Escola Politècnica Superior de Castelldefels
UNIVERSITAT POLITÈCNICA DE CATALUNYA

TRABAJO DE FIN DE CARRERA

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

TITULACIÓN: Ingeniería Técnica de Telecomunicaciones esp. Telemática

AUTOR: Víctor N. García Fernández

DIRECTOR: Eduard Garcia Villegas

FECHA: 5 de septiembre de 2005

Título: Redes mesh basadas en puntos de acceso inteligentes 802.11 open source (II)

Autor: Víctor N. García Fernández

Director: Eduard Garcia Villegas

Fecha: 5 de septiembre de 2005

Resumen

El creciente éxito que las tecnologías inalámbricas basadas en 802.11 experimentan actualmente, está haciendo que cada vez más se opte por implementar soluciones basadas en este estándar para redes de ámbito doméstico, corporativo o incluso metropolitano.

Gran parte de su éxito se basa en la compatiblidad con las redes ethernet actuales, unido a unos precios cada vez más asequibles y la libertad de usarla sin licencia gracias a la banda en la que opera, pero es el mismo éxito el que amenaza el futuro de este tipo de redes ya que 802.11 dispone de un espectro muy limitado y la coexistencia de las cada vez más numerosas redes resulta cada vez más difícil.

La solución pasa por diseñar una sistema que permita realizar una distribución frecuencial eficiente y global para minimizar el impacto de las interferencias entre los distintos nodos.
Con una herramienta de estas características será posible llevar a cabo una distribución dinámica de las frecuencias de modo que nuestro sistema pueda evitar, en la medida de lo posible, interferencias entre sus propios nodos y con sistemas externos que escapen a nuestra gestión.

El objetivo de este proyecto es encontrar una plataforma de código abierto con la suficiente potencia y flexibilidad para llevar a cabo esta tarea y desarrollar sobre ella un algoritmo distribuido capaz de construir, en cada nodo, un mapa completo de la red que reuna los datos necesarios para efectuar la distribución frecuencial.

Title: Smart 802.11 open source access point based mesh networking (II)

Author: Víctor N. García Fernández

Director: Eduard Garcia Villegas

Date: September, 5th 2005

Overview

The increasing success that the wireless technologies based on 802.11 experiment at the moment, makes this tecnology one of the main choices in order to provide solutions oriented to domestic, corporative or even metropolitan networks.

The compatibility between Ethernet and the 802.11 is one of the strongest points and also are the low prices that the devices have now in the market.
The free ISM frequency band also encourages users for moving to 802.11

Lamentably, this success is in danger because the grow in number of 802.11 coexisting networks fills the band.

The objective of this project is to develop a system capable to adapt its transmission to the environment in order to avoid interference between the network nodes and with another 802.11 networks.

This will be done by developing an algorithm capable to build on each node, a full map of the network with all the necessary information to carry out the frecuencial distribution.

ÍNDICE

ÍNDICE 8

CAPÍTULO 1. INTRODUCCIÓN 1

1.1. Motivación del proyecto. 3

1.2. Objetivos del proyecto. 4

1.3. Contenido de la memoria. 5

CAPÍTULO 2. LA PLATAFORMA ACCESS CUBE 7

2.1. MTX-1: El hardware del Access Cube 7
2.1.1. Vista general del sistema. 7
2.1.2. Especificaciones del hardware. 7
2.1.3. Especificaciones energéticas. 8
2.1.4. Posibilidades de expansión. 10
2.1.5. Principal vía de ampliación: El bus Mini PCI. 11

2.2. El software del Access Cube: Distribución linux Nylon. 12
2.2.1. ¿Qué es Nylon? 12
2.2.2. ¿Qué es OpenEmbedded? 12
2.2.3. ¿Cómo vamos a utilizarlos? 12
2.2.4. Descarga y compilación del paquete NylonBuild. 13
2.2.5. El compilador cruzado *mipsel-linux* 15
2.2.6. Resultado de la construcción 15
2.2.7. `Deploy`, el contenedor de resultados 16

CAPÍTULO 3. PUESTA EN MARCHA DEL ACCESS CUBE 18

3.1 Métodos de carga de la imagen 18
3.1.1 Carga de la imagen conectando mediante puerto serie. 18
3.1.2 Carga de la imagen conectando mediante Ethernet. 19

3.2. Utilización de la YAMON Network Console 20
3.2.1. YAMON como herramienta para la prueba de kernels 20
3.2.2. YAMON como herramienta de recuperación del sistema 22

CAPÍTULO 4. DESARROLLO DE SOFTWARE PARA EL ACCESS CUBE. . 25

4.1. Holamundo: Ejemplo de desarrollo paso a paso 25
4.1.1. Primera *fase: Creación del entorno de trabajo.* 25
4.1.2. Segunda fase: Preparación de OpenEmbedded. 26
4.1.3. Tercera fase: Creación del script `.bb` 27
4.1.4. Cuarta fase: Creación del paquete con bitbake. 28

4.2. Instalación de paquetes de software .ipk 30

CAPÍTULO 5. DESARROLLO DE LA PROPUESTA 32

5.1. Presentación del algoritmo WNRA 32

5.2. Contexto de actuación del WNRA..... 32

5.3. Estructura del algoritmo WNRA..... 34
5.3.1. Estructura del programa. 34
5.3.2. Estructura de los datos. 38

CAPÍTULO 6. PRUEBAS DEL ALGORITMO..... 42

6.1. Análisis detallado del proceso de convergencia. 42
6.1.1. Escenario 1: Dos nodos, uno de los cuales implementa WNRA. 42
6.1.2. Escenario 2: Dos nodos que implementan WNRA. 45
6.1.3. Escenario 3: Tres nodos, dos de los cuales implementan WNRA. 47

CAPITULO 7. CONCLUSIONES 51

BIBLIOGRAFIA I REFERENCIAS..... 54

A. Anexo A: código del algoritmo 55

CAPÍTULO 1. INTRODUCCIÓN

Una red mesh inalámbrica no es más que la implementación de la topología de malla sobre una red de nodos con capacidad de comunicación inalámbrica.

Resulta evidente que la topología de malla es la óptima para una red, dado que reduce a 1 el salto máximo para llegar de cualquier emisor a cualquier receptor además de que la caída de un nodo no afecta al resto.

Lamentablemente, no es una topología que se pueda materializar de forma sencilla ya que, hasta ahora, la instalación y mantenimiento del medio en redes cableadas, hacía este tipo de redes inviables.
Pese a estas limitaciones y en ocasiones en las que la seguridad lo requiere, se han empleado soluciones que adoptan topologías híbridas formando semi mallas o mallas parciales permitiendo así redundar las vías de transmisión en caso de fallo.

Actualmente y gracias a la gran proliferación de las redes inalámbricas basadas en 802.11, la apartada topología de malla vuelve a tomar fuerza como posibilidad plausible gracias a que el problema del coste y mantenimiento del cableado desaparece. La topología de malla ha dejado de ser una opción prohibitiva en ese aspecto.

A pesar de que el concepto no es algo nuevo, todavía no se ha introducido en el mercado, ya que las soluciones comerciales se encuentran aún en desarrollo.

De todas formas, en el campo militar, hace años que se utilizan sistemas similares. Este tipo de sistemas permite comunicar entre sí equipos terrestres de forma utilizando dispositivos intercomunicadores que a su vez funcionan de repetidores extendiendo el alcance de la red sin necesidad de una infraestructura troncal común.

Otra ventaja que aporta esta clase de sistemas, es el ahorro energético que supone respecto a sistemas radio convencionales si lo comparamos con sistemas como las two way radios, los sistemas de telefonía celular o los sistemas de red inalámbricos basados en 802.11 con puntos de acceso (APs, en adelante).

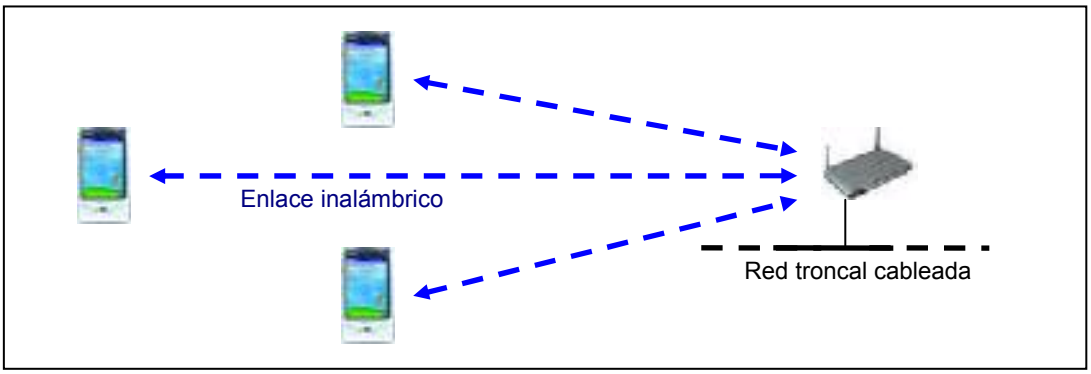


Fig. 1.1. Red inalámbrica 802.11 convencional (modo infraestructura).

Es este último caso sobre el que se centra este proyecto y es que el modelo de red mesh multiplica las posibilidades de las actuales redes LAN inalámbricas.

Encontramos un claro ejemplo en la contraposición de las figuras 1.1 y 1.2, en las que se muestra un sistema 802.11 convencional en modo infraestructura y una red mesh basada en el mismo estándar inalámbrico.

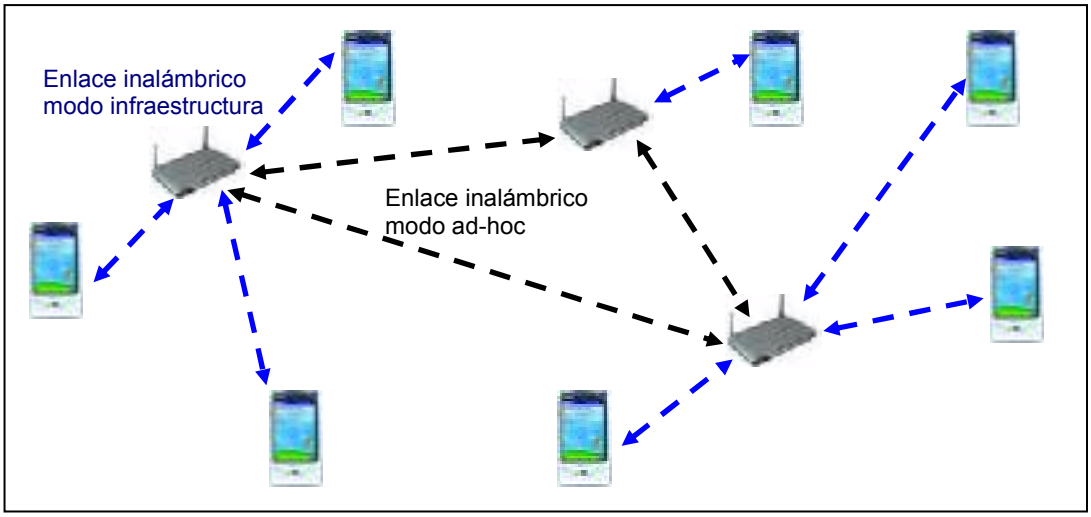


Fig. 1.2. Red mesh inalámbrica 802.11 basada en puntos de acceso inteligentes.

Hay que tener en cuenta, tal y como muestra la figura 1.2, que el modelo de red mesh inalámbrica con el que se trabajará en este proyecto en particular y el más utilizado en general tiene una topología híbrida que no forma una malla completa entre todos los nodos.

Esto es así porque de utilizar una topología estrictamente de malla (malla completa), los terminales móviles necesitarían dos interfaces al igual que los APs (una interfaz operando en modo infraestructura como cliente y otra en modo ad-hoc), lo que provocaría un aumento injustificado de la complejidad en los terminales.

Otra funcionalidad interesante de cara a eliminar la necesidad de disponer de una red troncal cableada, es la posibilidad de extender la red más allá del rango de cobertura del AP ya que en este sistema, la red troncal la forman los APs conectados entre sí por medio de su interfaz en modo ad-hoc.

El crecimiento descontrolado de una red de este tipo daría lugar a una saturación del espectro que perjudicaría las prestaciones de la red en términos de calidad de servicio desde el punto de vista de los terminales móviles y constituiría un grave problema de interferencias en caso de que no existiera la posibilidad de poder ejercer una administración centralizada y única de los puntos de acceso.

La administración centralizada puede ser viable en entornos privados y relativamente estáticos, pero de cara a la creación de redes públicas ciudadanas o para casos en los que haya una alta frecuencia de incorporación y desaparición de nodos, una reconfiguración manual resulta poco escalable.

La solución pasa por crear un sistema capaz de analizar la red en cada momento, reuniendo la información que cada nodo tiene de su entorno para formar un mapa completo de la red mesh sobre el cual poder llevar a cabo estrategias de planificación frecuencial.

La planificación frecuencial permitirá redistribuir los nodos de la red entre los canales disponibles para minimizar las interferencias entre ellos con el fin de mejorar el rendimiento global de la red y en última instancia, la experiencia del usuario.

El objetivo de este proyecto es la creación de un algoritmo que le proporcione a la herramienta de planificación frecuencial, en cada punto de acceso, un mapa completo de la red.

1.1. Motivación del proyecto.

El principal problema con el que se enfrentarán las redes LAN inalámbricas basadas en 802.11 en un futuro próximo, es la saturación del espectro.

La creciente popularidad de esta tecnología, unida al descenso en el precio de los equipos está fomentando un crecimiento bastante acelerado que pone en compromiso la suficiencia de los recursos radio con los que cuenta 802.11.

Las redes inalámbricas basadas en 802.11 con modulación DSSS (Direct Sequence Spread Spectrum) disponen de una banda frecuencial dividida en 14 canales de 5 MHz cada uno que se extienden en la banda libre ISM (International Scientific Medical band) que va de los 2,412 GHz a los 2,484 GHz.

En el caso de Europa, la ETSI (European Telecommunication Stardards Institute) autoriza como operativos los primeros 13 canales, pero a pesar de contar con estos 13 canales, el número real de canales que se pueden utilizar simultáneamente se reduce a 3.

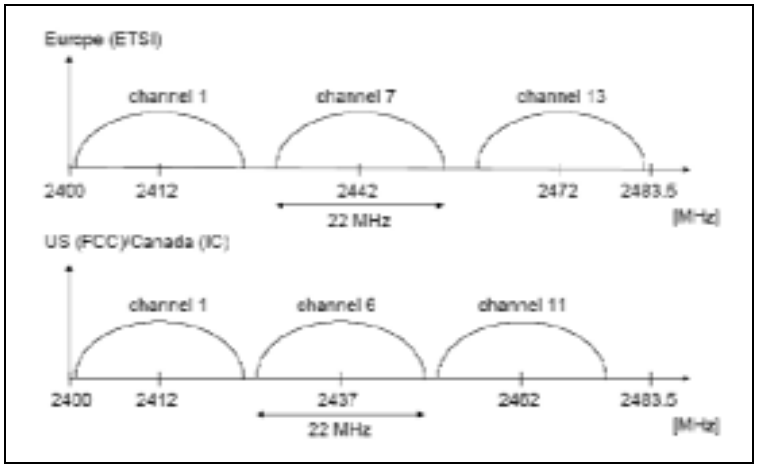


Fig. 1.3. Canales de uso recomendado por la ETSI, la FCC y la IC para Europa, EEUU y Canadá respectivamente.

Esto es debido a que los 22 MHz de ancho de banda de la señal de 802.11 modulada con DSSS exceden de los 5 MHz de los que dispone cada canal.
De modo que la única solución para evitar que las señales se solapen, es separarlas en el espectro.

Para Europa se recomienda usar los canales 1, 7 y 13, lo que inevitablemente nos conduce a preguntarnos: ¿Qué sucede cuando más de 3 redes inalámbricas coexisten de forma cercana?
Inevitablemente, se interfieren.

La solución a este problema es un plan de asignación dinámica de los canales frecuenciales con el objetivo de reducir las interferencias entre los APs y conseguir así mejorar el rendimiento global de la red.

1.2. Objetivos del proyecto.

En este proyecto se utilizará una variante [1] del algoritmo Dsatur [2], cuya función es la de asignar el canal óptimo para un AP, basandose en datos propios y datos de su entorno tales como el número de APs que le rodean, las frecuencias a las que operan, la potencia que recibe de cada uno de ellos y la que cada uno de ellos recibe de él.

El objetivo de este proyecto es encontrar un AP comercial con un sistema operativo basado en código abierto que permita el desarrollo de aplicaciones

con el fin de poder ejecutar en él el algoritmo de asignación frecuencial basado en Dsatur.

Para ello, habrá que desarrollar un algoritmo que provea a Dsatur de la información antes mencionada con su debido formato.

El AP elegido es el Access Cube (también conocido como Meshcube) de 4G Systems y el algoritmo desarrollado, que sido bautizado como WNRA (Wireless Network Roadmap Algorithm), ha sido concebido como un algoritmo distribuido capaz de proveer, en su convergencia, un mapa con todos los nodos y enlaces que integran así como información sobre sus potencias, canal de operación y utilización, datos que el algoritmo variante de Dsatur utilizará en sus cálculos.

Finalmente, podemos establecer que la consecución de objetivos que seguirá este proyecto es la siguiente:

- Elegir un AP comercial basado en código abierto que permita el desarrollo de aplicaciones y tenga la capacidad de proceso suficiente como para ejecutar una aplicación de estas características, que tenga las dos interfaces 802.11 necesarias y que ofrezca las mayores posibilidades de ampliación para usos futuros.
- Familiarizarse con el hardware y el software de la plataforma elegida, documentar sus características de forma detallada y aprender a utilizar sus herramientas de desarrollo, principalmente la herramienta de compilación cruzada.
- Diseñar, escribir y compilar para la plataforma una aplicación, pensando siempre en las limitaciones de un entorno empotrado, capaz de proporcionar los datos necesarios al algoritmo variante de Dsatur para que éste funcione correctamente.
- Comprobar el correcto funcionamiento del algoritmo, analizando en varios escenarios, su comportamiento y el fichero de salida (mapa de la red) que genera. Deberá corresponderse con la realidad del escenario y ser idéntico en todos los nodos.
- Documentar las pruebas realizadas y evaluar los resultados.
- Contraposición de los objetivos inicialmente planteados con los resultados obtenidos y redacción de las conclusiones

1.3. Contenido de la memoria.

La memoria se estructura en 7 capítulos. El primero, a modo de introducción, explica la situación actual en lo que a redes mesh se refiere, expone las motivaciones del proyecto, los objetivos que pretende conseguir y cómo se estructura su documentación a lo largo de la memoria.

