# UNIVERSIDAD CARLOS III DE MADRID

# ESCUELA POLITÉCNICA SUPERIOR


## PROYECTO FIN DE CARRERA

## INGENIERÍA DE TELECOMUNICACIÓN

# IMPLEMENTACIÓN DE REDES OVERLAY JERÁRQUICAS USANDO EL PROTOCOLO P2PP

**Autor:** Roberto González Sánchez

**Tutor:** Isaías Martínez Yelmo.

Octubre 2009

EL TRIBUNAL

**Presidente:**

**Secretario:**

**Vocal:**

Realizado el acto de defensa y lectura del Proyecto Fin de Carrera el día 30 de Octubre de 2009 en Leganés, Facultad de la Escuela Politécnica Superior de la Universidad Carlos III de Madrid, acuerda otorgarle la calificación de:

Fdo. Presidente Fdo. Secretario Fdo. Vocal

# AGRADECIMIENTOS

Es difícil pensar en todas las personas a las que debo agradecerles algo después de tantos años de carrera y tantos momentos en los que me han prestado su apoyo sin pedir nada a cambio.

En primer lugar, como no, los primeros a los que tengo que dar las gracias son a mi padre, mi madre y mi hermana. Aunque simplemente se podría decir que me lo han dado todo, en este caso, debo agradecerles sobre manera la confianza incondicional que han tenido en mí y que siempre están ahí, en todos los buenos momentos y sobre todo en los malos.

A Ángela, por aguantar mi mal humor en los días malos, por ayudarme en cuanto puede y animarme sin parar, por estar siempre junto a mí. Tampoco puedo olvidar su ayuda en la realización de esta documentación.

A mis amigos de toda la vida. Por alejarme de la universidad cuando estoy fuera de su recinto. Porque siempre hay alguno dispuesto a tomar una caña o jugar un partidillo y relajarse un rato.

A mis “nuevos” amigos de la Carlos III. Por sufrir las “peculiaridades” de esta universidad a mi lado. Por tanto apoyo, tantos apuntes y tantas explicaciones necesitadas. Pero sobre todo por estar ahí, por irnos de fiesta, por confiar en mí, por no hablar de cosas de la uni…

Por último, no me puedo olvidar de Isaías y de Carmen. Por guiarme a lo largo de todo el proyecto y por permitirme trabajar junto a ellos aprendiendo cosas que no se enseñan en las clases.

Gracias a todos.

# RESUMEN

En los últimos años, la transmisión de contenido multimedia se ha convertido en uno de los servicios más usados de Internet dando lugar a arquitecturas de red que se desmarcan de los tradicionales modelos cliente servidor. Sin embargo, la popularidad de estas aplicaciones no ha ido acompañada de un igual avance en su estudio y entendimiento.

Entre estas nuevas estructuras se encuentran las redes overlay, uno de cuyos usos más extendidos es la creación de redes P2P. Estas redes se caracterizan por crear una red sobre Internet con una topología lógica completamente diferente a la del nivel de red. Dependiendo de la manera en la que se genere esta topología las podemos denominar estructuradas o no estructuradas.

El objetivo principal de este proyecto es demostrar la mejora de rendimiento que supone el añadir un modelo jerárquico a una red overlay estructurada. Para ello buscamos una implementación disponible de una de estas redes y la modificamos para que funcione jerárquicamente. En este caso el protocolo elegido ha sido Peer-to-Peer Protocol (P2PP).

Tras realizar las modificaciones necesarias para que el software funcionara de acorde a nuestras necesidades, se paso a una fase de diseño y realización de pruebas en una red emulada usando el software ModelNet.

Finalmente se analizaron los resultados obtenidos llegando a la conclusión de que la inclusión de la jerarquía mejora el rendimiento de la red de una manera notable según crece el número de nodos participantes en la red.

# ÍNDICE

# ÍNDICE DE FIGURAS

# CAPÍTULO I

# INTRODUCCIÓN

# 1. INTRODUCCIÓN

En los últimos años, la transmisión de contenido multimedia se ha convertido en uno de los servicios más usados de Internet. Sin embargo, estos nuevos servicios necesitan aún de un amplio desarrollo.

Tanto las aplicaciones de compartición de archivos que se empezaron a crear con Napster a finales del siglo XX como las actuales aplicaciones que nos proporcionan comunicación de video y voz en tiempo real como Skype se encuentran con el problema de buscar información en una red enorme, cuyos identificadores son direcciones IP sin relación alguna con el contenido guardado en las máquinas que representan. Dentro de este marco podemos entender la importancia de las redes *overlay*.

La principal característica de una red *overlay* es que los terminales informáticos que la conforman, aún estando interconectados entre sí mediante routers IP, se organizan de tal manera que definen una nueva estructura de red superpuesta a la ya existente. Se trata por tanto de sistemas puramente distribuidos, y que pueden ser utilizados en muchos ámbitos de interés: por ejemplo, para encontrar contactos en un sistema de VoIP. Aunque quizá, las redes *overlay* más conocidas sean las denominadas *peer-to-peer* (P2P), muy utilizadas para la descarga eficiente de grandes volúmenes de información. Entre las redes *overlay* podemos distinguir dos tipos de redes *overlay* P2P: las estructuradas y las no-estructuradas.

Las redes P2P no-estructuradas, aunque requieren la presencia de al menos un controlador (*rendez-vous*), tienen la ventaja de que el proceso de búsqueda de la información a descargar es extremadamente sencillo y eficaz. Ello no ocurre cuando la red P2P es del tipo estructurada, aunque en este caso

la ventaja es que no se requiere de ningún controlador central y en consecuencia la red *overlay* es más escalable, robusta y también mucho más fácil de reorganizar frente a posibles cambios de contorno.

En las redes estructuradas, la construcción de la red *overlay* se realiza en base a un algoritmo conocido de antemano. Dicho algoritmo identifica a cada terminal mediante la aplicación de una función *hash* a su dirección MAC o IP. Conocido el identificador, el algoritmo es capaz de situar al terminal en una posición concreta de la red, pasando a ser un nodo de la red *overlay*. Por contra, en las redes no-estructuradas los terminales se sitúan en la red *overlay* gracias a la intervención de uno (o varios) terminales *rendez-vous* con funciones de gestión de la red.

El objetivo inicial de este proyecto es conseguir modificar una implementación actual de algún protocolo de redes overlay estructuradas para que se comporte de un modo jerárquico. De esta manera, conseguiremos que los nodos considerados cercanos se encuentren también cerca en la estructura de la overlay, esperando un descenso en el número de saltos necesarios para encontrar la información buscada.

## *1.1 Organización de la memoria*

El presente documento se divide en los capítulos que se enumeran a continuación:

- El Capítulo I nos hace una pequeña **Introducción,** presentando las redes overlay, y algunas de sus posibles aplicaciones, así como la motivación de este trabajo. También añadimos una pequeña descripción del contenido de este documento.
- El Capítulo II se corresponde con el **Estado del Arte** y describe el estado actual de la investigación en redes overlay jerárquicas. Este capítulo enumera, además, las posibles implementaciones de partida para el desarrollo de nuestro experimento.

- El Capítulo III versa sobre el **Diseño e implementación de la aplicación** y en él se presenta la estructura original de la implementación de partida, así como las modificaciones realizadas para adaptar su funcionamiento al de una red jerárquica.
- El Capítulo IV se basa en la **Verificación** de nuestro programa. En esta sección, tras una introducción que nos ayuda a comprender los resultados esperados, nos encontramos con las herramientas y las configuraciones usadas para realizar las pruebas. Por último se presentan y analizan los resultados obtenidos.
- El Capítulo V presenta las **Conclusiones** obtenidas durante todo el proceso de realización de este proyecto y finalmente se enumeran algunas líneas de trabajo futuro.
- En la sección de **Apéndices** nos encontramos con la memoria económica del proyecto, un glosario de términos y dos pequeños manuales para la comprensión e instalación de dos importantes herramientas como son Proxmox y ModelNet.
- Por último se ha añadido una **Bibliografía** con las fuentes de información usadas durante la elaboración del proyecto y su correspondiente documentación.

# CAPÍTULO II

# ESTADO DEL ARTE

# 2. ESTADO DEL ARTE

## *2.1 Redes Overlay Estructuradas (DHTs)*

Uno de los usos más extendidos de las redes overlay son las conocidas redes P2P. El término “peer-to-peer” se usa en muchos contextos con diferentes significados. En los últimos años se ha usado para referirse a la forma de cooperación que surgió con la aparición del programa de intercambio de archivos Napster. Con esta aplicación, los archivos de música eran intercambiados entre ordenadores dependiendo de un directorio central para conocer que peer tenía cada archivo. Napster dejó de funcionar años después debido a problemas legales relacionados con su tecnología, pero un gran número de sistemas como Gnutella o Freenet en los que el directorio central fue sustituido por un proceso de inundación en el que cada ordenador conectaba con peers aleatorios dentro de la red peer-to-peer y preguntaba a sus vecinos, los cuales repiten esta búsqueda hasta que esta es resuelta. Podemos considerar el grafo aleatorio generado como un ejemplo de de red overlay ya que es una red de nivel de aplicación que funciona sobre el nivel de transporte de Internet con su propia topología y enrutamiento.

Con el paso de los tiempos se vio que este esquema, en el que se generaba una red aleatoria, tenía cosas buenas pero también cosas malas. Las virtudes de este sistema radican en la simplicidad de la solución y su habilidad para distribuir completamente el control. La eliminación del control central es muy importante ya que evita puntos únicos de fallo y permite la construcción de sistemas distribuidos a gran escala. Por otro lado su mayor defecto se encuentra en la gran cantidad de tráfico producido que puede hacer la solución no escalable. La necesidad de hacer escalables las redes P2P resultó en la

creación de las redes overlay estructuradas, también llamadas DHTs (Distributed Hash Tables).

El enfoque principal tomado como solución, se basa en que los peers actúen como una estructura de datos distribuida con operaciones definidas de antemano, formando una tabla de hash distribuida. Las dos operaciones básicas son Put(key, value) y Get(key). La operación Put tiene como resultado el almacenamiento de un valor en uno de los peers de modo que cualquier otro peer pueda hacer una operación Get y recuperar el peer que tiene ese valor. La primera solución encontrada consistía en que cada peer conociera a todos los demás peer, y por tanto las operaciones Get se harían siempre con un solo salto. Como parece obvio, esta solución no es escalable, por lo que debemos introducir otra restricción: cada nodo solo debe conocer a un número "pequeño" de peers. Desde un punto de vista de teoría de grafos podemos pensar que cada nodo debe tener un número de vecinos en relación al radio del grafo.

### *2.1.1 Definiciones*

A continuación voy a enumerar algunas de las definiciones básicas para el entendimiento de las siguientes secciones de este documento. Hemos mantenido la notación inglesa debido a que la traducción de alguno de los términos puede causar confusión.

Values. El conjunto de valores V, pueden ser ficheros, entradas de directorio, usuarios de un programa VoIP… Cada valor tiene una key del conjunto Keys(V). Si el value es un archivo la key puede ser, por ejemplo, el checksum del archivo, si es un usuario puede ser un hash del nombre de usuario.

Nodes. El conjunto P de máquinas o procesos, también nos referimos a ellos como peers. Keys(P) es un conjunto de claves únicas para cada Nodo. Usualmente se usa la IP del nodo o su clave pública.

El espacio de identificadores (IDs). Una de las asunciones fundamentales de todos los DHTs es que las keys de los valores y los nodos son mapeados en un rango usando una función de hash. Por ejemplo, la IP de los nodos y el checksum de los archivo son codificados usando un algoritmo de hash (como

SHA-1 o md5) para obtener un identificador (ID) de n bits (habitualmente de 128 bits). El término ID es usado para referirnos a estos hashes. El término espacio de IDs hace referencia al rango de posibles identificadores y su tamaño normalmente es N.

Items. Cada uno de los pares key-value.

### 2.1.2 Chord

Chord [SMLN+03] es un protocolo de búsqueda distribuida. Este protocolo únicamente proporciona la operación de conseguir una clave, mapeando esta clave a un nodo. La localización de datos puede ser implementada fácilmente sobre Chord asociando una clave a cada objeto y almacenando la pareja clave-objeto en el nodo que mapea a esa clave.

Chord adapta fácilmente la entrada y salida de nodos, permitiendo las búsqueda aún cuando el entorno cambia continuamente.

El conjunto de claves está ordenado en una topología con forma de anillo en el que las claves están asociadas al nodo que mapea la clave más próxima a la suya.

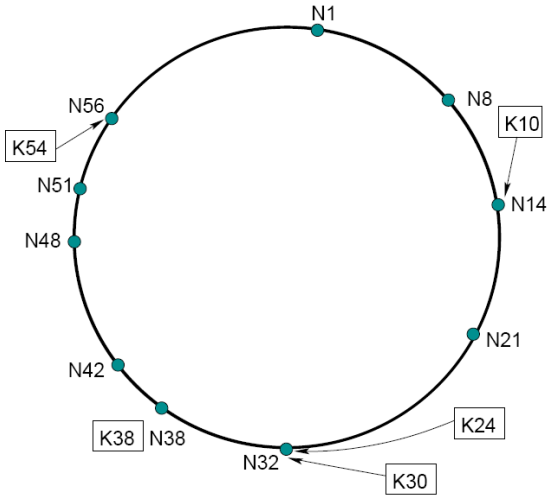


**Figura 1:** Ejemplo de red formada por Chord [1]

[1] Figura obtenida de [SMLN+03]

Los peers de Chord construyen una tabla de rutas (llamada *finger table*) de modo que conocen a más peers cuanto más cerca se encuentren de ellos en la topología.

### *2.1.3 Kademlia*

Kademlia [MM02] es una tabla de hash distribuida que coloca los nodos en un árbol binario. Este protocolo usa una métrica XOR para calcular la distancia entre las diferentes claves, lo que hace que todas las búsquedas por la misma clave se encaminen hacía el mismo destino.

