simulador.

```
puts "Empezando simulación..."
$ns_ run
```

En el apartado I.VII se puede observar el script completo de un escenario básico para redes wireless.

I.V. ANÁLISIS DE RESULTADOS

Una vez se ha realizado la simulación, se inicia la fase de análisis de resultados. NS-2 genera como resultado de su simulación un archivo .tr con formato de traza donde se registran todos los paquetes que se generan/envían/reciben durante la simulación. Estos ficheros, dependiendo del tiempo de la simulación, pueden llegar a ser de gran tamaño (varios Mbytes).

Para analizar los ficheros .tr (fichero de traza) generados por el simulador es conveniente el uso de herramientas para la creación de gráficos para poder observar las variables como el throughput (Mbps), jitter (ms), etc. a lo largo de la simulación.

Hay varias herramientas para el análisis de los archivos de traza .tr como el Xgraph [35] que nos viene con el paquete ns-allinone de NS-2 o el Tracegraph [37] que utiliza librerías de Matlab.

En este tutorial se utilizará Tracegraph ya que nos permite obtener muchos más parámetros de la simulación. Se puede descargar de la página Tracegraph [37]. En esta página se puede encontrar toda la información sobre como instalarlo tanto en Linux como en Windows.

Una vez instalado, Tracegraph nos permite analizar la traza que NS-2 ha generado a partir de nuestra simulación.

En primer lugar, se debe ejecutar Tracegraph en el directorio donde fue instalado:

```
/home/user/tracegraph/$ ./trgraph
```

Una vez ejecutado, se abren tres ventanas:

- Trace Graph 2.02 (version)

Es donde abriremos nuestro archivo de traza .tr para que sea analizado.

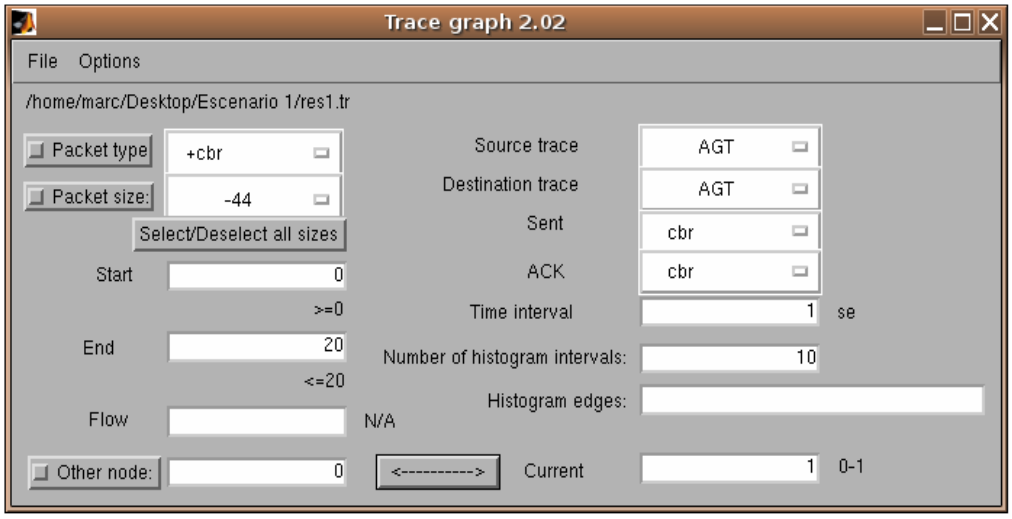

Fig. I.III Primera ventana Tracegraph

- Graphs

Una vez abierta la traza en la ventana Trace Graph en el menu -> 2D Graphs -> Tipo de gráfico (throghput, jitter, packet size, etc.)

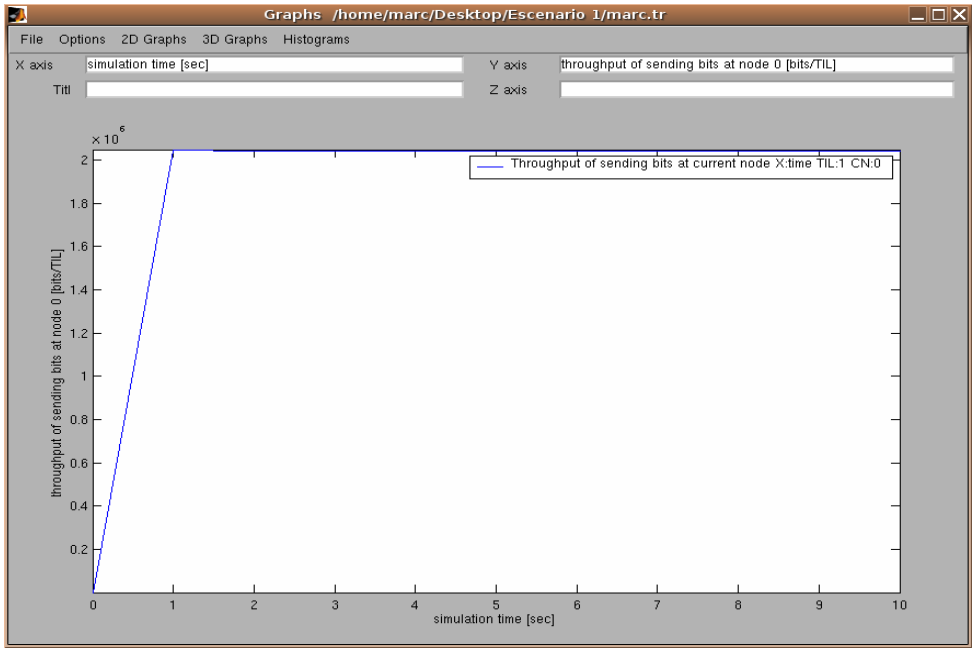

Fig. I.IV Análisi de traza en Tracegraph

- Network Information

En esta ventana (**Fig. I.V**) se puede consultar información de la simulación.

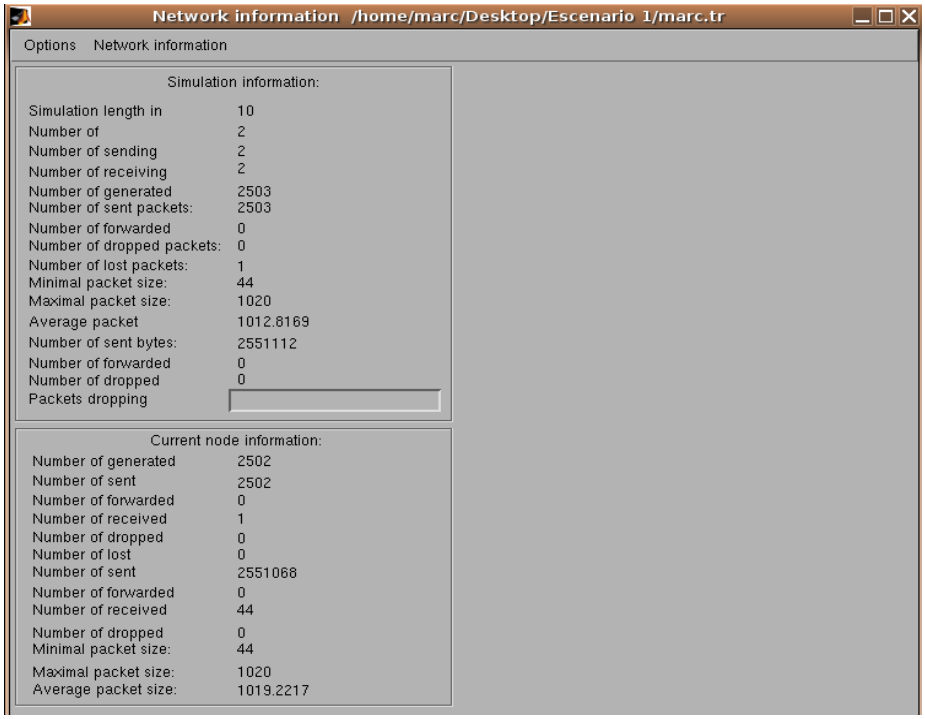

Fig. I.V Información sobre simulación en Tracegraph

Para que sea más cómodo el análisis es mejor guardar la traza a un fichero *.mat*.