El capítulo 2 trata las partes hardware y software del AP elegido. En primer lugar, las características del hardware del dispositivo y sus posibilidades. En segundo lugar, las características del sistema operativo y su estructura, así como el funcionamiento de las herramientas de desarrollo y compilación cruzada.

El capítulo 3 ofrece una toma de contacto con el conjunto de la plataforma Access Cube y documenta el proceso de configuración inicial del AP paso a paso.

El capítulo 4 aborda, con un ejemplo práctico, el procedimiento a seguir en el desarrollo de aplicaciones para el AP, partiendo desde el código fuente hasta conseguir un paquete software instalable en el sistema operativo del AP.

En el capítulo 5 se introduce el algoritmo desarrollado, se explica su funcionamiento y se muestran algunos detalles como los tipos de datos que maneja y el propósito de éstos.

El capítulo 6 presenta varios escenarios sobre los cuales se prueba el algoritmo, se adjuntan y analizan los resultados obtenidos con su correspondiente argumentación en cada caso.

Finalmente, en el capítulo 7, se pueden encontrar las conclusiones acompañadas de una valoración sobre el cumplimiento de los objetivos y una propuesta sobre futuras líneas de investigación.
El capítulo 7 también incluye el estudio de ambientalización.

CAPÍTULO 2. LA PLATAFORMA ACCESS CUBE

2.1. MTX-1: El hardware del Access Cube

2.1.1. Vista general del sistema.

La parte hardware de la plataforma Access Cube (en adelante, AC) se llama MTX-1 y se trata de un sistema empotrado con arquitectura MIPS cuya principal vía de expansión es un bus Mini PCI[1], mediante el cual puede expandir sus funcionalidades con la adición de tarjetas de red inalámbricas.

Como veremos más adelante, el bus Mini PCI no es la única vía de ampliación del MTX-1.

2.1.2. Especificaciones del hardware.

El MTX-1 se presenta en una caja de 7 x 5 x 7 centímetros y está formado por los siguientes componentes:

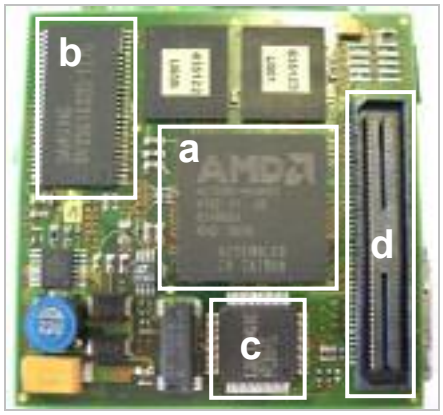


Fig. 2.1. Detalle de la placa base del MTX-1

- Procesador: AMD Alchemy Au1500 400MHz **(a)**
- Memoria RAM: 64 MB **(b)**
- Memoria Flash: 32 MB (28 MB para el sistema de ficheros)
- Controlador Ethernet: 100Mbps Fast Ethernet **(c)**
- USB: 1 x USB 1.1 modo cliente y 1 x USB 1.1 modo anfitrión

[1] http://www.pcisig.com/specifications/conventional/mini_pci/

- Bus Mini PCI: conector para los módulos de expansión del bus **(d)**

2.1.3. Especificaciones energéticas.

El MTX-1 se rige por un perfil de bajo consumo energético acorde con su tamaño y definición como sistema empotrado.

Su consumo en Watios oscila entre los 4 y los 6 W y depende del número de tarjetas Mini PCI instaladas.

Como referencia de consumo podemos considerar el de una tarjeta Intel PRO Wireless 2100 LAN 3B Mini PCI [2], cuyo consumo en función del estado se refleja en la siguiente tabla:

Tabla 2.1. Consumo energético de una tarjeta inalámbrica Mini PCI.

Transmisión	1,6 W
Recepción	1 W
Standby	135 mW
Sleep	45 mW

Tomando, como siempre, el peor caso (el más estricto) vemos que en caso de utilizar el máximo de ocho tarjetas Mini PCI que soporta el AC, éstas sumarían un consumo total de 12,8 Watios.

Hay que tener en cuenta, que en el tipo de escenarios que nos ocupan en este TFC, la configuración habitual será de dos tarjetas, que suman 3,2 Watios.

Como vía de suministro eléctrico, el MTX-1, cuenta de dos posibilidades. Se puede alimentar con un adaptador a la corriente convencional, que da una salida de 9,6 Watios o vía el puerto de red Ethernet, ya que cumple con las especificaciones PowerOverEthernet [3].

[2] http://help.nec-computers.com/eu/pib.asp?second=on&platform=spec_IntelProWireless_LAN3B&layout=1409

[3] Se puede encontrar una asequible referencia sobre Power Over Ethernet en la siguiente dirección: http://poweroverethernet.com/poe/content/view/full/349/

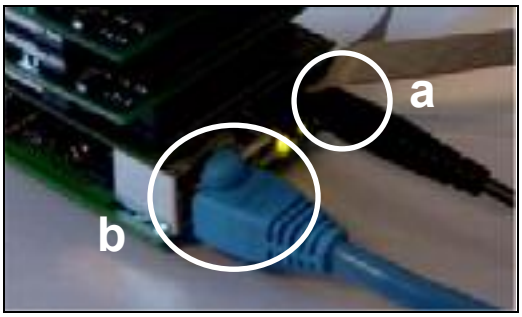


Fig. 2.2. Vías de suministro eléctrico para el MTX-1, adaptador a la corriente (a) y PowerOverEthernet por el puerto Ethernet con un cable de red (b).

La segunda opción, PowerOverEthernet estandarizado por la IEEE con el número 802.3af, es especialmente interesante para casos en los que no se disponga de infraestructuras de suministro de 220 voltios pero sí que se pueda hacer llegar un cableado de UTP Categoría 5.

En un caso cercano a la realidad, en la que el AC se encuentre conectado a una red troncal Ethernet, el mismo cable que transporta los datos puede suministrarle la electricidad necesaria.

En este escenario también se descubre otra ventaja de PowerOverEthernet, ya que el SAI (Sistema de Alimentación Ininterrumpida) se centraliza en los elementos de red tales como switches o routers y esa protección se extiende a la totalidad de elementos que se alimentan eléctricamente de la red Ethernet.

En cuanto a la potencia que se es capaz de suministrar siguiendo esta normativa, nos encontramos con que la diferencia de potencial puede ir de los 44 a los 57 Voltios, siempre por debajo de 60 Voltios, considerado el límite de seguridad para el cable UTP de Categoría 5.
El valor que se usa normalmente es de 48 Voltios (Un poco elevado si tenemos en cuenta que es el doble de lo que estipula la normativa de seguridad industrial para dispositivos de baja tensión).

Para el caso de la intensidad, se aplican diferentes márgenes al límite de seguridad del cable, fijado en 500 Mili Amperios, hasta dejarlo en un máximo de 350 Mili Amperios para este uso.

Para el cálculo de la potencia máxima que puede ser entregada, se utiliza la máxima intensidad (350 Mili Amperios) y el mínimo voltaje que contempla el estándar 802.3af (44 Voltios) menos 7 voltios que disiparía el cable más resistivo (estimado en 20 Ohmios) a la máxima intensidad posible (350 Mili Amperios). Es decir, 37 Voltios.

$$P = V \cdot I = [44\ V - (20\ \Omega \cdot 350\ mA)] \cdot 350\ mA = 12{,}95\ W \quad \textbf{(2.1)}$$

En total, tal y como se demuestra en (2.1) obtenemos una potencia máxima de 12,95 Watios, más que suficiente para alimentar el AC.

2.1.4. Posibilidades de expansión.

El MTX-1 está dotado de un puerto para la conexión a un bus I2C[4] mediante el cual expandir su hardware añadiendo nuevas funcionalidades con dispositivos GPIO (General Purpouse Input/Output).

El bus I2C, propiedad de Philips Electronics N.V., es a grandes rasgos, un protocolo multi-master de comunicaciones sobre un bus serie.

Es muy flexible ya que permite trabajar con dispositivos heterogéneos en cuanto a tiempos de respuesta y reloj, es adaptable ya que se pueden añadir y eliminar dispositivos fácilmente.
Uno de sus puntos fuertes es la robustez ya que permite solucionar cualquier situación de error, bloqueo o pérdida de datos.

En cuanto a aplicaciones prácticas con el AC, un ejemplo puede ser la implementación de periféricos sencillos de entrada y salida como un pequeño teclado para introducir datos o una pantalla LCD serie [5] para visualizar información de estado tal y como se muestra en la Fig. 2.2

Fig. 2.3. Ejemplo de teclado matricial y pantalla LCD incorporable al AC vía bus I2C.

Además de periféricos que ayuden a la interacción con el AC, también se le pueden añadir sensores de temperatura o sistemas servo para controlar motores eléctricos con los que, por ejemplo, poder desplazar el AC.
Un ejemplo de este último caso es el de Wifibot [6], que puede verse en la Fig. 2.4.

[4] Información general y especificaciones del I2C en : http://www.semiconductors.philips.com/markets/mms/protocols/i2c/
[5] Un modelo compatible es el LCD DSM-0822A
[6] http://www.wifibot.com

Fig. 2.4. Wifibot, robot desarrollado por 4G Systems y basado en el hardware del MTX-1.

2.1.5. Principal vía de ampliación: El bus Mini PCI.

El principal método de ampliación del AC es la adición de tarjetas inalámbricas 802.11a, b, g ó soluciones que permitan compatibilidad múltiple entre estándares.

La placa base del MTX-1 incluye un puerto de bus Mini PCI al que se pueden conectar módulos de expansión en cascada para aumentar su capacidad (en número de tarjetas).
Cada módulo de expansión cuenta con 2 ranuras para tarjetas Mini PCI.

En la gama del MTX-1 existen dos versiones de la implementación Mini PCI, la versión 1 (antigua) y la versión 2 (actual).

La principal diferencia entre las dos versiones es que la versión 1 necesita dos tipos de módulos de expansión del bus Mini PCI para manejar las interrupciones IRQ (Interrupt ReQuest) de las tarjetas, ya que la IRQ de cada uno de los dos zócalos Mini PCI está asignada por hardware.

En la versión 2, este problema desaparece y no es necesario combinar dos tipos de módulos de expansión para ocupar todas las IRQs, ya que puede programarse la IRQ deseada mediante el uso de jumpers.

2.2. El software del Access Cube: Distribución linux Nylon.

2.2.1. ¿Qué es Nylon?

Nylon es una distribución linux concebida para trabajar con redes inalámbricas y encaminamiento mesh.

En líneas generales sus principales características giran entorno al trabajo con redes, redes inalámbricas, encaminamiento y encaminamiento mesh.

Está dotada de capacidades de auto configuración y pensada con una especial atención en la seguridad ya que incorpora IPSec y soporta VPN (Virtual Private Network)

Nylon está basada en OpenEmbedded y especialmente creada por 4G Systems para el hardware MTX-1.

2.2.2. ¿Qué es OpenEmbedded?

OpenEmbedded (OE) es una herramienta de desarrollo que permite generar[7] paquetes software para sistemas empotrados.

Ha sido diseñado para manejar diferentes arquitecturas hardware, lo que la hace adaptable a una gran variedad de dispositivos.

2.2.3. ¿Cómo vamos a utilizarlos?

Nuestro objetivo es construir el entorno de desarrollo de OE para Nylon.
Este entorno será nuestra base para desarrollar aplicaciones para el MTX-1. Nos permitirá convertir el código fuente a código binario para el procesador MIPS.

El paquete NylonBuild, contiene todo lo necesario para construir el entorno de desarrollo OE y con él, compilar la distribución Nylon con todos sus paquetes ofreciendo como resultado, en última instancia, una imagen con el sistema operativo para cargar en la memoria del MTX-1.

[7] Mediante un compilador cruzado. En nuestro caso, compilar código en un PC para ser ejecutado en una máquina de otra arquitectura, el destinatario será un sistema empotrado con arquitectura MIPS.

2.2.4. Descarga y compilación del paquete NylonBuild.

En un primer paso, hay que preparar el sistema huésped. Para ello crearemos una carpeta donde alojar el archivo comprimido que contiene el código fuente del NylonBuild, nuestro entorno de desarrollo basado en OE.

El primer paso a realizar es liberar espacio en la partición destino (en caso de que sea necesario) por valor de 4 Gbytes, que es el espacio que ocupará NylonBuild durante la compilación.

Al finalizar el proceso, el entorno de desarrollo OE ocupará unos 2.3 Gbytes.

Al mismo tiempo debemos cerciorarnos de tener los paquetes necesarios para la compilación del NylonBuild, podemos obtenerlos mediante la herramienta apt-get:

- python
- ccache
- patch
- m4
- sed
- docbook
- bison
- make
- openjade
- subversion
- wget
- psyco (optional)
- bzip2
- cvs
- gawk
- libc6-dev
- gcc
- g++
- subversion
- sharutils

Una vez comprobada la instalación de la lista de paquetes requeridos, procedemos a la descarga del fichero comprimido de NylonBuild.

La última versión disponible a 17 de mayo de 2005 es la 0.8 estable [8]

Una vez descargada al directorio donde queremos crear nuestro entorno de desarrollo lo descomprimimos mediante el comando:

```
> tar -zxvf nylon-build-stable-0.8.tar.gz
```

Una vez descomprimido el archivo, accedemos al directorio creado y a continuación podemos proceder de dos formas:

1. Si vamos a tener conexión a Internet durante la compilación del NylonBuild (de 4 a 6 horas dependiendo de la velocidad de la máquina) podemos ejecutar directamente el comando `make`. De este modo, las fuentes se irán bajando a medida que sean necesarias durante el proceso de compilación.
2. Si queremos realizar la compilación sin conexión a Internet, podemos bajar las fuentes previamente mediante el comando:

   ```
   > wget -r -nd -P sources http://meshcube.org/nylon/stable/sources
   ```

   (El tamaño del conjunto de archivos a descargar es de 410 Mbytes).

   Esta orden creará un subdirectorio `sources` dentro del directorio `nylon-build-stable-0.8` donde descargará las fuentes para la compilación.

   Al ejecutar posteriormente el comando `make`, el script de construcción detectará que disponemos de las fuentes ya descargadas y las obtendrá de forma local desde el directorio `sources`.

La ejecución del comando `make` ejecuta por defecto todos los objetivos. Creará, durante varias horas de proceso, la herramienta de cross-compilación y los paquetes necesarios para construir todas las imágenes de Nylon.

Si sólo nos interesa construir un objetivo en particular deberemos utilizar el comando `make` acompañado del nombre del objetivo. Hay 5 en total:

Tabla 2.2. Relación de comandos y objetivos para construir el entorno de desarrollo.

Comando	Objetivo que construye
`> make`	Todos
`> make kernel`	Imagen del *kernel* de Nylon
`> make nylon-image-standard`	Imagen base mínima

[8] Descargable desde http://meshcube.org/nylon/stable/nylon-build-stable-0.8.tar.gz

`> make nylon-image-base`	Imagen estándar
`> make nylon-feed`	Todos los paquetes software.

2.2.5. El compilador cruzado *mipsel-linux*

El paquete NylonBuild incorpora la herramienta de sobre la que se basa todo el proceso de contrucción de los binarios para el AC: El cross-compiler, o compilador cruzado.

Básicamente, se trata de un conjunto de binarios compilados para PC, entre los cuales se encuentran los binarios encargados de compilar código para el lenguaje C y C++.

La compilación manual puede llevarse a cabo de la misma forma en la que se realiza con un compilador para PC convencional ya que en el caso del compilador de C, se trata de una versión del conocido GCC (Gnu Compiler Collection) para arquitectura MIPS:

```
> ./mipsel-linux-gcc prueba.c –o prueba
```

Examinando la información del binario compilado con el comando “file” obtenemos la siguiente información:

ELF 32-bit LSB MIPS-I executable, MIPS, version 1 (SYSV), for GNU/Linux 2.4.0, dynamically linked (uses shared libs), stripped

De ella se extrae que se trata de un binario para linux kernel 2.4.0 de 32 bit compilado para arquitectura MIPS que usa librerías compartidas.

2.2.6. Resultado de la construcción

Tras finalizar la construcción del entorno OE y la creación de las imágenes de Nylon, obtenemos como resultado una estructura de subdirectorios bajo nuestro directorio principal `nylon-build-stable-0.8`.

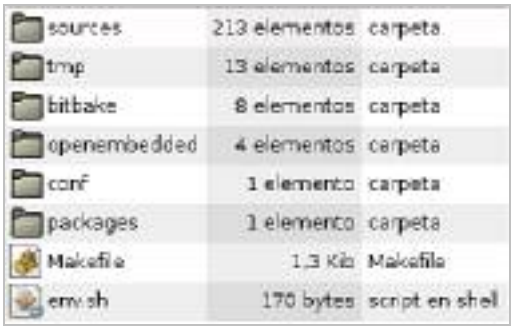


Fig. 2.5. Estructura del directorio principal de construcción `nylon-build-stable-0.8`

2.2.7. `Deploy`, el contenedor de resultados

Dentro del directorio `tmp`, se encuentra el directorio `deploy` donde se alojan los resultados de la compilación de la distribución Nylon.
Consta a su vez de dos subdirectorios: `images` e `ipk`.

2.2.7.1. Subdirectorio `/tmp/deploy/images`, contenedor de imágenes de Nylon.

En el directorio `/nylon-build-stable-0.8/tmp/deploy/images/`, encontramos los siguientes archivos que se corresponden con el juego completo de imágenes de nylon (cuando hacemos un `make` de todos los objetivos).

Los archivos cuya extensión acaba en *.gz* son archivos comprimidos que contienen las imágenes de las que toman el nombre.

Observamos que, principalmente, hay 3 tipos de imágenes: la kernel, la nylon-base y la nylon-standard:

Archivos `kernel-24.27-mtx1_0.8.*`

Son la imagen del kernel o núcleo de la distribución Nylon.

- `.flash.bin`: formato binario, puede ser escrito en la memoria flash desde un AC en funcionamiento.
- `.flash.srec`: formato preparado para ser escrita en la memoria flash de forma permanente vía TFTP.
- `.ram.srec`: formato pensado para cargarse en la memoria RAM vía TFTP.

Archivos `nylon-base-mtx-1_0.8.*`

Son la imagen con los paquetes mínimos para funcionar, no incluye los drivers de Wireless LAN lan ni extras.
Se utiliza como base para la creación de distribuciones personalizadas basadas en Nylon.
Ocupa al rededor de 6,1 MB en la memoria flash del MTX-1 (casi el 22% del total).

- `.imgz`: formato diseñado para ser escrito en la memoria flash mediante el script `install-image`. Contiene tanto el sistema de archivos como el kernel.

- `.rootfs.jffs2`: formato preparado para instalar el sistema de archivos mediante el script `install-image` o por NFS.

- `.rootfs.srec`: formato preparado para instalar el sistema de archivos vía TFTP.

