UNIVERSIDAD POLITECNICA DE VALENCIA

Escuela Técnica Superior de Ingeniería Informática

etsinf

Proyecto Final de Carrera

Interfaz de comunicaciones java para red inalámbrica

Alumno:

Roberto Peñaranda Cebrian

Director:

Carlos Tavares Calafate

Depto. Informática de Sistemas y Computadores

Valencia a 27 de Septiembre de 2010

PFC: VisualDNS

Índice

Índice de figuras

Índice de tablas

Introducción

Hoy en día y cada vez más, las tecnologías inalámbricas están están dominando en el mercado. Sobre todo para conexiones en internet, mucha gente usa routers inalámbricos WIFI para conectarse desde cualquier sitio, ya sea con móviles, PDA's, portátiles o incluso ordenadores de sobremesa.

Estas redes son centralizadas, donde hay una o más infraestructuras que se encargan del trabajo de mantenimiento de la red, para compartir recursos o comunicarse. Pero este tipo de red inalámbrica no es la única que existe. También hay un tipo de red distribuida, la red Ad-Hoc, en la que no se necesita ningún tipo de infraestructura para mantener la red, lo hacen los propios nodos que la utilizan.

Otro punto que está en auge es el de las comunicaciones. A fechas de hoy todo el mundo tiene una cuenta de mensajería instantánea, ya sea Messenger, Skype o muchos otros. Todos estos necesitan de internet aunque se utilicen de forma local. Por lo que si solo disponemos de un conjunto de ordenadores capaces de crear una red Ad-Hoc, no podremos utilizar estas herramientas.

Aun así hay herramientas que si lo permiten, como por ejemplo Microsoft NetMeeting, entre otras, que es capaz de trabajar sobre internet pero también sobre una red local. El único problema es que para usar esta aplicación tienes que conocer de antemano a los usuarios (sus IP's) de la red.

En este proyecto entraremos en la descripción más detallada de las redes Ad-Hoc e implementaremos una aplicación que funcione en este tipo de redes, capaz de permitir comunicarse y ofrecer servicios a los participantes de la red.

Además, esta aplicación no necesita de un conocimiento previo de los usuarios de la red, pues una de sus tareas es descubrirlos.

Objetivos

El principal objetivo de este proyecto es implementar una aplicación capaz de dar soporte a la comunicación, ya sea mediante mensajes escritos o videoconferencia, y ofrecimiento de recursos mediante las redes Ad-Hoc.

El trabajo de esta aplicación, en su mayor parte es de descubrir participantes de la propia red y ofrecer el servicio de chat o videoconferencia entre otros con los participantes descubiertos.

Como objetivos secundarios, que nos ayudarán ha entender mejor la materia con la que trabajamos, nos proponemos estudiar como funciona el estándar 802.11 y sus variantes. Después entraremos en detalle en las redes Ad-Hoc, viendo sus características mas importantes.

También veremos como funciona el enrutamiento en este tipo de redes y nos centraremos en algunos de ellos.

Por último, una vez implementada la aplicación, se validará la misma mediante una red Ad-Hoc de prueba compuesta por tres nodos. De estos tres nodos, uno podrá alcanzar a los otros nodos, pero estos dos últimos son inaccesibles entre si, lo cual obliga a que accedan mediante el primer nodo, que alcanza a los dos.

Metodología

En este apartado comentaremos la metodología que hemos seguido para realizar este proyecto.

Lo primero que se ha realizado es la búsqueda de información sobre las redes Ad-Hoc para tener un conocimiento sobre lo que se está trabajando.

Después se ha ido implementando la aplicación por partes. Esto es, lo primero que se implemento es la comunicación con dos nodos, que supuestamente son vecinos. Para este desarrollo no hizo falta trabajar en una red Ad-Hoc, pues solo se trabajaba con dos nodos. Así que se utilizó una red WIFI centralizada, con un punto de acceso.

Una vez terminada esa parte, se desplegó una red Ad-Hoc con tres nodos para ver como funcionaba la aplicación en el entorno en el que se supone que debe trabajar.

Una vez probado en una red Ad-Hoc y solucionados los problemas que han ido surgiendo, se han realizado pruebas en esta red Ad-Hoc para tomar tiempos que son interesantes a la hora de explicar ciertas propiedades, tanto de la aplicación como del comportamiento de este tipo de red.

Estructura del documento

En este apartado describiremos como está estructurado este documento, describiendo que es lo que se explicará en cada apartado.

En el capítulo siguiente, capítulo 1, explicaremos como funcionan las redes Ad-Hoc. Dentro de este capítulo describiremos el estándar 802.11, y las topologías típicas

de las redes Ad-Hoc. También describiremos como funciona los algoritmos de enrutamiento para estas redes, donde se explicará el funcionamiento de algunos de los algoritmos más importantes.

En el siguiente capítulo describiremos nuestra aplicación, describiendo como funciona. También explicaremos sus algoritmos principales y su arquitectura. También se hablará de otras aplicaciones o herramientas que nuestra aplicación utiliza durante su ejecución.

En el capítulo tres hablaremos sobre algunos detalles de implementación de la aplicación, donde se comentarán también algunos bloques de código de programación importantes.

A continuación, en el capítulo 4, explicaremos las pruebas que hemos realizado y mostraremos los resultados que hemos obtenido. También hablaremos sobre los resultados, si son los esperados o no.

El quinto y último capítulo, trata sobre las conclusiones que hemos obtenido al desarrollar esta aplicación y las perspectivas que vemos cara el futuro.

PFC: VisualDNS

Capítulo

1

Redes móviles Ad-Hoc

Una red Ad-Hoc es una red inalámbrica descentralizada, es decir, no hay un nodo central que se ocupa de hacer el trabajo de encaminamiento y de enrutamiento, como puede ser un punto de acceso. En este caso, estas y otras tareas se realizan de forma distribuida en cada nodo, realizando las funciones propias del mantenimiento de la red.

Las primeras redes Ad-Hoc aparecieron en 1972 en el proyecto PRNet (Packet Radio Network) de la DARPA (Defense Advanced Research Projects Agency). Los nodos y dispositivos (repetidores, de encaminamiento...) de la red PRNet eran todos moviles, aunque de forma muy limitada. Según se ha ido avanzando con la tecnología se ha logrado integrar los nodos y dispositivos de red en una sola unidad, el nodo Ad-Hoc.

A medida que se iban desarrollando las redes Ad-Hoc, se buscaba adaptación a los protocolos que ya existían para dar soporte a las redes IP. Por eso, en 1996 se formo un grupo de trabajo de estas redes en la IETF (Internet Engineering Task Force), el cual se encargaria de especificar interfaces y protocolos estándares para el soporte del funcionamiento de internet basado en IP sobre las redes Ad-Hoc móviles.

En 2003 se formó el consorcio Ad-Hoc Network, uniendo los esfuerzos de la industria, la academia y el gobierno para desplegar las redes Ad-Hoc en hogares inalámbricos, redes personales multimedia y hasta en redes de área amplia.

Las comunicaciones en estas redes son descentralizadas, como ya hemos comentado antes, por lo que no necesitamos ninguna infraestructura que realice los enrutamientos y otras tareas de mantenimiento de la red. Las comunicaciones se realizan de forma directa entre distintos nodos o a través de nodos intermedios, que actúan como routers. Para que esto funcione hace falta que se definan protocolos que difieren de los que se disponen en las redes tradicionales centralizadas.

1.1. Topología de la red

La topología en este tipo de redes es muy dinámica, es decir, al no ser un sistema centralizado, los nodos no tienen que estar alrededor de un nodo principal, como por ejemplo en el caso de conexiones Wifi con puntos de acceso donde todos los nodos

tienen que poder alcanzar al punto de acceso. En el caso de una red Ad-Hoc, la única condición es que para que un nodo participe a una red Ad-Hoc tiene que alcanzar como mínimo a otro nodo que ya pertenezca a esa red.

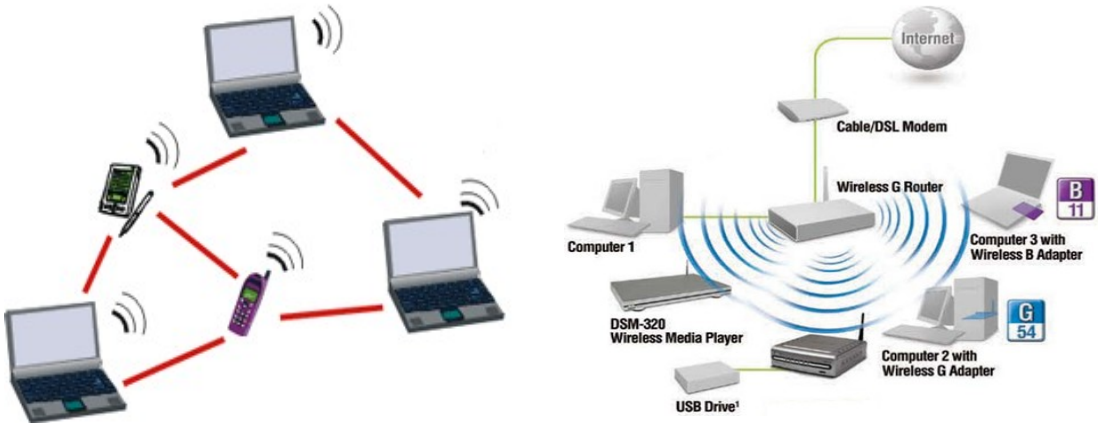

Figura 1.1: Ejemplo red no centralizada (izquierda) y red centralizada (derecha).

En las imágenes anteriores podemos diferenciar entre una red centralizada (derecha) y otra descentralizada y distribuida, es decir Ad-Hoc (a la izquierda). Como podemos ver en la imagen de la derecha, todos los nodos tienen que alcanzar al punto de acceso, ya que en otro caso no podrán pertenecer a esa red.

En la imagen de la izquierda (red Ad-Hoc) podemos ver que el nodo en el extremo izquierdo no logra alcanzar a todos los nodos pero sí alcanza a dos de ellos, con lo cual puede formar parte de la red Ad-Hoc.

1.2. Tecnología IEEE 802.11

IEEE 802.11 es un estándar que define la capa física y de enlace de datos de la arquitectura OSI para las redes WLAN. Este estándar utiliza como método de acceso al medio el protocolo CSMA/CA, y llega a alcanzar hasta 2 Mb/s.

Posteriormente surgieron mejoras y variaciones, que comentaremos a continuación:

- 802.11a: Este estándar trabaja en la banda de 5 GHz y alcanza una velocidad de 54 Mb/s. El estándar utiliza 12 canales, 8 para red inalámbrica y 4 para conexiones punto a punto. Al trabajar en la banda de 5 GHz, en vez de la de 2.4, obtenemos una ventaja, ya que la banda de 2.4 GHz es utilizada por más aplicaciones, como por ejemplo teléfonos inalámbricos o los hornos de microondas entre otros, lo que produce interferencias. Pero en esta banda, el

alcance de cobertura es menor, ya que sus ondas son se absorben más fácilmente.

- 802.11b: Esta variación usa el mismo método de acceso al medio que el estándar original, CSMA/CA. Dispone de una velocidad máxima de 11Mb/s. Se trabaja en una banda de 2.4 GHz con 3 canales, y se logra un alcance de hasta 300 metros. Pero esa banda también es utilizada por teléfonos inalámbricos, bluetooth y los hornos microondas, lo que produce interferencias.

- 802.11c: Esta variación es menos utilizada que las anteriores. Se utiliza para la comunicación entre redes del mismo tipo o de diferentes tipos, como por ejemplo los de tipo 802.1d, el utilizado para bridges MAC.