I.VI. SCRIPT FINAL

```
# Creamos el objeto a simular
set ns [new Simulator]

#Abrimos el fichero de trazado NAM
set nf [pen out.nam w]
$ns namtrace-al $nf

# Definimos el procedimiento "finish"
        poc finish{} {
                global ns nf
                $ns flush-trace
                # Cerramos el fichero de trazado
                close $nf
                # Ejecutamos NAM con el fichero de trazado
                exec nam out.nam &
                exit 0
        }

#Definimos los nodos que necesita la simulación
set n0[$ns node]
set n1[$ns node]
```

```
# Definimos los enlaces entre nodos
$ns duplex-link $n0 $n1 1Mb 10ms DropTail

#Crear un agente UDP y unirlo al nodo n0
set udp0 [new Agent/UDP]
$ns attach-agent $n0 $udp0

#Creamos una fuente de tráfico CBR que se une a udp0
set cbr0 [new Application/Traffic/CBR]
$cbr0 set paquetSize_500
$cbr0 set interval_0.005
$cbr attach-agent $udp0

#Creamos un agente de tipo NULL asociados al nodo 1
 set null0 [new Agent/Null]
$ns attach-agent $ns1 $null0

#Conectamos el tráfico de la fuente con el del receptor
 $ns connect $udp0 $null0

#Fijamos el funcionamiento de eventos de CBR

#Arrancará a los 0.5 seg de empezar la simulación
 $ns at 0.5 "$cbr0 start"

#Y hará la parada a los 4.5 segundos
 $ns at 4.5 "$cbr0 stop"

#Llamamos a la aplicación "finish" después de 5segundos de simulación
$ns at 5.0 "finish"

#Arrancamos la simulación planteada
$ns run
```

I.VII. SCRIPT FINAL CON WIRELESS

```
# ======================================================================
# Definición de opciones
# ======================================================================
set val(chan)         Channel/WirelessChannel  ;# tipo de canal
set val(prop)         Propagation/TwoRayGround ;# modelo de radio-propagación
set val(ant)          Antenna/OmniAntenna      ;# tipo de antena
set val(ll)           LL                       ;# tipo de capa de enlace
set val(ifq)          Queue/DropTail/PriQueue  ;# tipo de disciplina de cola
set val(ifqlen)       50                       ;# num. max de paquetes en cola
set val(netif)        Phy/WirelessPhy          ;# tipo de interficie de red
set val(mac)          Mac/802_11               ;# tipo de MAC
set val(rp)           DSDV                     ;# protocolo de rutado ad-hoc
set  val(nn)                    2                                    ;# numero de nodos moviles
======================================================================

# Programa Principal
# ======================================================================

#
```

```
# Se definen Variables Globales
#
set ns_             [new Simulator]
set tracefd     [open simple.tr w]
$ns_ trace-all $tracefd

# Configuramos la topografía del escenario
set topo       [new Topography]
$topo load_flatgrid 500 500

#
# Creamos God
#
create-god $val(nn)

#
#  Creamos los nodos y la conexión entre ellos
#  Se crean 2 nodos: node (0) and node (1)

# Configuración de nodo

      $ns_ node-config -adhocRouting $val(rp) \

                              -llType $val(ll) \
                              -macType $val(mac) \
                              -ifqType $val(ifq) \
                              -ifqLen $val(ifqlen) \
                              -antType $val(ant) \
                              -propType $val(prop) \
                              -phyType $val(netif) \
                              -channelType $val(chan) \
                              -topoInstance $topo \
                              -agentTrace ON \
                              -routerTrace ON \
                              -macTrace OFF \
                              -movementTrace OFF


         for {set i 0} {$i < $val(nn) } {incr i} {

                  set node_($i) [$ns_ node]
                  $node_($i) random-motion 0              ;# deshabilita el movimiento aleatorio
de los nodos
         }

#
# Posición inicial de los nodos
#

$node_(0) set X_ 5.0
$node_(0) set Y_ 2.0
$node_(0) set Z_ 0.0

$node_(1) set X_ 390.0
$node_(1) set Y_ 385.0
$node_(1) set Z_ 0.0

#
# Se definen los movimientos de los nodos durante la simulación
#
```

```
$ns_ at 50.0 "$node_(1) setdest 25.0 20.0 15.0"
$ns_ at 10.0 "$node_(0) setdest 20.0 18.0 1.0"
$ns_ at 100.0 "$node_(1) setdest 490.0 480.0 15.0"

# Configuramos el tipo de tráfico entre los nodos
# Connexiones TCP entre node_(0) y node_(1)

set tcp [new Agent/TCP]
$tcp set class_ 2
set sink [new Agent/TCPSink]
$ns_ attach-agent $node_(0) $tcp
$ns_ attach-agent $node_(1) $sink
$ns_ connect $tcp $sink
set ftp [new Application/FTP]
$ftp attach-agent $tcp
$ns_ at 10.0 "$ftp start"

#
# Se define cuando termina la simulación
#

for {set i 0} {$i < $val(nn) } {incr i} {
    $ns_ at 150.0 "$node_($i) reset";
}

$ns_ at 150.0 "stop"
$ns_ at 150.01 "puts \"NS EXITING...\" ; $ns_ halt"
proc stop {} {
    global ns_ tracefd
    $ns_ flush-trace
    close $tracefd
}

## Se ejecuta la simulación
#

puts "Empezando Simulación..."

$ns_ run
```

ANEXO II. MANUAL OPNET MODELER

I.I. INTRODUCCIÓN

Se define simulación como una técnica que imita el comportamiento de un sistema del mundo real conforme evoluciona el tiempo.

La simulación mediante software en el campo de redes de telecomunicaciones es actualmente una herramienta muy útil para empresas y Universidades del sector a la hora de recrear el comportamiento de un sistema en el cuál se están ejecutando diferentes servicios y/o aplicaciones en unas condiciones definidas por el usuario.

El hecho de simular permite descubrir cuellos de botella en una red, analizar la capacidad de un enlace o prever qué número de puntos de acceso son necesarios para dar cobertura a ciertos clientes inalámbricos por ejemplo. Todo esto sin tener que probarlo in situ, con lo que comporta una gran reducción de coste sobretodo para redes WAN de gran envergadura.

La elección del simulador depende de varios factores: número de protocolos que puede simular, la inteligibilidad del propio simulador, la facilidad de extraer resultados, y el coste del mismo entre otros.

El proceso de extracción y representación de resultados de un simulador se basa en el muestreo aleatorio del sistema. Por ello, los resultados posteriormente se han de evaluar y comprobar su lógica con las previsiones que se esperaban obtener. Para dotar de mayor fiabilidad a los resultados es aconsejable realizar varias simulaciones y obtener una media de las variables a analizar.

I.II. ¿QUÉ ES OPNET MODELER?

Modeler es un simulador basado en eventos orientado a la simulación de redes de telecomunicaciones creado por OPNET (Optimized Network Engineering Tools) [38].

Para ser más explícitos lo podríamos definir como un simulador dinámico y discreto que puede realizar simulaciones deterministas y/o aleatorias basándose en teorías de redes de colas.

- Dinámico porque la representación del sistema durante la simulación evoluciona con el tiempo.

- Discreto porque el comportamiento de los sistemas representados cambia únicamente en instantes de tiempo concretos, es decir, eventos.

- Una simulación es aleatoria cuando durante la simulación entran en juego variables aleatorias. En cambio, se define como determinista cuando no entra en juego ninguna variable aleatoria. En OPNET puedes definir gran cantidad de variables y asignarles un patrón determinista o aleatorio.