Archivos `nylon-standard-mtx-1_0.8.*`

La imagen estándar de Nylon para el MTX-1, ocupa aproximadamente 11 MB en la memoria flash (ocupa menos del 40% del total) e incorpora, además de los paquetes base, los drivers de Wireless LAN, enrutado Mesh, etc.

Los tipos de extensión de las imágenes `nylon-standard-mtx-1_0.8.*`, son los mismos que los de las imágenes `nylon-base-mtx-1_0.8.*`.

2.2.7.2. Subdirectorio `/tmp/deploy/ipk`, contenedor de paquetes de Nylon.

En el directorio `/nylon-build-stable-0.8/tmp/deploy/ipk/`, encontramos el conjunto de 2189 paquetes iPKG[9] con todo el software, librerías y documentación para Nylon.
En el mismo directorio se encuentra un fichero con la lista de paquetes, su nombre y una breve descripción de su funcionalidad, así como la URL para su descarga.

[9] Itsy Package, más información en http://www.uk-dave.com/tutorials/zaurus/ipkg.shtml

CAPÍTULO 3. PUESTA EN MARCHA DEL ACCESS CUBE

3.1 Métodos de carga de la imagen

Antes de trabajar con el MTX-1 es recomendable realizar una actualización del sistema operativo. Linux es un sistema en constante mejora, lo que comporta cambios frecuentes y Nylon no es una excepción.

Para proceder a cargar la imagen de la última versión[10] que hemos construido, podemos optar por los dos métodos universales que se aplican a la mayoría de equipos de red: por conexión serie o mediante red Ethernet.

A continuación se pasa a detallar los dos métodos.

3.1.1 Carga de la imagen conectando mediante puerto serie.

Para este procedimiento utilizaremos el cable serie especial que 4G Systeme suministra con el AC.
Estos pasos serán los que seguiremos en caso de que no tengamos acceso al AC vía red Ethernet, porque tenga la interfaz de red Ethernet caída.

La herramienta YAMON es bastante potente y su funcionamiento se detalla en el siguiente capítulo. Para el caso particular que nos ocupa sólo la utilizaremos para habilitar la interfaz Ethernet y poder así cargar la imagen de Nylon.

El proceso para conectar con el AC mediante terminal serie está descrito en el apartado 4.1. Tras realizar la conexión con éxito, una vez dentro de la consola YAMON, basta con ejecutar el siguiente comando:

```
YAMON> set ipaddr 192.168.0.250
```

Con esta orden, YAMON activará el puerto Ethernet y le dará una IP para que podamos entrar en el AC por red.

A continuación, podremos seguir los pasos descritos en el siguiente apartado tanto por el terminal serie en el que nos encontramos como por SSH.

[10] Pese a que pueden aparecer nuevas versiones a menudo, se recomienda utilizar como norma general las calificadas como estables, ya que garantizan una fiabilidad superior al resto.

3.1.2 Carga de la imagen conectando mediante Ethernet.

En el caso de que tengamos conexión por red con el AC, siempre resulta más cómodo actualizarlo por red, ya que no necesitamos acceder físicamente a él, sólo estar conectados a una misma red Ethernet.

El procedimiento se lleva a cabo mediante el script `install-image`[11].
El primer paso a dar es copiarlo al AC, para ello usaremos SSH[12], llamando al comando `scp` desde el directorio donde hayamos descargado el script:

```
> scp install-image 192.168.0.250:/root/install-image
```

(Suponemos 192.168.0.250 la dirección IP estándar del puerto Ethernet en el AC).
A continuación se inicia la sesión de copia por SSH y nos pide el password de root para el AC.

Una vez copiado el script en el AC, podemos proceder de dos maneras según dónde se aloje la imagen de Nylon:

- Si disponemos de un servidor web en nuestra red, podemos utilizarlo para servir la imagen de nylon.
  Para este caso, conectamos por SSH al AC y ejecutamos el script con los siguientes parámetros:

  ```
  > install-image –s 192.168.0.100/nylon/nylon.imgz
  ```

  Nótese que se ha renombrado el archivo `.imgz` por comodidad y que el prefijo http:// se sobreentiende. La dirección IP que aparece es de la máquina que tiene el servidor web.

- Si no disponemos de servidor web, podemos copiar el archivo de imagen al AC:

  ```
  > scp nylon.imgz 192.168.0.250:root/nylon.imgz
  ```

  Y a continuación, conectar por SSH al AC y ejecutar el script indicando la ruta local donde puede encontrar la imagen:

  ```
  > install-image –l /root/nylon.imgz
  ```

En ambos casos el resultado es el mismo, conseguimos instalar la distribución Nylon en el MTX-1.
Ya tenemos nuestro AC operativo.

[11] Última versión disponible en: http://meshcube.org/nylon/utils/install-image
[12] Secure Shell, Linux incorpora la versión OpenSSH: http://www.openssh.com/faq.html#1.0

3.2. Utilización de la YAMON Network Console

Yamon es un software básico de arranque que se encarga de cargar el kernel de Nylon en la memoria del MTX-1 durante el arranque.

Funciona de forma similar a GRUB o Lilo[13] en un PC, pero nos ofrece opciones más potentes y se encuentra a salvo en una parte imborrable de la memoria flash.

Esta herramienta se usa principalmente para dos objetivos:

- Cargar, con fines experimentales, versiones nuevas o modificadas del kernel del sistema operativo directamente en RAM (sin alterar el contenido de la memoria flash).

  De este modo conseguimos arrancar el AC con la versión del kernel que necesitamos, y al reiniciar el MTX-1 volverá a utilizar la versión de kernel almacenada en la flash.

- Recuperar ACs que hayan sufrido algún error durante el proceso de grabación de la imagen en flash[14].

  Ante errores de este tipo, el MTX-1 sólo responde mediante el puerto serie gestionado por Yamon, que le da las funcionalidades básicas para permitir un rescate del sistema.

Cabe destacar, que de los formatos de imagen detallados en el apartado 2.2.7.1, YAMON sólo soporta el manejo de imágenes con formato srec.

A continuación detallamos cada uno de los casos descritos.

3.2.1. YAMON como herramienta para la prueba de kernels

Para conectar con el AC mediante puerto serie, hace falta un cable propietario que facilita 4G Systeme y que lleva una electrónica interna que adecua los niveles de señal del puerto estándar serie de un PC a los propios del MTX-1.

Exteriormente es un cable con un conector hembra RS-232 en un extremo, y por el otro lado un conector hembra para 12 pines.

---

[13] Gestores de arranque del sistema utilizados por Linux. Pueden encontrarse referencias en http://tldp.org/HOWTO/LILO.html, para Lilo, y en http://www.gnu.org/software/grub/ para Grub.

[14] El proceso descrito en el Capítulo 3

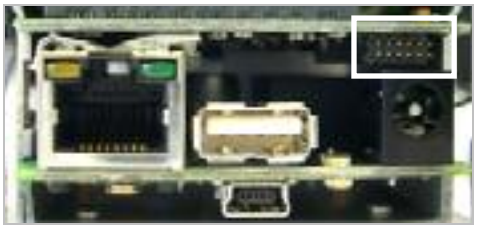


Fig 3.1. Conector para cable serie en el MTX-1.

Para conectar con el AC, podemos usar cualquier software para comunicaciones serie.

La configuración de la conexión se hace siguiendo los siguientes parámetros:

Tabla 3.1. Tabla de configuración del puerto serie para uso con el MTX-1.

Bits por segundo	115.200
Bits de datos	8
Paridad	Ninguno
Bits de parada	1
Control de flujo	Hardware

Una vez conectado al PC mediante el cable serie, se alimenta y el MTX-1 arranca.
Durante el arranque se debe presionar Ctrl+C para acceder al intérprete de comandos de Yamon, tal y como muestra la figura 5.1.

```
YAMON ROM Monitor, Revision 02.17mtx1.
Copyright (c) 1999-2000 MIPS Technologies, Inc. - All Rights Reserved.

For a list of available commands, type 'help'.

Compilation time =            Aug 19 2003  13:49:58
MAC address =                 00.0e.56.00.01.7b
Processor Company ID =        0x03
Processor ID/revision =       0x02 / 0x02
Endianness =                  Little
CPU =                         324 MHz
Flash memory size =           32 MByte
SDRAM size =                  64 MByte
First free SDRAM address =    0x8008ac24

Environment variable 'start' exists. After 2 seconds
it will be interpreted as a YAMON command and executed.
Press Ctrl-C to bypass this.

YAMON>
```

Fig 3.2. Arranque y carga del intérprete de comandos de Yamon.

El proceso que deberemos realizar a continuación es, básicamente, cargar una configuración IP mínima para escribir imagen del kernel en la RAM del MTX-1 desde un servidor TFTP[15].

Una vez dentro de la consola, debemos ejecutar los siguientes pasos:

1. Indicamos la dirección IP del servidor TFTP
   YAMON> set bootserver 192.168.0.150

2. Damos una dirección IP al puerto Ethernet del MTX-1 para que pueda conectarse al servidor.
   YAMON> set ipaddr 192.168.0.250

3. Ejecutamos el comando de carga *load* seguido del nombre de la imagen del kernel. La barra representa el directorio raíz del servidor TFTP.
   YAMON> load /kernel.ram.srec

   Nótese que la imagen que utilizamos en esta ocasión es la `kernel-2.4.27-mtx-1_0.8.ram.srec` ya que está preparada para ser escrita en memoria RAM y transferida por TFTP.

4. A continuación y una vez transferida y cargada en RAM la imagen del kernel, sólo queda ejecutarla.
   YAMON> go . / root=dev/mtdblock/0

   Con esta instrucción, comenzamos a ejecutar desde el primer bloque de la memoria RAM del MTX-1.

Una vez ejecutado el kernel, ya estamos en disposición de utilizarlo durante la sesión actual, ya que los cambios en RAM se pierden al reiniciar el hardware.

3.2.2. YAMON como herramienta de recuperación del sistema

En este apartado nos movemos en una dirección diferente al del apartado anterior, aunque comparten ciertas similitudes.

Yamon, como gestor de arranque que es, puede ser utilizado para cargar imágenes no sólo en la RAM y con propósitos de prueba sino que también es capaz de escribir en la memoria flash.

[15] Recomiendo el uso del software tftpd32, el mejor servidor de TFTP gratuito, para el caso de utilizar el entorno Windows. Disponible en http://tftpd32.jounin.net/

Utilizando el tipo de imagen correcta para el sistema de archivos y para el kernel, seremos capaces de cargar en la flash el sistema operativo Nylon para convertir el MTX-1 en un AC totalmente funcional.

Estos pasos pueden servirnos en tres situaciones:

- Caso 1:
  Error durante la escritura de la imagen Nylon en la memoria flash del MTX-1. (Nylon no será capaz de arrancar)

- Caso 2:
  MTX-1 nuevo.
  (Si tenemos un dispositivo nuevo con la memoria en blanco).

- Caso 3:
  MTX-1 con una imagen de Nylon errónea.
  (Si, por error, hemos escrito en flash la imagen base de Nylon en vez de la estándar, no podremos acceder al AC ya que no dispone de controladores de red)

Para comenzar con el proceso, es necesario configurar una conexión serie, igual que en el apartado anterior.

Si nuestro caso es el número 1, al reiniciar el MTX-1 obtendremos un error semejante al representado en la figura 5.3.

```
* Exception (user) : Reserved instruction *

CAUSE    = 0x00808028  STATUS   = 0x00000002
EPC      = 0xbfd00000  ERROREPC = 0x40400001
BADVADDR = 0x80011090

$ 0(zr):0x00000000  $ 8(t0):0x00000000  $16(s0):0x00000000  $24(t8):0x00000000
$ 1(at):0x00000000  $ 9(t1):0x00000000  $17(s1):0x00000000  $25(t9):0x00000000
$ 2(v0):0x00000000  $10(t2):0x00000000  $18(s2):0x00000000  $26(k0):0x00000000
$ 3(v1):0x00000000  $11(t3):0x00000000  $19(s3):0x00000000  $27(k1):0x00000000
$ 4(a0):0x00000002  $12(t4):0x00000000  $20(s4):0x00000000  $28(gp):0x00000000
$ 5(a1):0x80080af0  $13(t5):0x00000000  $21(s5):0x00000000  $29(sp):0x8008ac14
$ 6(a2):0x80043b48  $14(t6):0x00000000  $22(s6):0x00000000  $30(s8):0x8008ac14
$ 7(a3):0x04000000  $15(t7):0x00000000  $23(s7):0x00000000  $31(ra):0x8002e7e0

YAMON>
```

Fig 3.3. Error durante el arranque del sistema.

Para solucionar el problema y volver a escribir en la memoria flash las imágenes del kernel y el sistema de ficheros, debemos seguir los pasos descritos a continuación:

1. Borramos las zonas de la memoria flash dedicadas al kernel y al sistema de archivos.

```
YAMON> erase 0bfd00000 0xf0000
YAMON> erase -s
```

El segundo comando tardará sensiblemente más que el primero ya que borrará por completo los 28 MB de la memoria flash destinados al sistema.

2. Indicamos la dirección IP del servidor TFTP
   ```
   YAMON> set bootserver 192.168.0.150
   ```

3. Damos una dirección IP al puerto Ethernet del MTX-1 para que pueda conectarse al servidor.
   ```
   YAMON> set ipaddr 192.168.0.250
   ```

4. Ahora cargamos las versiones `.srec` (preparadas para transferencia por TFTP) de la imagen del sistema de archivos y del kernel.
   ```
   YAMON> load /standard.srec
   ```

   ```
   YAMON> load /kernel.flash.srec
   ```

   Nótese que se han utilizado las imágenes `nylon-standard-mtx-1_0.8.srec` y `kernel-2.4.27-mtx-1_0.8.flash.srec` debidamente renombradas para mayor comodidad.

Una vez cargadas ambas imágenes, ejecutamos el comando de arranque del sistema.

```
YAMON> $start
```

Si permanecemos conectados por el Terminal serie, visualizaremos el proceso de arranque de Nylon.
Una vez finalizado, generará en último lugar, las claves RSA y DSA para el servidor de SSH y mostrará el prompt pidiendo login (contraseña de root originalmente en blanco).

Llegados a este punto, la imagen del sistema operativo Nylon se encuentra cargada en el hardware MTX-1, con lo que podemos empezar a trabajar con el AC.

CAPÍTULO 4. DESARROLLO DE SOFTWARE PARA EL ACCESS CUBE.

4.1. Holamundo: Ejemplo de desarrollo paso a paso

A continuación abordaremos el proceso a seguir durante el desarrollo de software con el fin de crear paquetes de software para Nylon.

Realizaremos un proyecto a modo de ejemplo, basado en la programación, compilación, empaquetado e instalación de una aplicación sencilla.

Para la realización del proyecto usaremos el sistema CVS[16].

4.1.1. Primera *fase: Creación del entorno de trabajo.*

En primer lugar crearemos un subdirectorio para CVS en nuestro directorio de trabajo, si no lo tenemos ya.

A continuación, entramos en el directorio creado y para iniciar el proyecto escribimos la siguiente línea:

```
> cvs –d /home/victor/cvs init
```

Nótese que la ruta indicada debe ser la trayectoria hasta nuestro directorio `cvs` desde la raíz.

Esta orden creará un directorio llamado CVSROOT, donde guardará listados de los proyectos iniciados así como otra información de relevancia para el sistema CVS.

En segundo lugar, nos situamos en el directorio que contiene nuestras fuentes y ejecutamos el comando `cvs` con los siguientes parámetros:

```
> cvs –d /home/victor/cvs import holamundo TFC start
```

Llegados a este punto, hemos añadido un proyecto llamado “holamundo” a la lista de CVS. TFC es el nombre del autor o empresa que desarrolla ese código. Las fuentes de nuestro programa en desarrollo serán importadas a la base de datos de CVS para que cualquier usuario pueda trabajar en el proyecto “holamundo”.

En nuestro caso puntual, las fuentes del proyecto “holamundo” están compuestas por un solo archivo escrito en c:

[16] Concurrent Version System, más información en https://www.cvshome.org/

```
#include <stdio.h>
int main ( int argc, char *argv[] )
{
      printf ( "Cubo dice: ¡Hola Mundo!\n" );
      return 0;
}
```

Fig 4.1. Código de ejemplo `holamundo.c`

Una vez publicado el código fuente en la base de datos de CVS, el último paso es realizar una copia local para trabajar con él.

Fuera del directorio donde se aloja el código fuente crearemos nuestra copia de trabajo con:
> cvs –d /home/victor/cvs checkout holamundo

Con este comando, se crea una carpeta local con una copia del proyecto "holamundo", cuyo nombre está dado de alta en la base de datos de CVS.

De cara a trabajar en un proyecto con varios participantes, CVS permite publicar cambios para hacerlos disponibles al resto de desarrolladores así como actualizar las versiones locales del proyecto.

Puede consultarse un libro[17] o tutorial con el fin de conocer algunas de las funcionalidades adicionales de CVS.

4.1.2. Segunda fase: Preparación de OpenEmbedded.

El segundo paso para lograr nuestro objetivo, es la preparación del entorno OE para la creación de paquetes software para Nylon.

Para ello debemos crear, en la carpeta `nylon-build-stable-0.8`, un directorio llamado `packages`, donde alojaremos los ficheros `.bb` para cada proyecto.

Una vez creado el directorio, debemos configurar OE para que busque los archivos `.bb` en el directorio que acabamos de crear. Para ello añadimos la siguiente línea al archivo `/nylon-build-stable-0.8/conf/local.conf`:

```
BBFILES += "${OEBASE}/packages/*.bb"
```

[17] Consultar Bibliografia complementaria. Ref: CVS

Nótese que al usar la sintaxis "+=" estamos añadiendo un nuevo recipiente para la localización de archivos `.bb` y que la variable `${OEBASE}` en nuestro caso equivale a */nylon-build-stable-0.8/.*

Una vez efectuado el cambio en la configuración de OEprocedemos a crear el archivo `.bb` para nuestro proyecto.

4.1.3. Tercera fase: Creación del script `.bb`

El último paso previo a la propia creación del paquete software de nuestro proyecto es la creación del script `.bb` que interpretará `bitbake` y que contiene las instrucciones a seguir para la creación de nuestro paquete software en particular.

En primer lugar, debemos rellenar los campos descriptivos:

- `DESCRIPTION` (Descripción sobre la función del paquete software)
- `SECTION` (Categoría de software en la que se clasifica)
- `MANTAINER` (Responsable de la aplicación y de sus actualizaciones).
- `LICENSE` (Tipo de licencia que se aplica al paquete software)
- `SRC_URI` (ruta donde se encuentra el código fuente del proyecto)

En segundo lugar, debemos escribir las funciones que se corresponden a cada paso a seguir en la creación de un programa a partir de código fuente. (fetch, unpack, match, configure, compile, stage, install, package)

En nuestro caso particular, definimos la función de instalación `do_install` de la siguiente forma:

```
do_install(){
      install -d ${D}${bindir}
      install ${S}/holamundo ${D}${bindir}/
}
```

- Con la primera línea, se crea el directorio `bin` en la máquina destino (donde instalamos el software), por si no estaba creado ya.
- En la segunda, se copian los binarios del paquete al directorio `bin` de la máquina destino.