- 802.11d: Es un complemento del estándar original para permitir el uso internacional de las redes locales 802.11, permitiendo que distintos dispositivos intercambien información en rangos de frecuencia según lo que se permita en el país de origen de ese dispositivo.

- 802.11e: Este estándar mejora la calidad del servicio en la capa de enlace de datos. Esto consigue que se soporte tráfico de aplicaciones en tiempo real, por ejemplo aplicaciones de audio y video. Este estándar introduce un nuevo mecanismo de acceso, Hybrid Coordination Fuction(HCF), que define cuatro categorías de acceso al medio(de más a menos de prioridad): Background, Best Effort, Video, Voice. Dependiendo de la categoría de la que se trate se definen diferentes tiempos de acceso al medio y de tamaños de la ventana de contencion.

- 802.11f: Este punto se trata de una recomendación para los proveedores de puntos de acceso permitiendo que sean más compatibles entre si. Se utiliza el protocolo IAPP permitiendo que un usuario cambie de un punto de acceso a otro sin importar las marcas de estos puntos de acceso.

- 802.11g: Este estándar de modulación es la evolución del estándar 802.11b, trabajando también en la banda de 2,4 GHz. La principal novedad es que se logra una velocidad máxima teórica de 54Mb/s, similar a la del estándar 802.11a. Este estándar es compatible con su antecesor (802.11b). Es decir, un dispositivo que admite el estándar 802.11g, también funciona con el 802.11b.

- 802.11h: El objetivo de este estándar es modificar el estándar 802.11 para la coexistencia con sistemas radares o satélites y cumplir regulaciones europeas de usos de frecuencias y rendimiento energético.

- 802.11i: Se basa en la seguridad en la transferencia de datos y utiliza AES (estándar de cifrado avanzado) y TKIP (Protocolo de claves integra-seguras-

temporales) para cifrar transmisiones en las tecnologías 802.11a, 802.11b y 802.11g.

- 802.11Ir: Actualmente, este estándar es obsoleto. Se elaboró para poder usar señales infrarrojas en el estándar 802.11.
- 802.11j: Este estándar es similar al 802.11h, pero en el ámbito japonés.
- 802.11k: Este estándar permite a los puntos de acceso calcular y valorar los recursos de radiofrecuencia de los clientes para mejorar su gestión.
- 802.11n: Este estándar de modulación es el más novedoso. Trabaja tanto en la banda de 2,4 GHz como en la de 5 GHz, permitiendo la compatibilidad con todas las anteriores (802.11a, 802.11b y 802.11g). Fue ratificado en septiembre del 2009 con una velocidad teórica de 600 Mb/s, aumentando en mucho la velocidad de transmisión con respecto a sus antecesoras.
- 802.11p: Este estándar es para comunicaciones de corto alcance. Trabaja en la banda de 5,9 GHz. Se desea utilizar para el intercambio de datos entre vehículos y entre vehículos e infraestructuras en carretera.
- 802.11r: La principal característica de este estándar es permitir que, si un nodo va a cambiar de punto de acceso, se inicie los protocolos de seguridad en el punto de acceso nuevo antes de que abandone el punto de acceso donde estará actualmente.
- 802.11s: En este estándar se define la compatibilidad de fabricantes en protocolos de redes de mallas (redes donde se trabaja con topología Ad-Hoc y centralizada o con infraestructura).
- 802.11v: Este estándar permite la configuración remota de los clientes, mejorando la gestión de estos. Además de mejorar la gestión proporciona nuevas capacidades, que se dividen en cuatro categorías: mecanismos de ahorro de energía, posicionamiento, temporización y coexistencia.
- 802.11w: Este estándar aumentará, pues aun no está concluido, la seguridad de los protocolos de codificación y autenticación.
- 802.11y: Permite transmitir en la banda de 3650 a 3700 Mhz en EEUU.

1.3. Encaminamiento en redes Ad-Hoc

Los algoritmos clásicos de encaminamiento se basan en conocer la topología de la red para, una vez conocida, establecer las mejores rutas entre dos nodos cualesquiera.

Estas topologías no son cambiantes, es decir, son nodos estáticos donde las rutas de nodo a nodo no cambian.

Estos algoritmos no nos sirven para las redes Ad-Hoc, ya que tienen una topología muy dinámica. Estos algoritmos también tienen otra desventaja para las redes Ad-Hoc. Se trata que de estos algoritmos utilizan muchos mensajes para llegar a conocer la topología, y las redes Ad-Hoc no disponen de mucho ancho de banda, lo que podría dar lugar a una saturación de la red.

Por ese motivo se han desarrollado otros algoritmos que funcionan mucho más eficientemente en las redes Ad-Hoc. Estos algoritmos se pueden categorizar según su comportamiento para la búsqueda de rutas:

- Reactivos: Los algoritmos reactivos realiza la búsqueda de rutas solo cuando se necesite usar una ruta para enviar información y no tenga la ruta hacia el destino o tenga la ruta del destino pero no sea valida. En esta clase de algoritmos cabe esperar que el primer paquete tenga una latencia considerable, pues el primer paquete es el que desencadena la búsqueda del destino. Las rutas usadas se quedan activas por un tiempo, después de descartan.

- Proactivos: Estos algoritmos, a diferencia de los reactivos, intentan tener la información para el encaminamiento continuamente actualizada, sin tener que esperar a tener que utilizar una ruta para que sea descubierta. Cuando la topología cambie la red se inundará de paquetes para actualizar las tablas.

Aparte de esta clasificación según comportamiento para descubrir rutas, existe otra que depende del tipo de encaminamiento:

- Encaminamiento salto-a-salto: se trata de que un nodo fuente solo tenga que saber a qué nodo intermedio mandar un paquete para que este llegue al destino. Este nodo intermedio puede ser el destino o no. En caso de ser el destino, este lo procesaría. De no ser el caso, el nodo intermedio haría lo mismo que el nodo fuente: mandar el paquete a un nodo vecino que sí pueda alcanzar al nodo destino, de forma que es cada nodo el que decide qué salto es el mejor. La información de encaminamiento la tiene cada nodo.

- Encaminamiento en Origen: la ruta para alcanzar al nodo destino se ha de saber de antemano, y se almacena en el paquete. En cada nodo intermedio que alcanza solo mira cual es el siguiente nodo y lo envía, sin tener que calcular por donde mandarlo.

En la siguiente tabla se mostrará ejemplos de algoritmos para redes Ad-Hoc clasificados según las distintas categorías que se han explicado anteriormente:

	Encaminamiento salto a salto	Encaminamiento en Origen
Proactivos	DSDV, OLSR, CGSR, WRP, TBRPF	
Reactivos	AODV, LMR, TORA	DSR, LQSR

Tabla 1.1: Clasificación de algoritmos de encaminamiento.

A continuación entraremos un poco en detalle de algunos de los algoritmos mas conocidos.

1.3.1. DSDV

El algoritmo DSDV [1] (Destination Sequence Distance Vector) es un algoritmo proactivo y utiliza un encaminamiento salto-a-salto. En cada nodo se dispone de una tabla de encaminamiento con todos los destinos disponibles en la red, y el número de saltos que necesita para alcanzar el destino. También se dispone de un número de secuencia por cada destino para poder diferenciar entre rutas antiguas y rutas modernas.

Este algoritmo envía mensajes continuamente para poder tener las tablas de encaminamiento actualizadas. Para mejorar el trafico en este tipo de algoritmo se tienen dos tipos distintos de mensajes de actualización de tablas de encaminamiento. El primero es *full dump* y se encarga de mandar toda la información de encaminamiento. Este tipo de paquete, por el tamaño, puede llegar a mandarse en varios paquetes más pequeños. El otro tipo de paquete, *incremental*, se utiliza cuando hay pequeños cambios en la red, transportando solo la información de esos pequeños cambios.

La nuevas rutas que se almacenan contienen la dirección del nodo destino, el número de saltos que se deben de dar para alcanzar al nodo destino, el número de secuencia asociado al destino y un identificador de todo el mensaje. En caso de tener ya una ruta para ese nodo destino, se usará la que tenga un número de secuencia más moderno. En caso de que el número de secuencia sea igual, se seleccionará la ruta que alcance al nodo destino en el menor número de saltos.

Este algoritmo de encaminamiento funciona bien cuando es una red que no tiene mucha movilidad y en la cual muchos nodos de la red participan, ya que si hay muchos cambios en la topología de la red se tendrían que mandar muchos mensajes de actualización, y si no participan muchos nodos se haría mucho trabajo encontrando rutas a nodos que después no se van a utilizar.

1.3.2. AODV

El algoritmo AODV [2] evoluciona del DSDV y mantiene la idea de los números de secuencia y las tablas de encaminamiento, pero en vez de ser proactivo es

reactivo, o lo que es lo mismo, funciona bajo demanda, guardando solo la información de las rutas a nodos que participen en interconexiones.

En cada destino de la tabla de encaminamiento se tiene la dirección del siguiente nodo para alcanzar al destino (encaminamiento salto-a-salto), el número de saltos necesarios, el número de secuencia y el tiempo de vida. El número de secuencia nos sirve para evitar la creación de ciclos y para diferenciar entre rutas antiguas y las nuevas. El tiempo de vida nos sirve para descartar rutas de las que no se sabe nada desde hace tiempo.

El algoritmo de encaminamiento AODV, a diferencia del algoritmo DSDV, no mantiene la información de las rutas de todos los nodos. Las rutas de los nodos se van descubriendo según se vaya necesitando la comunicación con esos nodos. Este algoritmo utiliza dos tipos de mensajes para descubrir rutas : *RREQ* y *RREP*.

Cuando un nodo quiere mandar información a un nodo destino, manda un mensaje *RREQ* mediante broadcast, es decir, a todos los nodos que alcanza. Estos, en el caso de no ser el nodo destino, o no tener la ruta hacia el nodo destino, propagan el mensaje *RREQ* del nodo fuente, guardando la ruta hacia el nodo fuente.

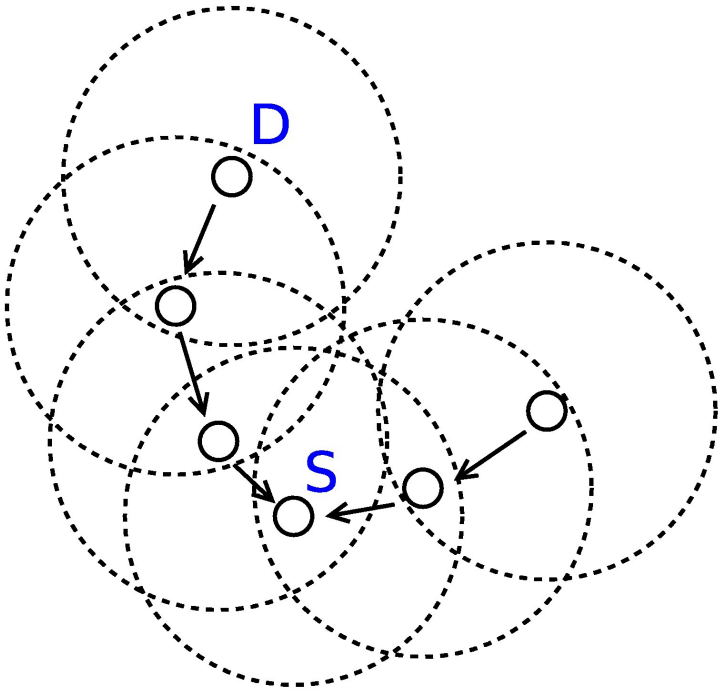

Figura 1.2: Ejemplo del algoritmo AODV (Fase1)

Una vez el mensaje *RREQ* ha alcanzado al nodo destino o a otro que conozca la ruta hacia el nodo destino, la red se encuentra en una situación parecida al del ejemplo mostrado en la imagen. Todos los nodos que hay entre el nodo fuente y el nodo destino conocen la ruta hacia el nodo fuente (flechas de la imagen) gracias al mensaje de petición (*RREQ*), con lo que el nodo destino, o el nodo que conozca la ruta hacia el nodo destino, puede mandar sin problemas el mensaje de respuesta, *RREP*, ya que se sabe la ruta hacia el nodo fuente. Una vez recibido este mensaje por parte del nodo fuente, este ya tiene asegurado un camino hacia el nodo destino, así que comienza la transmisión de paquetes de información.