OPNET Modeler es uno de los simuladores más avanzados en el campo de las redes de telecomunicaciones. Quizás, la característica más relevante es que es un simulador orientado a objetos, lo que permite interactuar al usuario sin problemas y ofrece una gran facilidad de interpretación y creación de escenarios a parte de tener en cada objeto una serie de atributos configurables.

Dispone de multitud de librerías, lo que permite simular gran diversidad de redes donde intervenga un amplio número de protocolos y variables específicas que el usuario podrá modificar y estudiar. Número de paquetes perdidos, throughput, jitter, caída de enlaces, potencia de transmisión son algunos de los parámetros que se pueden controlar.

Cabe destacar que OPNET permite entre otras cosas dotar de movilidad a los nodos de la red, modificar el código fuente de las librerías de los nodos para alterar su comportamiento ante diversas acciones, definir tipos de tráfico además de carga de la red debido a tipos de servicios, como por ejemplo HTTP, correo, VoIP, streaming, etc.

Este manual está basado en la versión 11.0 de dicho software.

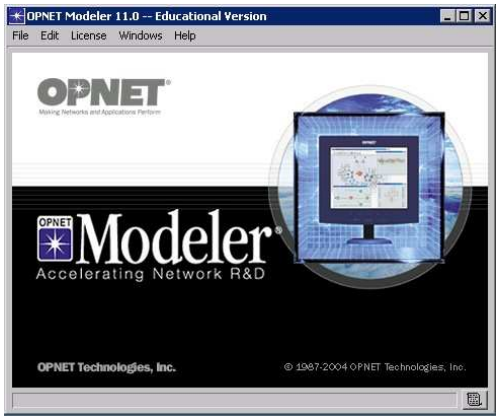

Fig. II.I Pantalla principal de OPNET Modeler v11.0

II.III. FUNCIONAMIENTO DE OPNET

Antes de entrar en contacto con el software hay que entender el funcionamiento interno de OPNET, es decir cómo se estructura un escenario que queramos simular.

OPNET utiliza una estructura jerárquica en tres niveles: modelo de red, modelo de nodos y modelo de procesos tal y como podemos ver en la **Fig. II.II**.

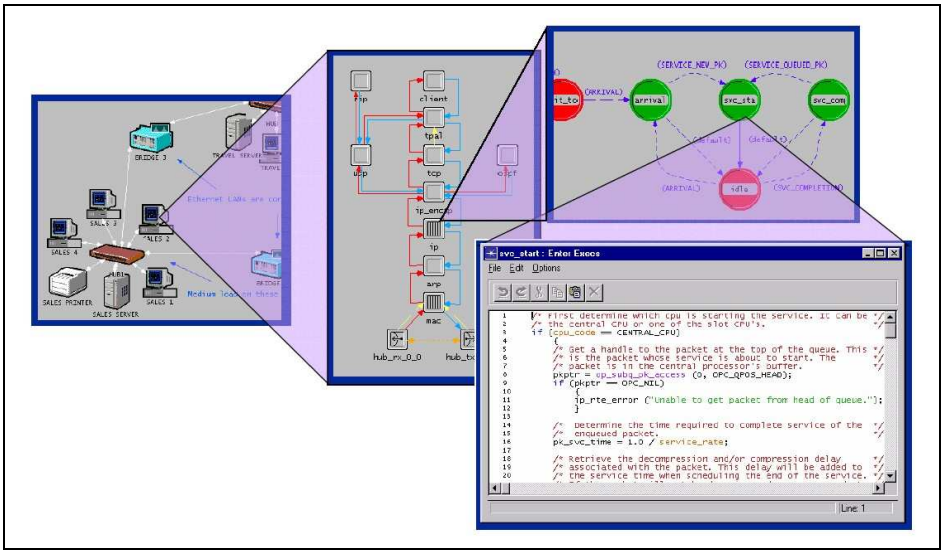

Fig. II.II Estructura jerárquica de OPNET Modeler

II.III.I. Modelo de Red

El modelo de red representa la topología de la red de comunicaciones, construida a partir de nodos y enlaces que los interconectan.

Una de las características más importantes del modelo de red de Modeler es la posibilidad de recrear redes con contexto geográfico donde se pueden definir características físicas del escenario tales como la distancia u obstáculos que puedan alterar el funcionamiento de nodos inalámbricos por ejemplo.

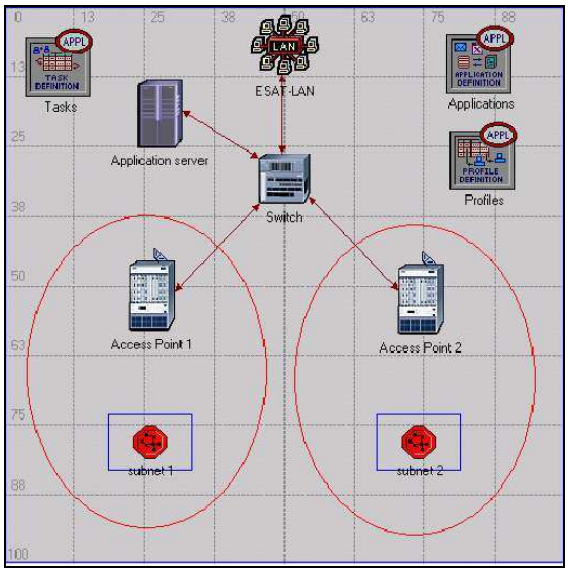

Fig. II.III Ejemplo de modelo de red

II.III.II. Modelo de nodos

Si entramos en un nodo, realizando doble-clic en cualquiera de ellos podemos ver la estructura de elementos funcionales que lo componen, denominados módulos. Los diferentes componentes de los protocolos están representados por módulos.

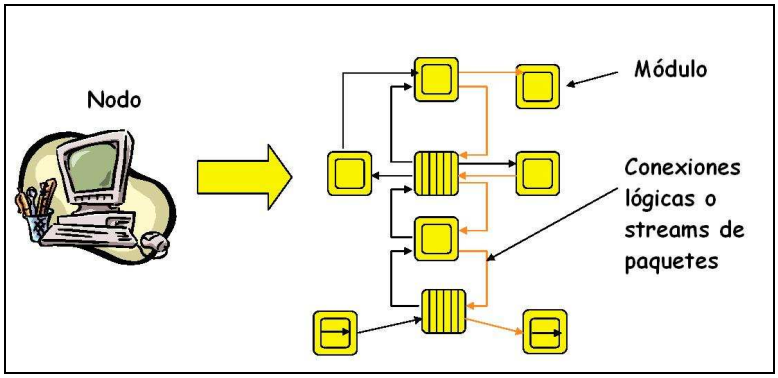

Fig. II.IV Ejemplo de modelo de nodos

II.III.III. Modelo de procesos

Cada módulo está compuesto por un modelo de procesos que representan protocolos, algoritmos, aplicaciones, etc. aproximándolos mediante máquinas de estados finitos (FSM *Finite State Machine*). La FSM está compuesta normalmente por estados forzados y no forzados. Los estados forzados son

estados en los que el proceso está bloqueado esperando a que un nuevo evento se genere para cambiar a un nuevo estado.

Las transiciones entre estados pueden ser condicionales o incondicionales. A cada estado se le puede asignar una serie de código que se ejecutará al entrar o al salir del estado. El funcionamiento interno tanto de estados como de transiciones implica la programación en lenguaje C/C++. En caso de utilizar modelos de nodos estándar no será necesaria la programación de procesos, ya estarán definidos en las librerías.