El resto de funciones se ejecutarán con las opciones por defecto.

El archivo `holamundo.bb` tendría, una vez finalizado, el aspecto que se muestra en la figura 4.2.

```
DESCRIPTION = "Victor OpenEmbeddded first test-package"
SECTION = "base"
PRIORITY = "optional"
```

```
MANTAINER = "Victor Garcia Fdez. <vgfdez@gmail.com>"
LICENSE = "GPL"
CVSDATE = "now"
SRC_URI =
"cvs://127.0.0.1/home/victor/TFC/cvs/;module=holamundo;method=ext;rsh=
ssh"
S = "${WORKDIR}/${PN}"

do_install(){
      install -d ${D}${bindir}
      install ${S}/holamundo ${D}${bindir}/
}
```

Fig 4.2. Archivo `.bb` de ejemplo para el proyecto `holamundo`.

En el script `.bb` se utilizan diversas variables, el significado de las cuales es el reflejado en la tabla 4.1

Tabla 4.1. Variables presentes en un script `.bb`

Variable	Significado
`S`	Source, directorio que contiene el código fuente.
`D`	Destino de los binarios compilados.
`PN`	Package Name, nombre del paquete software a crear. En nuestro caso es el del proyecto holamundo.
`WORKDIR`	Directorio de trabajo donde se encuentran los proyectos de CVS.
`bindir`	Directorio recipiente para binarios, por lo general, en los sistemas linux es /bin/.

Una vez creado el script `.bb` debemos hacer que la herramienta `bitbake` lo interprete.
Antes de poder usar `bitbake` debemos ejecutar el script de modificación de las variables de entorno que se encuentra en el directorio `nylon-build-stable-0.8:`

```
> . env.sh
```

Una vez ejecutado, se modifican las variables del sistema para usar el entorno de OE. Ahora el sistema sabe dónde encontrar el entorno de desarrollo para poder utilizar `bitbake`.

4.1.4. Cuarta fase: Creación del paquete con bitbake.

4.1.1.1. ¿Qué es `bitbake`?

Bitbake es una herramienta para la ejecución de tareas, que comúnmente se utiliza para construir paquetes software, ya que el proceso requiere de la consecución de una serie de pasos: fetch, unpack, patch, configure, compile, stage, install, package.

A grandes rasgos, su tarea es descargar desde un repositorio de paquetes o un servidor de CVS alojado en internet o en una red local el código fuente de un proyecto de software, prepararlo para su construcción, compilarlo y empaquetarlo para que pueda ser instalado en un sistema destino.

4.1.1.2. Ejecución de `bitbake` con el script `.bb`

Llegados a este punto, es el momento de llamar a `bitbake` para que interprete nuestro script `.bb`:

```
> bitbake –b /home/victor/TFC/nylon-build-stable-0.8/packages/holamundo.bb
```

Durante la ejecución, bitbake irá ejecutando por orden cada una de las funciones que completan la creación del paquete software.

En la fase de `fetch`, `bitbake` se comporta de forma especial, ya que el entorno de desarrollo OE que proporciona 4G Systeme busca las fuentes, en primer lugar, en meshcube.org.
Esta operación tiene el fin de conseguir la versión más actualizada del proyecto, ya que se usa meshcube.org como repositorio para los paquetes de software más comunes.

Al no encontrar el proyecto `holamundo` en los repositorios de meshcube.org, lo buscará en la base de datos del servidor de CVS para obtener una copia de las fuentes.

Una vez finalizados todos los pasos (algunos pasos no son siempre necesarios, como el match, para aplicar parches en el software que empaquetaremos).

Finalmente, el resultado del proceso (el paquete `.ipk` de nuestro proyecto) se encuentra dentro del directorio contenedor de paquetes `/nylon-build-stable-0.9/tmp/deploy/ipk` bajo el nombre de `holamundo_1.0-r0_mipsel.ipk`.

Ahora tenemos un paquete de software listo para instalar en cualquier AC.

4.2. Instalación de paquetes de software .ipk

Llegados a este punto, el procedimiento es el mismo que se sigue para instalar cualquier paquete de software como firefox o ethereal.

En primer lugar debemos descargar al AC el paquete de software que hemos creado, en nuestro caso lo transferimos del PC al AC:

```
> scp holamundo_1.0-r0_mipsel.ipk 192.168.0.250:/root/ holamundo_1.0-r0_mipsel.ipk
```

Una vez copiado, conectamos por SSH al AC, accedemos al directorio root (dónde lo acabamos de transferir) y lo instalamos con la herramienta `ipkg`:

```
> ipkg install holamundo_1.0-r0_mipsel.ipk
```

La instalación seguirá los pasos indicados en nuestro script `.bb`, copiando los archivos binarios (en este caso sólo uno: `holamundo`) al directorio `/bin/` del AC.

Como Nylon incluye, como cualquier distribución de linux, el directorio `/bin/` en la variable de sistema PATH, ahora podemos ejecutar el comando `holamundo` desde cualquier directorio.

```
> holamundo
  Cubo dice: ¡Hola Mundo!
>
```

Llegados a este punto, hemos visto como construir una aplicación para Nylon preparada para funcionar sobre el hardware del MTX-1 y a instalarla lista para ejecución en el AC.

Para la creación de los paquetes necesarios para este proyecto, los pasos a seguir seran los mismos que en el ejemplo anterior.

```
DESCRIPTION = "Algoritmo variante de dSatur"
SECTION = "base"
PRIORITY = "optional"
MANTAINER = "Eduard Garcia Villegas < eduardg@mat.upc.edu>”
LICENSE = "closed"
CVSDATE = "now"
SRC_URI =
"cvs://127.0.0.1/home/victor/TFC/cvs/;module=dsatur;method=ext;rsh=ssh
"
S = "${WORKDIR}/${PN}"

do_compile() {
      oe_runmake exemple
```

```
}

do_install(){
      install -d ${D}${bindir}
      install ${S}/dsatur ${D}${bindir}/
}
```

Fig 4.3. Archivo `.bb` para la creación del paquete Dsatur.

En la figura anterior se muestra el script de bitbake adecuado para la correcta construcción del paquete de software del algoritmo variante de Dsatur.

En el caso del script .bb para la creación del paquete del algoritmo desarrollado durante el proyecto, el procedimiento es exactamente el mismo, ya que sólo cambia el nombre del proyecto CVS (“wnra”), el objetivo compilado en el makefile y la descripción:

DESCRIPTION = "Wireless Network Roadmap Algorithm"
MANTAINER = "Víctor N. Garcia Fdez. < vgfdez@gmail.com>”

Llegados a este punto, prodemos construir e instalar cualquier paquete de software en el AC partiendo de su código fuente.

CAPÍTULO 5. DESARROLLO DE LA PROPUESTA

5.1. Presentación del algoritmo WNRA

El algoritmo desarrollado durante este proyecto, consta de un solo ejecutable, escrito en lenguaje C, que usa memoria dinámica, sockets unix y llamadas a sistema de bajo nivel, así como multiproceso basado en threads sincronizados , que efectúan las tareas de cliente y servidor simultáneamente.

Se escogió el lenguaje C por su indiscutible potencia y por tratarse de un lenguaje totalmente abierto, libre de ataduras comerciales o de propiedad que sí suceden en otros lenguajes que, a priori, podrían resultar más cómodos para el desarrollo de la mayoría de aplicaciones modernas orientadas a red, como es el caso de Java[18].

Relacionado con este último punto y dentro del marco de open source que caracteriza este proyecto, no podría ser de otra manera.
Por otra parte el AP escogido sólo cuenta con herramientas de compilación cruzada para C y C++.

El WNRA toma el nombre de las siglas Wireless Network Roadmap Algorithm, ya que, en su convergencia, pretende conseguir que cada nodo tenga un "mapa de carreteras" de la red completa, dándole a conocer tanto el resto de nodos de la red y la forma en la que están conectados entre sí, como las frecuencias en las que operan y el nivel de señal que cada uno recibe de sus vecinos.

5.2. Contexto de actuación del WNRA.

Para entender por qué ha sido necesaria la creación del WNRA, hay que situar al lector en el contexto de una red inalámbrica con múltiples puntos de acceso y multitud de clientes conectados a éstos.
Un buen ejemplo podría ser una red inalámbrica corporativa.

Supongamos una red inalámbrica como la de la siguiente figura.

[18] Léase "La trampa de Java" http://www.gnu.org/philosophy/java-trap.es.html

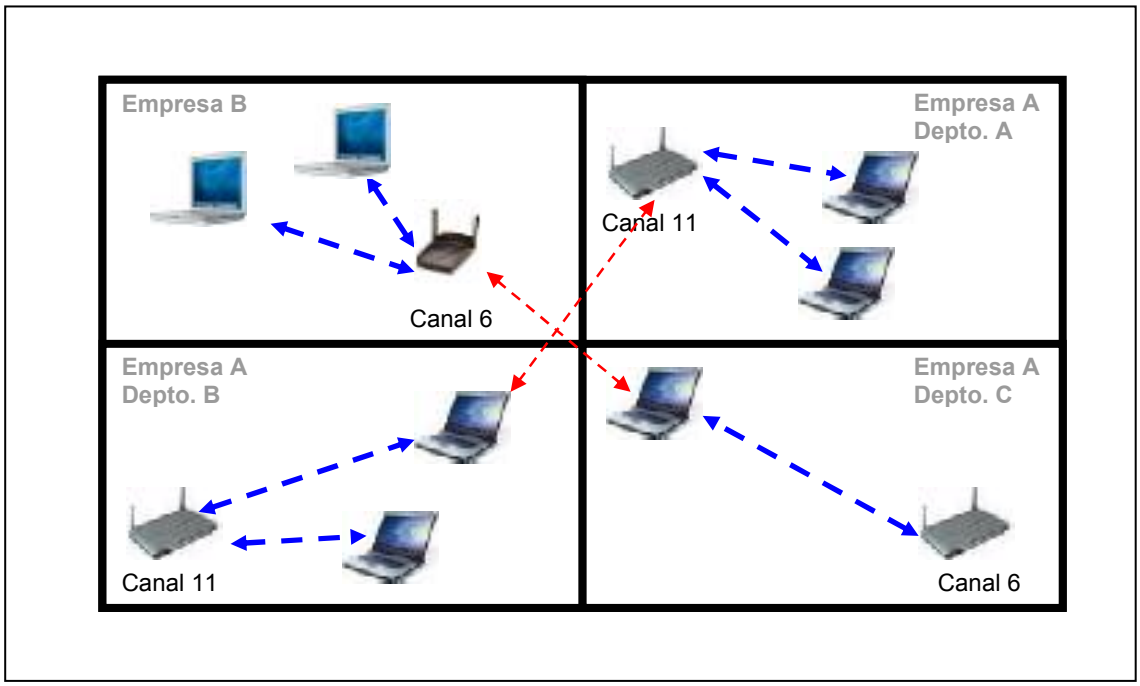


Fig 5.1. Escenario: Red inalámbrica corporativa.

Como se puede observar tiene varios problemas de interferencias, representados con las flechas rojas, todos ellos frecuentes en casos en los que varias redes inalámbricas se ven obligadas a coexistir.

Los problemas que afectan a los APs de la red corporativa pueden ser fácilmente solucionados si el administrador de la red realiza una planificación frecuencial más oportuna y configura cada uno de los APs de forma que no interfieran unos con otros.

Bastaría con separar los canales frecuenciales de cada AP hacia cada uno de los tres recomendados por la ETSI para Europa: 1, 7 y 13.
De esta forma evitaríamos la interferencia interna y el único problema sería que de cara a la ampliación de la red con nuevos APs, no dispondríamos de frecuencias donde situarlos para que coexistieran con los actuales sin sufrir interferencias o solapaciones de las señales entre canales próximos.

El problema que no podemos solucionar es el de las interferencias externas, ya que no tenemos acceso a la configuración de los APs ajenos para realizar una planificación frecuencial conjunta.

Como sólo podemos ejercer control sobre nuestra propia red corporativa, necesitamos un sistema que planifique una distribución frecuencial que evite, en la medida de lo posible, los efectos negativos de la saturación del espectro y que a la vez sea dinámico para poder adaptar esa planificación a las condiciones del entorno y de los sistemas externos que escapan a nuestra gestión.

5.3. Estructura del algoritmo WNRA

5.3.1. Estructura del programa.

5.3.1.1. *Diagrama de flujo*

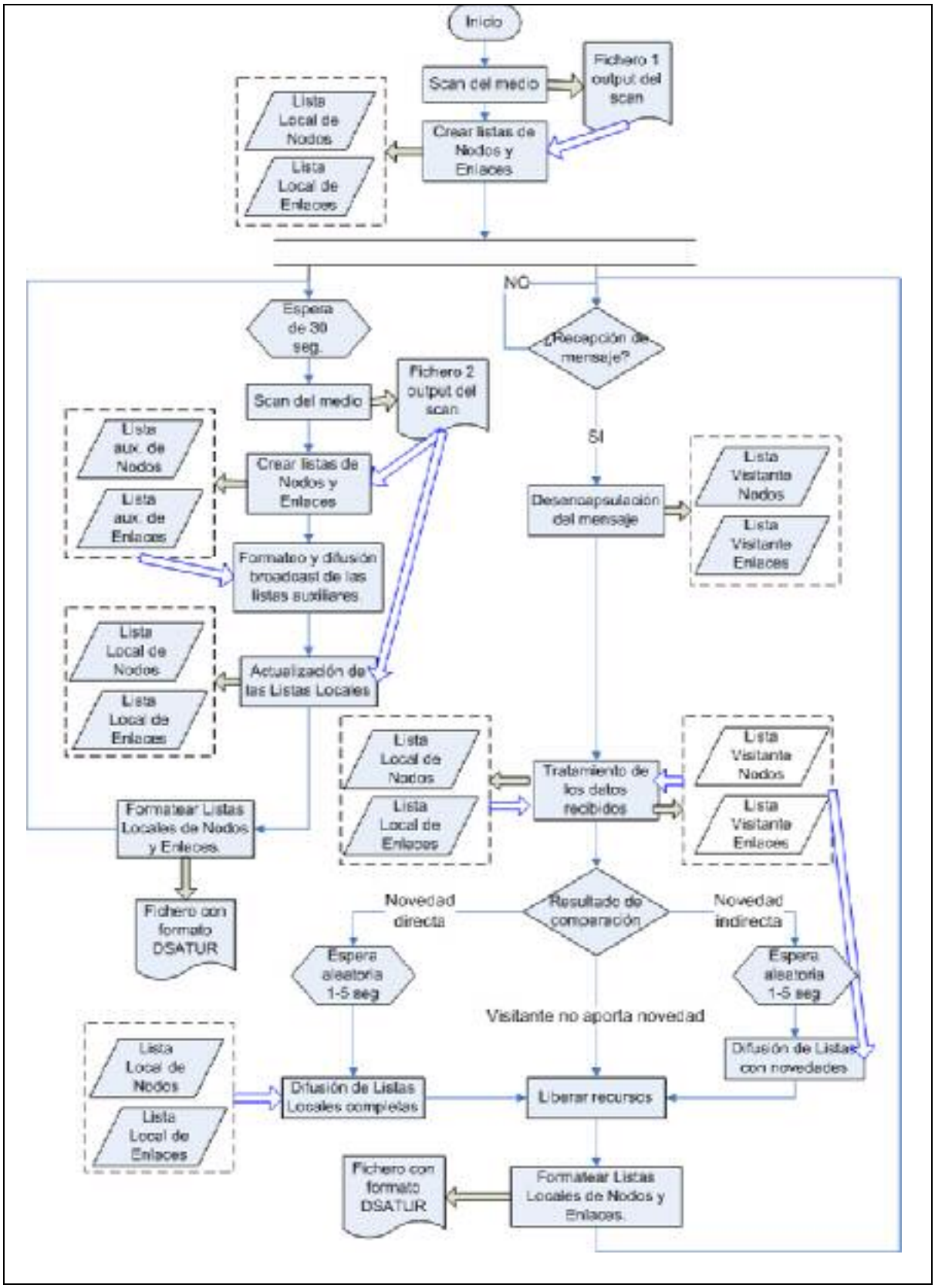


Fig 5.2. Diagrama de flujo del algoritmo desarrollado

En la figura anterior están representados, en forma de bloques funcionales, todos los procesos que participan en la ejecución del algoritmo WNRA.

Para consultar la funcionalidad y el significado de los símbolos utilizados, puede consultarse la leyenda del diagrama, en la siguiente figura.

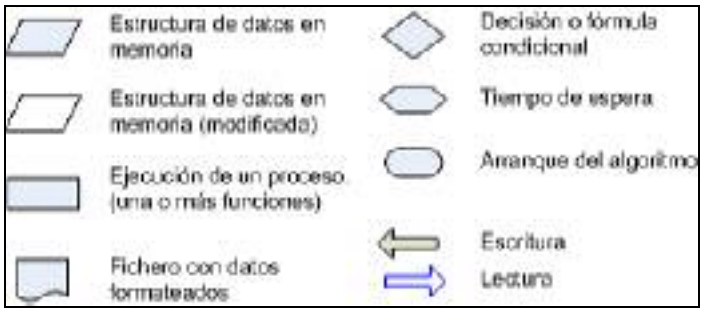


Fig 5.3. Leyenda del diagrama de flujo del algortimo desarrollado.

Como soporte adicional puede consultarse, en el siguiente apartado, el recorrido funcional del algoritmo que sigue los pasos del algoritmo simulando su ejecución.

5.3.1.2. *Funcionamiento*

Como se puede observar en el diagrama de flujo del algoritmo, al arrancar el binario del wnra, la primera acción que se lleva a cabo es el escaneo del medio, volcando su resultado a un fichero que se aloja dentro de la ruta del sistema de archivos /var/.

El alojamiento de todos los ficheros se realizará siempre en esa ruta ya que se monta en la memoria RAM y teniendo en cuenta que el AC monta el sistema de archivos en una memoria Flash, escribir en cualquier otra ruta provocaría un desgaste injustificado de los ciclos de reescritura.
Esto es debido a que la vida de las memorias Flash en cuanto a ciclo de reescrituras es bastante inferior al de las memorias RAM y el algoritmo reescribirá, en el mejor de los casos (sólo un nodo funcionando) cada 30 segundos.