Para cada nodo, nosotros podemos dividir el árbol en un conjunto de sub-arboles que no contienen al nodo. El de mayor nivel sería la mitad del árbol original que no contiene al nodo, el siguiente, la mitad del árbol restante que no contiene al nodo y así sucesivamente. De modo que para un nodo con prefijo 0011, estos sub-árboles serían los que tengan raíz 1, 01, 000 y 0010. Kademlia se asegura de que cada nodo conozca al menos a otro nodo en cada uno de los sub-árboles generados.

De esta manera, todo nodo puede preguntar iterativamente a sus nodos conocidos si saben cómo encontrar la clave requerida, obteniendo en cada salto una reducción del árbol mayor o igual a la mitad.

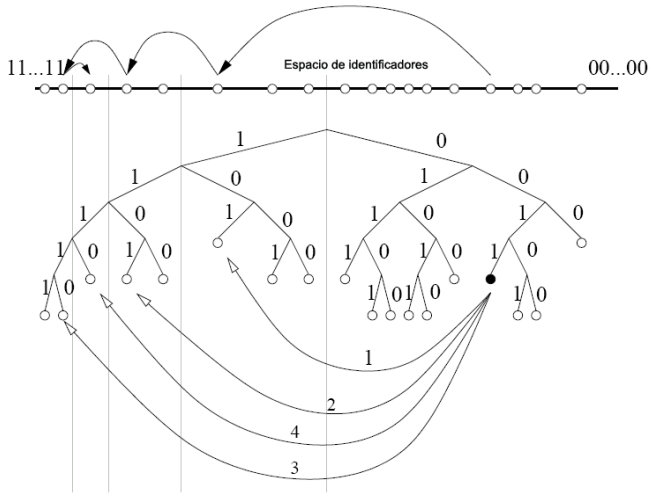


**Figura 2:** Búsqueda del nodo 1110 por el nodo 0011 en Kademlia [2]

[2] Figura obtenida de [MM02]

Como vemos en la imagen anterior, siguiendo el ejemplo de [MM02], vemos que si el nodo con raíz 0011 quiere preguntar por el nodo con raíz 1110. Deberá preguntar al nodo que conozca del sub-árbol con raíz 1 (en este caso el 101). Este le guiará al nodo que conozca en el sub-árbol con raíz 11 (el 1101), el cual a su vez le indicará el siguiente salto dándole información sobre su conocido en el sub-árbol con raíz 111 (el 11111), el cual ya conoce la dirección del nodo buscado.

Este sistema de búsqueda binario puede ser modificado para cubrir una mayor cantidad de bits en cada salto, de manera que se genere un árbol que no sea binario.

## *2.2 Redes Overlay Jerárquicas*

Una red overlay jerárquica es aquella en la que los nodos participantes se organizan en grupos, los cuales están a la vez comunicados mediante otra red overlay. Parece claro que las redes jerárquicas son la evolución natural de las redes estructuradas, ya que permiten una solución más escalable para un número alto de nodos.

En estas redes los peers forman grupos según un criterio dependiente de la aplicación para la que se vaya a usar la red overlay. Esta ordenación puede realizarse según criterios de proximidad geográfica, pero esté no es el único criterio posible ya que según el caso puede ser que esta ordenación no se pueda llevar a cabo o que otra sea beneficiosa.

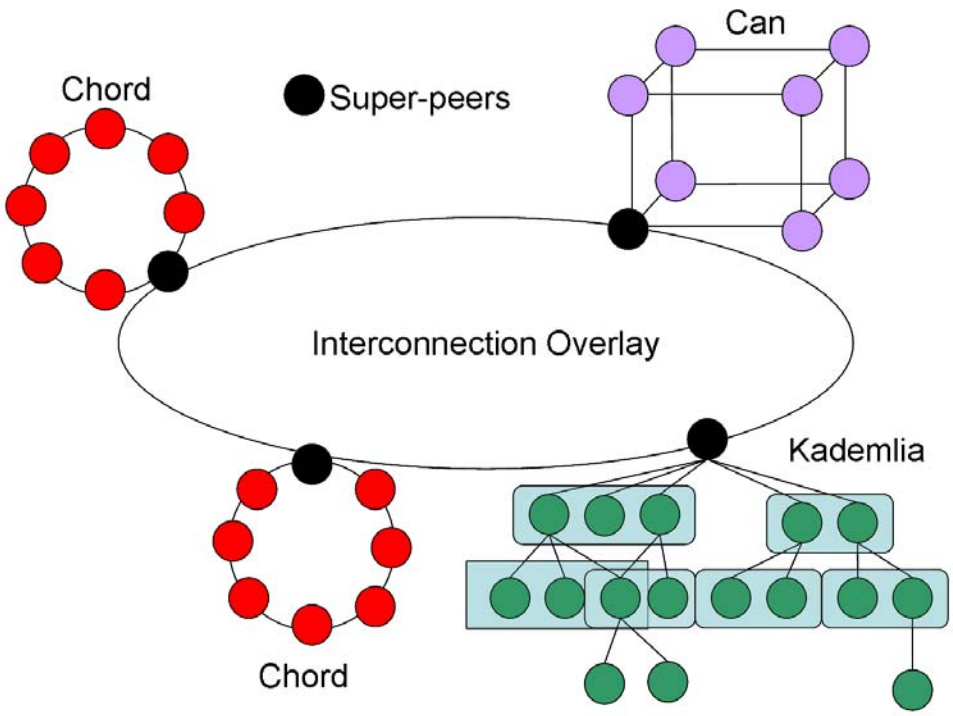


**Figura 3:** H-P2PSIP permite usar diferentes protocolos [3]

Aunque pueden estar organizadas en varios niveles, en nuestro estudio nos vamos a basar únicamente en redes de dos niveles, de modo que tendremos un nivel superior formado por nodos a los que llamaremos super-peers y un nivel inferior formado por nodos normales. A cada uno de los grupos de nivel inferior lo llamaremos dominio o cluster, y cada dominio estará conectado con el resto de dominios a través de de la red de interconexión.

Hay diferentes propuestas de redes jerárquicas con distintos objetivos algunas de las cuales podemos encontrar en [MS03a], [MS03b], [XMH03], [GGGM04] o [ALAS05]. Nosotros hemos decidido basar nuestra implementación H-P2PSIP cuyo diseño podemos ver parcialmente en [MYBG+08a] y [MYBC+08] ya que esta implementación permite tener cualquier red estructurada en cada uno de los dominios, cosa que las otras no permiten.

Para la arquitectura de dos niveles propuesta necesitaremos usar identificadores únicos tanto para los peers, como para los dominios. Estos

[3] Figura obtenida de [MYBC+09]

identificadores deben ser usados para realizar el enrutamiento de los paquetes en cada red. De modo que se use el identificador de nodo en el nivel inferior y el identificador de dominio en la red de interconexión.

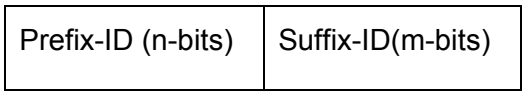


**Figura 4:** ID jerárquico

Cada dominio debe tener uno o más super-peers. Podríamos entender el caso en el que todos los nodos son super-peers como una red plana, sin embargo, en este caso no tendría sentido el poseer dos identificadores diferentes.

Los super-peers de cada dominio actúan como gateways entre grupos. Esta función hace que los super-peer tengan una mayor carga computacional y de red que los nodos normales. Además, en principio, es necesario que cada uno de los nodos conozca la IP de al menos uno de los super-peers encargados de su dominio para poder comunicarse con los peer de otro dominio, aunque esta condición puede sortearse con una elección adecuada del protocolo de enrutamiento. De este modo, cuando un peer quiera realizar una operación de look-up sobre un valor que no se encuentre en su dominio, podrá enviar la query a uno de los super-peers conocidos, y en caso de que el valor se encuentre en su dominio realizar la petición como si de una red plana se tratara. Existe la posibilidad de que el valor buscado sea de otro dominio pero el nodo no conozca la dirección de ningún super-peer. En este caso, el nodo inicial puede enviar la query a otro nodo vecino, el cual, en caso de conocer la dirección del super-peer, procederá a reenviarle la query, y en caso contrario actuará como en el caso anterior.

### *2.2.1 Servicio de look-up jerárquico*

En nuestro sistema de dos capas podemos definir de una manera sencilla la forma en la que se realizará una operación de look-up. Lógicamente, separaremos la búsqueda en dos tipos: las búsquedas de un valor de nuestro propio dominio (Si somos del dominio “A” buscamos a “x@A”) y las de un valor de otro domino (Siendo del dominio “A” buscamos a “x@B”).

En el caso de que se realice una búsqueda sobre un valor de nuestro propio dominio, la query se enrutará de acuerdo al protocolo P2P usado, en este caso realizaremos una búsqueda exactamente igual que si de una red plana se tratase.

Si el valor a buscar no forma parte de nuestro dominio (x@B) debemos entender el caso como si tuviéramos tres redes planas diferenciadas. En primer lugar, cuando un nodo normal detecte que el valor buscado no forma parte de nuestro dominio, este nodo debe enviar el mensaje al super-peer responsable del dominio. Una vez llegado el mensaje al super-peer este iniciará la búsqueda del dominio solicitado en la red de interconexión (Busca el recurso publicado por el responsable del dominio B), para una vez encontrado el mismo pedir a su super-peer responsable que realice la búsqueda en su dominio (En este caso ya se pide directamente la búsqueda de x@B). El super-peer responsable del segundo dominio iniciará ahora una búsqueda normal del valor pedido. Por último, una vez encontrado el valor, se debe recorrer el camino inverso para informar de su localización al nodo que inicio el proceso.

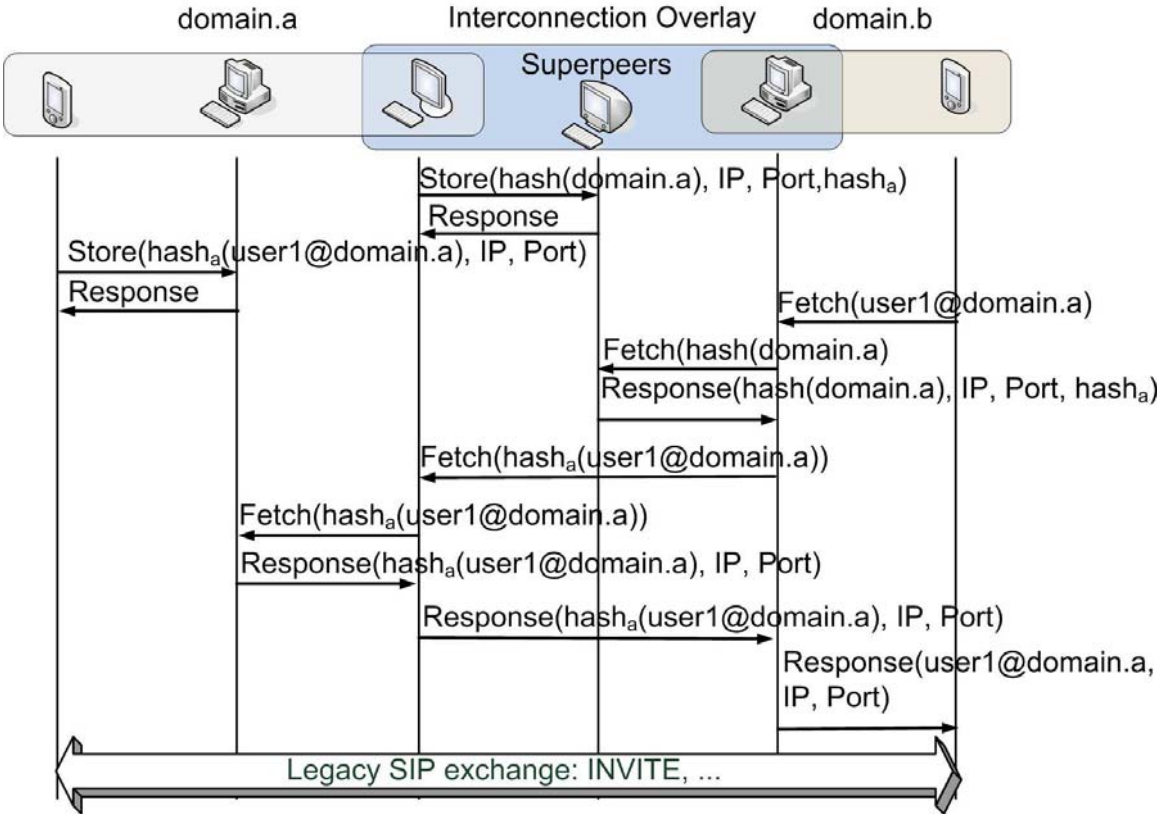


**Figura 5:** Señalización H-P2PSIP en entornos móviles [4]

## *2.3 P2PSIP*

El proyecto P2PSIP Working Group (P2PSIP WG) [P2PS09] del Internet Engineering Task Force (IETF) tiene el objetivo de desarrollar una versión peer-to-peer del protocolo SIP llamada P2PSIP en el cual podamos usar una red peer-to-peer estructurada para hallar recursos y servicios de forma descentralizada.

P2PSIP WG intenta desarrollar protocolos y mecanismos para el uso de SIP en entornos donde el servicio de establecimiento y mantenimiento de sesiones es llevado a cabo por una colección de puntos inteligentes, en lugar de por un servidor SIP centralizado.

[4] Figura obtenida de [MYBC+09]

No obstante, el alcance de P2PSIP no se limita a la obtención de una versión distribuida de los Proxy y Registrar de SIP, sino que puede ser también usado para otros fines o en combinación con otros protocolos de señalización.

El protocolo de P2PSIP está diseñado para soportar cualquier tipo de red basada en DHTs. Cada red overlay desarrollada, está identificada por un nombre de overlay y los participantes en esta arquitectura pueden soportar dos perfiles: peers o clientes. Los peers son nodos activos de la red overlay que poseen un identificador único llamado Node ID. Por otro lado, los clientes son entidades que usan los recursos ofrecidos por la red peer-to-peer pero no participan en el mantenimiento de la red. Este rol está reservado para dispositivos con capacidad muy limitadas como pueden ser teléfonos móviles.