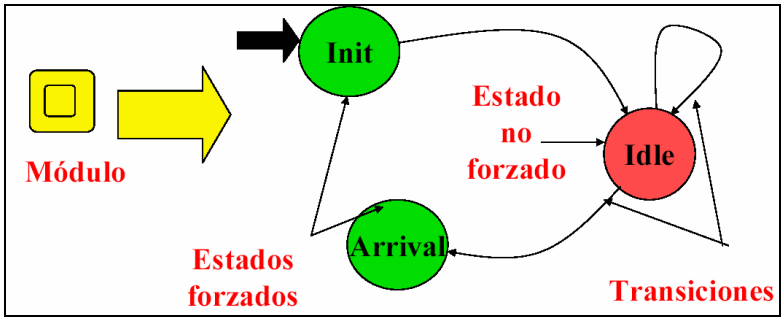

Fig. II.V Ejemplo de modelo de procesos

El hecho de llevar código C/C++ conlleva a tener instalado en el ordenador un compilador de C++ como por ejemplo el Microsoft Visual C++.

Una vez se han diseñado los tres niveles de jerarquía pasaremos a definir el tráfico que se generará en la red. Posteriormente se escogerán las variables a analizar, se ejecutará la simulación y obtendremos los resultados.

Estos pasos se verán reflejados con mayor claridad en el siguiente apartado donde se realiza la simulación de una red WLAN (Wireless Local Area Network).

II.IV. CREACIÓN DE UNA RED WLAN EN OPNET

A continuación se explicará detalladamente cómo realizar una simulación en OPNET Modeler de una red de área local inalámbrica (WLAN) donde el escenario estará compuesto por 2 nodos inalámbricos y un punto de acceso. También definiremos el tráfico que correrá por la red y se mostrará como recoger y analizar resultados.

II.IV.I. Creación escenario

II.IV.I.I. Abrir el simulador OPNET Modeler

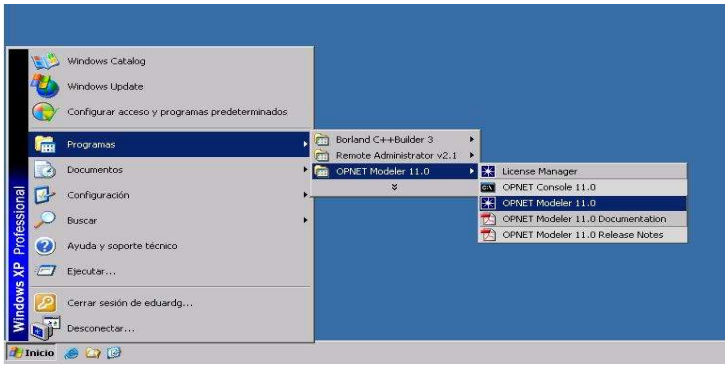

Fig. II.V Acceso al simulador

*II.IV.I.II. Crear un nuevo proyecto **File** → **New** → **Project***

II.IV.I.III. Asignar nombre al proyecto y al escenario.

En la primera casilla le asignaremos un nombre al proyecto. En nuestro caso WLAN. El nombre del escenario puede ser distinto al del proyecto, ya que en un mismo proyecto puede haber varios escenarios para comparar diferentes casos. En nuestro caso le hemos llamado "2nodos_1AP". La siguiente casilla es aconsejable que esté activada ya que así nos ayudará a crear el escenario.

Fig. II.VI Asignación de nombre al proyecto y al escenario

II.IV.I.IV. Elección del tipo de escenario.

A continuación se especifica el tipo de escenario. Escogeremos la primera opción, para crear un escenario vacío y le daremos a siguiente. A continuación escogeremos la opción “Office” ya que la red inalámbrica la queremos simular en un espacio cerrado. Seguidamente aceptaremos los valores por defecto, cambiando únicamente las dimensiones del espacio, en nuestro caso 300 m x 300 m. De esta manera podremos observar en otras simulaciones los efectos del modelo de propagación. A partir de una distancia de 300 m entre nodos, OPNET está diseñado para no recibir señal. Las siguientes opciones las aceptaremos tal y como vienen por defecto hasta que cliquemos sobre el botón de “finish”.

II.IV.I.V. Paleta de objetos.

El escenario representado como una cuadrícula aparece vacío. Allí tendremos que colocar los diferentes objetos que encontraremos en la “paleta de objetos” ordenada por familia de protocolos.

La paleta de objetos aparece abierta en este momento, pero en caso de cerrarla siempre podremos acceder a ella mediante el primer botón de la izquierda que encontramos en la barra de menú. En esta ocasión escogeremos la familia *wireless_lan* donde encontraremos los nodos necesarios para crear una red inalámbrica.

II.IV.I.VI. Configuración de cuadrícula.

Antes de nada, configuraremos la cuadrícula para poder medir mejor la distancia entre los nodos. Por defecto cada celda es de 25 m x 25 m. Es mejor tener unas celdas de 5 m x 5 m, sobretodo para espacios pequeños y que necesitemos bastante precisión para realizar cálculos.

Para ello, en la barra de menú iremos a la opción View → Background → Set Properties. Lo único que se debe cambiar es poner un 5 en la casilla de “Division”. En nuestro caso también hemos cambiado el color de las líneas pero por decisión propia. Debería quedar tal como aparece en la **Fig. II.VII**. Para salir y guardar los cambios haremos clic en el botón “Close”.

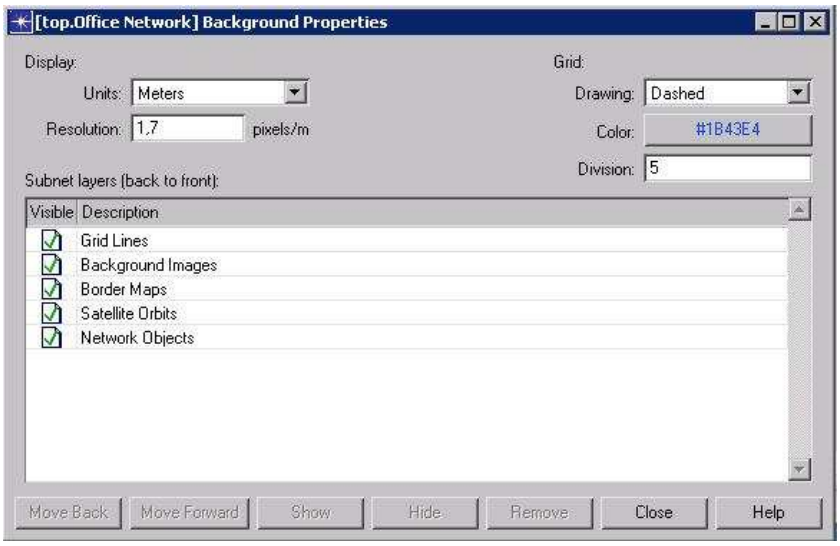

Fig. II.VII Ventana para cambiar formato cuadrícula

II.IV.I.VII. Selección de elementos.

Desde la paleta de objetos, con el ratón escogeremos el objeto *wlan_wkstn (mobile)* y lo arrastraremos al escenario. Haciendo clic otra vez en el botón izquierdo del ratón colocaremos otro nodo de este tipo. Para no colocar más nodos de este tipo haremos clic con el botón derecho del ratón encima del escenario. Este tipo de nodo simulará un cliente de WLAN con soporte para los protocolos UDP, IP, TCP y IEEE 802.11 entre otros. Además, permiten conectarnos de manera inalámbrica con velocidades de 1, 2, 5.5 y 11 Mbps.

Para el papel de punto de acceso escogeremos el objeto *wlan_server (mobile)*. Lo colocaremos en el escenario de la misma forma que el objeto anterior. Este objeto dispone de una interfaz IEEE 802.11, que también permite realizar conexiones sobre IP y cambiar la velocidad de transmisión.

Cambiaremos el nombre de los objetos para poderlos distinguir mejor. Para ello, debemos hacer clic con el botón derecho del ratón en cada objeto y seleccionar la opción “Set name”. En nuestro caso hemos llamado *client1* y *client2* a los clientes y *Server1* al servidor que realizará las funciones de punto de acceso.