Una vez se tiene el resultado del scan del medio en el fichero se insertarán, uno a uno y de forma ordenada, todos los nodos detectados en una lista local de nodos.
A continuación se creará la lista local de enlaces que, en este caso, establecerá un enlace del nodo local con cada uno de los nodos detectados.

Una vez finalizada la inicialización, se arrancan los procesos de refresco y de atención, que se ejecutarán paralelamente en el tiempo mediante el uso de threads de modo que ambos trabajarán sobre las listas locales de nodos y enlaces. Para evitar conflictos en lo que a acceso a las variables compartidas se refiere ambos procesos se sincronizan mediante variables de exclusión

mútua, que bloquearán el acceso a las variables cuando uno de los dos procesos las esté utilizando.

El proceso de refresco se encargará de escanear el medio, enviar un mensaje broadcast que refleje el resultado de la exploración, actualizar las listas locales y escribir el fichero de salida para el algoritmo variante de Dsatur.

Este ciclo lo repetirá cada 30 segundos, para garantizar una observación constante de la aparición de nuevos nodos que no incorporen el algoritmo WNRA (que no podrían ser detectados de otro modo) y para mantener a los nodos vecinos informados sobre los cambios el propio entorno.

Paralelamente, el proceso de atención, escuchará y atenderá los mensajes entrantes.
Ante la llegada de un mensaje, éste se desencapsula dando lugar las listas “visitantes” de nodos y enlaces.
A continuación, estas listas serán comparadas con las listas locales con el fin de evaluar si aportan alguna novedad.

Si no lo hacen, se liberará la memoria que ocupan y se procederá a escribir el fichero de salida para el algoritmo variante de Dsatur y se reiniciará el ciclo de atención.

Si, por el contrario, las listas visitantes aportan algún tipo de novedad ésta será discriminada por su forma de procedencia, lo que significa que si el emisor del mensaje es protagonista de alguna novedad se interpretará como novedad Directa. En caso contrario, nos encontraremos ante una novedad Indirecta.

Ante una novedad Directa, interpretamos que el nodo que protagoniza la novedad está a nuestro alcance, ya que recibimos un mensaje directamente de él. El mensaje de respuesta generado ante una novedad Directa incluirá toda la información de las listas locales con el propósito de transmitir toda la información disponible al nuevo nodo.

Ante una novedad Indirecta, interpretamos que el nodo que protagoniza la novedad no está a nuestro alcance y por lo tanto recibimos la novedad retransmitida por otro nodo, con lo que no tendría sentido hacer un volcado de toda nuestra información ya que no la recibiría el nuevo nodo.
En este caso, el procedimiento que se sigue es la difusión de un mensaje con las novedades, con el propósito de seguir retransmitiendo la novedad.

Tanto si la novedad es directa como indirecta, se espera un tiempo pseudoaleatorio de entre 1 y 5 segundos antes de enviar el mensaje, con el propósito de evitar que las respuestas de todos los nodos coincidan en el tiempo.
A continuación se libera la memoria ocupada por las listas visitantes y se escribe el fichero de salida para el algoritmo variante de Dsatur.

5.3.2. Estructura de los datos.

Como se ha explicado en el ejemplo anterior, WNRA se encarga de mantener un mapa estructurado de la red sobre el cual recoje datos de relevancia que son de utilidad para las operaciones del algoritmo Dsatur.

Este mapa se representa mediante 2 elementos (nodos y enlaces) que podríamos representar mentalmente como objetos.

5.3.2.1 *Estructura de representación de nodos.*

Los nodos representan a cada uno de los AP que un nodo que ejecuta WNRA tiene a su alcance.

Sobre ellos, WNRA recopila, en cada AP, diversa información destinada a diferentes fines:

- Dirección MAC: Se utiliza de identificador único y para ordenar alfabéticamente las listas de nodos, lo que agiliza las tareas de búsqueda.

- Utilización: Cada nodo calcula y difunde su propia utilización con el fin de que Dsatur valore el grado de importancia de la interferencia.

- Canal: El canal en el que trabaja cada punto de acceso le servirá a Dsatur para comprobar que nodos se interfieren entre sí.

- Señal recibida: Se almacena la potencia en dBm con la que se recibe la señal de cada nodo a fin de valorar el impacto que causa en caso de interferir.

La siguiente figura ayuda a visualizar el tipo de estructura de la que estamos hablando.

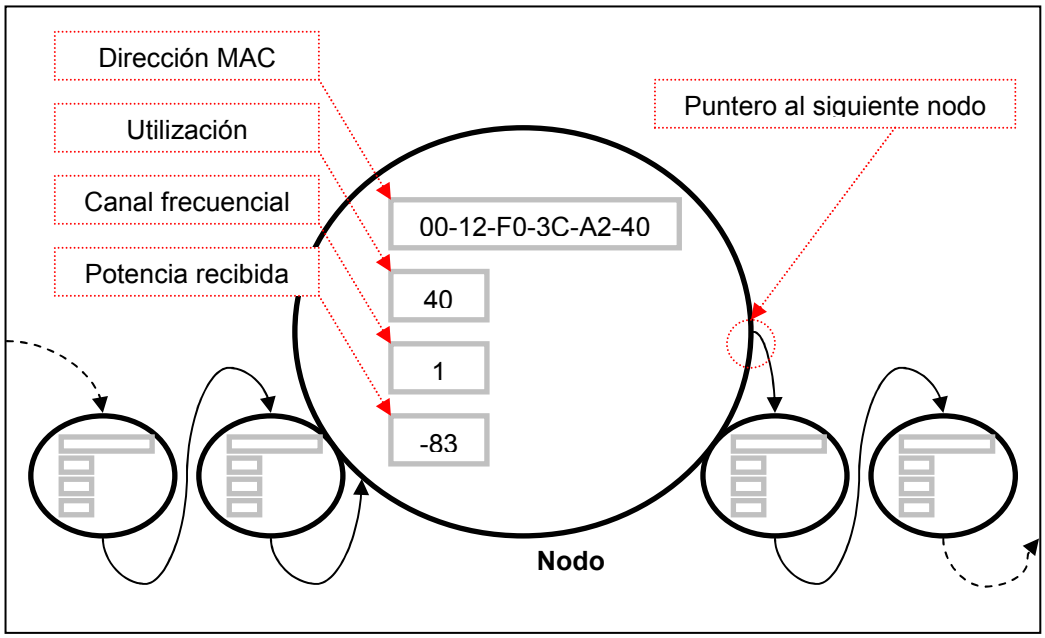


Fig 6.4. Abstracción de una lista enlazada de nodos.

Como podemos observar, la información se organiza en una lista enlazada y ordenada alfabéticamente, según la dirección MAC, cuyos elementos son estructuras de tipo nodo con las variables anteriormente comentadas.

La utilización de memórica dinámica nos permite ajustar el uso de la memoria a lo estrictamente requerido por el escenario.
El orden en la lista favorece la reducción de tiempo en las operaciones de búsqueda.

Ambos ahorros de en materia de memoria y proceso son interesantes de cara al trabajo con sistemas empotrados.

5.3.2.2 *Estructura de representación de enlaces.*

Cada enlace representa a cada uno de los enlaces punto a punto que unen los nodos de la red, cualquiera que sea su topología.

Cada enlace está caracterizado por cuatro parámetros que lo identifican unequívocamente:

- Direcciónes MAC 1 y 2: Constituyen los dos extremos del enlace.

- Señal recibida de 1 desde 2 y viceversa: Es el nivel de potencia recibida, de cada nodo, en su extremo opuesto. En dBm.

La información se organiza, como en el caso de los nodos, en forma de lista enlazada, sólo que en este caso los enlaces se insertan a medida que se crean o se actualizan.

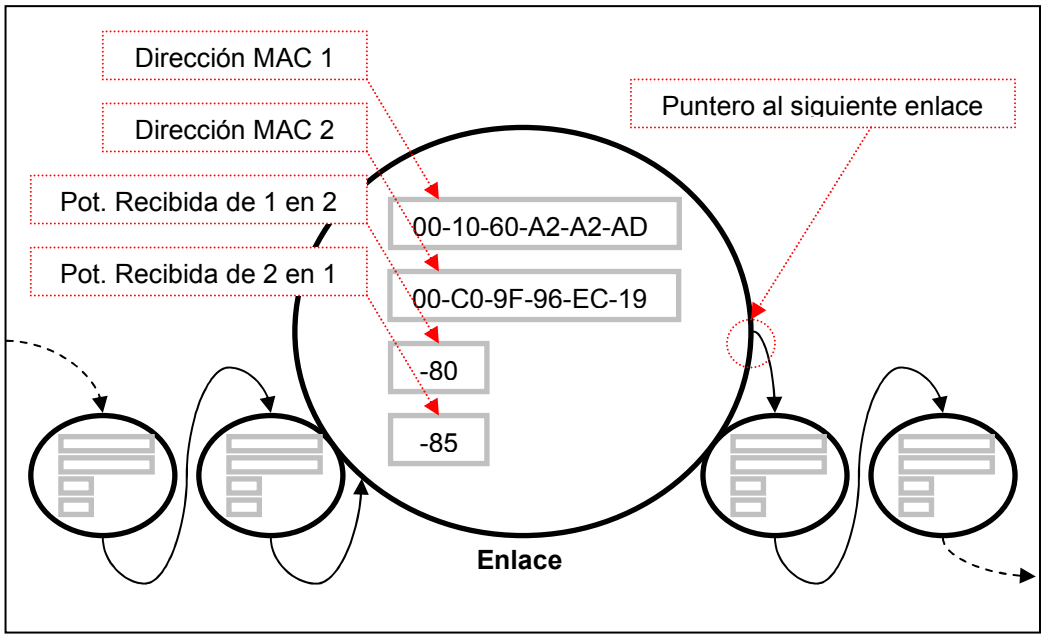


Fig 6.4. Abstracción de una lista enlazada de enlaces.

Para el caso de las listas de enlaces la información no se almacena ordenadamente ya que contamos con dos posibles índices (MAC 1 y MAC 2).

5.3.2.3 *Estructura de los mensajes.*

Los mensajes que envía el algoritmo WNRA tienen un formato predefinido, ya que combina dos tipos de elementos: nodos y enlaces.

El mensaje es una cadena de carácteres en la que se imprimen cada uno de los nodos de la lista de nodos a enviar, separados por delimitadores.

La información de cada nodo se imprime como:

- **00:12:F0:3C:A2:40 40 1/**
  Dirección MAC del nodo, utilización y canal frecuencial separados por espacios. Al final se añade un delimitador.

En cuanto a la información de enlaces, cada enlace se añade a continuación de su nodo y antes del delimitador de la siguiente forma:

- 00:12:F0:3C:A2:40 40 1 **00-10-60-A2-A2-AD 0 -82**/
  Dirección MAC del nodo enlazado, potencia que recibe el nodo enlazado del nodo local y potencia que recibe el nodo local del nodo enlazado.

Como se puede observar, la primera potencia figura como 0 ya que hasta que no recibamos información complementaria del nodo enlazado, no podemos

saber la potencia con la que recibe la señal del nodo local. En cuanto se recibe un mensaje de refresco del nodo enlazado, este valor de potencia se actualiza.

CAPÍTULO 6. PRUEBAS DEL ALGORITMO

6.1. Análisis detallado del proceso de convergencia.

A continuación se adjuntan los resultados de la convergencia en diferentes casos a modo de ejemplo. Se acompaña de capturas realizadas con un analizador de protocolos y diagramas que ayudan a entender el funcionamiento del algoritmo paso a paso.

6.1.1. Escenario 1: Dos nodos, uno de los cuales implementa WNRA.

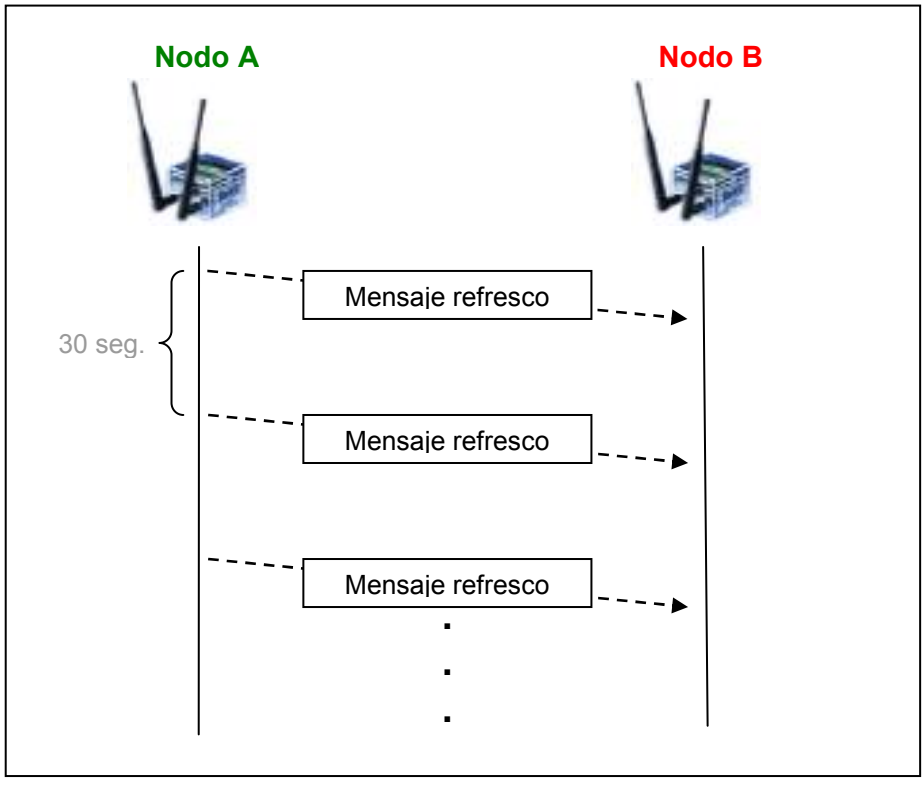


Fig 7.1. Diagrama de mensajes enviados para el caso analizado en el escenario 1.

En este primer caso, representado en la anterior figura, tenemos dos nodos cuyos principales datos son los siguientes:

Nodo A (Implementa WNRA)
- Dirección IP: 192.168.0.6
- Dirección MAC: 00:0C:29:47:14:E2

- Canal: 1
- Utilización 70%

Nodo B (No implementa WNRA)
- Dirección IP: 192.168.0.7
- Dirección MAC: 00:0C:29:E9:E7:BC
- Canal: 6
- Utilización 50%

Como en este caso sólo uno de los nodos estará ejecutando el algoritmo, los mensajes deWNRA sólo viajarán en una dirección, no obtendrán respuesta y se enviarán cada 30 segundos (mensaje de refresco).

Podemos observar como, pasado el tiempo de espera estipulado, el nodo A envía un mensaje con la información que conoce.

Fig 7.2. Captura de la consecución de 3 mensajes de refresco.

Tal y como se ve en la figura 8.1, los mensajes son enviados a la dirección broadcast de la red 192.168.0.0/24 desde la dirección 192.168.0.6, correspondiente al Nodo A.

```
-----> Mensaje ENVIADO (0):
00:0C:29:47:14:E2 70 1 00:0C:29:E9:E7:BC 0 -83/00:0C:29:E9:E7:BC 0 6/
-----> Mensaje ENVIADO (1):
00:0C:29:47:14:E2 70 1 00:0C:29:E9:E7:BC 0 -83/00:0C:29:E9:E7:BC 0 6/
-----> Mensaje ENVIADO (2):
00:0C:29:47:14:E2 70 1 00:0C:29:E9:E7:BC 0 -83/00:0C:29:E9:E7:BC 0 6/
```

Fig 7.3. Salida de la ejecución del algoritmo WNRA en el nodo A del escenario 1.

La figura anterior muestra el output de la ejecución de WNRA. Sólo aparecen los mensajes de refresco en los que aparece la información formateada que analizamos a continuación.

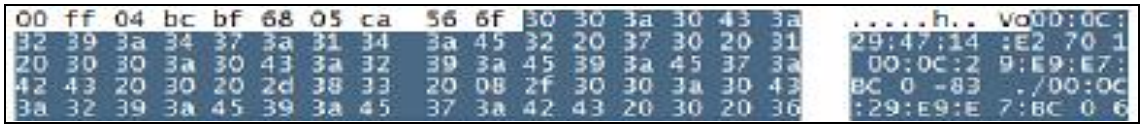


Fig 7.4. Contenido del campo de datos del datagrama UDP para el mensaje de refresco del escenario 1.

Siguiendo las pautas marcadas en el capítulo anterior, podemos fácilmente interpretar el contenido del mensaje formateado:

- **00:0C:29:47:14:E2** 70 1 00:0C:29:E9:E7:BC 0 -83/**00:0C:29:E9:E7:BC** 0 6/
- El escenario consta de dos nodos.

- 00:0C:29:47:14:E2 70 1 **00:0C:29:E9:E7:BC** 0 -83/00:0C:29:E9:E7:BC 0 6/
- El primer nodo (emisor del mensaje) tiene un enlace con otro nodo, que en este caso resulta ser el el segundo nodo.

- 00:0C:29:47:14:E2 70 1 00:0C:29:E9:E7:BC 0 -83/**00:0C:29:E9:E7:BC** 0 6/
- El segundo nodo no tiene ningún enlace.

Del mensaje también podemos extraer que el primer nodo (nodo A, el emisor) tiene una utilización del 70% y opera en el canal 1.

Del segundo nodo (nodo B) sólo conocemos el canal en el que opera y que el nodo A recibe su señal con -83 dBm.

No conocemos ni la utilización ni la señal que el nodo B recibe del nodo A ya que no implementa WNRA.

Como podemos comprobar, los resultados se corresponden con los datos reales y así queda reflejado en última instancia en el fichero destinado a ser interpretado por el algoritmo dsatur.

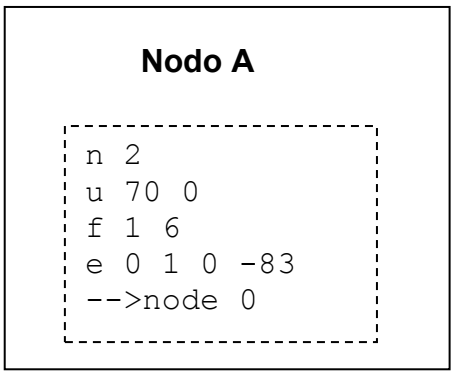


Fig 7.5. Ficheros graph.dsatur correspondientes a los nodos A y B del escenario 2.

Una referencia de como interpretar el grafo del algoritmo Dsatur puede encontrarse en [1].