Los mensajes de petición, *RREQ*, disponen de la dirección del origen de la petición y del destino buscado. Además, tiene un identificador del mensaje para controlar la inundación. También almacena el número de secuencia que tiene el origen de la última ruta hacia el destino (que será cero si no conocía ninguna ruta) para que no se responda con una ruta mas antigua, y el número de secuencia del propio mensaje de petición, por si se mueve el nodo origen.

Los mensajes de respuesta, *RREP*, también disponen de las direcciones del origen y del destino, además del número de secuencia del destino y del número de saltos necesarios para ser alcanzado.

A la hora de mantener las rutas, las rutas descubiertas se mantienen vivas (activas) solo durante un coto periodo de tiempo, para no disponer de rutas caducadas que mantienen una información que puede no ser válida.

Cada nodo mantiene la información de sus vecinos con mensajes *HELLO*, con un TTL=1. Si un nodo detecta que un vecino se ha movido, y participa en una ruta activa del nodo, este avisará con el mensaje de error *RERR*. En el caso de que el nodo desaparecido no participara en ninguna ruta activa, no se haría nada.

1.3.3. DSR

El algoritmo DSR [3], a diferencia de los anteriores, no es de encaminamiento salto a salto, si no de encaminamiento en origen. Es decir, los nodos que conocen una ruta, no solo conocen el siguiente nodo al que transmitir, sino que conocerá todo el tramo.

Los nodos disponen de una tabla donde, por cada destino, se desglosa de una lista de nodos por los que tiene que pasar para llegar al destino.

Para descubrir una ruta, el nodo origen manda un mensaje de petición donde se almacena el identificador (para no propagar por duplicado), la dirección del nodo destino y también la dirección propia del nodo. Cuando este mensaje llega a otro nodo,

si no es el nodo destino y no conoce la ruta hacia él, éste retransmite el mensaje añadiendo su dirección en él. Si se trata del nodo destino, éste mandará un mensaje de respuesta al nodo origen, pues sabe la ruta gracias al mensaje de petición recibido que ha ido almacenando los nodos por los que ha pasado, incluyendo en este mensaje la lista de nodos visitados del mensaje de petición recibido. También podría responder un nodo intermedio que tenga en su caché la ruta del destino buscado, pero en su mensaje de respuesta, además de insertar la lista de nodos visitados del mensaje de petición recibido, añadirá la lista de nodos de su caché para ese destino. A continuación se mostrara un ejemplo de petición con una imagen de una red, donde el nodo A es el origen y busca una ruta hacia el nodo D.

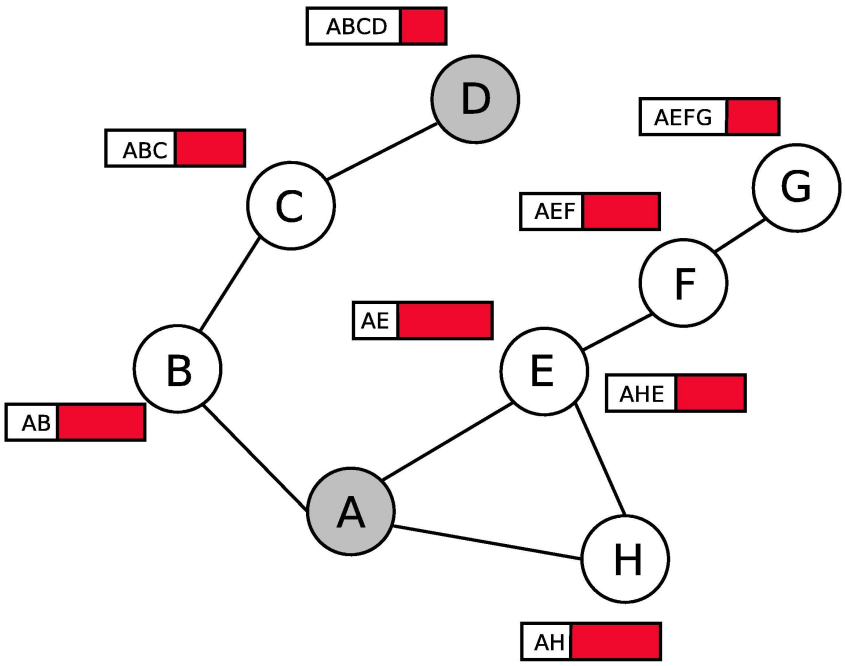

Figura 1.3: Algoritmo DSR: Fase de petición.

Como podemos observar, al nodo D le llega el mensaje de petición con los nodos por los que ha pasado. Ahora veremos le respuesta:

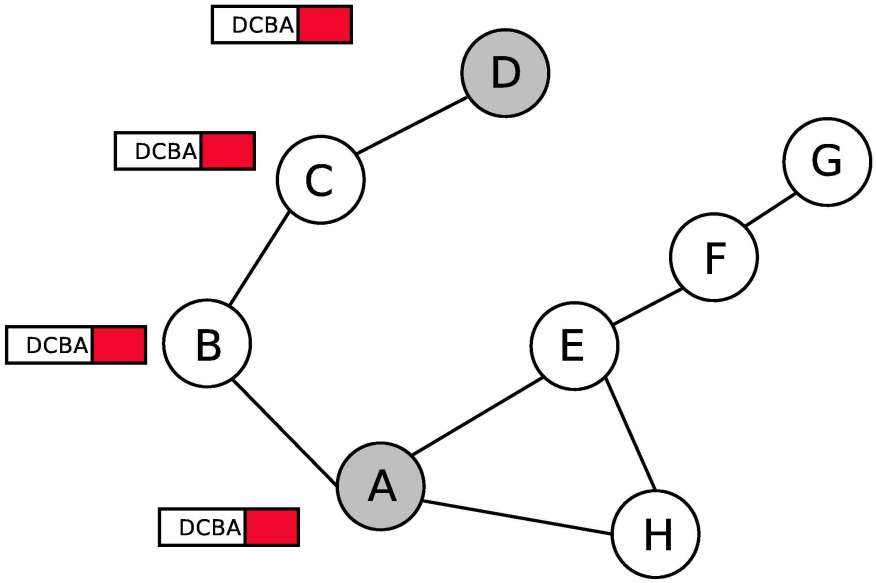

Figura 1.4: Algoritmo DSR: Fase de respuesta.

Como se puede ver el nodo destino, D, manda un mensaje de respuesta con la dirección que se ha conseguido en la petición. Cuando llega al origen, este guarda esta dirección en su caché.

Respecto al mantenimiento de las rutas, cuando un nodo se encuentra con un problema al comunicarse con otro nodo, envía un mensaje de error. Quien recibe este mensaje de error mira en su caché y, elimina las entradas de su tabla de encaminamiento que hacen uso del enlace que se ha perdido.

Capítulo

2

Aplicación Visual DNS

En este capítulo trataremos de explicar como funciona nuestra aplicación y las herramientas que utiliza y requisitos necesarios.

2.1. Requisitos

Para que nuestra aplicación funcione, tiene que haber una red Ad-Hoc configurada, con un algoritmo de encaminamiento establecido, y donde todos los nodos deberán tener establecidas sus IP. También puede trabajar en una red local típica, con una infraestructura principal, como una red WIFI normal, o incluso por Ethernet, pero eso no es competencia de nuestro proyecto.

Nuestra aplicación puede trabajar en varios sistemas operativos. Puede trabajar en los sistemas operativos Windows XP o superiores, así como en sistemas Linux que dispongan de administrador de archivos Nautilus o Konqueror y tengan instalada la herramienta Xdg-Open.

Cabe destacar que, para algunas utilidades de la aplicación, es necesario cumplir algunos requisitos. Como por ejemplo, para utilizar la videoconferencia necesitamos tener instalada eficientemente la herramienta Ekiga. También decir que para anunciar que se tiene un servidor web, se deberá tener un servidor web en marcha, o de lo contrario los otros usuarios no podrán acceder a ese servicio. Para el funcionamiento de los recursos compartidos deberemos tener en los sistemas Linux la herramienta SAMBA, para intercomunicarse con sistemas Windows.

2.2. Visión general de la arquitectura de la aplicación

Como hemos dicho anteriormente, esta aplicación funciona en redes Ad-Hoc donde se ha configurado con antelación los protocolos de encaminamiento. Una de sus funciones más importantes de esta aplicación es la de descubrir a los usuarios de la red Ad-Hoc a la que pertenece. Una vez descubiertos, la aplicación ofrece unos servicios que dependen de lo que hayan anunciado otros usuarios. También nos da información de los usuarios, como por ejemplo el nombre y la dirección IP. En la siguiente imagen podemos ver la aplicación una vez descubiertos algunos usuarios:

Figura 2.1: Ejemplo VisualDNS

A la hora de realizar las comunicaciones entre nodos, hemos usado una técnica de combinación de los protocolos TCP y UDP, donde utilizamos UDP para transmitir los anuncios de información de la que se dispone, y TCP para pasar esa información solo a los nodos vecinos.

El protocolo de descubrimiento se basa en que cada nodo envía por broadcast, mediante UDP, la lista de los nodos de la red que conoce, incluyéndose él mismo. Cuando recibe uno de estos mensajes, compara la lista de nodos que ha recibido con los nodos que ya conoce. Si entre la lista de nodos recibidos hay alguno que no se conocía todavía, se solicita la información de ese nodo nuevo al nodo vecino que lo ha anunciado. En la siguiente imagen se muestra un ejemplo del flujo de datos entre nodos.

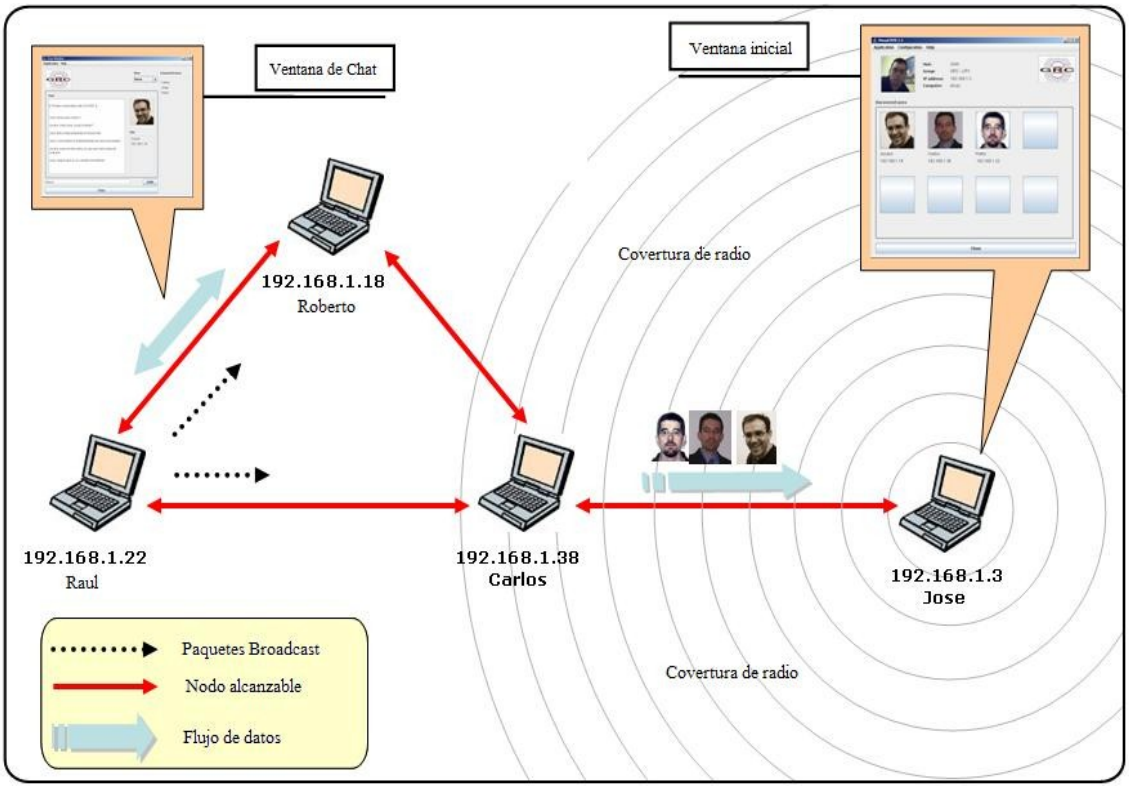

Figura 2.2: Ejemplo de flujo de datos.