Para poder generar tráfico debemos definir un servicio para hacerlo correr en el servidor. Para definir un servicio/aplicación debemos introducir en el escenario el objeto *Application Config* y para definir el perfil de tráfico se tiene que acompañar a este nodo con el objeto *Profile Config*.

El escenario debería quedar de la siguiente manera:

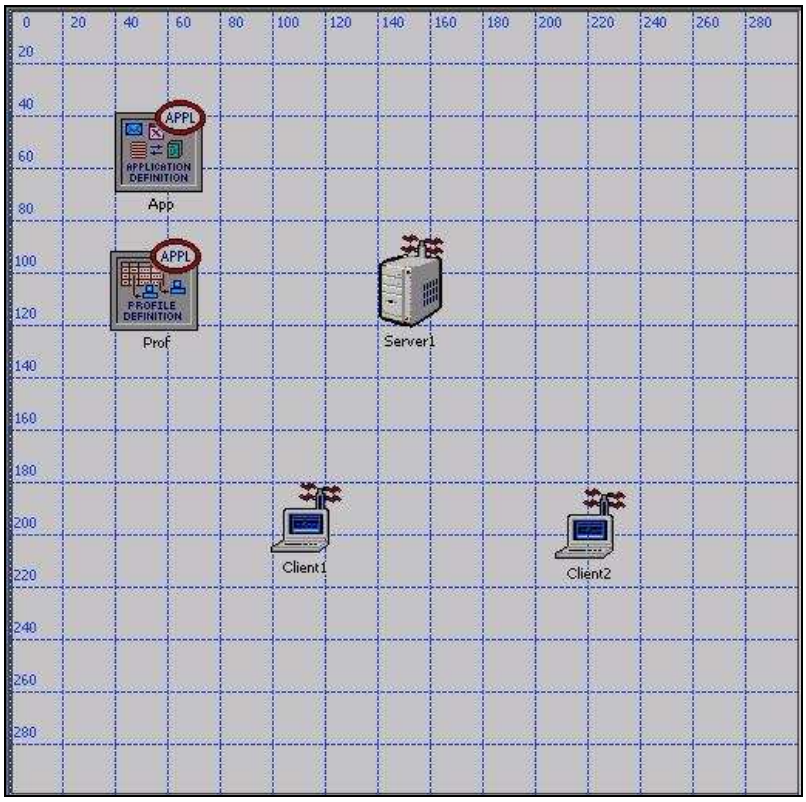

Fig. II.VIII Modelo de red a simular

II.IV.I.VIII. Interconexión de nodos.

En nuestro caso no hace falta conectarlos ya que utilizan el aire como medio físico. Si quisiéramos conectarlos vía ethernet por ejemplo, deberíamos utilizar los objetos que representan los diferentes tipos de enlaces. Para comprobar que los nodos están bien conectados utilizaremos la opción "Verify links" que podemos encontrar en el siguiente botón:

*II.IV.I.IX. Guardar los cambios en **File** → **Save**. Esta opción la realizaremos cuando cierto tiempo para evitar problemas.*

II.IV.II. Definir Tráfico/Aplicaciones

Para poder generar tráfico dentro de la red, primero se deben definir los servicios que se van a utilizar y posteriormente definir el perfil de tráfico que deseamos utilizar.

II.IV.II.I. Definir Aplicación.

Con el botón derecho del ratón haz clic en el objeto *Application Config* y selecciona la opción *Advanced Edit Attributes*. En esta opción podremos configurar todas las aplicaciones soportadas en la red. Únicamente definiremos un servicio FTP, donde Server1 ejercerá la función de Servidor FTP.

Expandimos la opción *Application Definitions.* Hacemos clic en la derecha de *rows* y seleccionamos el número 1. De esta manera decimos que en nuestro escenario solamente configuraremos una aplicación. Aparece una nueva lína (*row0*). La expandimos y le asignamos un nombre. En nuestro caso hemos puesto “Heavy FTP”.

Expandimos la opción *Description*. Aquí podremos elegir el tipo de aplicación que definiremos. Hacemos clic a la derecha de FTP y seleccionamos la opción *Edit.* Aquí estamos definiendo los parámetros de nuestro servicio FTP. El primer valor define el porcentaje de GETs dentro del total de peticiones. En nuestro caso hemos puesto un 100%, ya que de esta manera conseguimos que los clientes únicamente se bajaran ficheros del servidor y no a la inversa. Cabe destacar que hemos definido un tamaño de fichero muy grande para así ocupar al máximo la red. Los parámetros se muestran en la **Fig. II.IX**. Una vez definidos estos parámetros aceptamos las dos ventanas para guardar los cambios.

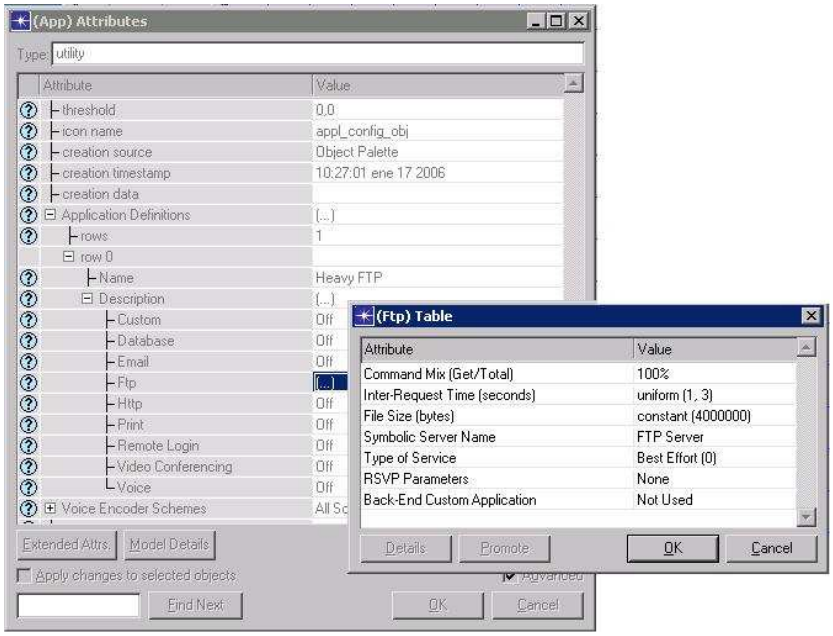

Fig. II.IX Parámetros de la aplicación FTP

II.IV.II.II.Definir Perfil.

Con el botón derecho del ratón haz clic en el objeto *Profile Config* y selecciona la opción *Advanced Edit Attributes*. En esta opción podremos configurar el perfil del tráfico FTP, como por ejemplo parámetros de cuando empiezan a transmitir los nodos.

Expandimos la opción *Profile Configuration.* Hacemos clic en la derecha de *rows* y seleccionamos el número 1. De esta manera decimos que en nuestro escenario solamente configuraremos un perfil de tráfico. Aparece una nueva lína (*row0*). La expandimos y le asignamos un nombre en *Profile Name*. En nuestro caso hemos puesto “Test FTP”.

Expandimos la opción *Applications*. Creamos una nueva fila. Volvemos a expandir la *row 0*. A la derecha de la variable *Name* seleccionamos la aplicación que hemos definido como *Heavy FTP.* Editamos la variable *Start Time Offset* y le asignamos un constant (0) para decir que empiece al inicio. En *Duration* ponemos la opción *End of Profile*. Los demás valores los dejamos por defecto y aceptamos los cambios. Debería quedar tal como aparece en la **Fig. II.X**.