6.1.2. Escenario 2: Dos nodos que implementan WNRA.

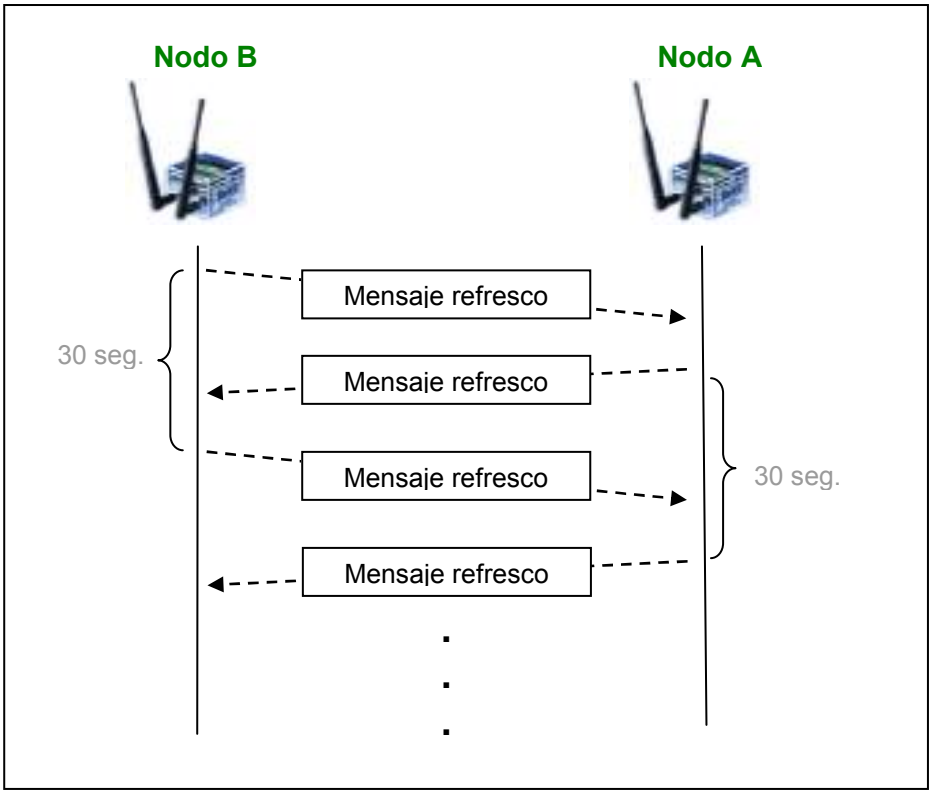


Fig 7.6. Diagrama de mensajes enviados para el caso analizado en el escenario 2.

En este segundo caso, representado en la anterior figura, volvemos a contar con los mismos nodos que en el escenario anterior, por lo que sus datos no varían.

La diferencia reside en que ambos implementan WNRA y por lo tanto, aunque no se generará novedad (ambos se ven entre sí), deberán actualizar sus listas locales hasta tener un mapa completo de la red, en el momento de la convergencia.

Como en este caso el proceso es simétrico desde el punto de vista de los nodos, sólo nos centraremos en un nodo, el nodo B.

Time	Source	Destination	Protocol	Info
31.093074	192.168.0.7	192.168.0.255	UDP	Source port: 1147 Destination port: 49000
32.952030	192.168.0.6	192.168.0.255	UDP	Source port: 1214 Destination port: 49000
64.110118	192.168.0.7	192.168.0.255	UDP	Source port: 1147 Destination port: 49000
65.753231	192.168.0.6	192.168.0.255	UDP	Source port: 1214 Destination port: 49000

Fig 7.7. Captura del intercambio de los primeros 4 mensajes de refresco.

Observamos en la figura anterior, que el Nodo B es el primero en enviar su primer mensaje de refresco y envía 2 mensajes de refresco consecutivos, en un espacio de 60 segundos.

Donde vamos a fijar la atención es en el contenido de los mensajes de refresco que envía el nodo B ya que en este escenario, no será siempre el mismo.

```
-----> Mensaje ENVIADO (0):
00:0C:29:E9:E7:BC 50 6 00:0C:29:47:14:E2 0 -86/00:0C:29:47:14:E2 0 1/
<----- Mensaje RECIBIDO (0):
00:0C:29:47:14:E2 70 1 00:0C:29:E9:E7:BC -86 -83/00:0C:29:E9:E7:BC 50
6/
-----> Mensaje ENVIADO (1):
00:0C:29:E9:E7:BC 50 6 00:0C:29:47:14:E2 -83 -86/00:0C:29:47:14:E2 70
1/
<----- Mensaje RECIBIDO (1):
00:0C:29:47:14:E2 70 1 00:0C:29:E9:E7:BC -86 -83/00:0C:29:E9:E7:BC 50
6/
```

Fig 7.8. Salida de la ejecución del algoritmo WNRA en el nodo B del escenario 2.

En la figura anterior, podemos observar los mismos 4 mensajes que se reflejan en la captura de paquetes de la figura 8.7, con una especial atención sobre el contenido de los mensajes.

En este caso vemos que el segundo mensaje enviado por el nodo B incluye más información que el primero y eso es debido a que el primer mensaje recibido en el nodo B actualiza sus listas locales con información que en un principio desconocía (resaltada):

- Potencia con que el nodo A recibe la señal del nodo B.
- Utilización del nodo A.

En el caso de los mensajes enviados por el nodo A, son todos iguales. Esto es debido a que, al ser el último en enviar el primer mensaje de refresco, su primer mensaje ya está actualizado con los datos provenientes del nodo B.

De nuevo comprobamos que los datos de los que dispone cada nodo, son idénticos y se corresponden con la realidad y así lo atestiguan los fichero graph.dsatur de cada nodo.

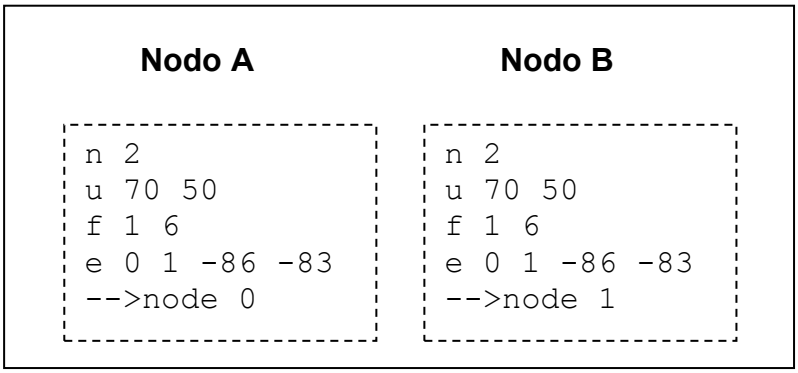


Fig 7.9. Ficheros graph.dsatur correspondientes a los nodos A y B del escenario 2.

En este caso, el tiempo de convergencia de este escenario es de, como máximo 60 segundos. (concatenación del tiempo de refresco de ambos nodos).

6.1.3. Escenario 3: Tres nodos, dos de los cuales implementan WNRA.

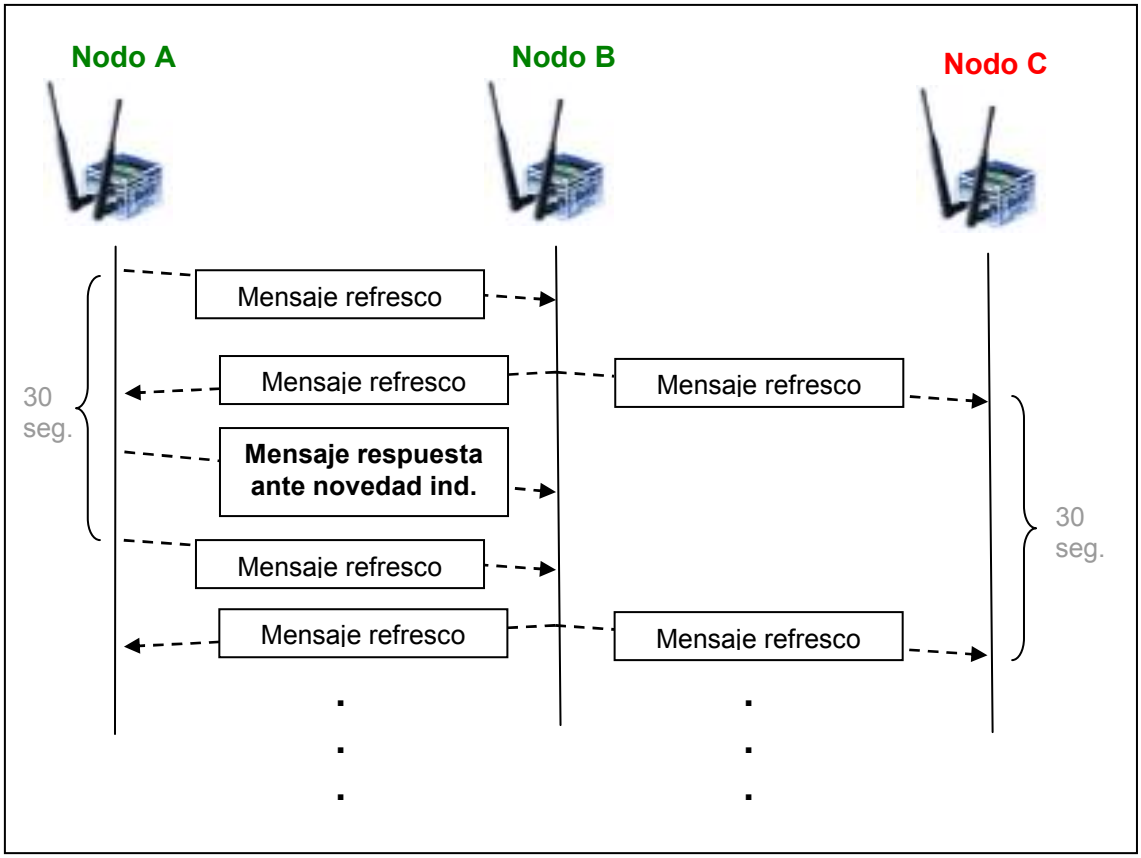


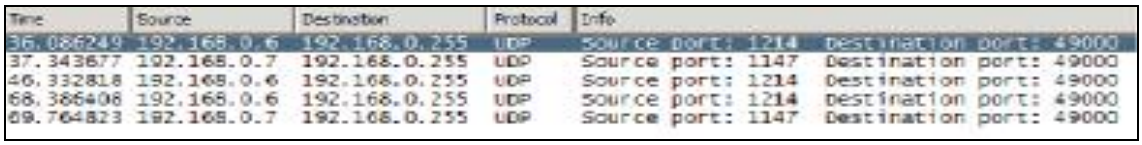
Fig 7.10. Diagrama de mensajes enviados para el caso analizado en el escenario 3.

En este tercer caso, los nodos A y B son los mismos que en los escenarios anteriores. El nodo C se incorpora ahora, pero no implementa WNRA.

Los datos más relevantes sobre el nodo C son los siguientes:

Nodo C (No implementa WNRA)
- Dirección IP: 192.168.0.8
- Dirección MAC: 00:0C:29:48:1A:E5
- Canal: 11
- Utilización 60%

El interés de este escenario reside en que el nodo A y el nodo C se encuentran a demasiada distancia como para recibir señal el uno del otro, por lo que el Nodo A tendrá constancia de la existencia del nodo C de forma indirecta, a través del nodo B.



Time	Source	Destination	Protocol	Info
36.086249	192.168.0.6	192.168.0.255	UDP	Source port: 1214 Destination port: 49000
37.343677	192.168.0.7	192.168.0.255	UDP	Source port: 1147 Destination port: 49000
46.332818	192.168.0.6	192.168.0.255	UDP	Source port: 1214 Destination port: 49000
68.386406	192.168.0.6	192.168.0.255	UDP	Source port: 1214 Destination port: 49000
69.764823	192.168.0.7	192.168.0.255	UDP	Source port: 1147 Destination port: 49000

Fig 7.11. Captura del intercambio de los primeros 4 mensajes de refresco y el mensaje de respuesta ante novedad indirecta.

Ahora centraremos la atención en analizar el contenido de los mensajes intercambiados, tanto como de los mensajes de refresco como el de respuesta ante novedad indirecta.

```
-----> Mensaje ENVIADO (0):
00:0C:29:47:14:E2 70 1 00:0C:29:E9:E7:BC 0 -83/00:0C:29:E9:E7:BC 0 6/
<----- Mensaje RECIBIDO (0):
00:0C:29:E9:E7:BC 50 6 00:0C:29:47:14:E2 -83 -86 00:0C:29:48:1A:E5 0 -
78/00:0C:29:47:14:E2 70 1/00:0C:29:48:1A:E5 0 11/
EVENTO ACTIVADO ::: NOVEDAD INDIRECTA
-----> Respuesta ENVIADA:
00:0C:29:47:14:E2 70 1 00:0C:29:E9:E7:BC -86 -83/00:0C:29:48:1A:E5 0
11 00:0C:29:E9:E7:BC -78 0/00:0C:29:E9:E7:BC 50 6/
-----> Mensaje ENVIADO (1):
00:0C:29:47:14:E2 70 1 00:0C:29:E9:E7:BC -86 -83/00:0C:29:48:1A:E5 0
11 00:0C:29:E9:E7:BC -78 0/00:0C:29:E9:E7:BC 50 6/
<----- Mensaje RECIBIDO (1):
00:0C:29:E9:E7:BC 50 6 00:0C:29:47:14:E2 -83 -86 00:0C:29:48:1A:E5 0 -
78/00:0C:29:47:14:E2 70 1/00:0C:29:48:1A:E5 0 11/
```

Fig 7.12. Salida de la ejecución del algoritmo WNRA en el nodo A del escenario 3.

Como puede apreciarse tanto en la captura como en la salida por consola del algoritmo en el nodo A, el primer nodo en enviar el mensaje de refresco es el nodo A.

A continuación recibe constancia, por mediación del mensaje de refresco del nodo B, de la existencia del nodo C, lo que activa el evento de novedad indirecta y genera una respuesta (resaltada en la figura 8.12).

Lo que hace el algoritmo ante este tipo de evento es difundir las novedades recibidas tras un periodo de espera pseudoaleatorio.

En este caso tan sencillo de tan sólo 3 nodos, da la casualidad de que las novedades más la información del propio nodo emisor de la respuesta conforman la totalidad del escenario. Por lo tanto, el mensaje de respuesta y los sucesivos mensajes de refresco, serán iguales.

En casos más complejos, con un número de nodos mayor, se apreciaría una diferencia notable entre el tamaño de un mensaje de actualización y uno que enviara toda la información de las listas locales.

En cuanto al contendido del mensaje de respuesta, podemos advertir, comparándolo con el primer mensaje de refresco enviado, los siguientes cambios:

- Se ha añadido el nivel de potencia con que el nodo B recibe la señal del nodo A (-83 dBm).
- Se ha añadido la utilización del nodo B (50%).
- Se ha añadido el nuevo nodo (nodo C, 00:0C:29:48:1A:E5) y su enlace con el nodo B.

Los únicos datos que A y B conocen del nodo B son la potencia con la que llega su señal hasta el nodo B (-78 dBm) y el canal en el que opera (11). Los datos restantes no se conocen puesto que el nodo C no implementa WNRA.

Una vez llegados a este punto, comprobamos de nuevo que los datos con los que cuenta cada nodo se corresponden con la realidad del escenario montado y así se refleja en el fichero graph.dsatur

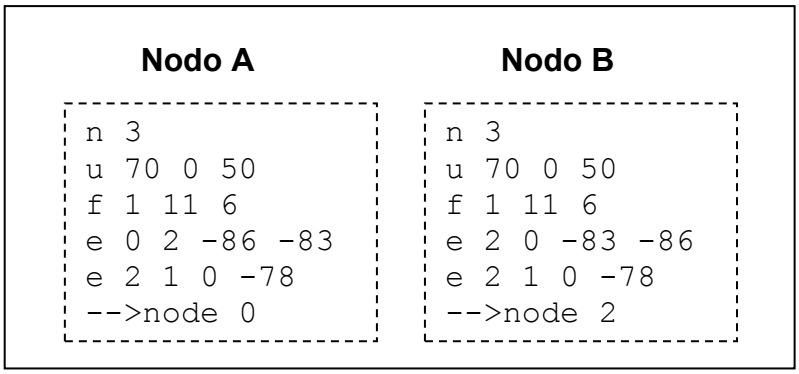


Fig 7.13. Ficheros graph.dsatur correspondientes a los nodos A y B del escenario 3.

Finalmente, podemos comprobar que en el caso actual el tiempo de convergencia vuelve a ser de, como máximo, 60 segundos.

CAPITULO 7. CONCLUSIONES

Para poder extraer conclusiones sobre el conjunto del proyecto, recapitularemos hasta el objetivo inicial.

En resumen, se puede dividir la realización del proyecto en dos etapas de una duración aproximadamente similar.

Durante la primera etapa se realizó una investigación sobre las posibilidades que ofrecía el mercado de AP basados en tecnología inalámbrica 802.11 y con sistemas operativos de código abierto.

La oferta no era muy extensa, ya que la mayor parte del sector estaba copado por el Linksys WRT54G, sobre el que la mayoría de comunidades wireless del mundo, basaban sus desarrollos.

Hubo que internarse en un terreno de soluciones más profesionales para encontrar la potencia y la flexibilidad que las gama comerciales de puntos de acceso que ofrecían los principales fabricantes no podían ofrecer.

Finalmente se optó por la plataforma Access Cube, de 4G Systems por su potencia (muy superior a la de un AP convencional), flexibilidad en cuanto a ampliación del número de interfaces.
El hecho de que una empresa privada se encontrase tras el desarrollo y perfección del dispositivo, que también empleaba para soluciones industriales a medida, prometía expectativas de ampliar sus posibilidades en futuras versiones de Nylon, la distribución linux que incluye.

La toma de contacto con el dispositivo tardó en llegar por un gran retraso en la entrega de los access cube, así que hubo que apresurarse en el aprendizaje de la plataforma para comprender el funcionamiento básico de su sistema operativo y de las herramientas de desarrollo.
Esto fue debido a que la información online disponible sobre el dispositivo era muy escueta, al tratarse de un dispositivo tan nuevo y de altas prestaciones que no está demasiado extendido todavía.

Llegados a este punto, comenzó la segunda etapa del proyecto, consistente en el desarrollo y prueba del algoritmo.

Los objetivos que debía cumplir el algoritmo eran los de proporcionar , en su convergencia, un mapa completo de la red en cada nodo combinado con otros datos que le servirían al algoritmo variante de Dsatur para poder llevar a cabo los cálculos de planificación frecuencial.

El desarrollo de la aplicación ha consumido una gran parte de las horas dedicadas a este proyecto ya que el uso de, principalmente threads y memoria dinámica, hacían del desarrollo de este algoritmo distribuido una dura tarea.