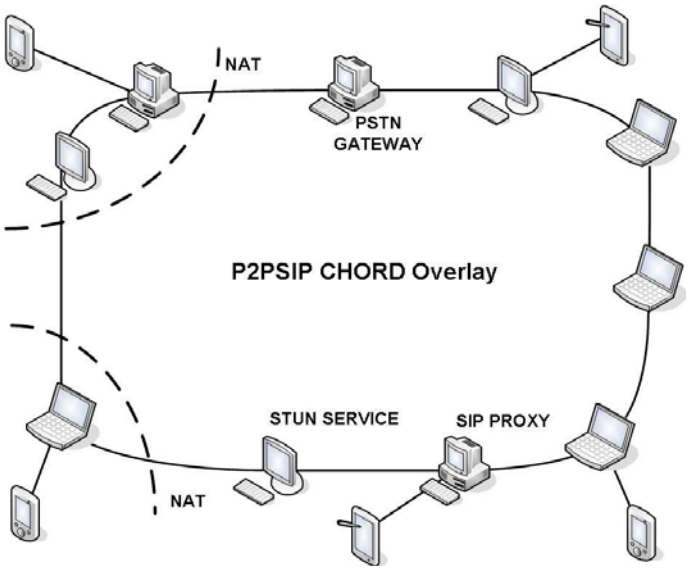


**Figura 6:** Modelo de referencia de P2PSIP [5]

La información almacenada en la red peer-to-peer está formada por registros de recursos que están asociados a recursos disponibles en la red. Estos recursos están identificados por un Resource ID y pueden almacenar servicios proveídos por un peer que a su vez se identifica por un Node ID. Como estos peers y servicios normalmente están identificados por un nombre en el formato URI (Uniform Resource Identifier), necesitamos definir un mecanismo que mapee los URI de usuario y servicio a un ID. Sin embargo, los detalles de

[5] Figura obtenida de [MYBC+09]

este mapeo dependen de cada implementación y son independientes de la funcionalidad ofrecida por el protocolo P2PSIP. Además, el protocolo debe soportar las primitivas básicas de una red peer-to-peer como son un servicio de bootstrapping, la unión a la red, la localización de recursos, o el mantenimiento de la red overlay y de la conexión entre peers y clientes (incluso cuando se encuentren tras un NAT). Obviamente, todos estos requerimientos incrementan la complejidad de la solución.

### *2.3.1 P2PP*

Peer-to-Peer Protocol (P2PP) es un protocolo de nivel de aplicación que puede ser usado para formar y mantener una overlay entre nodos participantes. Provee mecanismos para que los nodos puedan unirse, abandonar, publicar y buscar recursos en la overlay.

La overlay puede ser creada usando protocolos de P2P estructurados o no estructurados como pueden ser Bamboo, Chord, Pastry, Kademlia, Gnutella y Gia. P2PP usa transporte seguro, tiene un API para aplicaciones, mecanismos para atravesar NATs y firewalls, intercambio de capacidades de los nodos y de información de diagnostico. Al igual que P2PSIP, P2PP está diseñado para soportar una red SIP P2P, pero puede ser usado para otras aplicaciones.

P2PP fue uno de los primeros candidatos en el P2PSIP WG como protocolo de P2PSIP pero finalmente se consensuó que se unieran las diferentes propuestas en una sola para tener lo mejor de cada una, dando lugar al protocolo RELOAD.

### *2.3.2 RELOAD*

REsource LOcation And Discovery (RELOAD) [P2PS08] es un protocolo de señalización peer-to-peer usado sobre Internet cuyo precursor es P2PP. RELOAD provee un servicio de DHT, que permite a los nodos participantes leer y escribir entradas en una tabla hash que está almacenada de forma colectiva entre los participantes.

El protocolo es binario y ligero, y provee varias de las funcionas que son críticas para un protocolo P2P funcional. Algunas de estas funciones son:

**Security Framework**. Para un protocolo de P2P la seguridad es uno de los problemas más difíciles de resolver. En RELOAD se definen un marco de seguridad que permite la autorización de las funciones del protocolo P2P y la autentificación de los datos almacenados en la red. Este marco no elimina todos los ataques pero reduce en gran medida el espacio de posibles ataques.

**Usage Model**. RELOAD está diseñado para soportar varias aplicaciones entre las que se incluyen las comunicaciones P2P multimedia con SIP. Cada aplicación tiene sus propios tipos de datos y necesita almacenarlos y recuperarlos del DHT. Cada una define una estructura de datos, autorización, políticas, tamaño e información en el DHT. RELOAD puede ser usado por nuevas aplicaciones a través de un proceso simple de implantación.

**Nat traversal**. Las operaciones para atravesar NATs son parte del diseño base, incluyendo el establecimiento de nuevas conexiones RELOAD, la implantación de túneles SIP o los requerimientos de cualquier otra aplicación. RELOAD hace uso de ICE para crear la red P2P y el establecimiento de los canales usados por las aplicaciones. También se define como los peers pueden actuar como servidores STUN o TURN y como estos peers pueden ser encontrados. De este modo RELOAD puede ser usado en situaciones en las que casi todas las peers se encuentran detrás de NATs.

**High Performance Routing**. La naturaleza de los algoritmos DHT introduce el requisito de que los peers participantes en la red P2P realicen el enrutamiento en nombre de otro peers. Esto introduce una carga en estos otros peers tanto en ancho de banda como capacidad de proceso. RELOAD intenta reducir esta carga utilizando un protocolo binario muy ligero y definiendo una estructura de paquetes que reduce la complejidad.

**Transport Flexibility**. RELOAD permite el uso de DTLS y TLS como protocolo de transporte. Es obligatoria la implementación de DTLS sobre UDP aunque se prefiere TLS sobre TCP ya que suele mejorar el rendimiento y la estabilidad de las conexiones.

**Pluggable DHT Algorithms**. RELOAD ha sido diseñado con un interfaz que permite implementar una gran variedad de algoritmos, tanto estructurados como no estructurados sobre él. Es obligatoria la implementación de Chord.

## *2.4 Implementaciones relacionadas con P2PSIP disponibles*

A continuación vamos a exponer las diferentes implementaciones de protocolos similares a P2PSIP que hemos barajado como base para nuestra implementación ya que aun no existe ninguna implementación de RELOAD disponible.

**P2PNS** [P2PN09]. Peer-to-Peer Name Service es un sistema de nombres distribuido sobre una red peer to peer. El proyecto está siendo desarrollado por el grupo del profesor Zitterbart de la universidad de Karlsruhe. En la actualidad el proyecto está centrado en proveer un sistema seguro y eficiente de resolución de nombres SIP para una red VoIP descentralizada.

Esta implementación desarrollada en C está compuesta por más de 2000 archivos. Está dividida en 2 partes, un proxy SIP modificado y un marco que forma la overlay. Estas dos partes están conectadas entre sí mediante un interfaz XML-RPC.

**ViaSIP_NG** [VIAP09]. El proyecto ViaSIP_NG usará el último draft del IETF para desarrollar una versión de P2PSIP de código abierto.

El proyecto iniciado en 2005 no ha encontrado apoyo en ninguna institución o universidad. Desde 2007 dejó de estar alojada en sourceforge y el proyecto fue adoptado por AgileCO.

**Sip-Communicator** [SIPC09]. Este proyecto tiene como objetivo implementar una arquitectura SIP tradicional usando tecnología P2P. Puede encontrar otros usos de los DHT, y obtener rutas a través de él.

**Olyo** [OLYO09]. Olyo es un proyecto de código abierto promovido por MINE lab, BUPT. Su objetivo es implementar en C un prototipo de P2PSIP y conseguir transmitir contenidos multimedia en tiempo real sobre este prototipo.

**P2PP** [P2PP09]. Peer to Peer Protocol es un proyecto desarrollado por la universidad de Columbia. Actualmente hay completamente desarrolladas dos versiones del prototipo, uno con soporte para NAT transversal y otro sin él.

El programa está implementado en C++, está compuesto por más de 300 archivos y tiene una implementación orientada a objetos que permite fácilmente la introducción de nuevos protocolos.

Este proyecto está muy bien documentado y ha sido probado en el entorno PlanetLab con aproximadamente 1000 nodos.

# CAPÍTULO III

# DISEÑO DE LA APLICACIÓN

# 3. DISEÑO E IMPLEMENTACIÓN DE LA APLICACIÓN

## *3.1 Requisitos de una red overlay jerárquica*

Una red overlay jerárquica tiene una serie de particularidades que la hacen diferente de una red overlay plana. Estas peculiaridades se pueden traducir en la siguiente serie de requisitos:

**Separación de los peers en grupos.** Para el diseño de nuestra overlay jerárquica necesitamos desarrollar una red en la que podamos separar los nodos en diferentes grupos o clusters. Además, estos grupos deben estar conectados entre sí por una red de superpeers.

**Independencia entre grupos.** Los diferentes grupos de la red deben ser independientes entre sí, pudiendo incluso tener diferentes protocolos de enrutamiento en cada uno de ellos. Dado que las simulaciones se realizaron con el protocolo de P2P estructurado Kademlia, nuestro objetivo es realizar una red con este mismo protocolo que funcione de forma jerárquica, aunque se podrían diseñar otros protocolos que interactuaran entre sí.

**Uso de identificadores jerárquicos.** Debemos usar identificadores que nos permitan diferenciar la información de la red, de la del usuario. Decidimos que la primera parte del identificador sea el hash del dominio y la segunda sea el

hash asociado a la información guardada. En una aplicación de VoIP sería la información del usuario.

**Peers y superpeers participan en el enrutamiento.** De este modo, para cada peer y query, si la primera parte del hash de la query es la del propio peer, el mensaje se enrutará dentro de nuestro propio cluster. En cambio, si este hash es diferente, el peer enrutará el mensaje hacía el superpeer encargado de su dominio, y este último será el encargado de buscar al encargado del dominio en el que debemos buscar y este encargado será finalmente el que deba buscar en su dominio al usuario buscado.

## *3.2 Elección de la implementación de partida*

Tras analizar detalladamente las implementaciones disponibles finalmente nos decidimos por usar P2PP como base para nuestro experimento.

| | P2PNS | ViaSIP_NG | Sip-Communicator | Olyo | P2PP |
|---|---|---|---|---|---|
| **Lenguaje de programación** | C | - | Java | C | C++ |
| **Código libre** | Si | Si | Si | Si | Si |
| **Última actualización** | 2008 | 2007 | 2009 | 2007 | 2009 |
| **Continúa** | - | No | Si | No | Si |
| **Complejidad** | Alta | - | Muy Alta | Media | Alta |
| **Adaptación a nuestras necesidades** | Baja | - | Media | Baja | Muy Alta |

**Figura 7:** Comparación de las implementaciones disponibles

Como podemos ver, P2PP es la aplicación que mejor se adapta a nuestros requisitos ya que nos proporciona exclusivamente una red P2P que nos servirá de punto de partida. La otra alternativa a considerar es Sip-Communicator, pero la desechamos por su elevada complejidad, al proporcionar muchísimas funcionalidades que no son necesarias para nuestro experimento.

Pese a que la implementación elegida es anterior a la estandarización de P2PSIP por parte del IETF, tiene desarrollados suficientes recursos para implementar una overlay en la que probar las ventajas de las redes jerárquicas.

P2PP lanzó su primera versión en marzo de 2008. Desde entonces llevan tiempo probándolo y han desarrollado una gran variedad de protocolos P2P. En la actualidad tienen una versión capaz de atravesar NATs y sobre la que se puede instalar OpenWengo (un teléfono VoIP de código libre).

Otra razón para la elección de esta implementación es que está desarrollada en C++ y orientada a objetos. La orientación a objetos nos ayudará a realizar las modificaciones necesarias para llevar a cabo nuestro experimento. De este modo, añadiendo nuevas clases sobre las ya existentes podemos implementar otro tipo de overlay sin necesidad de modificar la base del programa, aprovechando de esta manera casi toda la estructura del programa y de este modo solo necesitamos añadir un nuevo protocolo P2P al programa.

## *3.3 Estructura de la implementación de partida*

En la actualidad hay 2 versiones diferentes de P2PP. La versión 0.2 del programa se diferencia de la 0.1 en que dispone de un sistema capaz de atravesar NATs, para ello usa la librería PJNATH, sin embargo en la web del programa indican que si no se va a necesitar esta utilidad es mejor usar la versión 0.1. Debido a esto finalmente usamos como punto de partida la implementación 0.1 de P2PP que fue liberada el 3 de marzo de 2008.

Nada más ver la implementación descubrimos que genera 2 ejecutables diferentes, uno llamado p2pmain que es el programa principal encargado de formar los nodos de la overlay. El segundo programa, llamado checker, es usado para realizar comprobaciones en la red. Este ejecutable será el que usaremos para realizar las pruebas de la red, ya que nos permite empezar cualquier query desde cualquier peer de la red.

Además de estos dos ejecutables vemos que el código está distribuido en distintas carpetas según su funcionalidad.

***Bigint***. Esta carpeta contiene la clase BigInt que implementa funciones adicionales sobre números enteros grandes, lo que nos ayudará en las funciones de routing al dejarnos comparar números de forma binaria.