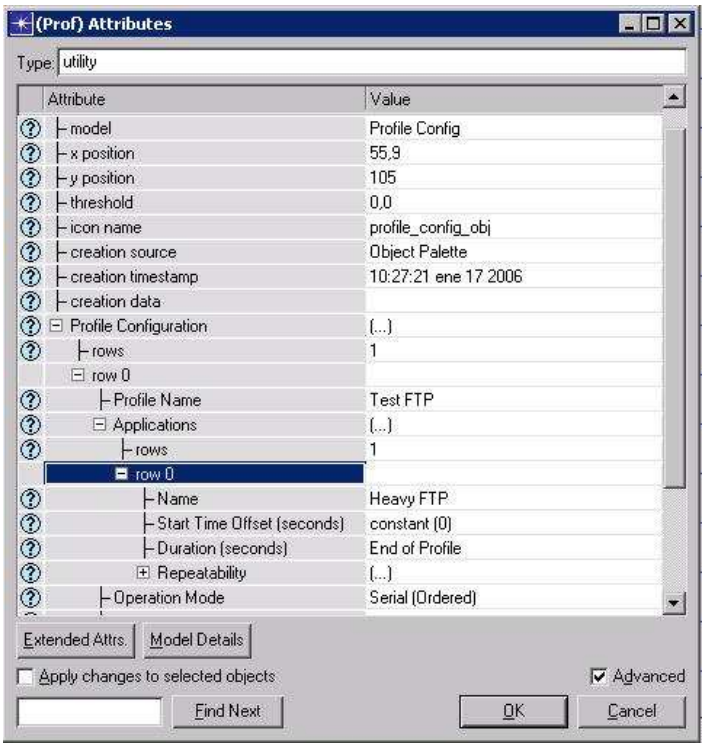

Fig. II.X Parámetros del perfil de la aplicación FTP

II.IV.III. Configuración de los objetos

II.IV.III.I.Variables del servidor.

Con el botón derecho del ratón haz clic en el objeto *server1* y selecciona la opción *Advanced Edit Attributes*. En esta opción podremos configurar todas las variables del funcionamiento interno del nodo. La mayoría las dejaremos por defecto. Únicamente cambiaremos las referentes a la interfaz WLAN y a la definición de aplicación.

Desplegamos el menú de *Wireless LAN* y seguidamente el de *Wireless LAN Parameters*. Aquí encontraremos multitud de parámetros que podemos modificar. Presionamos en el valor del parámetro de *BSS Identifier*, escogemos *Edit* y escribimos el número 1 por ejemplo. Este parámetro identifica la red inalámbrica. Todos los nodos de esta red tienen que tener el mismo identificador. Asignamos el valor *Enabled* a la variable *Access Point Functionality* para que realice funciones de punto de acceso. Escogemos la velocidad de 11 Mbps y decimos que trabaje en el canal 1. Cambiamos el valor de *Transmit Power* a 0,03 W para ponerle un valor estándar. La variable *Packet Reception-Power Threshold* indica la sensibilidad del receptor. Para la tasa de 11 Mbps, según dice el estándar debemos poner una sensibilidad de -81 dBm. Por último desactivaremos la opción de *CTS-to-self Option.*

Una vez configurada la interficie WLAN, le asignaremos un nombre al servidor como tal. Para ello, editaremos la variable *Server Ardes* y le asignaremos un nombre. En nuestro caso, lo hemos nombrado “SERVER FTP ESSID 1”.

Por último, queda asignarle la aplicación que utilizará. Para ello expandimos la opción *Applications.* A continuación editamos la opción *Application: Supported Services.* Aparecerá una ventana. Abajo a la izquierda asignamos el valor 1 a *rows* para añadir una fila. En la nueva fila, hacemos clic en el nombre del campo. Allí seleccionaremos la aplicación definida, que en nuestro caso es “Heavy FTP”. Aceptamos los cambios.

Todos los demás parámetros los dejamos por defecto tal y como se muestra en la **Fig. II.XI**:

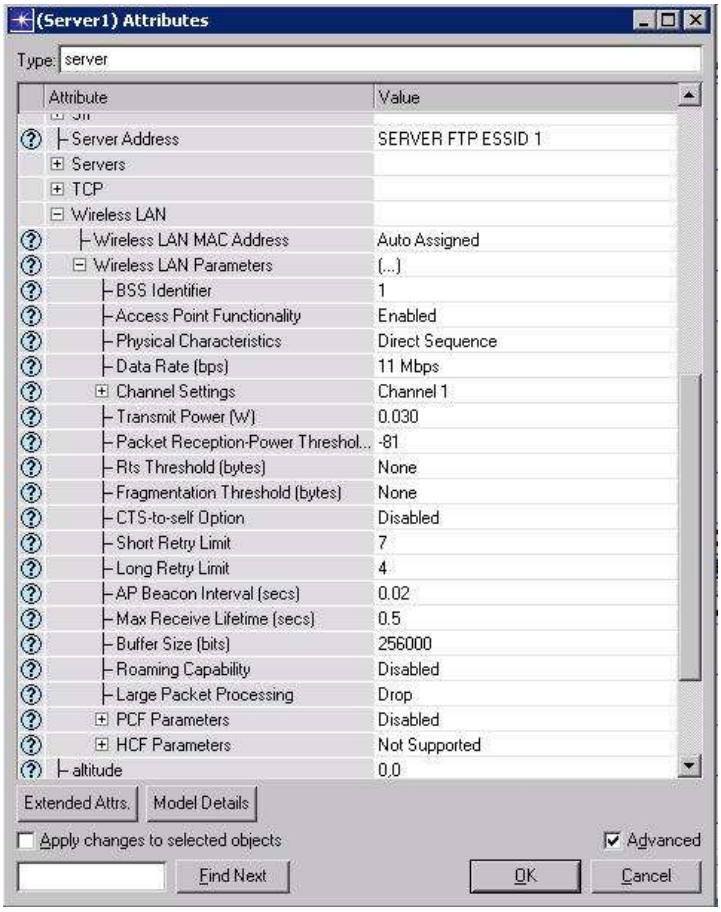

Fig. II.XI Parámetros WLAN del servidor

II.IV.III.II. Variables de los clientes inalámbricos.

Con el botón derecho del ratón haz clic en uno de los clientes y selecciona la opción *Select Similar Nodes*. Esto seleccionará todos los nodos del mismo tipo. Vuelve a hacer clic con el botón derecho en alguno de los clientes y selecciona la opción *Advanced Edit Attributes*. Antes de nada marca la casilla *Apply changes to selected objects*. De esta manera se guardarán los cambios en todos los nodos seleccionados.

En este caso cambiaremos el parámetro *BSS Identifier* tal como se ha descrito anteriormente y le asignaremos el mismo valor, en nuestro caso el número 1. También pondremos los mismos parámetros que el servidor como por ejemplo tasa 11 Mbps, canal 1, potencia transmitida a 0,03 W y sensibilidad a -81 dBm. Hay que recordarse de desactivar la opción de punto de acceso y la de CTS. Los cambios deben quedar tal como aparecen en la **Fig. II.XII**.