Como podemos ver en el ejemplo, los nodos 192.168.1.18, 192.168.1.22 y 192.168.1.38 ya se conocen entre si. Además, todos esos nodos también se alcanzan entre si. Después aparece el nodo 192.168.1.3, que solo alcanza al nodo 192.168.1.38. Este último manda un mensaje broadcast anunciando que conoce a los demás nodos. Cuando el nodo 192.168.1.3 recibe el mensaje, los compara con los que conoce. Como no conoce a ninguno pide la información de los nodos nuevos. Una vez hecho esto, aunque no está representado en la imagen, el nodo 192.168.1.38, que ya conoce al nodo 192.168.1.3, lo anuncia en el siguiente mensaje broadcast. Los nodos 192.168.1.22 y 192.168.1.18 verán que hay un nuevo participante en la red y pedirán información de este último.

El siguiente algoritmo muestra que es lo que hace un nodo cuando recibe un mensaje de anuncio de vecinos:

POR (*cada mensaje recibido*)
HACER
    Construir lista para comparar
    SI(*Hay nuevos miembros*)
        Obtener información por TCP
        Actualizar la lista de miembros conocidos
        Actualizar interfaz gráfica

Por último decir que el tiempo entre anuncios de nodos conocidos es configurable mediante la interfaz gráfica. El tiempo entre anuncios de nodos conocidos es importante ya que, cuanto mayor sea el tiempo, más se tardará en saber los participantes de la red en cada nodo. Pero tampoco se puede configurar con un tiempo pequeño porque la red se inundaría de paquetes de anuncios de nodos conocidos.

2.3. Herramientas utilizadas

2.3.1. Start

Esta herramienta es un comando de la consola de Windows. Se utiliza para ejecutar un programa o un archivo. Esto último nos es de utilidad a la hora de lanzar el navegador web en nuestra aplicación, ya que ejecuta el navegador web que tiene asignado como predeterminado el usuario.

A continuación se explicara el uso de esta aplicación:

start [“título”] [/D ruta] [/I] [/MIN] [/MAX] [/SEPARATE | /SHARED]

[/LOW | /NORMAL | /HIGH | /REALTIME | /ABOVENORMAL |/BELOWNORMAL]

[/AFFINITY <afinidad hex>] [/WAIT] [/B] [comando o programa] [parámetros]

Opción	Utilidad
título	Título que se mostrarse en la barra de título de la ventana.
ruta	Con esta opción se indica el directorio de inicio
B	Con esta opción se inicia la aplicación sin crear una nueva ventana. La aplicación omite el manejo de ^C. A menos que la aplicación habilite el procesamiento de ^C, sólo se podrá interrumpir la aplicación con ^Inter
I	El nuevo entorno será el entorno original pasado a cmd.exe, y no el entorno actual.
MIN	Inicia la ventana minimizada
MAX	Inicia la ventana maximizada

SEPARATE	Inicia un programa de Windows de 16 bits en un espacio de memoria separado
SHARED	Inicia un programa de Windows de 16 bits en un espacio de memoria compartido
LOW	Inicia aplicación en la clase de prioridad Inactiva
NORMAL	Inicia aplicación en la clase de prioridad Normal
HIGH	Inicia aplicación en la clase de prioridad Alta
REALTIME	Inicia aplicación en la clase de prioridad Tiempo real
ABOVENORMAL	Inicia aplicación en la clase de prioridad Arriba de lo normal
BELOWNORMAL	Inicia aplicación en la clase de prioridad Debajo de lo normal
AFFINITY	La nueva aplicación tendrá la máscara de afinidad de procesador especificada, expresada como un número hexadecimal.
WAIT	Inicia aplicación y esperar a que finalice.
Comando o programa	Si se trata de un comando cmd interno o un archivo por lotes, el procesador de comandos se ejecuta con el modificador /K en cmd.exe, lo que significa que la ventana permanecerá después de que el comando se ejecute. Si no es un comando cmd interno ni archivo por lotes, entonces se considera un programa y se ejecutará como una aplicación de ventana o aplicación de

	consola.
Parámetros	Los parámetros transmitidos al comando o programa

Tabla 2.1: Parámetros Start.

NOTA: las opciones SEPERATE y SHARED no se admiten en plataformas de 64 bits.

Si en vez de un ejecutable seleccionamos un archivo no ejecutable, el comando Start lo ejecutará con su programa asociado. Esta parte es la que nosotros aprovechamos para utilizar el navegador web que en ese momento está asociado para abrir una URI, en nuestro caso la dirección del servidor web de otro cliente.

En el caso de no encontrar el fichero que le hemos indicado, intentará abrir un directorio con el nombre de ese fichero con el Explorer.

2.3.2. Explorer

Explorer [4] es el administrador de archivos oficial de los sistemas operativos de Microsoft desde que surgió Windows 95 hasta las más actuales. Se trata de una interfaz gráfica que te permite administrar los archivos y directorios de un equipo y crear, modificar o borrar archivos o directorios.

También permite la navegación por archivos compartidos en la red. Por lo que lo utilizaremos en nuestra aplicación a la hora de poder ver los recursos que ofrecen otros usuarios.

En Windows, cuando se inicia el sistema operativo, la aplicación Explorer se ejecuta de forma automática, en el escritorio, cuando hacemos doble clic en alguna carpeta, pero también se puede ejecutar por linea de comandos. Si no se le pasa nada como parámetro, abre el directorio principal del usuario por defecto de forma gráfica con el explorador de Windows. Pero también se le puede pasar como parámetro un directorio, de forma que se comportara como si hiciésemos doble clic en el icono de ese directorio.

Cuando le pasamos un directorio por parámetro, este directorio no tiene por qué ser local, y esto es lo que utilizamos en nuestra aplicación. Si ejecutamos “explorer \\host\recurso” abrirá el recurso de ese host.

Otra cosa a destacar es que el protocolo que se utiliza para trabajar con archivos compartidos en Windows es CIFS, o también llamado SMB.

2.3.3. Nautilus

Al igual que vimos para Windows y Explorer, en Linux tenemos otro administrador de archivos [5]. En este caso vamos ha hablar del administrador de archivos del entorno gráfico de Gnome, Nautilus.

Este administrador de archivos permite administrar los archivos locales del equipo. También permite administrar archivos alojados en otro equipo en red. Permite trabajar con archivos de red basados en FTP, WebDAV, SSH y archivos compartidos mediante Windows por medio de SAMBA, implementación UNIX de el protocolo SMB o CIFS.

Como hemos dicho, Nautilus puede trabajar con los archivos compartidos de Windows. Esto nos es muy útil para nuestra aplicación, ya que podemos mostrar los archivos compartidos que publique un nodo que se haya descubierto.

El modo en el que lo utilizamos en nuestra aplicación es el siguiente, lo ejecutamos con el parámetro siguiente: smb://máquina/recurso. Con lo cual, "smb" indica que se debe utilizar samba, "maquina" sería la ip del maquina donde se tiene el recurso y "recurso" el recurso compartido.

2.3.4. Konqueror

Si para el entorno Gnome tenemos el administrador de archivos Nautilus, para el entrono KDE tenemos Konqueror [6]. Como se puede observar empieza por la lera 'k' por pertenecer al entorno KDE.

En este caso, Konqueror no es solo un administrador de archivos, ya que también se puede utilizar como explorador web como visor de archivos, por lo que permite navegar por páginas web y también ver documentos pdf o de texto entre otros.

Como los demás administradores de archivos que hemos visto, Konqueror también puede administrar archivos alojados en otros equipos en la red. También se le puede añadir multitud de extensiones mediante plugins, como por ejemplo para acceder a archivos ZIP.

A la hora de utilizarlo en nuestra aplicación, usamos la misma operación que con Nautilus, es decir, ejecutamos el programa con el parámetro del recurso que queremos abrir, smb://máquina/recurso. Con lo que accederemos al recurso "recurso" del equipo "maquina".

2.3.5. Xdg-Open

Esta herramienta es un comando que está presente en las últimas distribuciones de Linux. Esta herramienta trabaja de forma parecida al "start" de Windows. El funcionamiento de esta herramienta es muy sencillo: se le pasa como argumento un fichero o una URL.

En el caso de ser un fichero, busca el programa asociado a este fichero de forma predeterminada y abre el fichero con dicho programa. Por ejemplo, si se trata de un archivo de música, Xdg-Open localizará el reproductor de audio predeterminado por el usuario.

Si el argumento que se le pasa se trata de una URL, ya sea HTTP o FTP, Xdg-Open abrirá la URL con el explorador Web predeterminado por el usuario. Es decir, si ejecutamos lo siguiente, "xdg-open http://www.upv.es", se ejecutará el navegador web con la URL "http://www.upv.es".

Esta última utilidad es la que utilizaremos en nuestra aplicación para poder abrir las paginas web de los nodos descubiertos que publiquen que tienen un servidor web, en el caso de que se este trabajando en una maquina con una distribución de Linux.

En nuestro caso utilizamos el comando"xdg-open http://host:puerto", donde "host" es la IP del nodo que tiene el servidor Web y "puerto" es el puerto donde tiene escuchando el servidor Web.

2.3.6. Ekiga

Ekiga es una aplicación que permite realizar videoconferencias y llamadas IP. Para hacer las videoconferencias se puede disponer de una cuenta SIP, que es como un nombre de usuario, para que cuando alguien quiera comunicarse no tenga que saberse la IP de la máquina con la que quiere comunicarse, si no que a través de su cuenta SIP puede resolver su IP. También se puede trabajar con IP directamente.

Este software hace conexiones punto a punto, es decir, cuando una máquina quiere hacer una videoconferencia con otra no participa una tercera, a no ser que se utilice una cuenta SIP que tendrá que ser traducida a una IP. Por lo tanto, esta aplicación se puede utilizar en una red local que no tenga conexión a internet.

En nuestra aplicación, se usa Ekiga para dar soporte a la videoconferencia. Cuando un cliente quiere dar el servicio de videoconferencia, la aplicación lanza Ekiga. Si quiere ejecutar Ekiga para comunicarse con otro usuario se hace la llamada a Ekiga con los siguientes parámetros:

"ekiga -c sip:host"

Donde el parámetro “-c” indica que le va a hacer una llamada, “sip” indica cual es el protocolo de la llamada y “host” es la dirección de la máquina a la que se va a llamar.