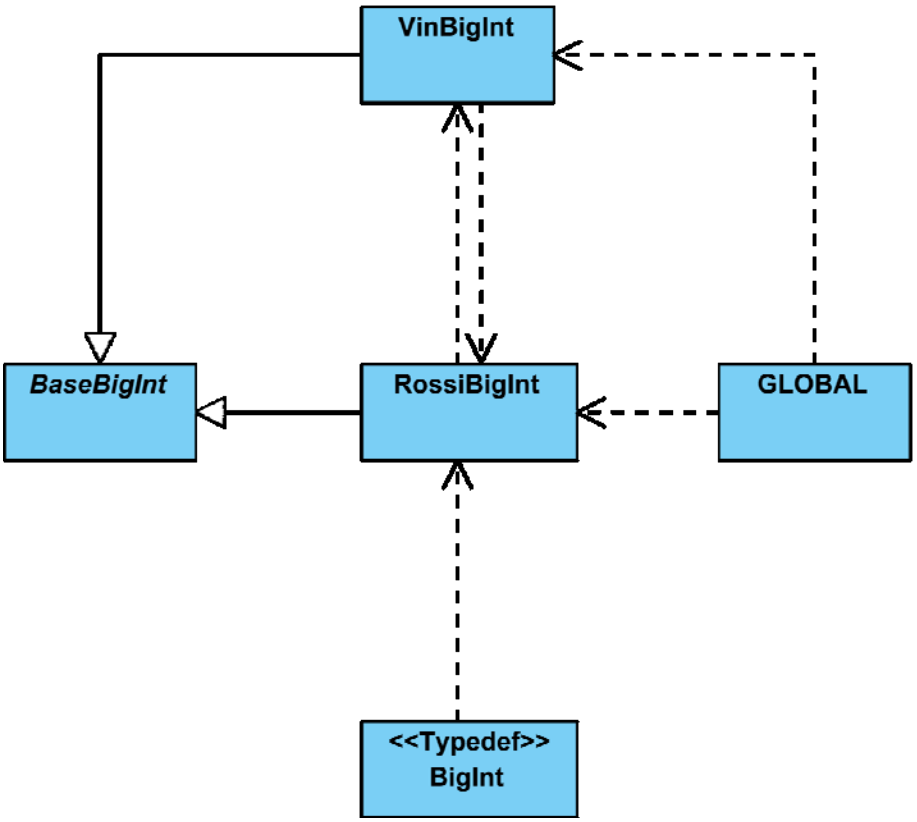


**Figura 8:** Diseño de clases: Bigint

***Common***. Contiene dos archivos de cabecera, uno define variables como la longitud de las direcciones IP usadas. El otro es usado para la recolección de trazas que nos ayudarán a depurar el código.

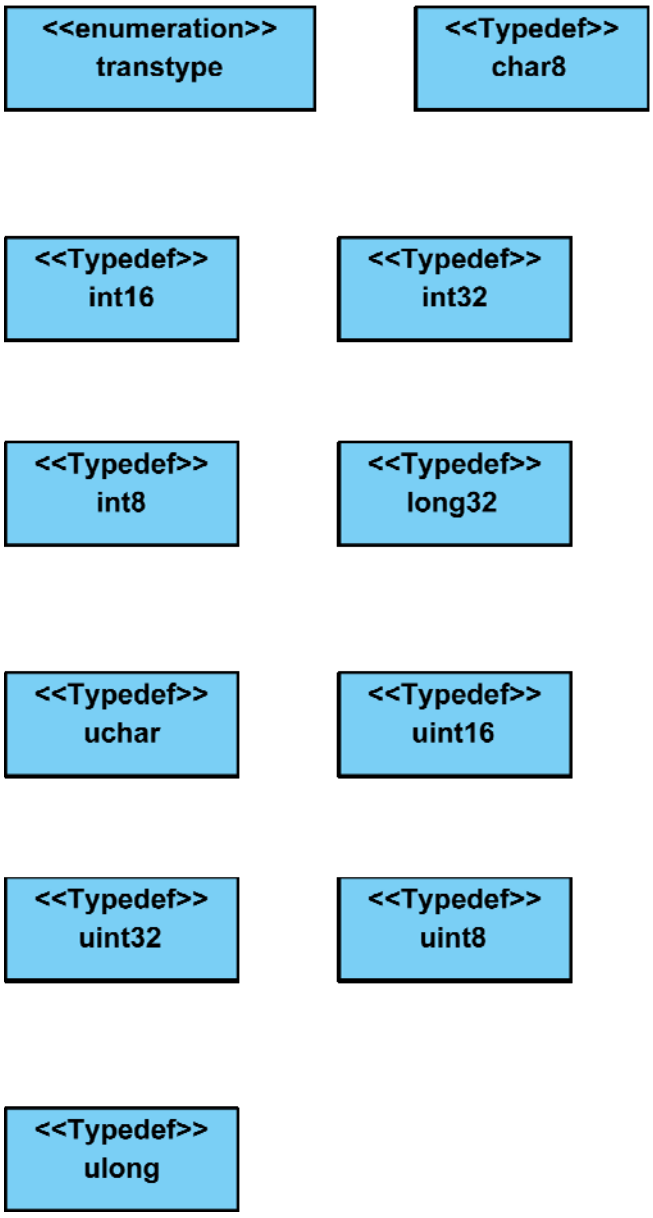


**Figura 9:** Diseño de clases: Common

***Distance***. Las clases de este apartado son las encargadas de calcular la distancia para los distintos protocolos P2P. Así para los distintos peer conocidos usaremos como siguiente salto el que menor distancia al destino tiene para cada protocolo.

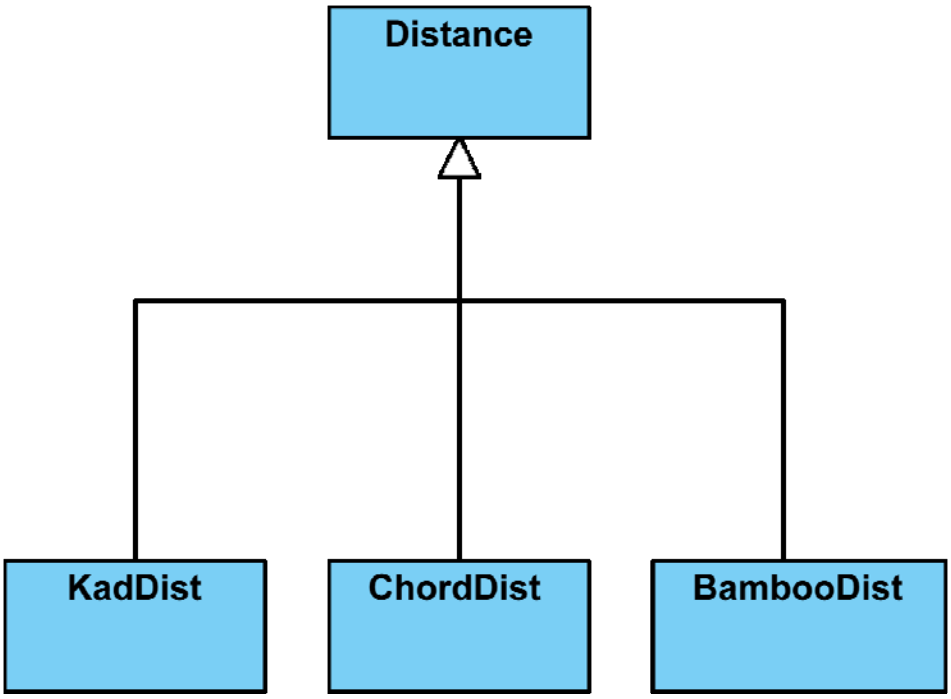


**Figura 10:** Diseño de clases: Distance

***Utils***. En esta carpeta se encuentran las funciones para la realización del hash necesario sobre los distintos identificadores.

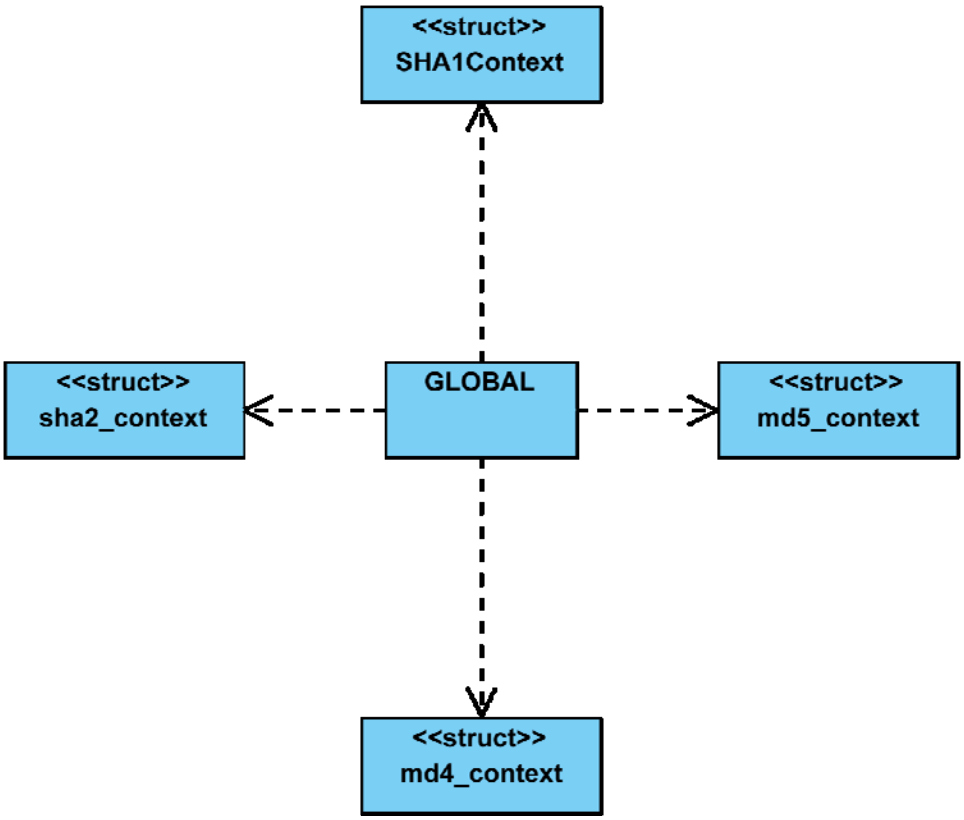


**Figura 11:** Diseño de clases: Utils

***Msg***. Contiene el archivo msgheader.cpp y msgheader.h en los que están definidos todos los mensajes definidos en el draft de P2PP. También proporciona funciones para pasar de los mensajes binarios a variables y viceversa.