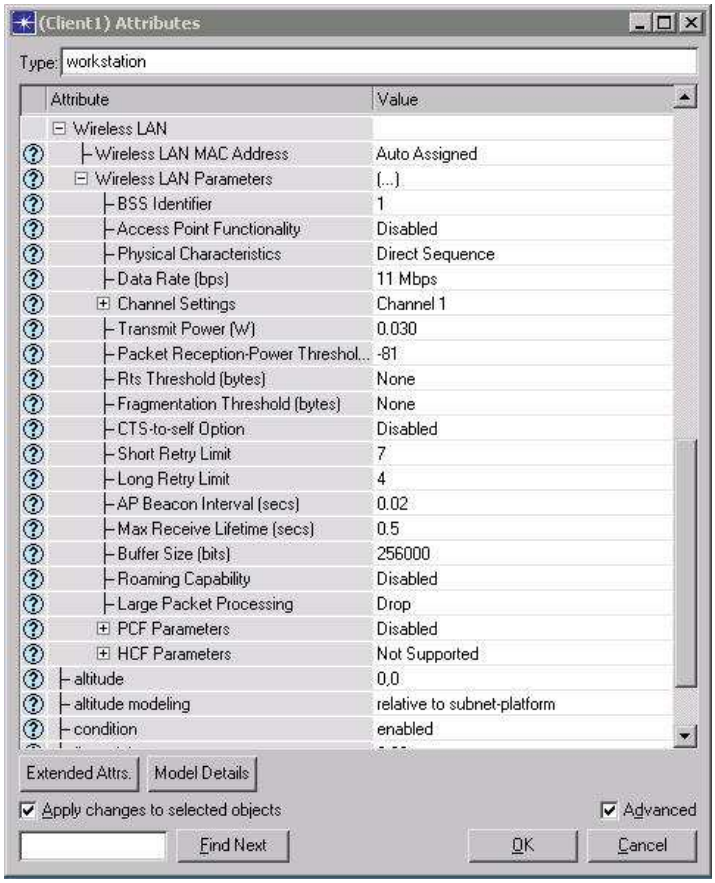

Fig. II.XII Parámetros WLAN de los clientes

Ahora pasamos a definir el uso de las aplicaciones. Para ello expandimos el menú de *Applications*. A continuación editamos la variable *Application: Destination Preferences.* Insertamos una línea (*1 row*) en la ventana que nos aparece. En la casilla de *Application* escogemos la aplicación definida, en nuestro caso *Heavy FTP*. En la casilla de *Symbolic Name* escogemos "*FTP Server"* en nuestro caso. Seguidamente hacemos clic en *Actual Name* y en *Name* escogemos el nombre del servidor al cuál se conectarán los dos clientes, en nuestro caso *SERVER FTP ESSID 1*. Aceptamos todos los cambios para cerrar las dos ventanas abiertas.

A continuación definiremos los perfiles soportados por el cliente. Para ello, dentro del menú *Applications* expandimos el submenú *Application: Supported Profiles.* Aquí añadiremos una fila como ya hemos hecho otras veces. Expandimos la *row 0* y en *Profile Name* escogemos el perfil definido. En nuestro caso es *Test FTP.* Más o menos los cambios deben quedar tal y como aparece en la **Fig. II.XIII**.

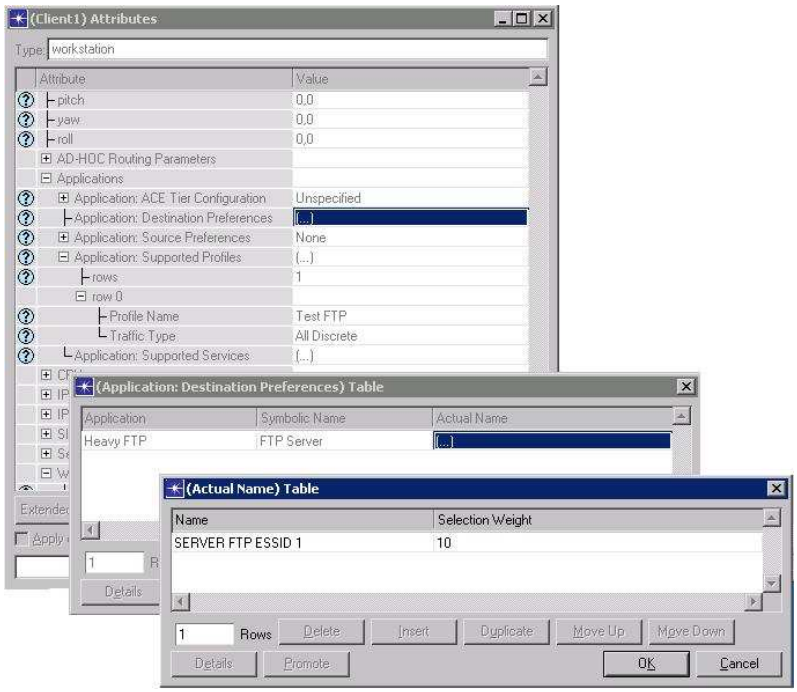

Fig. II.XIII Parámetros aplicaciones de los clientes

De esta forma tendremos el escenario bien configurado. A continuación pasamos a escoger las variables a analizar durante la simulación.

II.IV.IV. Elección de estadísticas

La selección de las estadísticas se puede hacer de diferentes maneras. Se pueden obtener resultados globales del escenario, resultados específicos de cada nodo, resultados de enlaces y resultados de demanda de tráfico. En nuestro caso escogeremos resultados globales y algunos específicos de los nodos.

Hay que tener en cuenta que una misma variable se puede analizar a partir de diferentes tipos de estadísticas, pero no se pueden repetir, es decir si por ejemplo se escoge la variable de *Tráfico enviado* en estadísticas globales no se podrá seleccionar también en estadísticas de nodo. En caso contrario puede generar un error en la simulación y no mostrar ninguna de las dos variables.

II.IV.IV.I. Selección de variables globales.

Hacemos clic con el botón derecho en cualquier sitio del escenario que no contenga ningún objeto y escogemos la opción *Choose Individual DES Statistics*. Podemos ver que aparecen tres menús que representan los tipos de variables. Entramos en *Glonal Statistics* y seleccionamos todas las de *FTP* y *Wireless Lan.* De esta manera analizaremos los

tráficos globales de todo el escenario, es decir, la suma de los resultados de cada nodo.

Ahora pasamos a escoger las variables globales de cada nodo. Para ello entramos en *Node Statistics* y escogeremos las estadísticas *Client Ftp* y *Wireless Lan* y aceptamos los cambios. Estas estadísticas las tomará para cada nodo de la red. En la **Fig. II.XIV** se pueden observar las características seleccionadas *Wireles Lan*.

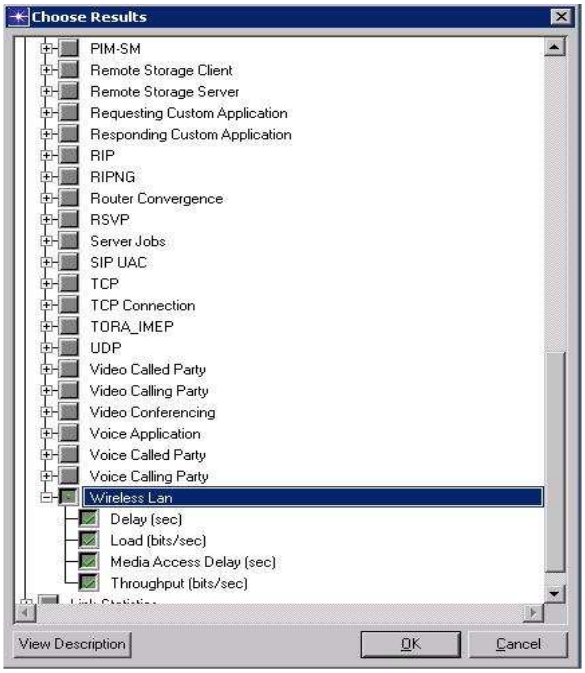

Fig. II.XIV Selección de variables globales

Para ver qué significa cada variable podemos seleccionar la variable y presionar en la opción *View Description*.

II.IV.IV.II. Selección de variables específicas.

Con estas variables podremos ver por ejemplo resultados específicos de un solo nodo, por ejemplo el *client1* o *client2*. Para ello hacemos botón derecho encima de dicho nodo y seleccionamos la opción *Choose Individual DES Statistics*. Como podemos ver, la distribución de variables ha cambiado.

Dentro del menú *Module Statistics → wlan_port_rx_0_0.channel [0]→ radio receiver* se han escogido las variables *bit error rate, received power* y *signal/noise ratio*.

II.IV.IV.III. Choose Statistics (Advanced)

Una vez escogidas las estadísticas, para asegurar que has escogido todas las variables a estudiar se puede utilizar la herramienta *Choose Statistics (Advanced)* del menú DES de la barra de herramientas.