La extensión del periodo de desarrollo del algoritmo ha ido en detrimento del tiempo disponible para hacer pruebas y aunque no se ha podido probar en todos los escenarios que hubiese sido deseable, se ha conseguido perfilar de forma satisfactoria el comportamiento del algoritmo para cumplir las espectativas que se esperaban mediante frecuentes pruebas a lo largo de todo el desarrollo que pretendían comprobar poco a poco el correcto funcionamiento del diagrama de flujo diseñado a medida que se incorporaban mejoras o se corregían errores.

A falta de tiempo para realizar más pruebas, se puede concluir que en los escenarios básicos probados que se documentan en el proyecto, el algoritmo cumple su cometido correctamente.

Las líneas futuras, por lo tanto, pasarían por el diseño de nuevos escenarios para seguir probando y perfeccionando el algoritmo.

Algunas mejoras que se podrían añadir al conjunto podrían ir encaminadas a mejorar la eficiencia del transporte, como 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.
En caso de seguir utilizando sockets convencionales, la incorporación de funciones de multicast ayudaría también a mejorar la versión actual del algoritmo reduciendo el tráfico de datos.

Finalmente y en cuanto a el estudio de ambientalización del proyecto, cabe decir que la única vía de impacto mediambiental que cabe destacar en este caso es la del cumplimiento de la legislación española vigente en cuanto a materia de uso del espectro radioléctrico.

Pese a la polémica social que puedan suscitar las tecnologías que operan en la banda de 2,4GHz no se ha podido demostrar que, con las normativas que cumplen los dispositivos certificados por la ETSI, la emisión radioeléctrica de la tecnología que utiliza el 802.11 pueda resultar dañino para la salud.

Las interfaces Mini PCI 802.11b que utiliza el AC cumple con los niveles de potencia máximos fijados por la ETSI.

Este proyecto cumple la legislación al operar en una banda libre y usa dispositivos comerciales que cumplen las normativas eléctricas europeas de seguridad y como tecnología inalámbrica que es, evita el impacto muy superior que supone la instalación de redes cableadas.

BIBLIOGRAFIA I REFERENCIAS

Referencias

[1] E. Garcia, R. Vidal, and J. Paradells, "New Algorithm for Distributed Frequency Assignments in IEEE 802.11 Wireless Networks," in 11th European Wireless Conference 2005, EW05, vol. 1, pp. 211-217, April 2005.

[2] D. Brélaz, "New Methods to Color the Vertices of a Graph," Communications of the ACM, vol. 22, pp. 251-256, 1979.

Bibliografia

YoLinux.com. Página web (URL: http://www.yolinux.com)
Tutorial: POSIX thread (pthread libraries)
<http://www.yolinux.com/TUTORIALS/LinuxTutorialPosixThreads.html>

Linksys.com. Página web (URL: http://www.linksys.com)
Sitio web del fabricante del Linksys WRT54G.

Valenciawireless.com
Tutorial: POSIX thread (pthread libraries)
<http://www.yolinux.com/TUTORIALS/LinuxTutorialPosixThreads.html>

Meshcube.org. Página web (URL: http://www.meshcube.org)
Meshcube Wiki (referencia web y documentación sobre AC y Nylon).
<http://www.meshcube.org/meshwiki/>

OpenEmbeddded.org. Página web (URL: http://www.openembedded.org)
Sitio web del proyecto OpenEmbedded.

Acrónimos

CVS: Concurrent Version System
AC: Access Cube
AP: Access Point
OE: OpenEmbedded.
SSH: Secure Shell
RSA: Rivest, Shamir y Adleman (sistema criptográfico asimétrico)
DSA: Digital Signature Algorithm

Escola Politècnica Superior
de Castelldefels
UNIVERSITAT POLITÈCNICA DE CATALUNYA