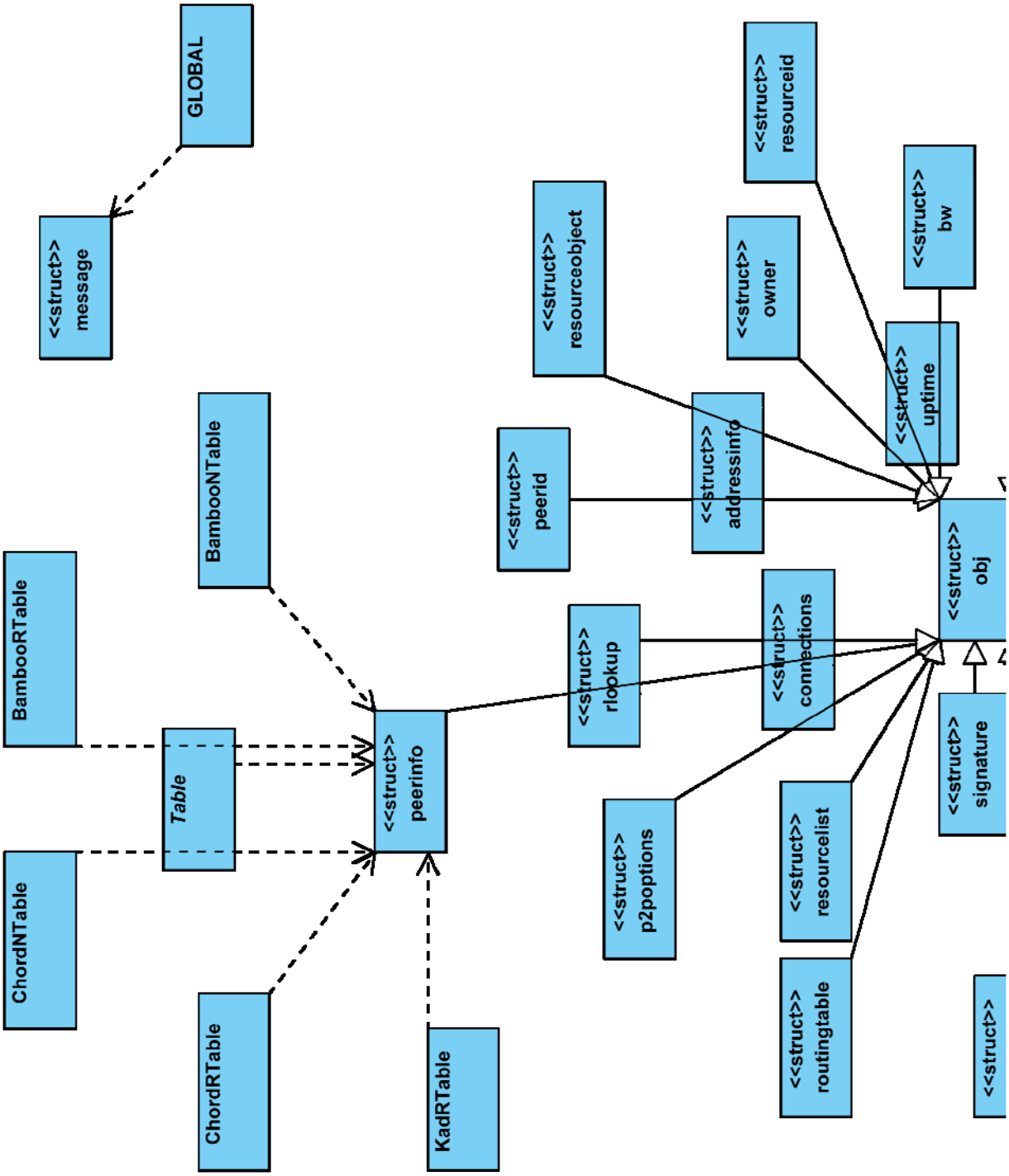


**Figura 12:** Diseño de clases: Msg (1)

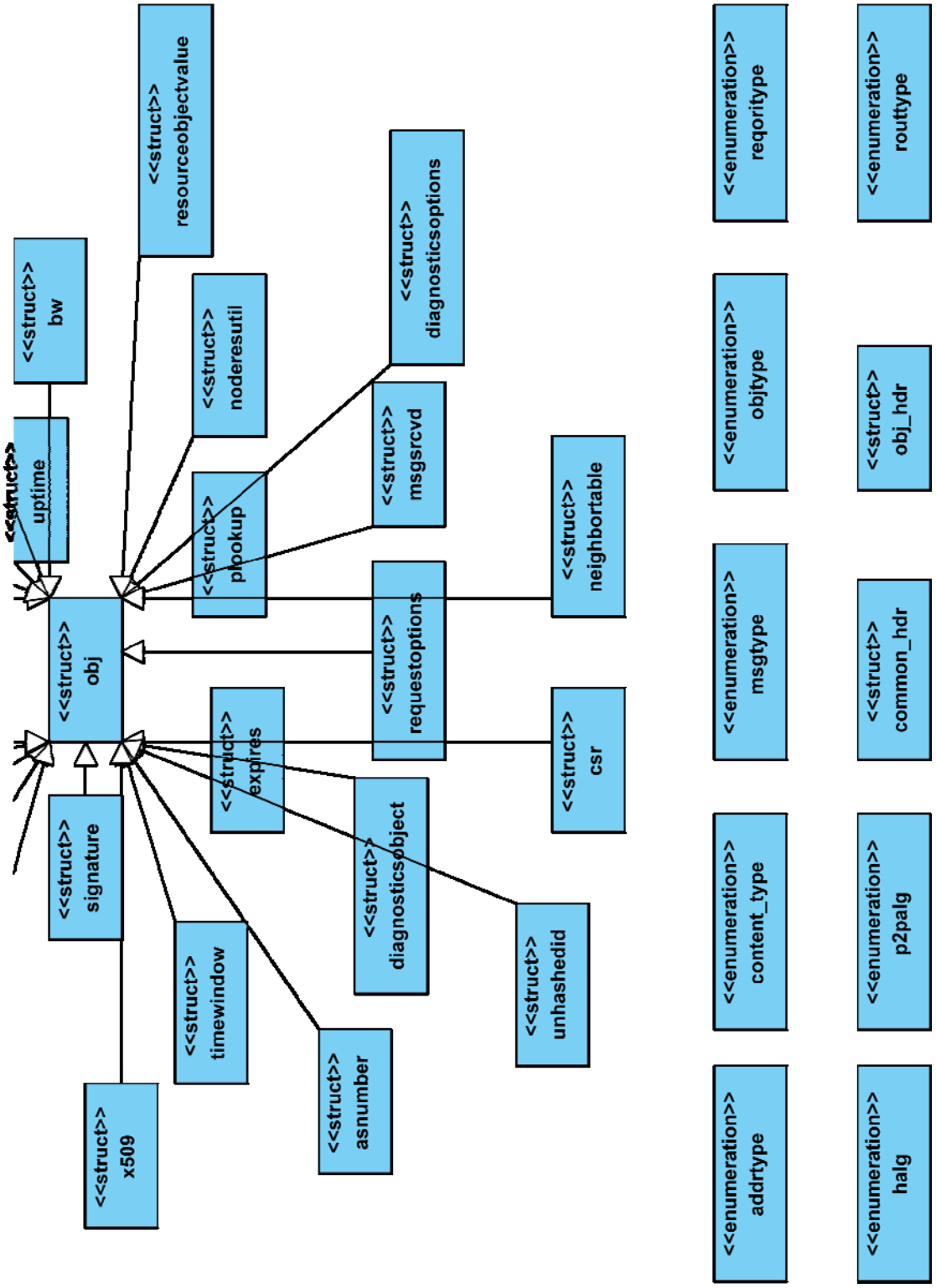


**Figura 13:** Diseño de clases: Msg (2)

***Net***. En esta carpeta podemos encontrar las clases encargadas de la comunicación de red. Contiene una clase encargada de manejar los sockets y otra encargada de la gestión de la capa de transporte.

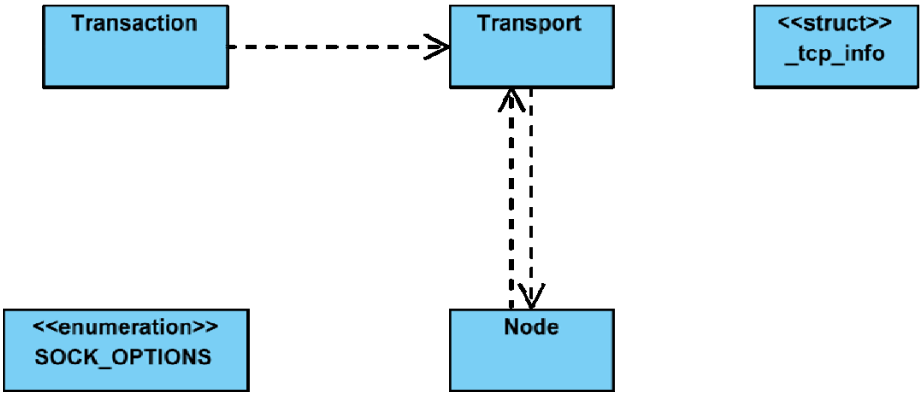


**Figura 14:** Diseño de clases: Net

***p2pconfig***. La clase P2PConfig es la encargada de leer el archivo de configuración y crear las variables necesarias para que al iniciar el programa cada peer tenga la configuración adecuada.

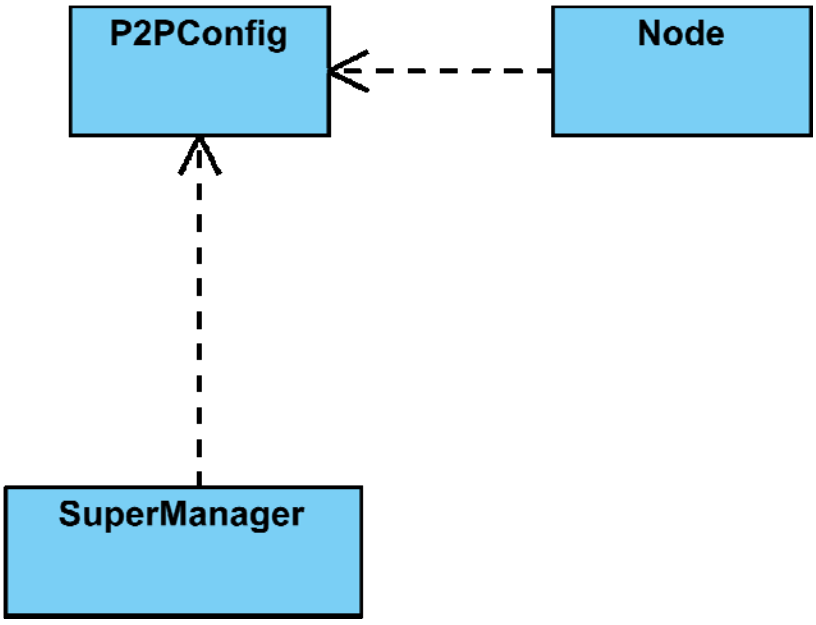


**Figura 15:** Diseño de clases: p2pconfig

***Sys***. Contiene funciones de ayuda para el programa como son la gestión de threads, comparación de cadenas, o la comparación de tiempos.

***Table***. Aquí están implementadas las distintas tablas, tanto de peers conocidos como de recursos para cada uno de los protocolos P2P.

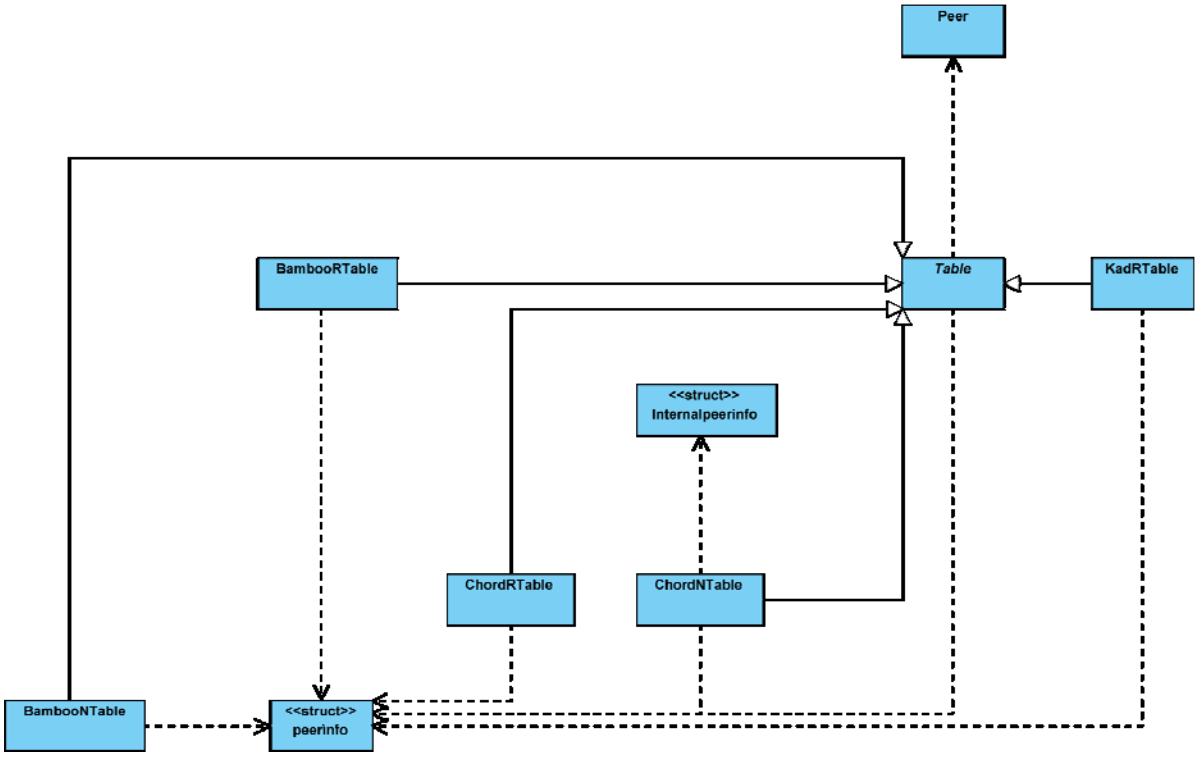


**Figura 16:** Diseño de clases: Table

***Node***. Aquí encontramos el centro del programa. En esta carpeta están definidas las clases encargadas de actuar como un peer para cada protocolo P2PP, así como de las funciones de bootstrap y checker. También se encuentran aquí las clases encargadas del mantenimiento de la red, la gestión de los recursos y del control de las distintas transacciones entre peers.