PFC: VisualDNS

Capítulo

3

Detalles de implementación

Esta aplicación se ha desarrollado en el lenguaje de programación JAVA (J2SE) en el entorno de desarrollo NetBeans IDE 6.8. Al estar implementado en JAVA, nuestra aplicación puede ejecutarse en cualquier sistema que disponga una maquina virtual de JAVA, aunque está especificado y probado en sistemas Windows y Linux (Ubuntu).

Cuando la aplicación se ejecuta, lo primero que se debe hacer es configurar los parámetros. Estos parámetros se pueden separar en tres categorías: parámetros de usuario, de red y de difusión.

Los parámetros de usuario son tanto el nombre como la ruta de la fotografía. También en estos entran los servicios que el usuario quiere publicar. Cada usuario se representa en la aplicación como un objeto, él cual dispone de: una variable de tipo String para el nombre, otra también de tipo String para guardar el nombre de la imagen y, por último, un vector de objetos *service*, que dispondrá tantos elementos como servicios se publiquen. El objeto *service* dispone de dos variables String para especificar el nombre del servicio y los parámetros necesarios (en el caso de que no se necesiten parámetros esta variable estará vacía).

Cuando la aplicación recibe un nodo nuevo recibe el objeto que lo representa y después pide la imagen. Esta imagen la guarda en un directorio llamado *discover* donde se almacenarán las imágenes de los nodos descubiertos. Si este directorio no existiera se crearía automáticamente. En este punto hubo un pequeño problema, porque cuando tratábamos de almacenar la imagen por su nombre, no solo teníamos el nombre de la imagen, sino todo la ruta del directorio donde está la imagen de forma local en la máquina correspondiente. Esto se solucionó al principio solicitando al sistema el separador predeterminado para las direcciones de ficheros. Pero esto no era correcto, pues si se estaba en un sistema Linux y se había recibido una imagen de un sistema Windows, los separadores eran distintos. Así que se optó por buscar en la ruta de la imagen los dos tipos de separadores, / y \. Primero buscamos uno, y si no lo encontramos buscamos el siguiente. Nos quedamos con la última posición del separador encontrado y separamos el nombre de la ruta, con lo cual ya tenemos el nombre de la imagen. A continuación se muestra el código que se utiliza para dividir el nombre de la imagen:

```
String filename=usu.photo;
int pos=usu.photo.lastIndexOf("\\");
if(pos<0)pos=usu.photo.lastIndexOf("/");
if(pos>0)filename=usu.photo.substring(pos+1);
```

Los parámetros de red están constituidos por la IP del usuario y la máscara de red. Estos datos se pueden escribir a mano o se pueden resolver de manera automática. Si se elige la manera automática, la aplicación recupera la IP de la máquina mediante la función *java.net.InetAddress.getLocalHost()*, y después ejecutará *ipconfig* o *ifconfig,* dependiendo de si se trata de Windows o Linux, de donde obtendrá la máscara de red correspondiente a esa IP.

Por último, los parámetros de difusión se constituyen por el tiempo entre anuncios de usuarios conocidos por broadcast y el tiempo entre comprobaciones de nodos activos. Una vez configurados todos los parámetros, la aplicación se puede poner en marcha.

Cuando la aplicación se pone en funcionamiento, una vez insertados los parámetros, lanza varios hilos. Al principio lanzaba tres hilos, uno que se encargaba de recibir los mensajes de anuncios de usuarios conocidos, que vienen por UDP y cuyo algoritmo a sido explicado anteriormente. A continuación mostraremos el código utilizado:

```
DatagramSocket socket = new DatagramSocket(4000);
DatagramPacket packet = new DatagramPacket(buffer,buffer.length);
 while(true) {
        socket.receive(packet);
        if (packet.getAddress().getHostAddress().equals(oMain.vIP))
                //El paquete es mio
        else {
                recvList = NodeList.fromByteArray(packet.getData());
                compareLists(packet.getAddress().getHostAddress(),recvList);
                oMain.RefreshList(actualList);
        }
}
```

Como podemos ver, al principio se espera ha recibir un datagrama UDP por el puerto 4000 y una vez se ha recibido se obtiene su información, una lista de nodos. Después se llama a la función *compareList* que compara esta lista obtenida con la que tiene la aplicación. En el caso de nuevos nodos, se le pedirá la imagen al nodo que ha mandado el datagrama, la cual se guardará en el directorio *discover*. Por último se llama a la función *RefreshList* que refresca la interfaz gráfica para que aparezcan los nuevos usuarios, en el caso de que hubieran nuevos.

El segundo hilo se encarga de crear los propios anuncios de usuarios conocidos y mandar los mensajes por broadcast en un bucle donde, por cada pasada, se esperará el tiempo indicado en el parámetro de configuración. A continuación se mostrará el algoritmo utilizado:

MIENTRAS(*siempre*)
Se copia la lista de usuarios conocidos
Se añade el propio usuario a la copia de la lista
Se manda la lista
Se esperá el tiempo configurado.

También mostraremos la implementación en código de este algoritmo:

```
addInitNode();

buffer = sentList.toByteArray();

DatagramSocket socket = new DatagramSocket();

DatagramPacket packet = new DatagramPacket( buffer, buffer.length,
InetAddress.getByName("255.255.255.255") ,4000);

while (true) {

        socket.send(packet);

        sentList.clear();

        i++;

        System.out.println(i + " - Mensaje enviado a " +
        packet.getAddress().getHostName() + ", puerto=" + packet.getPort());

       time = Integer.valueOf(oMain.vTime.trim()).intValue() * 1000;

        Thread.sleep(time);


        copyList(oUdpReceiver.getActualList());
```

```
            addNode(sentList);
            buffer = sentList.toByteArray();
            packet = new DatagramPacket(buffer,buffer.length,
InetAddress.getByName("255.255.255.255"),4000);
        }
```

Podemos ver que lo primero que se realiza es añadir en la lista que se va a enviar un nodo que nos representa a nosotros mismos con la función *addInitNode*. Después se manda el datagrama UDP al pueto 4000, *socket.send(packet)*. A continuación se borra la lista y se espera el tiempo que se a configurado para el siguiente envío. Por último, se copia la lista de nodos conocidos, se añade otra vez el nodo que nos representa y se vuelve a enviar y a realizar todo el proceso.

Por último, el tercer hilo se basa en recibir las conexiones TCP realizadas en el algoritmo del primer hilo, que pide información (imagen) de los usuarios nuevos por esta conexión TCP. Cuando el tercer hilo recibe una conexión TCP, recibe por esta conexión una lista de usuarios que este usuario debe conocer, de los que se solicita información. Después, por cada usuario de la lista se le transmite la información de ese usuario al usuario que la ha pedido abriendo esta conexión TCP. Este es el algoritmo utilizado:

POR (*cada conexión TCP*)
    Se recibe una lista de usuarios
    POR (*cada usuario de la lista*)
        Se transmite la información de ese usuario al solicitante

Después de tener estos algoritmos implementados, se quiso implementar una nueva funcionalidad. Se trata de saber si un usuario está disponible o no, es decir, si un usuario ha dejado la red o no. Es de suponer que si un usuario ha dejado la red ya no podemos utilizar sus servicios ni comunicarnos con él mediante chat o videoconferencia. Por lo tanto, esta funcionalidad debería eliminar de los usuarios conocidos aquellos que dejen de estar en la red.

Para su implementación se han creado dos hilos más. El primero se encarga de hacer peticiones de conexión TCP a los nodos conocidos.

```
while(sw)
{
        time = Integer.valueOf(main.vTime2.trim()).intValue() * 1000;
        sleep(time);//Tiempo de espera
```

```
            if(main.myList.size()>0)
            {
                i=i%main.myList.size();
                try {
                    cliente = new Socket(main.myList.get(i).IP, 7777);
                } catch (Exception ex) {
                    main.myList.remove(i);
                    main.RefreshList(main.myList);
                }
                i++;
            }
        }
```

Como se puede observar, por cada pasada por el bucle se espera el tiempo configurado y después, por cada nodo que se conocen, se intenta realizar una conexión TCP. Si la conexión falla se elimina ese nodo y se actualiza la interfaz gráfica.

La razón por la que se utiliza conexiones TCP en vez de UDP es porque el tráfico TCP está orientado a conexión, por lo cual, en caso de no poder establecer conexión da un error que podemos manejar para saber si un nodo sigue en la red o no. En cambio, el tráfico UDP no está orientado a conexión, se basa en la transmisión de mensajes. Una vez que se manda un mensaje no se sabe si ha llegado a su destino o no, por lo que no podemos utilizar este protocolo para saber si un nodo está en la red o no.

Se empieza con el primer nodo y, si funciona, se continua con el siguiente. En el caso de que falle la conexión, se elimina este nodo de la lista de nodos conocidos y se actualiza la interfaz gráfica. Cuando termina de comprobar todos los nodos se esperá el tiempo configurado en los parámetros y se vuelve a empezar. El segundo hilo se encarga solamente de recibir las conexiones TCP de los otros nodos. Está funcionalidad es muy similar a ir haciendo ping ha los nodos conocidos. El tiempo entre estos intentos de conexión es importante, ya que si se pone un valor pequeño la red se puede inundar de peticiones de intentos de conexión.

Al resolver esta última funcionalidad surgió un problema. El problema era el siguiente: si un nodo eliminaba la red, cuando otro nodo se daba cuenta lo eliminaba de entre sus conocidos. Pero no todos los nodos se daban cuenta a la vez, por lo que un

nodo podría haberlo eliminado mientras que otro no. Si el que no lo ha eliminado manda un anuncio de nodos conocidos, el que lo ha eliminado detecta que conoce al nodo que ha abandonado la red y le pide la información, añadiéndolo como nodo conocido. De forma que suele crearse un bucle no deseado.

Este problema se solucionó modificando el algoritmo de petición de información de nuevos nodos. Cuando se recibía un anuncio de nodos enviado por otro nodo A, se obtenía una lista de los nodos que no se conocen y se solicita información de éstos al nodo A. Si antes de solicitar la información de los nuevos nodos, comprobamos que estos nodos están todavía en la red, se elimina este bucle indeseado. Si no se encuentran en la red, lógicamente no pedimos información sobre esos nodos.

Otra parte interesante es la implementación de la interfaz gráfica principal. Al principio se desarrollo para mostrar 8 botones que representarían a los nodos conocidos de forma fija. Si habían menos de 8 nodos, se mostraban botones vacíos y si habían más solo se mostraban los 8 primeros. Esto se soluciono cambiando la implementación de forma que solo se mostraran los botones de los nodos que se conocen y si habían tantos que no cupiesen en la interfaz aparecería un scroll para poder visualizar todos, es decir, implementando los botones de forma dinámica.