II.IV.V. Configurar la simulación

Ya se puede proceder a ejecutar la simulación. Para ello accederemos a partir de la opción *Configure/Run Discrete Event Simulation* o a través del siguiente acceso directo:

II.IV.V.I. Configurar la simulación.

En la ventana que aparecerá (**Fig. II.XV**), en el menú *Common* se especifica las características generales de la simulación como por ejemplo la duración de la simulación (no es tiempo real), y el número de eventos.

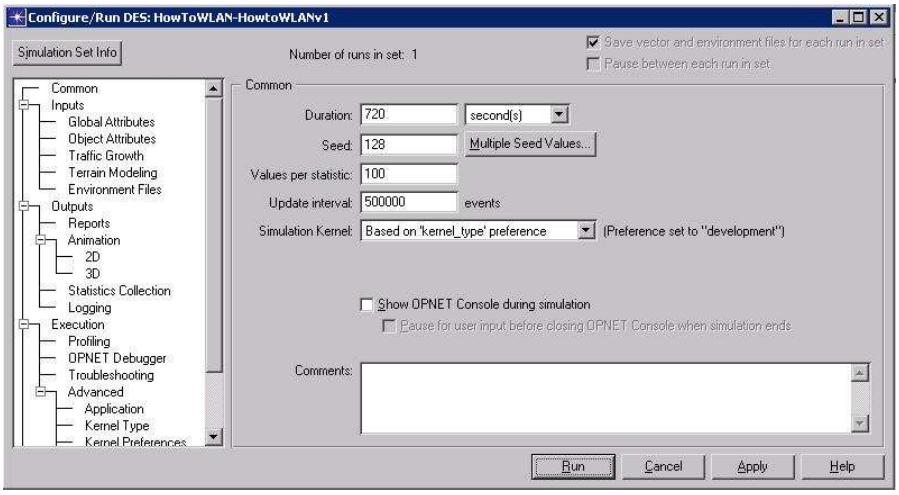

Fig. II.XV Ventana para configurar la simulación

Lo único que cambiaremos será la duración. En vez de 1 hora pondremos 720 segundos porque sino tendríamos que esperar mucho tiempo. Además, con esta duración los resultados ya muestran estadísticas fiables.

El número de eventos es el número de llegadas o salidas que se producen durante la simulación. Se establece que a mayor número de eventos, en iguales condiciones, la fiabilidad se ve incrementada.

Es pues, de especial importancia, prestar atención a que el número de eventos que se sucedan en una determinada simulación sea suficiente para garantizar que los resultados son estadísticamente fiables.

II.IV.V.II. Arrancar la simulación.

Los demás valores los dejaremos por defecto y apretaremos el botón *Run*. Cuando acabe la simulación cerraremos la ventana.

II.IV.VI. Analizar resultados

II.IV.VI.I. Analizar Resultados

Para analizar los resultados iremos a la opción *DES → Results → View Results (Advanced)* o mediante el siguiente acceso directo:

*II.IV.VI.II. Allí nos aparecerá una ventana tal como la que aparece en la **Fig. II.XVI**, donde podremos visualizar los resultados obtenidos.*

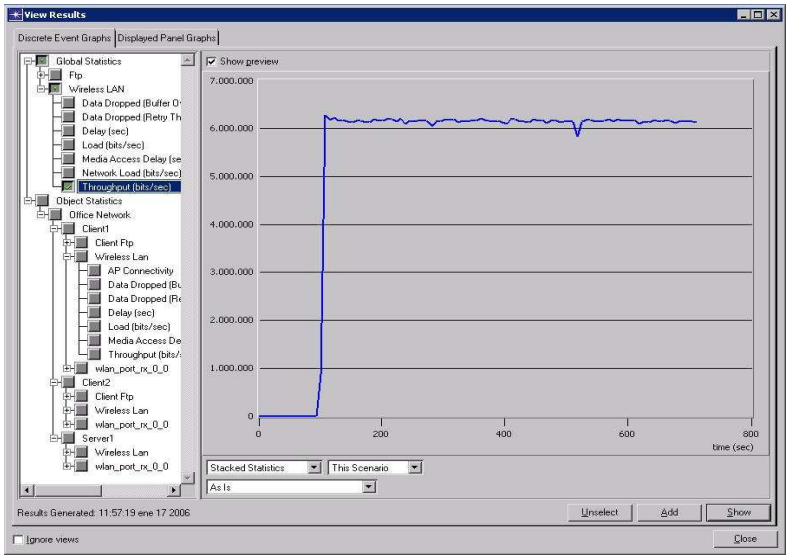

Fig. II.XVI Ventana para visualizar los resultados

De esta manera se concluye este manual. Para ver un escenario más complejo sobre redes inalámbricas simuladas en OPNET Modeler podemos recurrir a los ejemplos de proyectos que incorpora el propio software. Por ejemplo podemos

ver el proyecto *WLAN* que lo podemos abrir a partir de *File → Open → example_networks → WLAN.*

Para ver los diferentes escenarios que contiene el proyecto deberemos ir a *Scenarios → Switch to scenario* y allí escoger uno. Todos los escenarios se pueden simular y ver los resultados. Un ejemplo de estos escenarios lo podemos en la **Fig. II.XVII**.

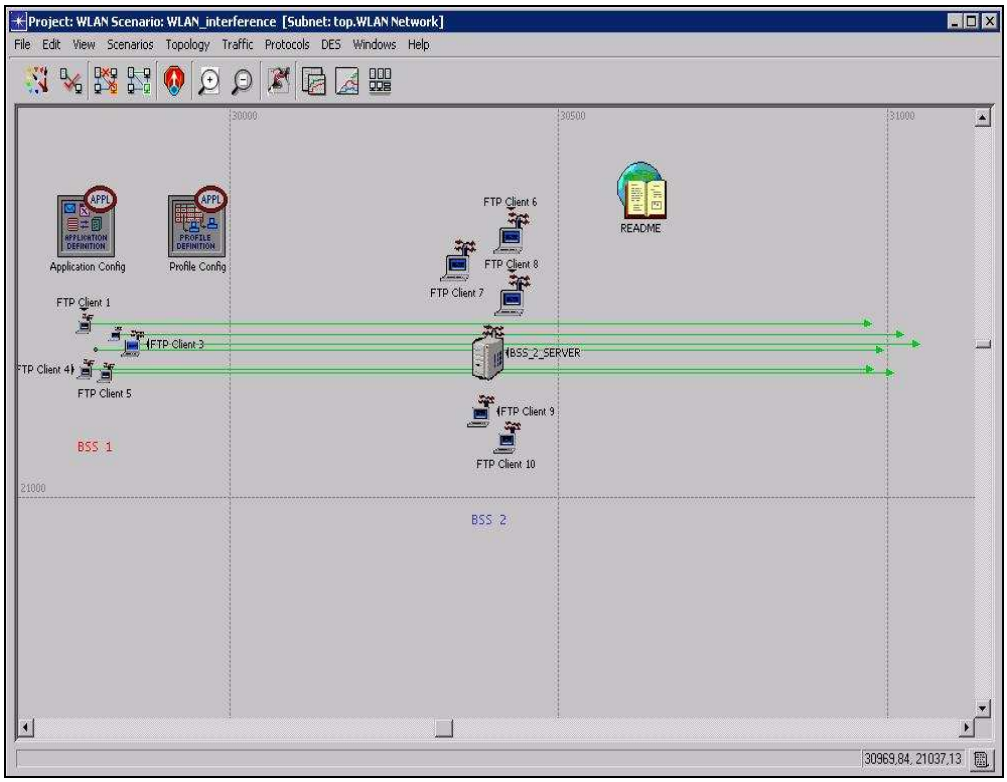

Fig. II.XVII Proyecto ejemplo *WLAN* de OPNET Modeler