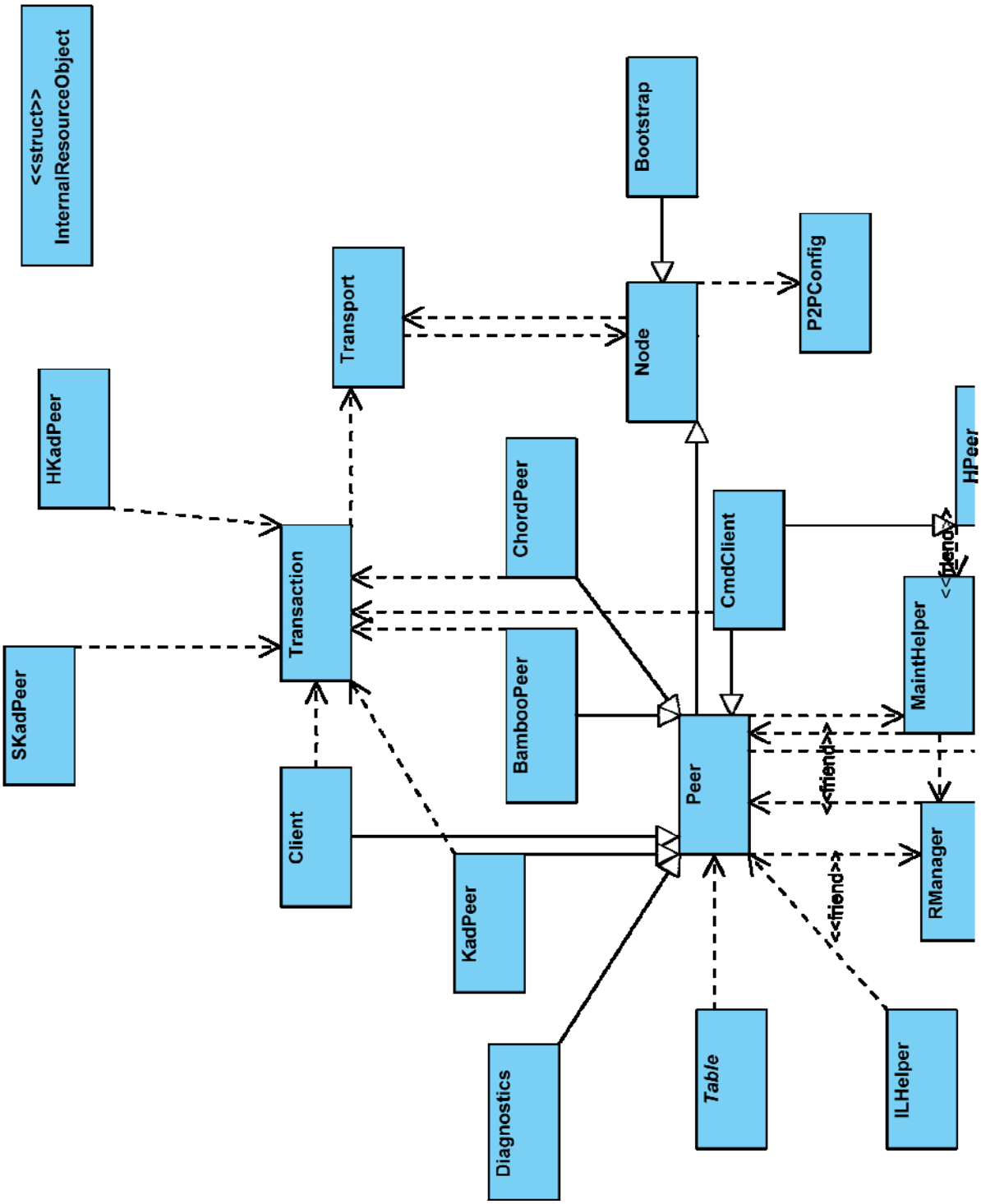


**Figura 17:** Diseño de clases: Node (1)

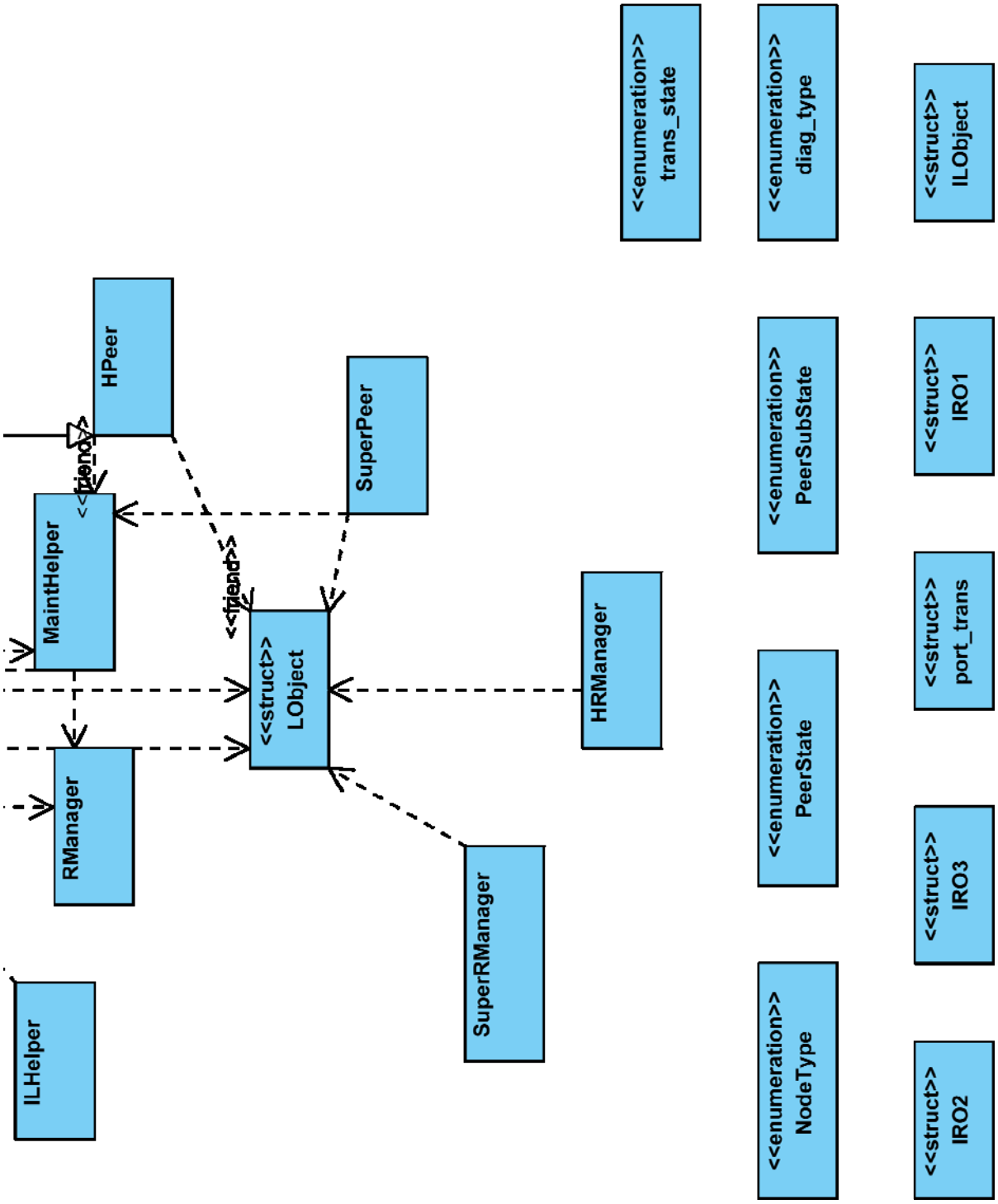


**Figura 18:** Diseño de clases: Node (2)

## 3.4 Modificaciones para el cumplimiento de los requisitos de partida

### 3.4.1 El protocolo

Para el cumplimiento de los requisitos de partida hemos tenido que modificar algunas de las clases y crear otras. En la estructura original del programa, las clases que implementaban el protocolo P2P tenían una estructura lógica en la que la clase con el protocolo en si heredaba de la clase Peer la cual a su vez heredaba de la clase Node. Para la realización de nuestro esquema intentaremos modificar esta estructura lo menos posible.

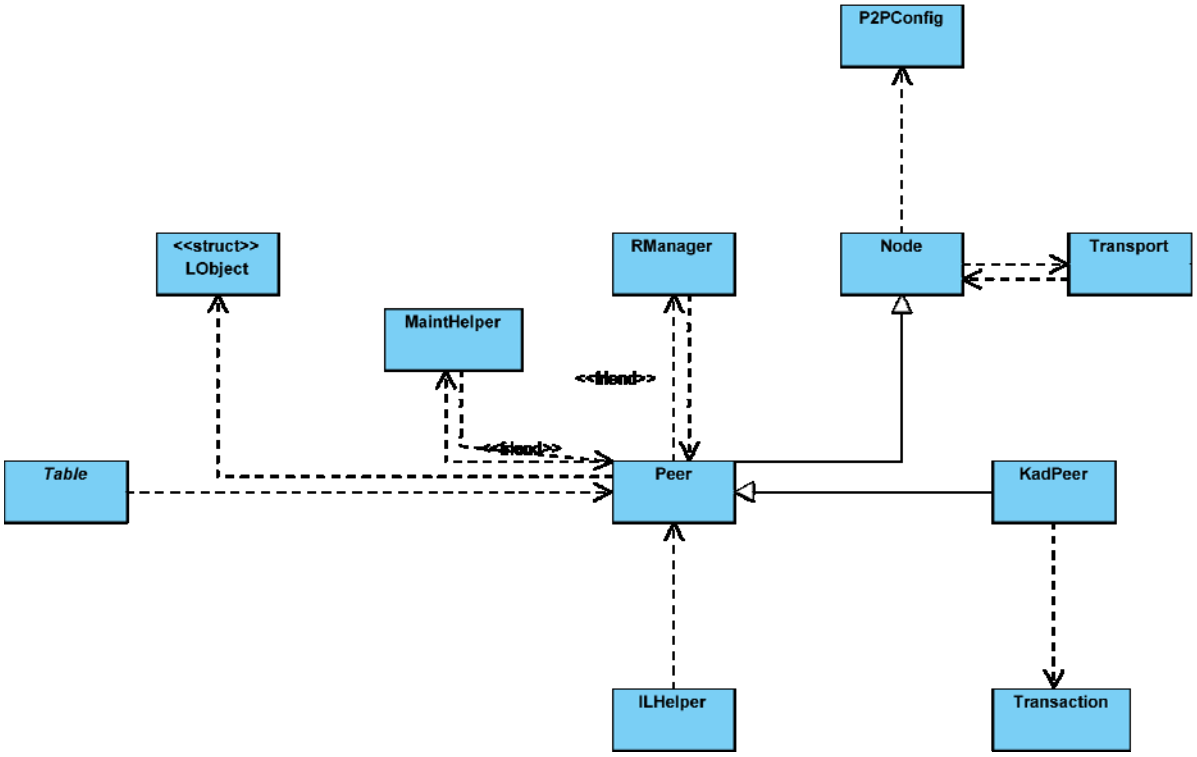


**Figura 19:** Esquema de clases: Esquema original

En primer lugar introducimos un paso lógico intermedio entre la clase Peer generalista y la clase que implementa el comportamiento propio de cada protocolo. Para ello creamos la clase HPeer. Esta clase añade 2 funcionalidades básicas a la clase Peer.

La primera de estas funcionalidades es la creación de ID jerárquicos. Como explicamos anteriormente, necesitamos que el routing dentro de nuestro cluster se haga en función del suffix ID y que en la red de superpeers se haga mediante el prefix ID.

Con el objetivo de no modificar el routing original, vamos a realizar el routing sobre el ID completo, pero en todos los Peer y recursos del mismo cluster tendremos el mismo prefijo. De ese modo el prefix ID será siempre el hash del dominio, mientras que el suffix ID será para los Peer el hash de su IP y para los recursos el hash del nombre de usuario completo.

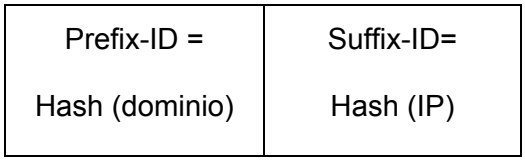


**Figura 20:** Identificador jerárquico de los peers

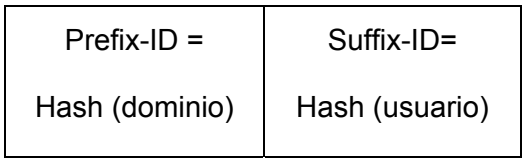


**Figura 21:** Identificador jerárquico de los recursos

En la red de superpeers podemos usar identificadores normales, ya que en este caso el dominio será el recurso a buscar. Por ello usaremos como ID de los superpeer el hash de su IP y como ID de los recursos el hash del dominio.

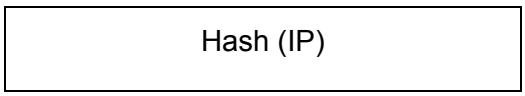


**Figura 22:** Identificador de los superpeers

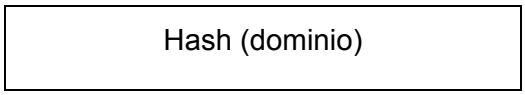


**Figura 23:** Identificador de los recursos en la red de interconexión

La segunda funcionalidad añadida en la clase HPeer es la de detectar cuando una petición es para nuestro dominio y cuando la debemos reenviar a la red de superpeers. De este modo, hemos modificado el método HPeer::getNextHop, que en la implementación original no hacía nada, para que cuando el destino no esté en nuestro cluster añada la información del superpeer encargado de nuestro dominio. En este caso también se debe controlar cuando

el peer encargado del routing del mensaje es a la vez el superpeer, si esto sucede, debemos continuar la operación a través de la otra red, para lo cual devolveremos la información a un manejador de eventos (SuperManager) que explicaremos más adelante.

De esta clase HPeer hereda la clase HKadPeer que representa el funcionamiento de los nodos normales en Kademlia jerárquico. Esta clase se basa en la clase KadPeer original de la versión 0.1 de P2PP. Durante la implementación de nuestro prototipo intentamos modificarla lo menos posible, con el objetivo de que casi todas las operaciones compartidas entre los distintos protocolos jerárquicos se hicieran en la clase HPeer, no obstante hay que dotarla de funcionalidad suficiente para poder usar los métodos mejorados en esta última clase.

La novedad más importante en este caso se encuentra en la modificación del método HKadPeer::getNextHop, que era el encargado de buscar en la tabla de rutas cual es el siguiente salto. Ahora, si el método HPeer::getNextHop nos dice que somos el encargado del dominio, podremos enrutar la query hacía la red de superpeers. En el caso de que el nodo en cuestión no sea un superpeer, en este método, cogeremos la información sobre el responsable del dominio para dirigir a este la query, en lugar de usar la tabla normal de rutas.

Al igual que las dos clases anteriores encargadas del control de los peer normales se han añadido otras dos encargadas del control de la red de superpeers. Estas clases son SuperPeer, heredera de la clase Peer, y SKadPeer heredera, a su vez, de SuperPeer.

En la clase SuperPeer se ha añadido el método SuperPeer::LookupNormalUser que es el encargado de iniciar una búsqueda por un recurso de otro cluster. Este método es necesario ya que en el caso de que a un superpeer le llegue una consulta para un recurso de otro dominio, una vez encontrado al responsable de este dominio debe reenviar la consulta por el recurso directamente a este responsable, sin hacer caso al routing normal en su red.

En el caso de la clase SKadPeer, básicamente, sus métodos han sido adaptados para funcionar con clases del tipo SuperPeer en lugar de las del tipo Peer.

Además de las clases de manejo del protocolo se han creado las clases propias para manejar los recursos tanto para la red de peers normales como para la de superpeers. La clase RManager, original de P2PP, es la encargada de gestionar los mensajes que llegan a un peer. A partir de esta clase debemos generar otras que se adapten a nuestras necesidades.

Para la red de peers normales creamos la clase HRManager. En ella se ha modificado el método HRManager::LookupUser para que maneje los identificadores jerárquicos. También se ha añadido código en el método HRManager::OnLookupObjectRequest para que maneje el caso en el que nos están preguntando por un objeto perteneciente a otro dominio y por tanto debemos enviar esta información al SuperPeer responsable del dominio. Otra modificación necesaria ha sido la del método HRManager::PublishUserInfo para que cree correctamente el objeto a publicar con el ID jerárquico. Por último se ha añadido el método HRManager::setCallBack que nos permite añadir una función de “callback” en esta clase para devolver información a los Manager de nivel superior.

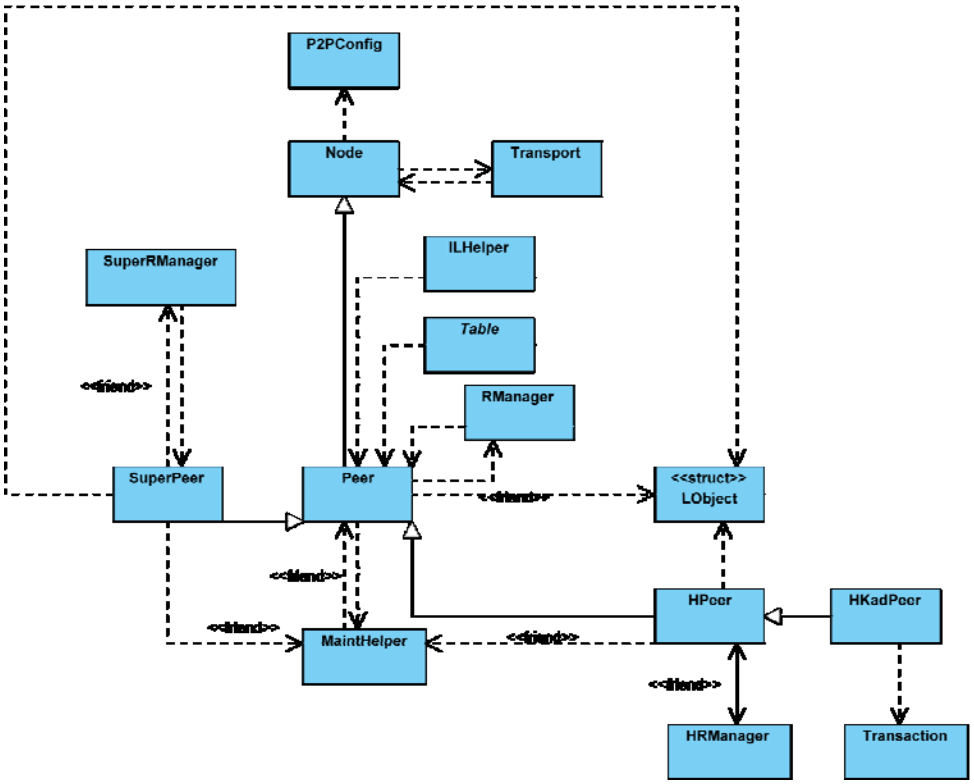


**Figura 24:** Esquema de clases: Esquema final

Para la red de superpeers hemos creado la clase SuperRManager. En ella hemos añadido el método SuperRManager::LookupNormalUser encargado de gestionar la búsqueda de un usuario normal desde la red de superpeers. Los métodos SuperRManager::OnLookupObjectRequest y SuperRManager::PublishUserInfo también han sido modificados para adaptarse a los identificadores jerárquicos al igual que en la clase HRManager.