El código empleado es el siguiente:

```
for (i = 0; i < p_list.size(); i++ ) {
        usu=p_list.get(i);
        String id=""+i;
        String filename=usu.photo;
        int pos=usu.photo.lastIndexOf("\\");
        if(pos<0)pos=usu.photo.lastIndexOf("/");
        if(pos>0)
        filename=usu.photo.substring(pos+1);
        photo = new ImageIcon("discovered/" + filename);
        btnUser = new javax.swing.JButton();
        btnUser.setFocusPainted(false);
        btnUser.setIcon(photo);
        btnUser.setText(id);
```

```
btnUser.addActionListener(new java.awt.event.ActionListener() {
        //funcion de chat
        …
});
txtNICK=new javax.swing.JTextField();
txtIP=new javax.swing.JTextField();
txtNICK.setEditable(false);
txtNICK.setBorder(null);
txtIP.setEditable(false);
txtIP.setBorder(null);
txtNICK.setText(p_list.get(i).name);
txtIP.setText(p_list.get(i).IP);
jPopupMenu = new javax.swing.JPopupMenu();
jMenuItem = new javax.swing.JMenuItem();
jMenuItem.setText("Chat");
jMenuItem.addActionListener(new java.awt.event.ActionListener() {
        //funcion de chat Boton derecho
        …
});
jPopupMenu.add(jMenuItem);
if(p_list.get(i).services!=null)
{
        for(int j=0;j<p_list.get(i).services.length;j++)
        {
                //Por cada servicio que publica se añade una opción en el
                menú del botón derecho
        }
```

```
            }
            btnUser.setComponentPopupMenu(jPopupMenu);
            this.jPanel1.add(btnUser);
            btnUser.setBounds((i%4)*(130+10)+10, (i/4)*(130+60)+10, 130, 130);
            this.jPanel1.add(txtNICK);
            this.jPanel1.add(txtIP);
            txtNICK.setBounds((i%4)*(130+10)+10, (i/4)*(130+60)+140, 130, 20);
            txtIP.setBounds((i%4)*(130+10)+10, (i/4)*(130+60)+160, 130, 20);
    }
```

Por cada nodo que se conoce, se obtiene el nombre de su imagen. También un identificador porque como los botones se crean de forma dinámica, no se puede saber a priori con que botón se trabaja cuando se dispara un evento ni a que nodo representa.

Después se coloca la imagen en el botón y se le añade las acciones que se deben hacer en el evento de clic izquierdo en el botón, que abrirá la ventana del chat. A continuación se crean las etiquetas que tendrán la información del nodo: el nombre y la IP. Se añade el menú despegable para el evento del clic derecho en el botón, con las opciones de ejecutar los distintos servicios según haya publicado el nodo al que está representando este botón. Por último se coloca el botón y las etiquetas de información de forma correcta con el algoritmo del final, que genera filas de 4 botones cada fila de forma adecuada.

A la hora de ejecutar las herramientas ajenas a nuestra aplicación, siempre tenemos que descubrir si estamos trabajando en un sistema Linux o se trata de un sistema Windows. Eso lo hacemos con el siguiente código:

```
if(System.getProperty("os.name").toUpperCase().indexOf("LINUX") == -1)
        ejecuciónWindows();
else
        ejecuciónLinux();
```

Lo único que se realiza es consultar en las propiedades del sistema si se está trabajando en un sistema Linux. Si no es el caso pues sabemos que es Windows, no se ha desarrollado el caso de que no sea ni Windows ni Linux.

Para ejecutar el explorador de archivos, como ya sabemos si se trabaja en Linux o en Windows, la tarea es sencilla. En Windows tenemos que ejecutar el siguiente código:

```
Runtime obj = Runtime.getRuntime();
obj.exec(new String[]{"cmd.exe", "/C", "explorer", "\\\\",ip,"\\",source});
```

Lo que ejecuta un proceso independiente al de la aplicación que en este caso es *cmd.exe* con el parámetro *explorer*, lo que equivale a iniciar Explorer en el cmd de Windows. Los parámetros *IP* y *source* equivalen a la IP del nodo que tiene un recurso compartido y el recurso compartido respectivamente.

El caso de Linux es distinto porque dependiendo de la distribución que se utiliza se debe ejecutar su explorador predeterminado. Para la distribución KDE debemos ejecutar Konqueror, mientas que para la GNOME debemos ejecutar Nautilus.

```
Runtime obj = Runtime.getRuntime();
 if(getLinuxDesktop().equals("kde"))
        obj.exec("konqueror smb://"+ip+"/"+source);
if(getLinuxDesktop().equals("gnome"))
        obj.exec("nautilus smb://"+ip+"/"+source);
```

La función *getLinuxDesktop()* consulta en las variables de entorno para descubrir en que distribución estamos trabajando y dependiendo de ello ejecuta un explorador u otro.

Para el caso de abrir una URI tenemos una solución muy parecida. Para averiguar si se trata de Linux o no, utilizamos la solución que hemos explicado anteriormente. Para el caso de Windows utilizamos la herramienta *start*:

```
Runtime obj = Runtime.getRuntime();
obj.exec(new String[]{"cmd.exe", "/C", "start", uri});
```

Ejecuta por consola la herramienta *start* pasando como parámetro la dirección web del servidor al que se desea conectar. Para Linux es muy parecido pero utilizando la Herramienta Xdg-open:

```
Runtime obj = Runtime.getRuntime();
obj.exec("xdg-open "+uri);
```

Por último, explicaremos como se ejecuta la herramienta Ekiga desde nuestra aplicación. En este caso también diferenciamos entre sistemas Linux y Windows porque no tienen el mismo comportamiento en los dos sistemas.

Este es el código implementado en el caso de sistemas Linux:

```
Runtime obj = Runtime.getRuntime();
try {
        if(vEkiga==null)vEkiga=obj.exec("ekiga" + uri);
        else obj.exec("ekiga" + uri);
} catch (IOException ex) {
        //mostrar mensaje de error
        ...
}
```

En el caso de Ekiga, guardamos una variable con el proceso para poder destruirlo al cerrar nuestra aplicación. Por eso, lo que primero que se comprueba es si esta variable está vacía o no. En el caso de que este vacía, es la primera vez que se ha ejecutado Ekiga, se guarda su proceso en la variable. Puede haber dos casos de ejecución de Ekiga. El primero es el caso en el que el usuario quiere publicar que ofrece el servicio de videoconferencia, por lo tanto no quiere contactar con nadie en ese momento. En este caso el parámetro "*uri"* con el que se ejecuta estará vacío. El segundo caso ocurre cuando alguien quiere contactar con otra persona que ha publicado el servicio de videoconferencia. En este caso el parámetro "uri" dispone la dirección correspondiente al usuario con el que se quiere contactar. En el primer caso citado, cuando se comprueba la variable del proceso Ekiga siempre estará vacía. Pero en el segundo caso puede que ya este en marcha y que la variable ya tenga el proceso de Ekiga y se este ejecutando en background. Esto no supone un problema en Linux, pues se ejecuta de igual forma sin ningún problema. En sistemas Windows, esto si que supone un problema. Si se intenta realizar una videoconferencia mediante un comando nos muestra un error informándonos de que la aplicación ya se está ejecutando. Por eso hemos optado por esta implementación:

```
boolean ok=true;
String Dir = System.getenv("ProgramFiles");
if(vPathEkiga==null)this.vPathEkiga="\""+Dir+"\\Ekiga\\ekiga.exe\"";
if(vEkiga!=null)
```

```
        {
                vEkiga.destroy();
                Runtime obj = Runtime.getRuntime();
                try {
                        vEkiga = obj.exec(vPathEkiga + uri);
                } catch (IOException ex) {
                        ex.printStackTrace();
                }
        }
        else{
                do
                {
                        Runtime obj = Runtime.getRuntime();
                        try {
                                vEkiga = obj.exec(vPathEkiga + uri);
                                ok=true;
                        } catch (IOException ex) {
                                //mostrar error y dar a elegir la ruta de Ekiga
                        }
        }while(!ok);
}
```

Lo primero que se realiza es obtener la ruta de Ekiga por defecto. Una vez se ha obtenido, si no se ha ejecutado ya antes, es decir, si la variable donde se guarda el proceso de Ekiga está vacía, se ejecuta guardando el proceso resultante en la variable. En caso de que no se encuentre la herramienta Ekiga en su ruta por defecto, se lo comunica al usuario y da la posibilidad de explorar en busca de la ruta alternativa o de cancelar la acción. Si ya se había ejecutado antes, es decir, la variable donde se guarda el proceso no está vacía, hay que destruir ese proceso y crear uno nuevo. Si la acción es para iniciar una comunicación la variable "*uri*" contendrá la dirección necesaria para conactar.

También cabe destacar que cuando se cierra nuestra aplicación se comprueba si la variable que guarda el proceso de Ekiga está vacía. Si no lo está, al cerrar la aplicación se cierra este proceso para cerrar Ekiga a la vez que se cierra nuestra aplicación.

Capítulo

4

Validación y pruebas

En este apartado iniciaremos la aplicación en una red que se ha montado para la prueba de la aplicación. Esta red está constituida por tres nodos con sistemas Linux, y obviamente se trata de una red Ad-Hoc, donde dos de ellos solo verán a un nodo, el cual verá a los dos tal y como muestra la siguiente imagen:

Figura 4.1: Ejemplo de red de pruebas.

Como se puede ver en la imagen, el nodo del medio se encargará de encaminar los paquetes entre los otros dos nodos.

Dos de los nodos no pueden alcanzarse, utilizando para ello las Iptables. Pero las restricciones no se hacían a nivel de IP, ya que, aunque los paquetes se enrutaran por el nodo restante, estos llegarían con la IP del origen y también se descartarían. Por eso se utilizo Iptables pero restringiendo mediante la dirección MAC:

iptables -A INPUT -m mac –mac-source “dirección MAC” -j DROP

El nodo restante debe tener algún tipo de enrutamiento, para que cuando le llegue un paquete que no es para él, lo mande a quien le pertenece. Esto lo conseguimos con la siguiente instrucción:

echo 1 > /proc/sys/net/ipv4/ip_forward

Una vez la red está montada, ejecutamos la aplicación. Al tratarse de una aplicación en formato JAR se deberá ejecutar con el siguiente comando, dentro del directorio donde se encuentre el ejecutable:

```
java -jar Visual_DNS.jar
```

Después se debería de configurar los parámetros de la aplicación según se crea conveniente.

En este apartado, lo primero que haremos es explicar como se ejecutaría la aplicación y como se configuraría. Después veremos como, dependiendo del parámetro de tiempo entre anuncio de nodos conocidos, un nodo nuevo que se añade a la red puede tardar más o menos a la hora de conocer todos los nodos de la red.

A continuación explicaremos paso a paso como poner en marcha la aplicación y como usar todas las utilidades que ofrece. Como hemos dicho, esta aplicación debe ejecutarse en una red que tenga algún algoritmo de enrutamiento, y no necesariamente deberá ser una red Ad-Hoc. De hecho puede tratarse de una red WIFI con puntos de acceso o incluso una red cableada Ethernet.

Una vez ejecutada la aplicación hay que configurar los parámetros. Para eso hay que hacer clic en el menú Configuration como se muestra en la figura 4.1. Como se puede observar, hay tres submenús que corresponden a los parámetros de usuario, los de red y por último los de difusión (broadcast).