ANEXOS

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

TITULACIÓN: Ingeniería Técnica de Telecomunicaciones esp. Telemática

AUTOR: Víctor N. García Fernández

DIRECTOR: Eduard Garcia Villegas

FECHA: 5 de septiembre de 2005

A. Anexo A: código del algoritmo

```
/*
################################################################################################
######################
   TFC/PFC - Redes Mesh basadas en puntos de acceso inteligentes 802.11 open source -
Victor N Garcia Fernandez

################################################################################################
###################### */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <netinet/in.h>
#include <netinet/ip.h>
#include <sys/socket.h>
#include <pthread.h>
#define PORTNUM 49000
#define MAXNODES 100
#define MAXBUFFER 1474
#define NOV_INDIRECT 1
#define NOV_DIRECT 0
#define NO_NOV 3

// Estructura de datos para los nodos
struct node
{
        char U;
        char cell;
        char signal;
        char mac[18];
        struct node *next;
};

// Estructura de datos para los enlaces
struct edge
{
        char mac1[18];
        char mac2[18];
        char signal12;
        char signal21;
        struct edge *next;
};

/**
 * Construye la lista de enlaces segun la informacion recibida del nodo mac
 *
 * @param mac         MAC del nodo origen de la informacion recibida
 * @param nodesM      Lista de nodos vecinos de mac (enviada por mac)
 * @param nodes               Lista de nodos de referencua a comparar con la lista
recibida de mac
 * @param edgesM      Lista de enlaces recibida de mac
 * @param edges               Lista de enlaces de referencia a comparar con la lista
recibida de mac
 * @return                    Lista de enlaces de mac que representan novedad
 */
struct edge *listToSend(char *mac, struct node *nodesM, struct node **nodes, struct edge
*edgesM, struct edge **edges);

/**
 * Añade la lista de nodos vecinos de mac a la lista de enlaces.
 *
 * @param mac  MAC del nodo de referencia
 * @param nodes       Lista de vecinos de mac
 * @param edges Lista de enlaces a rellenar
 */
void addEdges(char *mac, struct node *nodes, struct edge **edges);

/**
 * Crea en la variable msg2send,un mensaje formateado listo para enviar
 *
 * @param nodelist    Lista de nodos a enviar
 * @param linklist    Lista de enlaces a enviar
```

```
 */
void serialize(struct node *nodelist, struct edge **linklist);

/**
 * Deserializa un mensaje recibido en msg2rcv y lo separa en dos listas
 *
 * @param nodelist    Lista de nodos a rellenar
 * @param linklist    Lista de enlaces a rellenar
 */
void ezilaires(struct node **nodelist, struct edge **linklist);

/**
 * Actualiza la lista de nodos con la info del nodo nuevo.
 *
 * @param nouNode     Puntero al nodo nuevo
 * @param nodes              Puntero a la direccion de la lista de nodos a actualizar
 * @return                   0 si nouNode no aporta informacion nueva
 *                                  1 si nouNode no existia en la lista
 *                                  2 si ya existia, pero aporta información nueva
 */
int existingNode(struct node *nouNode, struct node **nodes);

/**
 * Actualiza la lista de enlaces con la informacion del enlace edgetmp.
 * edgetmp tambien se actualiza si en la lista hay mas info.
 *
 * @param edgetmp     Puntero a informacion con nuevo enlace
 * @param edges              Puntero a direccion con la lista de enlaces a actualizar
 * @return                   0 si edgetmp no aporta informacion nueva
 *                                  1 si edgetmp no existia en la lista
 *                                  2  si  ya  existia,  pero  aporta  informacion  nueva
(potencias)
 */
int existingEdge(struct edge **edges, struct edge *edgetmp);

/**
 * Elimina un elemento de la lista de enlaces.
 *
 * @param prev Puntero a la direccion del elemento inmediatamente anterior al objetivo
 * @param post Puntero a la direccion del elemento posterior anterior al objetivo
 */
void deletEdge(struct edge **prev, struct edge **post);

/**
 * Escribe en el archivo graph.dsatur el grafo de interferencias entre los APs del
escenario.
 * @param nl   Lista de nodos
 * @param el   Lista de enlaces
 */
void writeDsatur(struct node *nl, struct edge *el);

/**
 * Devuelve el numero que ocupa el nodo mac dentro de una lista de nodos.
 *
 * @param mac         Direccion MAC del nodo a buscar
 * @param db2count    Lista de nodos donde buscar
 * @return                   Numero de orden (0 si es el primero y -1 si no aparece en
la lista)
 */
char nodeNumber(char *mac, struct node *db2count);

/**
 * Añade al FINAL de un mensaje la variable de un nodo.
 *
 * @param msg         String del mensaje
 * @param elem        Elemento a añadir
 * @param type        Formato ("d" para numeros y "s" para caracteres)
 */
void add2msg(char *msg,char *elem,char type);

/**
 * Clona una lista de enlaces cualquiera a la lista enlazada "dolly"
 *
 * @param source      Lista de enlaces a clonar
 */
void cloneEdgeList(struct edge *source);
```

```
/**
 * Cuenta el número de elementos de una lista de nodos
 *
 * @param db2count    Lista de nodos en la que contar.
 * @return                    Numero de elementos
 */
char nodeCounter(struct node *db2count);

/**
 * Libera la memoria ocupada por una lista de enlaces
 *
 * @param list Puntero a la direccion de la lista de enlaces a liberar.
 */
void freeEdgeMem(struct edge **list);

/**
 * Libera la memoria ocupada por una lista de nodos
 *
 * @param list Puntero a la direccion de la lista de nodos a liberar.
 */
void freeNodeMem(struct node **list);


/**
 * Carga valores en una lista de nodos
 *
 * @param file Nombre del fichero output del scan (contiene los nodos que cargaremos en
la lista).
 * @param opt  Lista de nodos destino (Si es 1, los nodos se cargan en localDB, si es 0
en la auxDB)
 */
void loadDB(char *file, char opt);

/**
 * Obtiene la propia dirección MAC
 *
 * @param mac  String donde guardar la MAC
 */
void getmymac(char *mac);

/**
 * Envía a broadcast puerto 49000 el mensaje contenido en el string "msg2send"
 *
 * @param opt  Si es 0 lo envía de inmediato, si es 1 espera antes un tiempo aleatorio
entre 1 y 5 seg.
 */
void sendMsg(char opt);

/**
 * Procesa el mensaje recibido en el string "msg2rcv".
 * toma las medidas necesarias en caso de que haya novedad directa, indirecta o no haya
novedad.
 */
void processMsg();

/**
 * Tarea de Refresco.
 * espera 30 seg y escanea el medio, enviar un mensaje broadcast con el resultado de la
exploracion,
 *  actualiza las listas locales y escribe el fichero de salida para el algoritmo
variante de Dsatur.
 */
void *taskRefresh();

/**
 * Tarea de Atencion.
 * espera la recepcion de un mensaje y lo procesa.
 */
void *taskEngine();

/**
 * Recibe un mensaje en el puerto 49000 y lo guarda en "msg2rcv"
 *
 * @return     0 Si el emisor del mensaje es el propio nodo
 *                     1 Si el emisor del mensaje es otro nodo
 */
char rcvMsg();
```

```
/**
 * Obtiene el propio canal de la interfaz wlan1
 *
 * @return     canal frecuencial en el que opera la interfaz
 */
char getmychan();

/**
 * Obtiene la propia utilizacion de la interfaz wlan1
 *
 * @return     utilizacion de la interfaz
 */
char getmyU();

// Punteros para las listas de nodos
pthread_mutex_t mutex_localDB = PTHREAD_MUTEX_INITIALIZER; struct node *localDB;
struct node *updateDB;
struct node *auxDB;

// Punteros para las listas de enlaces
pthread_mutex_t mutex_localLink = PTHREAD_MUTEX_INITIALIZER; struct edge *localLink;
pthread_mutex_t mutex_dolly = PTHREAD_MUTEX_INITIALIZER; struct edge *dolly;
pthread_mutex_t mutex_list2send = PTHREAD_MUTEX_INITIALIZER; struct edge *list2send;
struct edge *updateLink;
struct edge *auxLink;

// String para almacenar los mensajes a enviar + variable de bloqueo para evitar
exclusion mutua.
pthread_mutex_t mutex_msg2send = PTHREAD_MUTEX_INITIALIZER; char msg2send[MAXBUFFER];

// String para almacenar los mensajes recibidos
char msg2rcv[MAXBUFFER];

char sndrMAC[18], event=NO_NOV, myMAC[18];
int cnt_env=0,cnt_rec=0;

int main()
{
        getmymac(myMAC);
        pthread_t thread_engine, thread_refresh;
        int iret1,iret2;

    localDB=NULL;

        system("iwlist wlan1 scanning > /var/scan1");
    loadDB("/var/scan1",1);
        addEdges(myMAC,localDB,&localLink);


        iret1 = pthread_create( &thread_refresh, NULL, taskRefresh, NULL);
        iret2 = pthread_create( &thread_engine, NULL, taskEngine, NULL);


        pthread_join( thread_refresh, NULL);
        pthread_join( thread_engine, NULL);
}

void *taskRefresh()
{
        while(1)
        {
                // Esperamos 30 segundos entre ejecuciones
                sleep(30);

                // Bloqueamos el acceso ajeno a las variables globales requeridas
                pthread_mutex_lock( &mutex_localDB );
                pthread_mutex_lock( &mutex_localLink );
                pthread_mutex_lock( &mutex_dolly );
                pthread_mutex_lock( &mutex_msg2send );

                // Escaneamos el medio y enviamos el resultado.
                system("iwlist wlan1 scanning > /var/scan2");
                loadDB("/var/scan2",0);
                addEdges(myMAC,auxDB,&auxLink);
                msg2send[0]='\0';
                serialize(auxDB,&auxLink);
```

```
                sendMsg(0);

                // Liberamos memoria ocupada por las listas auxiliares
                freeEdgeMem(&auxLink);
                freeNodeMem(&auxDB);

                // Actualizamos las listas locales
                loadDB("/var/scan2",1);
                addEdges(myMAC,localDB,&localLink);

                // Creamos el fichero de salida para el algoritmo variante de Dsatur.
                writeDsatur(localDB,localLink);

                // Desbloqueamos el acceso ajeno a las variables globales requeridas
                pthread_mutex_unlock( &mutex_localDB );
                pthread_mutex_unlock( &mutex_localLink );
                pthread_mutex_unlock( &mutex_dolly );
                pthread_mutex_unlock( &mutex_msg2send );

        }
}

void *taskEngine()
{
        char in=0;
        while(1)
        {
                in=rcvMsg();

                // Si recibimos un mensaje
                if(in==1)
                {
                        // Bloqueamos el acceso ajeno a las variables globales requeridas
                        pthread_mutex_lock( &mutex_localDB );
                        pthread_mutex_lock( &mutex_localLink );
                        pthread_mutex_lock( &mutex_dolly );
                        pthread_mutex_lock( &mutex_msg2send );

                        // Procesamos el mensaje
                        processMsg();

                        //  Desbloqueamos el acceso ajeno a las variables globales
requeridas
                        pthread_mutex_unlock( &mutex_localDB );
                        pthread_mutex_unlock( &mutex_localLink );
                        pthread_mutex_unlock( &mutex_dolly );
                        pthread_mutex_unlock( &mutex_msg2send );
                }
        }
}

void processMsg()
{
        // Extraemos la MAC del emisor del mensaje
        strncpy(sndrMAC,msg2rcv,17);
        sndrMAC[17]='\0';

        // Deserializamos el mensaje
        updateDB=NULL; updateLink=NULL;
        ezilaires(&(updateDB),&(updateLink));

        // Comparamos las listas locales con las recibidas
        list2send=listToSend(sndrMAC,updateDB,&localDB,updateLink,&localLink);

        msg2send[0]='\0';

        // Si hay una novedad...
        if(event!=NO_NOV)
        {
                // Si la novedad se recibe directamente, serializamos y enviamos todo.
                if (event==NOV_DIRECT)
                {
                        dolly=NULL;
                        cloneEdgeList(localLink);
                        serialize(localDB,&dolly);
                        sendMsg(1);
                }
```

```
                // Si la novedad se recibe indirectamente, serializamos y enviamos las
novedades.
                if (event==NOV_INDIRECT)
                {
                        serialize(localDB,&list2send);
                        sendMsg(1);
                }
        }

        // Liberamos la memoria ocupada por las listas de actualizacion
        freeEdgeMem(&updateLink);
        freeEdgeMem(&list2send);
        freeNodeMem(&updateDB);

        // Escribimos el grafo de entrada para el algoritmo variante de Dsatur
        writeDsatur(localDB,localLink);
}


char nodeCounter(struct node *db2count)
{
        char cnt=0;
        struct node *focus;
        focus=db2count;

        while (focus!=NULL)
        {
                cnt++;
                focus=focus->next;
        }
        return(cnt);
}

void getmymac(char *mac)
{
        FILE *mymacfd;
        char b[13];

        system("ifconfig wlan1 > /var/mymac");
        mymacfd=fopen("/var/mymac","r");

        while ((strcmp(b,"HWaddr")!=0)&&(!feof(mymacfd)))
                fscanf(mymacfd,"%s",&b);

        fscanf(mymacfd,"%s",mac);

        fclose(mymacfd);
        system("rm /var/mymac 2> /dev/null");
}

char getmychan()
{
        FILE *mychanfd;
        char b[8],channel[4];

        system("iwlist wlan1 frequency > /var/mychan");
        mymacfd=fopen("/var/mychan","r");

        while ((strcmp(b,"(channel")!=0)&&(!feof(mychanfd)))
                fscanf(mychanfd,"%s",&b);

        fscanf(mychanfd,"%s",channel);

        fclose(mychanfd);
        system("rm /var/mychan 2> /dev/null");

        return((channel[0]-48)*10+(channel[1]-48));
}

char getmyU()
{
        FILE *myUfd;
        char b[4];

        system("./utilizacion.sh > /var/u");
        myUfd=fopen("/var/u","r");
```

```
        fscanf(myUfd,"%s",&b);

        fclose(myUfd);
        system("rm /var/myu 2> /dev/null");

        if (strlen(b)==3) return((b[0]-48)*100+(b[1]-48)*10+(b[2]-48));
        if (strlen(b)==2) return((b[0]-48)*100+(b[1]-48));
}



int existingNode(struct node *nouNode, struct node **nodes){
        struct node *tmp=*nodes;
        struct node *tmp2=NULL;
        struct node *newNode = NULL;

        // Si la lista esta vacia
        if(*nodes==NULL) {
                *nodes = (struct node *)malloc(sizeof(struct node));
                memcpy(*nodes, nouNode, sizeof(struct node));
                (*nodes)->next=NULL;
                return 1;
        }

        while(tmp!=NULL) {
                // Si existe,actualiza datos y sale
                if(strcmp(nouNode->mac, tmp->mac)==0) {
                        // Si no va sobre mi.
                        if(strcmp(nouNode->mac, myMAC)!=0){
                                tmp->U=nouNode->U;
                                tmp->cell=nouNode->cell;
                        }
                        return 0;
                }
                // Si la nueva MAC es menor que tmp->mac, insertamos el nuvo nodo
                if(strcmp(nouNode->mac, tmp->mac)<0) {

                        newNode = (struct node *) malloc(sizeof(struct node));
                        memcpy(newNode, nouNode, sizeof(struct node));
                        newNode->next=tmp;
                        if(tmp2!=NULL)
                                tmp2->next=newNode;
                        else *nodes = newNode;
                        return 1;
                }
                tmp2 = tmp;
                tmp = tmp->next;
        }

        //Si hemos llegado hasta aqui, nouNode va al final de la lista
        newNode = (struct node *) malloc(sizeof(struct node));
        memcpy(newNode, nouNode, sizeof(struct node));
        newNode->next = NULL;
        tmp2->next = newNode;
        return 1;
}



void loadDB(char *file, char opt)
{
        FILE *scanfd;
        char dBm[11];
        char buffer[15],mac[18],mode[8];
        char cell,insert=0;
        struct node *focus, *head, *tail,*v1,*v2, *me;
        head=localDB;
    tail=localDB;

        if (opt==1)
        {
                // Nos insertamos a nosotros mismos en la lista.
                me=(struct node *)malloc(sizeof(struct node));
                me->next=NULL;
                me->cell=getmychan();
                strcpy(me->mac,myMAC);
```

```
                me->U=getmyU();
                me->signal=0;
                head=me;
                tail=me;
                localDB=me;
        }

    scanfd=fopen(file,"r");
        while (!feof(scanfd))
        {
                // Extraemos el canal del fichero de scan a una variable aux.
         while ((strcmp(buffer,"Cell")!=0)&&(!feof(scanfd)))
             fscanf(scanfd,"%s",&buffer);
                if(strcmp(buffer,"Cell")!=0) break;

         fscanf(scanfd,"%d",&cell);

                // Extraemos la MAC del fichero de scan a una variable aux.
         while (strcmp(buffer,"Address:")!=0)
             fscanf(scanfd,"%s",&buffer);

         fscanf(scanfd,"%s",&mac);

         // Buscamos el modo en el que opera el nodo.
         while (strncmp(buffer,"Mode:",5)!=0)
             fscanf(scanfd,"%s",&buffer);

                // Si el nodo opera como Master, creamos el nuevo nodo, le volcamos las
variables auxiliares y a continuación
                // la potencia recibida en dBm.
                if (strcmp(buffer,"Mode:Master")==0)
                {
            // Construimos el nuevo nodo.
            focus=(struct node *)malloc(sizeof(struct node));
                        focus->next=NULL; // Nodo independiente.

                        // Volcado de las variables auxiliares a las variables del nodo.
                        focus->cell=cell;
                        strcpy(focus->mac,mac);
                        focus->U=0;

                        // Extraccion del nivel de señal desde el fichero scan a la
variable del nodo.
            while (strcmp(buffer,"Signal")!=0)
                fscanf(scanfd,"%s",&buffer);
            fscanf(scanfd,"%s",&dBm);

                        // Conversion manual del nivel de señal (de string a val.
numerico).
                        if (dBm[9]=='\0') focus->signal=-1*((dBm[7]-48)*10+(dBm[8]-48));
                        else     focus->signal=-1*((dBm[7]-48)*100+(dBm[8]-48)*10+(dBm[9]-
48));

                        // Se inserta el nuevo nodo ordenadamente (MAC) dentro de la lista
local de nodos.
                        existingNode(focus,&localDB);
                }
        }
    // Cerramos el descriptor del fichero scan.
    fclose(scanfd);
}


void add2msg(char *msg,char *elem,char type)
{
    char temp[18];
        if (type=='s') sprintf(temp,"%s",elem);
        if (type=='d') sprintf(temp,"%d",*elem);
        strcat(msg,temp);
        if (strcmp(elem,"\b/")!=0) strcat(msg," ");
}


void deletEdge(struct edge **prev, struct edge **post)
{
    struct edge *aux;
```

```
        // Si nos encontramos en el primer elemento de la lista.
    if (prev==post)
    {
        prev=&((*prev)->next);
        aux = *prev;
        free(*post);
        *post=aux;
        prev=post;
    }
    else // Resto de casos.
    {
        aux = (*prev)->next;
        (*prev)->next=(*post)->next;
        free(aux);
        post=&((*prev)->next);
    }
}



void serialize(struct node *nodelist, struct edge **linklist)
{
    serializeMe(nodelist,linklist);

        struct node *Nfocus;
    struct edge **Efocus1, **Efocus2;
        Nfocus=nodelist;
        char match=0;
        while(Nfocus!=NULL)
        {
         if ((strcmp(Nfocus->mac,myMAC)!=0)==1)
                {

                        add2msg(msg2send,Nfocus->mac,'s');
                        add2msg(msg2send,&(Nfocus->U),'d');
                        add2msg(msg2send,&(Nfocus->cell),'d');
                        Efocus1=linklist;
                        Efocus2=linklist;
                        while(*Efocus1!=NULL)
                        {
                                match=0;
                                if (strcmp((*Efocus1)->mac2,Nfocus->mac)==0)
                                {
                                        add2msg(msg2send,(*Efocus1)->mac1,'s');
                                        add2msg(msg2send,&((*Efocus1)->signal21),'d');
                                        add2msg(msg2send,&((*Efocus1)->signal12),'d');
                                        deletEdge(Efocus2,Efocus1);
                                        match=1;
                                }
                                else if (strcmp((*Efocus1)->mac1,Nfocus->mac)==0)
                                {
                                        add2msg(msg2send,(*Efocus1)->mac2,'s');
                                        add2msg(msg2send,&((*Efocus1)->signal12),'d');
                                        add2msg(msg2send,&((*Efocus1)->signal21),'d');
                                        deletEdge(Efocus2,Efocus1);
                                        match=1;
                                }
                                if (match==0)
                                {
                                        Efocus2=Efocus1;
                                        Efocus1=&((*Efocus1)->next);
                                }
                        }
                        // Antes de cambiar de nodo, añadimos una marca delimitadora
                        add2msg(msg2send,"\b/",'s');
                }
                Nfocus=Nfocus->next;
    }
}

void ezilaires(struct node **nodelist, struct edge **linklist)
{
        char *point, tmpMAC[18],tmp1[4],tmp2[4];
        struct node Nfocus;
        struct edge Efocus;
        FILE *fd;
```

```
// Extraemos al emisor (primer nodo) del mensaje.
fd=fopen("ezilaires","w");
point=strtok(msg2rcv,"/");
fprintf(fd,"%s",point);
fclose(fd);

// Leemos y registramos la MAC del emisor del mensaje.
fd=fopen("ezilaires","r");
fscanf(fd,"%s",tmpMAC);
strcpy(Nfocus.mac,tmpMAC);

// Leemos y registramos la utilizacion del emisor del mensaje.
fscanf(fd,"%s",tmpMAC);
Nfocus.U=(char)atoi(tmpMAC);

// Leemos y registramos el canal del emisor del mensaje.
fscanf(fd,"%s",tmpMAC);
Nfocus.cell=(char)atoi(tmpMAC);


Nfocus.next=NULL;
existingNode(&Nfocus,nodelist);
// Extraemos los enlaces del primer nodo.
while (!feof(fd))
{
        fscanf(fd,"%s",tmpMAC);
        strcpy(Efocus.mac2,tmpMAC);

        fscanf(fd,"%s",tmpMAC);
        Efocus.signal12=(char)atoi(tmpMAC);

        fscanf(fd,"%s",tmpMAC);
        Efocus.signal21=(char)atoi(tmpMAC);

        strcpy(Efocus.mac1,Nfocus.mac);
        Efocus.next=NULL;
        if (strlen(Efocus.mac2)==17) existingEdge(linklist,&Efocus);
}
fclose(fd);

// Extraemos el resto de nodos del mensaje.
while ((point = strtok( NULL, "/" )) != NULL )
{
        system("rm ezilares");
        fd=fopen("ezilaires","w");
        fprintf(fd,"%s",point);
        fclose(fd);

        // Leemos y registramos la MAC del emisor del mensaje.
        fd=fopen("ezilaires","r");
        fscanf(fd,"%s",tmpMAC);
        strcpy(Nfocus.mac,tmpMAC);

        // Leemos y registramos la utilizacion del emisor del mensaje.
        fscanf(fd,"%s",tmpMAC);
        Nfocus.U=(char)atoi(tmpMAC);

        // Leemos y registramos el canal del emisor del mensaje.
        fscanf(fd,"%s",tmpMAC);
        Nfocus.cell=(char)atoi(tmpMAC);


        Nfocus.next=NULL;
        existingNode(&Nfocus,nodelist);
        // Extraemos los enlaces del primer nodo.
        while (!feof(fd))
        {
                fscanf(fd,"%s",tmpMAC);
                strcpy(Efocus.mac2,tmpMAC);

                fscanf(fd,"%s",tmpMAC);
                Efocus.signal12=(char)atoi(tmpMAC);

                fscanf(fd,"%s",tmpMAC);
                Efocus.signal21=(char)atoi(tmpMAC);

                strcpy(Efocus.mac1,Nfocus.mac);
```

```
                        Efocus.next=NULL;
                        if (strlen(Efocus.mac2)==17) existingEdge(linklist,&Efocus);
                }
                fclose(fd);

        }
}


// Difunde a broadcast el mensaje creado.
void sendMsg(char opt)
{
    // Declaracion de variables locales
    int sockfd,i;
    struct sockaddr_in everybody;

    // Configuramos la estructura destino
    everybody.sin_family=AF_INET;
    everybody.sin_addr.s_addr=inet_addr("192.168.0.255");
    everybody.sin_port=htons(PORTNUM);

        // Creamos el socket UDP y fijamos sus opciones
    sockfd=socket(AF_INET, SOCK_DGRAM, 0);
    int a=1;
    setsockopt(sockfd,SOL_SOCKET,SO_BROADCAST,&a,sizeof(a));

        if (opt==1) sleep(1+(int) (5.0*rand()/(RAND_MAX+1.0)));

        sendto(sockfd,msg2send,sizeof(msg2send),0,(struct                              sockaddr
*)&everybody,sizeof(everybody));
        close(sockfd);

}


int existingEdge(struct edge **edges, struct edge *edgetmp) {
        int trobat;
        struct edge *edg, *tmp;
        edg = *edges;
        trobat = 0;


        //Si la llista estava buida, afegim enllaÃ§ i sortim
        if(edg==NULL) {
                *edges = (struct edge *) malloc(sizeof(struct edge));
                memcpy (*edges, edgetmp, sizeof(struct edge));
                (*edges)->next=NULL;
                return 1;
        }

        while(edg!=NULL) {
                //Si EnllaÃ§ existeix i macs en mateix ordre: actualitzem senyals
                if(strcmp(edgetmp->mac1,   edg->mac1)==0   &&   strcmp(edgetmp->mac2,   edg-
>mac2)==0) {
                        trobat=1;
                        if(edgetmp->signal12!=0) {
                                if(edg->signal12==0)
                                        trobat = 2;
                                edg->signal12 = edgetmp->signal12;
                        } else if(edg->signal12!=0)
                                edgetmp->signal12 = edg->signal12;

                        if(edgetmp->signal21!=0){
                                if(edg->signal21==0)
                                        trobat = 2;
                                edg->signal21 = edgetmp->signal21;
                        } else if(edg->signal21!=0)
                                edgetmp->signal21 = edg->signal21;

                        return trobat;
                }
                //Si EnllaÃ§ existeix i macs en ordre invers: actualitzem senyals
                if(strcmp(edgetmp->mac1,   edg->mac2)==0   &&   strcmp(edgetmp->mac2,   edg-
>mac1)==0) {
                        trobat=1;
                        if(edgetmp->signal12!=0){
```

```
                                if(edg->signal21==0)
                                        trobat = 2;
                                edg->signal21 = edgetmp->signal12;
                        } else if(edg->signal21!=0)
                                edgetmp->signal12 = edg->signal21;

                        if(edgetmp->signal21!=0){
                                if(edg->signal12==0)
                                        trobat = 2;
                                edg->signal12 = edgetmp->signal21;
                        } else if(edg->signal12!=0)
                                edgetmp->signal21 = edg->signal12;

                        return trobat;
                }
                tmp = edg;
                edg = edg->next;
        }
        //Si enllaÃ§ no hi era (era nou)
        if(trobat==0) {
                //Afegim al final
                tmp->next = (struct edge *) malloc(sizeof(struct edge));
                memcpy (tmp->next, edgetmp, sizeof(struct edge));
                (tmp->next)->next=NULL;
                return 1;
        }
        return 0;

}


void addEdges(char *mac, struct node *nodes, struct edge **edges)
{
    struct edge *newEdge;
        struct node *newNode;
        newNode = nodes;

        if(nodes== NULL)
                return;

        while(newNode!=NULL)
    {
         if(strcmp(newNode->mac, mac)!=0)
         {
             //Construim nou enllac
                        newEdge = (struct edge *)malloc(sizeof(struct edge));
                        strcpy(newEdge->mac1, mac);
                        strcpy(newEdge->mac2, newNode->mac);
                        newEdge->signal12 = 0;
                        newEdge->signal21 = newNode->signal;

             //Si Aquest enllac ja existia, allibera memoria reservada per nou enllac
                        existingEdge(edges, newEdge);
                        free(newEdge);
                }
                newNode = newNode->next;
        }
}


char nodeNumber(char *mac, struct node *db2count)
{
        char cnt=0;
        struct node *focus;
        focus=db2count;

        // Mientras queden elementos, aumenta el contador.
        while (focus!=NULL)
        {
                if(strcmp(focus->mac, mac)==0)
                        return cnt;
         cnt++;
         focus=focus->next;
        }
    return(-1);
}
```

```
void writeDsatur(struct node *nl, struct edge *el)
{
	char *use, *ch;
	char n = nodeCounter(nl);
	struct node *focus;
	struct edge *edg;
	int i;
	FILE *fout;

	use = (char *) malloc(n*sizeof(char));
	ch = (char *) malloc(n*sizeof(char));
	focus = nl;
	edg = el;
	i = 0;

	while(focus!=NULL) {
		use[i] = focus->U;
		ch[i] = focus->cell;
		focus = focus->next;
		i++;
	}

	fout = fopen("graph.dsatur", "w");

	fprintf(fout, "n %d\nu", n);
	for(i=0; i<n; i++)
		fprintf(fout, " %d", use[i]);

	fprintf(fout, "\nf");
	for(i=0; i<n; i++)
		fprintf(fout, " %d", ch[i]);

	while(edg!=NULL) {
		fprintf(fout,	"\ne	%d	%d	%d	%d",	nodeNumber(edg->mac1,
nl),nodeNumber(edg->mac2, nl), edg->signal12, edg->signal21);
		edg = edg->next;
	}
	fprintf(fout, "\n");
	fprintf(fout, "-->node %d\n", nodeNumber(myMAC, nl));
	free(ch);
	free(use);
	fclose(fout);
}

char rcvMsg()
{
	// Declaracion de variables locales
    int sockfd,i;
    char nNodes=0;
    struct sockaddr_in servidor;
	int servidorl;
	servidorl = sizeof(servidor);

	sockfd = socket(PF_INET,SOCK_DGRAM,0);

	servidor.sin_family=AF_INET;
	servidor.sin_port=htons(PORTNUM);
	servidor.sin_addr.s_addr=INADDR_ANY;

	bind(sockfd,(struct sockaddr *)&servidor,sizeof(servidor));

	setsockopt(sockfd,SOL_SOCKET,SO_BROADCAST,&i,i);

	printf("antes de rcvfrom");
			printf("\n");
	recvfrom(sockfd,msg2rcv,2000,0,(struct sockaddr *)&servidor,&servidorl);
	printf("despues de rcvfrom");
	printf("\n");

	// Si el emisor del mensaje es igual al receptor, he enviado yo el mensaje y lo
descarto.
	if (strncmp(myMAC,msg2rcv,17)==0)
	{
		msg2rcv[0]='\0';
		close(sockfd);
```

```
                return 0;
        }
        else
        {
                close(sockfd);
                return 1;
        }
}


struct edge *listToSend(char *mac, struct node *nodesM, struct node **nodes, struct edge
*edgesM, struct edge **edges)
{
        struct node *tmp=nodesM;
        struct edge *tmp2=edgesM;
        struct edge *result=NULL;
        //char myMAC[18];
        event = NO_NOV;

        //Comprovem per tots els nodes de la llista nova (nodesM)
        //Si hi son a la llista global (nodes)
        while(tmp!=NULL)
        {
                if(existingNode(tmp, nodes)==1)
                {//Node tmp es nou (s'ha afegit a la llista)
                        event = NOV_INDIRECT;
                        if(strcmp(mac,  tmp->mac)==0)  //    a  mes,  missatge  es  rebut
directament d'ell
                                event = NOV_DIRECT;
                }
                tmp = tmp->next;
        }

        while(tmp2!=NULL) {
                if(existingEdge(edges, tmp2) != 0) //Hi ha novetat (afegida a la llista)
                        existingEdge(&result, tmp2);

                tmp2 = tmp2->next;
        }

        //Afegir els enllaços propis sempre
        tmp2 = *edges;
        while(tmp2!=NULL){
                if(strcmp(tmp2->mac1, myMAC)==0 || strcmp(tmp2->mac2, myMAC)==0)
                        existingEdge(&result, tmp2);
                tmp2 = tmp2->next;
        }


        return result;
}
void freeEdgeMem(struct edge **list)
{

        struct edge *Efocus = *list;

        while(Efocus!=NULL)
        {
                Efocus = (*list)->next;
                free(*list);
                *list=Efocus;
        }
        *list=NULL;
}

void freeNodeMem(struct node **list)
{
        struct node *Nfocus = *list;

        while(Nfocus!=NULL)
        {
                Nfocus=(*list)->next;
                free(*list);
                *list=Nfocus;
        }
        *list=NULL;
}
```

```
void cloneEdgeList(struct edge *source)
{
        struct edge *Sfocus, *Dfocus;
        Sfocus=source;
        Dfocus=NULL;
        char round=0;
        while(Sfocus!=NULL)
        {
                if (round==0)
                {
                        Dfocus = (struct edge *) malloc(sizeof(struct edge));
                        memcpy(Dfocus,Sfocus, sizeof(struct edge));
                        Sfocus=Sfocus->next;
                        dolly=Dfocus;
                }
                else
                {
                        Dfocus->next = (struct edge *) malloc(sizeof(struct edge));
                        memcpy(Dfocus->next,Sfocus, sizeof(struct edge));
                        Sfocus=Sfocus->next;
                        Dfocus=Dfocus->next;
                }
                if (Sfocus!=NULL)
                {
                        Dfocus->next = (struct edge *) malloc(sizeof(struct edge));
                        memcpy(Dfocus->next,Sfocus, sizeof(struct edge));
                        Sfocus=Sfocus->next;
                        Dfocus=Dfocus->next;
                        round++;
                } else Dfocus->next=NULL;
        }
        Dfocus->next=NULL;
}
```