Otra clase que se ha debido modificar para manejar los identificadores jerárquicos ha sido la clase Bootstrap, encargada de facilitar el primer contacto entre los diferentes peers. Esta clase es la que le indica a cada nodo el ID que debe usar, por ello hemos tenido que modificar el método Bootstrap::OnReceive para que si algún peer pide un identificador para Kademlia jerárquico le devuelva un ID con la forma adecuada.

Por otro lado la clase Bootstrap ha tenido que ser modificada para poder ser usada en nuestro entorno de trabajo. Ya que puede haber varios bootstrap ejecutándose en la misma máquina es importante que no todos guarden su log

en el mismo archivo, para lo cual se crean archivos en relación al dominio al que pertenecen.

Por último, para terminar con las modificaciones en el protocolo, se ha creado la clase SuperManager encargada del control de los superpeers.

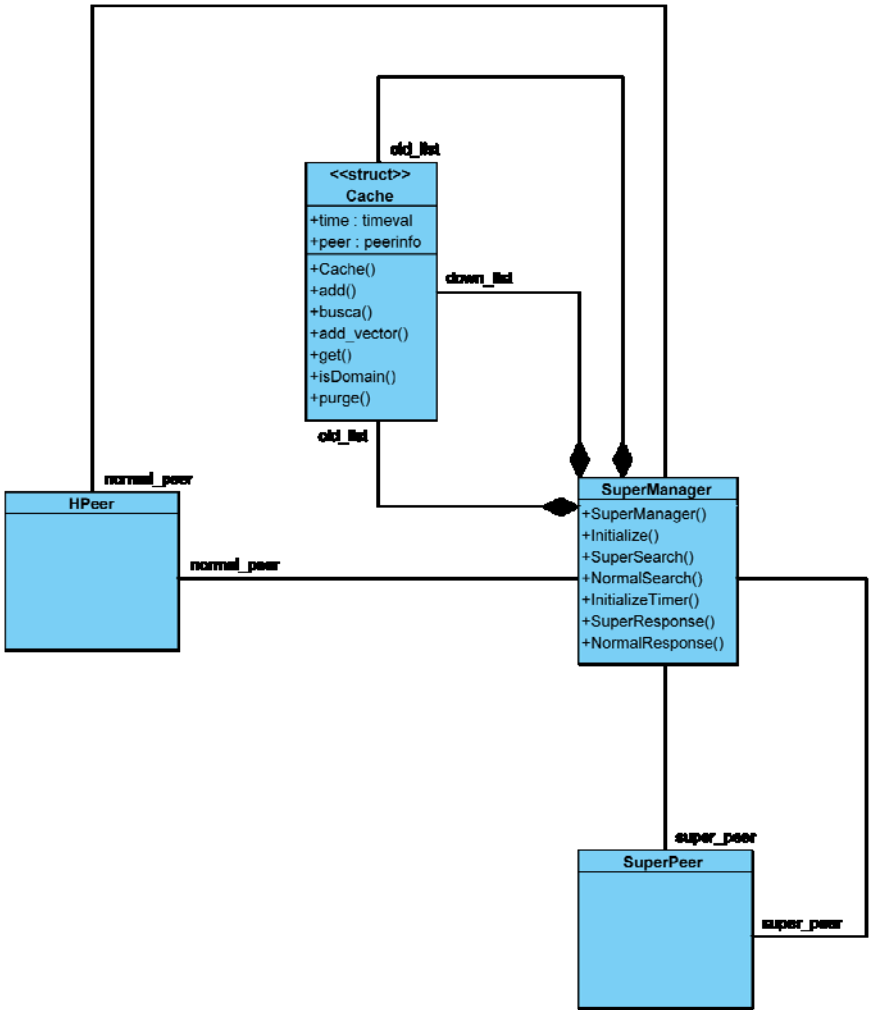


**Figura 25:** Esquema de clases relacionadas con SuperManager

En nuestra implementación todos los nodos que forman parte de la red de superpeers están corriendo a la vez dos instancias del protocolo Kademlia en diferentes hilos, una para relacionarse con la red de peers y otra con la red de superpeers. Estas dos instancias deben relacionarse para poder llevar a cabo su función y para tal fin hemos creado un manejador de eventos.

La clase SuperManager es la encargada de la inicialización de estas dos instancias. Para ello disponemos del constructor de la clase y del método

SuperManager::Initialize encargados de la creación e inicialización de los nodos encargados de ambas redes.

Esta clase se apoya en dos funciones de callback para realizar su función, una de ellas recibe la información de la red de peers y la otra de la red de superpeers. Estas funciones miran la información recibida y a partir de ellas deciden a que método deben llamar.

Existen cuatro métodos diferentes para gestionar estas búsquedas, SuperManager::SuperSearch, SuperManager::NormalSearch, SuperManager::SuperResponse y SuperManager::NormalResponse.

El método SuperSearch es llamado cuando a un superpeer le llega una query que no es para su propio dominio. Este método comprueba si el responsable del dominio a buscar es conocido, en cuyo caso manda la búsqueda al superpeer a este responsable tras añadir esta búsqueda a la lista de tareas pendientes. Si el dominio no es conocido por el superpeer añadiremos la búsqueda a la lista de tareas pendientes y se iniciará la búsqueda del responsable del dominio en la red de superpeers.

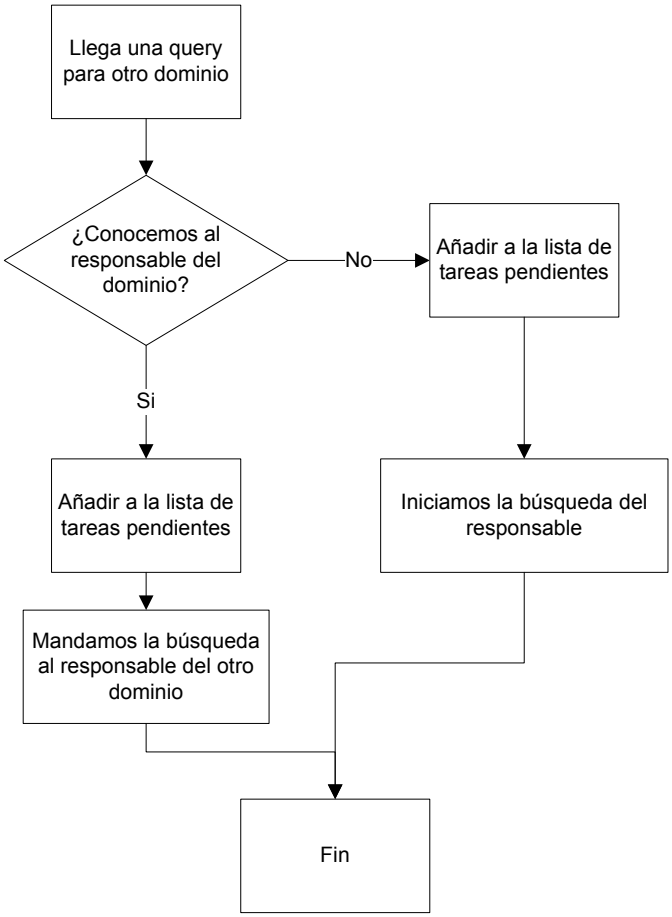


**Figura 26:** Organigrama SuperManager::SuperSearch

El método NormalSearch es llamado cuando un superpeer recibe una query preguntando por un peer en su dominio a través de la red de superpeers. En este caso solo debemos guardar la búsqueda en la lista de tareas pendientes y después iniciar la búsqueda a través de la red de peers normales.

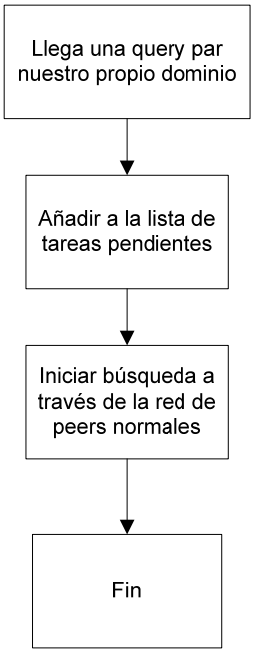


**Figura 27:** Organigrama SuperManager::NormalSearch

El método SuperResponse es llamado cuando recibimos una respuesta a una query a través de la red de superpeers. En este caso debemos comprobar si hemos recibido una respuesta satisfactoria, en cuyo caso si la respuesta es por el responsable de un dominio debemos mandarle la búsqueda original a este, mientras que si la respuesta tiene la información sobre un peer debemos reenviar esta información al peer que inicio la búsqueda. En el caso de recibir un código de error debemos reenviar este código al peer que inicio la búsqueda.

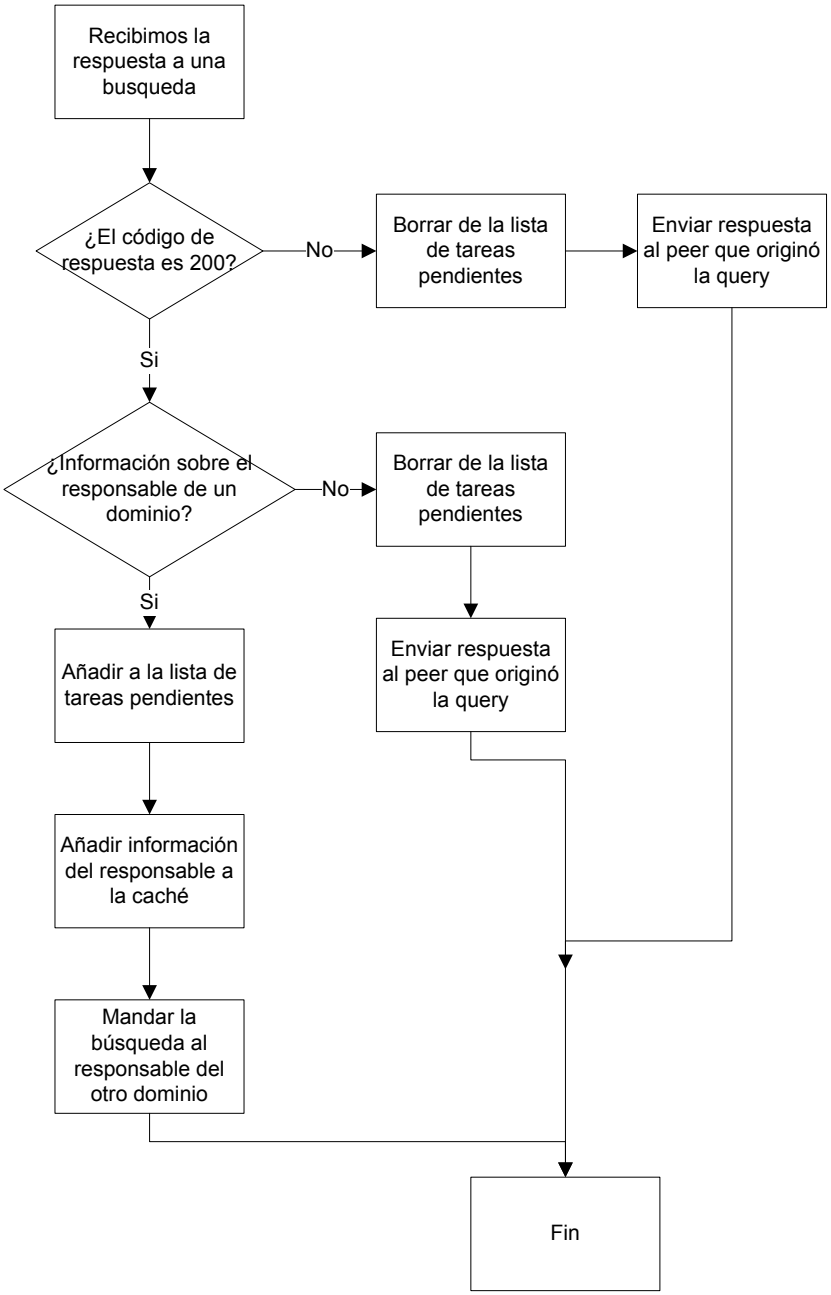


**Figura 28:** Organigrama SuperManager::SuperResponse

Por último, el método NormalResponse es llamado cuando recibimos una respuesta a través de la red de peers normales, en este caso lo único que debemos hacer es mandar la información al superpeer que había mandado anteriormente la búsqueda.

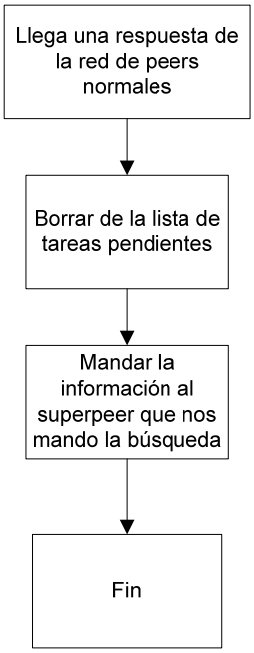


**Figura 29:** Organigrama SuperManager::NormalResponse

### *3.4.2 Trazas*

Para poder recuperar los resultados hemos tenido que realizar modificaciones la clase SuperManager y la clase HKadPeer, además de los archivos checker.cpp y p2pmain.cpp.