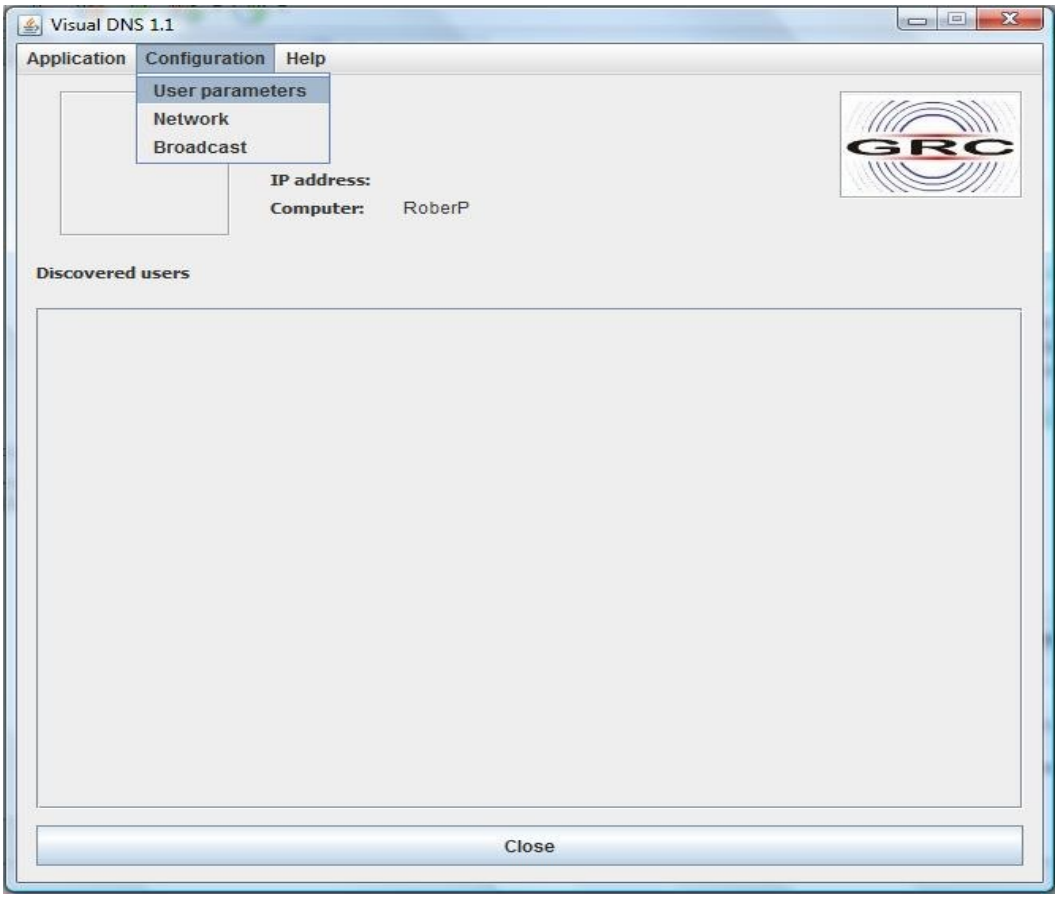

Figura 4.2: menú configuration.

Empezaremos explicando la configuración de los parámetros de usuario. La ventana de configuración corresponde a la figura 4.2 que está más abajo. Aquí debemos poner el nombre por el que queramos que se nos conozca, y también podemos añadir a qué grupo de trabajo pertenecemos, en caso de que estemos divididos en diferentes grupos. También podemos añadir el nombre de la imagen que nos va a representar, si lo sabemos, o podemos buscarla pulsando el botón a la derecha del hueco del nombre de la imagen, lo que abrirá una ventana de exploración.

Una vez hecho esto, pasamos a la parte de servicios que se ofrecen. Podemos ofrecer hasta tres servicios. El primero se trata sobre el servidor web. Al activarlo, podemos especificar por qué puerto está escuchando el servidor web, (por defecto el puerto es el 80).

El segundo es el servicio de recursos compartidos, donde una vez activo debemos especificar el nombre del recurso que se comparte. Si no se especifica ningún recurso, cuando un usuario acceda podrá ver todo lo que comparte nuestra máquina.

Y por último nos queda la opción de la videoconferencia, la cual, si la activamos ejecutará de forma automática la herramienta Ekiga.

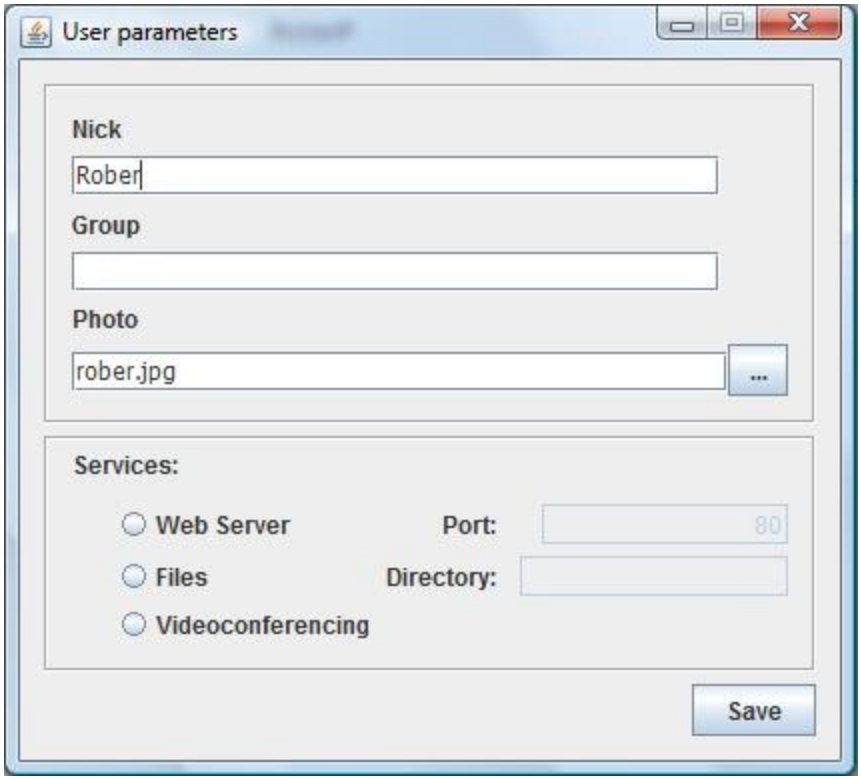

Figura 4.3: parámetros de usuario.

Ahora pasaremos a los parámetros de red (Ver Figura 4.3). Primero tenemos un combobox que nos muestra las interfaces que hay disponibles. Después tenemos un recuadro donde podemos especificar la IP. Por último tenemos que especificar la máscara de red. Todo esto se puede hacer de forma automática si se pulsa el botón Load, que rellenará los campos de IP y máscara de red de forma automática. A continuación mostramos la ventana de configuración de los parámetros de red.

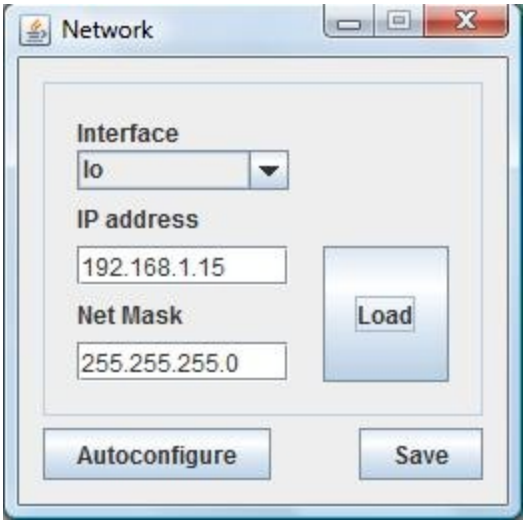

Figura 4.4: parámetros de red.

Por último pasamos a configurar los parámetros de difusión. Esta parte de configuración quizás sea la más fácil, pero no por ello pierde importancia, ya que un mal uso de estos parámetros puede hacer que la red pierda calidad. En este caso solo hay que configurar dos parámetros, el tiempo entre anuncios de nodos conocidos y el tiempo entre intentos de conexión, en segundos. En la siguiente imagen se muestra la ventana de configuración.

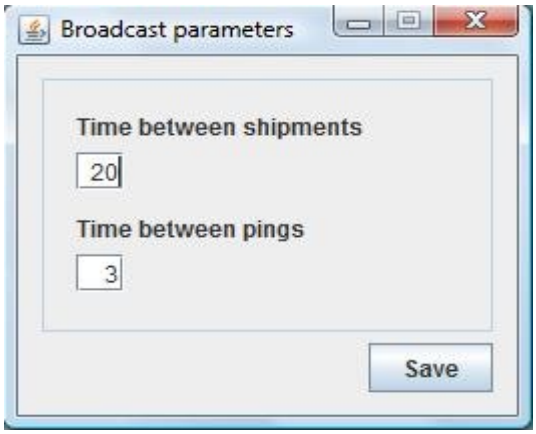

Figura 4.5: parámetros de difusión.

Una vez echa la configuración de los parámetros ya se puede poner en marcha la aplicación. En la ventana principal, en el menú Application, dándole clic a Start la aplicación empezara a mandar mensajes y a recibirlos.

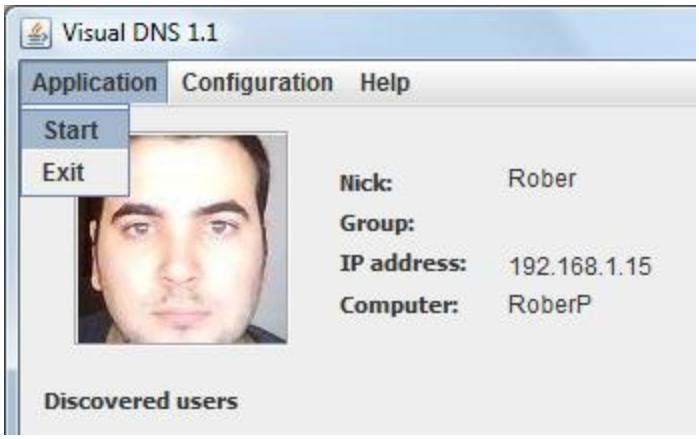

Figura 4.6: Puesta en marcha de la aplicación.

Una vez ya está en marcha la aplicación solo es cuestión de tiempo que se nos muestren todos los usuarios de la red (en nuestra prueba serán solo dos). Una vez ya hay usuarios nos van saliendo en la ventana principal. Ahora se puede acceder a los servicios que publiquen los usuarios. Siempre podemos acceder al chat, que es propio de la aplicación, haciendo clic izquierdo en el usuario deseado. A continuación se mostrará la ventana de chat que se abre al hacer clic izquierdo en el usuario.

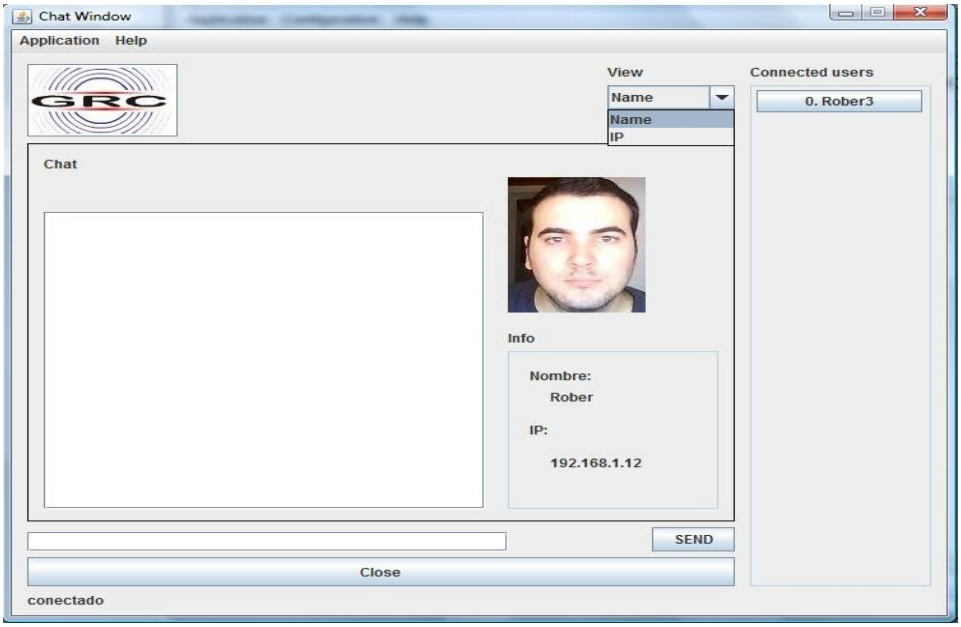

Figura 4.7: Ventana de chat.

En el panel de la derecha aparecerán una lista de botones con los demás usuarios conectados, que pueden estar representados por su nombre o por su IP. Al pulsar uno de estos botones se abrirá otra ventana de chat con ese usuario.

Para acceder a otros servicios de los usuarios, tendremos que, estando en la ventana principal, hacer clic derecho en algún usuario. Aparecerá un menú con las opciones disponibles como muestra la siguiente figura:

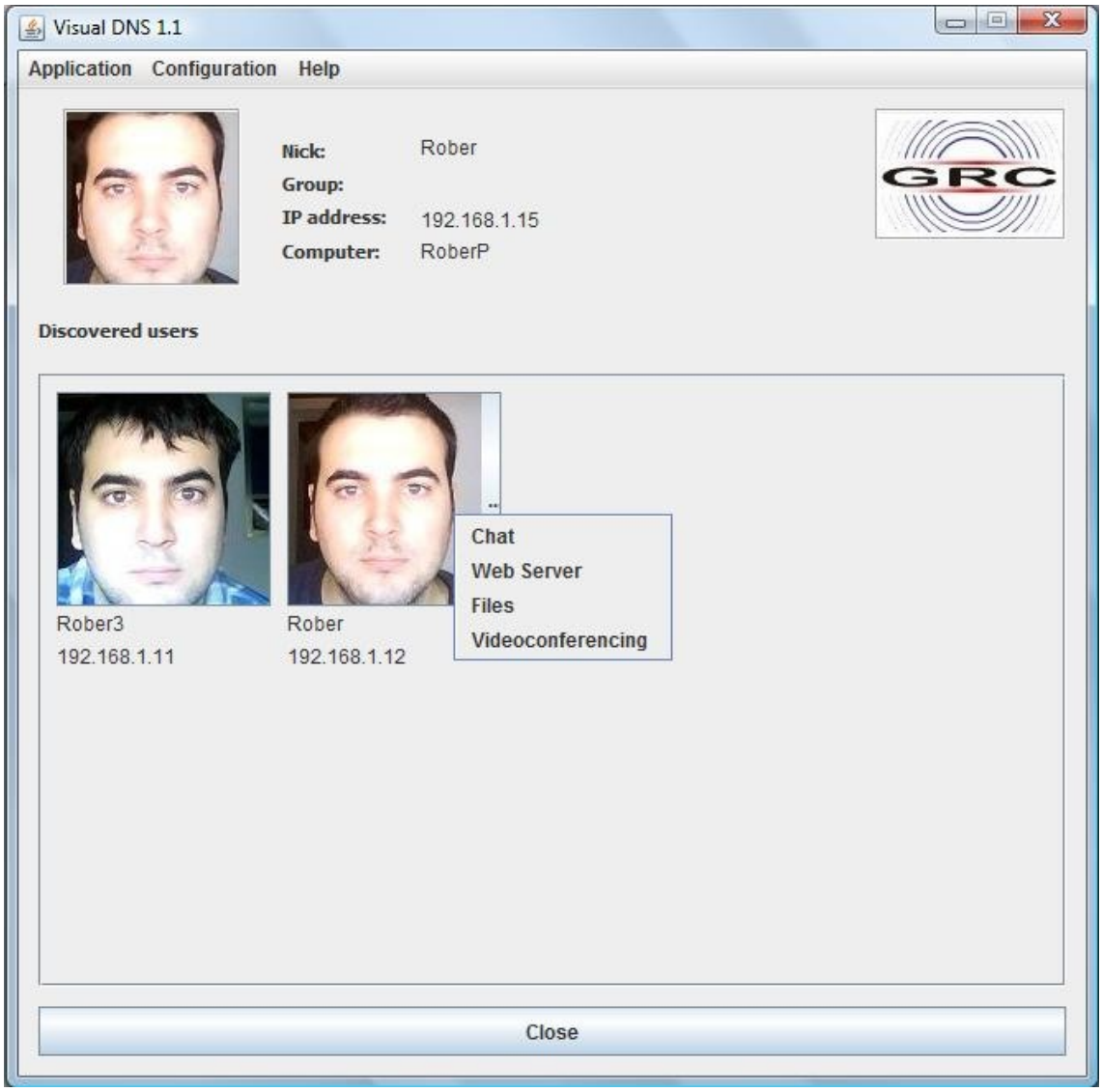

Figura 4.8: Servicios publicados.

Como podemos ver salen los distintos servicios que ese usuario ha publicado. También está el servicio chat, dando la posibilidad de acceder también de esta forma.

Una vez visto esto veamos el comportamiento de la aplicación según el tiempo configurado entre anuncios de nodos conocidos. Para hacer esta prueba en nuestra red, hemos medido la latencia que tiene un nodo al conectarse por primera vez hasta que consigue conocer a los nodos de la red. Como dijimos con anterioridad, tendremos tres nodos: uno que alcanza a los otros dos y puede ser alcanzados por ellos, y los otros dos que solo se alcanzan a través del primero. El nodo que conectamos para medir los tiempos es uno de los últimos, de los "extremos". Aunque el nodo vecino al que se conecte tiene un tiempo fijo entre anuncios, el tiempo de latencia no es constante, pues puede que entre en la red en mitad del tiempo de espera para anunciar los nodos conocidos, o en el peor de los casos que este justo en la situación después de haber

mandado ya el anuncio. Por ese motivo, se han asignado diferentes tiempos de latencia entre envíos de anuncios de nodos conocidos y se han hecho varias pruebas con cada latencia, obteniendo tiempos promedios. A continuación se mostrará una gráfica que expresa los resultados obtenidos.

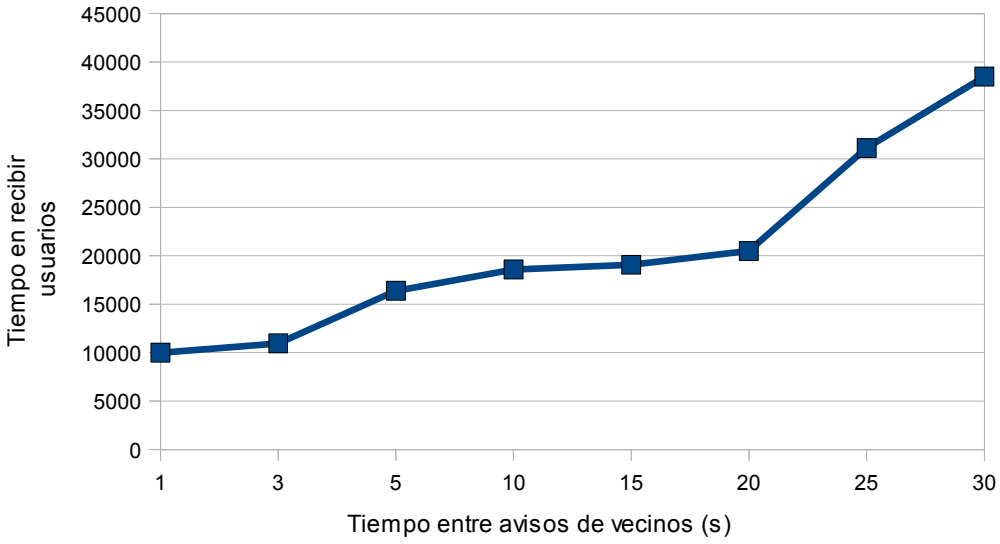

Figura 4.9: gráfica de tiempos entre avisos de vecinos.

Como se puede ver, tiene un comportamiento semejante a uno lineal. A mayor tiempo entre anuncios, más tardará el nodo vecino en tener la representación de los usuarios de la red. Eso ya se intuía, por lo que podíamos pensar en poner un tiempo pequeño entre avisos, o mejor aun no poner ningún tiempo. Esto perjudicaría el rendimiento de la red, pues la red se inundaría de paquetes, algo indeseable. Para solucionar esto tendríamos que saber como funciona la red y escoger un tiempo entre avisos que perjudique lo menos posible el rendimiento de la red y que disminuya en la medida de lo posible el tiempo que va ha tardar un usuario nuevo en recibir a los demás usuarios de la red.

Capítulo

5

Conclusiones y trabajo futuro

En esta memoria se ha presentado una aplicación capaz de funcionar en una red Ad-Hoc, descubriendo los usuarios de la red y dando la posibilidad de ofrecer chat entre ellos, además de servicios de videoconferencia, compartición de archivos y servidor web.

Después de un duro trabajo, se consiguió alcanzar la meta esperada. Se pudo implementar la aplicación y que tuviera un funcionamiento correcto.

Al principio hubo algún problema, pues en la carrera no se habla mucho de las redes Ad-Hoc, pero se fue obteniendo información de este tipo de red y se pudo tener bastante información como para poder realizar la aplicación a medida.

Otro de los problemas a la hora de construir una red Ad-Hoc ha sido el lograr tener un número de portátiles suficiente para validar la aplicación.

Una vez solucionados los problemas se implemento la aplicación VisualDNS, capaz de descubrir los nodos que hay en la red, ya sea una red Ad-Hoc o de otro tipo. También se ofrece la posibilidad de publicar servicios para que puedan ser vistos por otros usuarios que pertenezcan a la red. Esta aplicación puede ser vista parecida a otras, como por ejemplo Messenger. La principal diferencia es que esta aplicación trabaja de forma local y no necesita de conexión a Internet. También puede ser semejante a la aplicación Microsoft NetMeeting, pero, a diferencia de esta aplicación, la nuestra no se necesita saber de antemano la dirección de los demás usuarios, ya que nuestra aplicación descubre a los usuarios de la red.

Cabe destacar que las redes inalámbricas están en auge, tanto en el mercado como en el desarrollo, ya que son más cómodos de utilizar por no tener cables y ofrecer mucha más movilidad. Por eso, muchos dispositivos actuales disponen de tecnología inalámbrica para poder conectarse. Uno de los problemas que podemos encontrar es que no siempre hay infraestructuras para conectarse inalámbricamente. Es en ese caso cuando se puede optar por utilizar una red Ad-Hoc. Estas redes, en las que no se necesitan ningún tipo de infraestructura se están popularizando ya que, si varias personas se unen en un lugar donde no hay ninguna infraestructura, pueden montar una red Ad-Hoc y estar comunicados entre si. Por ese motivo, se cree que este tipo de red aumentará su uso.

En un futuro, se podrían agregar más servicios a nuestra aplicación, o incluso hacerla más portable para poderla ejecutar en PDA's y móviles por ejemplo.

También se pueden implementar nuevos servicios, se podría añadir la posibilidad de poder categorizar a los usuarios descubiertos por sus características, como los que ofrecen videoconferencia, o por gustos de usuario (amigos, familia …). también se puede modificar la aplicación para que no solo funcione en redes locales si se dispone de conexión a Internet, permitiendo la comunicación entre nodos a través de internet.

Otra de las opciones de trabajo cara el futuro para mejorar esta aplicación sería la implementación de las herramientas que utiliza nuestra aplicación, es decir, que nuestra aplicación no necesite de otras herramientas para dar los servicios que actualmente da, que sea independiente.

Referencias Bibliográficas

- [1] PERKINS, C. E., BHAGWAT, P.: “Highly Dynamic Destination-Sequenced Distance-Vector Routign (DSDV) for Mobile Computers”, Comp. Commun, 1994.
- [2] PERKINS, C. E.: “Ad Hoc Networking”, Addison-Wesley, 2001.
- [3] JOHNSON, D. B., MALTZ, D. A.: “Dynamic Source Routing in Ad-Hoc Wireless Networks”, Mobile Computing, 1996.
- [4] Wikipedia, Explorador de Windows, 25 jul. 2010. Disponible en: http://es.wikipedia.org/wiki/Explorador_de_Windows .
- [5] Wikipedia, Nautilus, 21 jul. 2010. Disponible en: http://es.wikipedia.org/wiki/Nautilus_(informática) .
- [6] Wikipedia, Konqueror, 25 jul. 2010. Disponible en: http://es.wikipedia.org/wiki/Konqueror .