La clase Checker ha sido modificada para guardar en un archivo las respuestas recibidas en formato CSV. Para ello cuando recibimos una respuesta, analizamos la información y guardamos en un archivo datos como el código recibido, el número de saltos empleado o el tiempo tardados.

Las otras modificaciones realizadas se han llevado a cabo para poder guardar en un log, cada 30 segundos, el número de entradas en la tabla de rutas que tiene cada nodo.

Como en la clase SuperManager ya teníamos un hilo ejecutándose periódicamente para controlar las cachés, solo es necesario añadir la lógica para que dentro de este hilo se guarde cada 30 segundos la información de la tabla de rutas.

En el caso de los peers normales la cosa es más complicada, ya que no hay ningún evento periódico, por lo que creamos el método

HKadPeer::InitializeLog que será encargado de guardar la información de la tabla de rutas cada 30 segundos. Este método se ejecuta dentro de otro hilo, que será lanzado al iniciar la aplicación desde el archivo p2pmain.cpp tras lanzar el hilo en el que se ejecutarán las tareas relacionadas con el protocolo.

### *3.4.3 Otras modificaciones*

Además de las modificaciones propias del nuevo protocolo usado han sido necesarias otras modificaciones para poder probar P2PP en nuestro entorno.

En primer lugar ha sido necesario modificar el método Transport::getAddrPort para que reconociera la interfaz sobre la que debía enviar los paquetes en nuestro sistema de virtualización. Para ello debemos leer una variable de entorno creada por ModelNet con los datos necesarios.

Por otro lado al añadir nuevos parámetros al archivo de configuración como pueden ser la dirección y puerto del superpeer o el definir nuevos nombres de protocolo ha sido necesario modificar la clase P2PConfig para que analice también los nuevos parámetros.

En último lugar es importante destacar que los archivos Checker.cpp y P2Pmain.cpp encargados de generar los ejecutables del programa también han debido ser modificados para funcionar con los nuevos protocolos.

# CAPÍTULO IV

# VERIFICACIÓN

# 4. VERIFICACIÓN

## *4.1 Introducción*

Uno de los mayores problemas con los que nos encontramos al intentar verificar el comportamiento de un protocolo P2P es que necesitamos una red muy grande para poder obtener resultados significativos. En la actualidad tenemos varias formas para evaluar el comportamiento de un protocolo concreto, estas son, en orden de complejidad creciente, el modelado analítico, la simulación, la emulación y la prueba en un entorno real.

Los modelos analíticos son el método formal por excelencia en la ciencia. Usados para derivar y validar modelos de fenómenos reales, los modelos analíticos nos proporcionan un gran entendimiento de las fuerzas en juego y nos ofrece la posibilidad de explorar un escenario del cual tenemos un control completo. El modelado analítico de escenarios y fenómenos complejos es una cuestión complicada. Los modelos siempre intentan reducir la complejidad y son demasiado esquemáticos, ya que si no, pueden ser demasiado complicados y su proceso de validación muy tedioso.

Muchos de estos modelos son limitados y no valen en la práctica si otros datos tienen que ser añadidos al modelo. Es posible ignorar, o no tener en cuenta, algunos factores en el proceso de modelado. A pesar de su dificultad, su forma es válida en muchos campos científicos, como las matemáticas. Estos modelos deben ser verificados y validados.

En estos últimos años los ordenadores han sido usados como una herramienta para validar modelos analíticos. La computación compleja y la simulación por ordenador son una valiosa herramienta para chequear modelos y validarlos ante la comunidad científica. Con el aumento de la capacidad de

cómputo de los ordenadores convencionales ha sido posible aumentar la complejidad de los escenarios a validar e investigar más profundamente en el entendimiento del modelo. La investigación en redes también aprovecha estos simuladores, entre los cuales se encuentran el NS-2 que puede ser usado para obtener unos resultados de un modo más sencillo que usando modelos de colas.

No debemos olvidar que los simuladores están construidos sobre software que se ejecuta en máquinas físicas. Por tanto, hacer disponibles públicamente las APIs del software y los scripts de simulación, es probablemente la única manera de depurar las herramientas de simulación. Sin embargo, esta no es una práctica común en el mundo de las aplicaciones distribuidas y la mayoría de las APIs de simulación están incompletas, mal documentadas o focalizadas en un tipo de aplicación concreto. Esta situación empeora si consideramos que la mayoría de los simuladores no permiten ejecutar implementaciones o aplicaciones reales. Aunque algunos simuladores permiten ejecutar aplicaciones sin modificar sobre ellos, normalmente las implementaciones de los protocolos tienen que ser creadas sin un paso de simulación previo, lo que incrementa el riesgo de bugs y errores de implementación.

Otro problema de la simulación es que los recursos de hardware no son infinitos y muchas veces no son suficientes. Cuando tratamos con redes a gran escala este es un problema serio. La simulación de escenarios simples y pequeños no nos puede asegurar en la mayoría de los casos la escalabilidad de la solución y que las propiedades a pequeña escala se mantendrán si el tamaño de la red crece.

Aunque la velocidad del hardware no sea un factor clave en una simulación de eventos (excepto si el tiempo de simulación se hace demasiado largo), estos introducen una gran carga de uso de recursos, especialmente CPU y memoria. Por ejemplo, si un paquete debe ser reenviado desde un nodo hasta otro en el escenario de simulación, el paquete soltado por el nodo genera un evento en el simulador de eventos. Cuando ese evento ha sido procesado, otro evento es generado y guardado, la llegada del paquete a su destino. Para introducir las características de una red “real”, el tiempo de latencia debe ser computado teniendo en cuenta las condiciones actuales y futuras de la red.

Entonces, la transmisión de paquetes genera sobrecarga de CPU y memoria que crece linealmente con el tamaño de la red. Otras operaciones y transacciones pueden hacer que la memoria usada crezca exponencialmente, en otras palabras, la simulación de una red de gran escala puede requerir una ingente cantidad de recursos, tomando años para finalizar y generar resultados.

Cuando los investigadores tratan de evitar estas limitaciones tienen tres caminos principales que seguir,

I. Incrementar la cantidad de recursos comprando más hardware para aumentar el poder de procesamiento o hacer uso de tecnologías de computación paralela.

II. Simplificar el modelo de la red para que este genere menos eventos y uso de recursos.

III. Intentar una clase diferente de herramienta de evaluación de redes.

Cuando elegimos la tercera opción, debemos optar por emuladores o por pruebas en escenarios reales. En la siguiente ilustración se muestra en qué situación es mejor usar una solución u otra en términos del realismo del experimento y su complejidad. Los modelos analíticos y las simulaciones están en la parte inferior-izquierda ya que aportan un menor realismo y necesitan menos requerimientos que los emuladores y las pruebas en entornos reales.

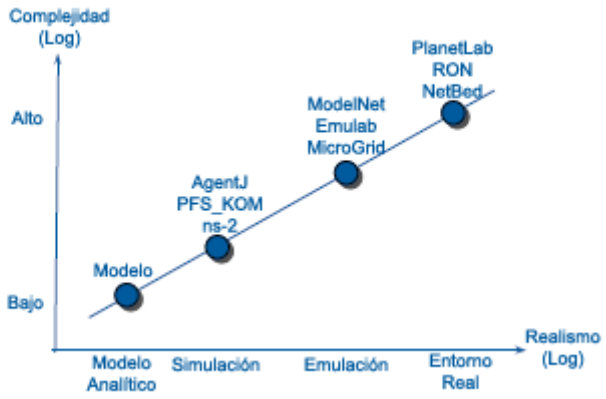


**Figura 30:** Herramientas, complejidad vs. realismo

En nuestro caso partimos del modelo analítico descrito en [MYBC+09], para redes P2P jerárquicas, usaremos el simulador PeerFactSim.KOM [KKL+07] para validar este modelo y finalmente usaremos ModelNet para emular una red en la que ejecutaremos instancias reales de nuestra implementación.

### *4.1.1 El modelo analítico*

En este caso el modelo descrito por [MYBC+09] se basa en [MYCGM08] para describir el comportamiento en el routing de una red P2P jerárquica.

Para ello empieza describiendo una lista de parámetros del modelo analítico:

K: El número de dominios.

$M_k$: El número de peers en un el dominio k.

N: Todos los peer de todos los dominios. En nuestro caso, consideramos que un peer no puede pertenecer a varios dominios, por tanto $N = \sum_{i=1}^{K} M_i$ .

$S_k$: El número de super-peers en el dominio k.

$\rho_{ij}$: La probabilidad de lanzar una query desde el dominio i hasta el dominio j.

C(x): El número de saltos necesarios para encontrar un super-peer en la red de interconexión dependiente del número de super-peers x. Este valor depende del tipo de red usado en la red de interconexión.

$D_k(x)$: El número de saltos necesarios para encontrar un peer en una red plana del tipo k en función del número de peers x en el dominio.

Para este estudio asumimos que todos los peers en un dominio conocen el super-peer asignado a este. Esta condición implica que solo es necesario un salto para encontrar al super-peer. El enrutado dentro de un mismo dominio no cambia y es el mismo que en una red plana. En todo caso, si una query debe ser enrutada a otro dominio, debe haber, en todo caso, un salto hasta alguno de los superpeers. El peor caso sucede cuando todos los super-peer de un dominio están conectados a la red de interconexión. Cuando el número de super-peers

conectados crece, el número de saltos para encontrar un recurso en la red de interconexión también crece. Sin embargo, este incremento es marginal: entre uno y tres saltos, dependiendo del número de super-peers por dominio en la red de interconexión.

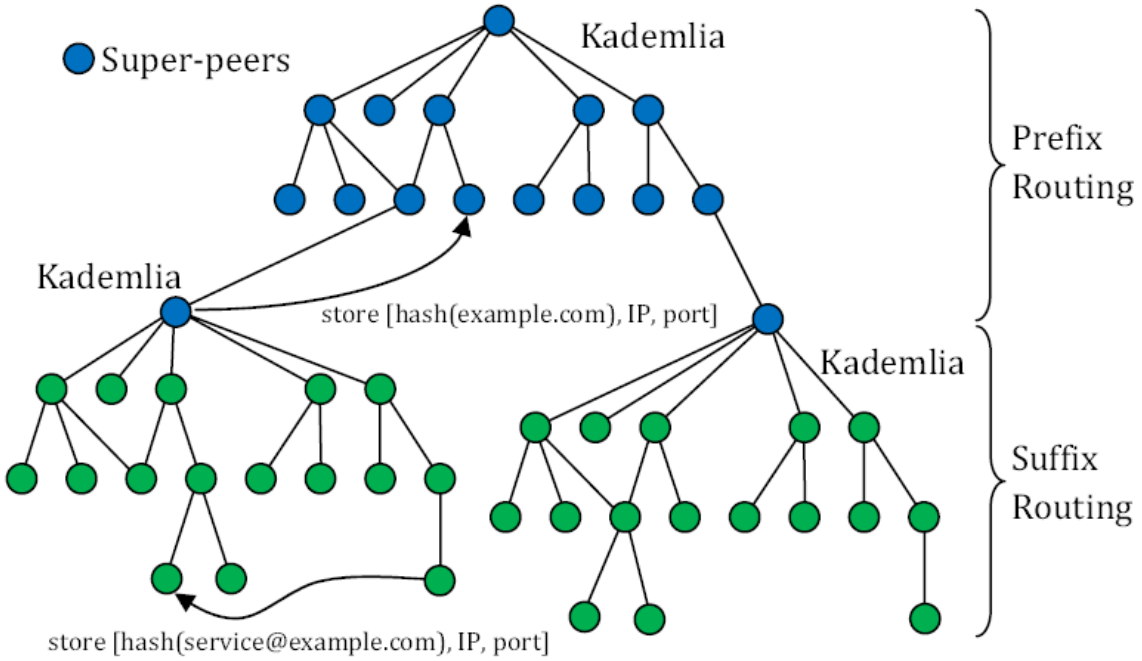


**Figura 31:** Red overlay de Kademlia jerárquico [6]

Teniendo en cuenta las definiciones de arriba, podemos obtener un modelo del comportamiento de la red (Routing Performance, RP) en este tipo de overlay jerárquica basado en DHT. Antes de nada, definimos el coste de encontrar un peer en cada overlay:

- $D_k(M_k)$: El coste de encontrar un peer en su propio dominio.
- $C(\sum_{k=1}^{K} S_k)$: El coste de encontrar un super-peer en la overlay de interconexión.

Si la probabilidad de obtener un ítem en un dominio de su super-peer es considerada insignificante y contando que con que el número de peers en un

[6] Figura obtenida de [MYBC+09]

dominio des N/K con N>>K, el Routing Performance medio experimentado por un peer en el dominio i puede ser descrito como sigue:

$$RP_i = \rho_{ii} \cdot D_i(M_i) + \sum_{j=1, j\neq i}^{K} \rho_{ij} \cdot [1 + D_j(M_j) + C(\sum_{k=1}^{K} S_k)]$$

El primer término es la suma del coste de buscar algo en su propio dominio, mientras que el segundo término es el coste de buscar en otro dominio.

El número medio de saltos viene dado por:

$$RP = \frac{1}{N} \cdot \sum_{i=1}^{K} M_i \cdot RP_i$$

Lo que si el número de peers es igual en todos los dominios, se simplifica a:

$$RP = \frac{1}{K} \cdot \sum_{i=1}^{K} RP_i$$

Como hemos asumido que el número de peers es igual en cada dominio y cada búsqueda en la red es considerada aleatoriamente independiente, obtenemos que la probabilidad de buscar un peer conectado a otro dominio esta equitativamente distribuida entre el resto de dominios. Además, la probabilidad de buscar un peer en nuestro propio dominio es diferente de la probabilidad de buscar a un peer en otro dominio. Entonces, la probabilidad de buscar entre dominios es $\rho_{ij} = \frac{1-\rho_{ii}}{K-1}$ y podemos expresar la ecuación anterior como:

$$RP_i = \rho_{ii} \cdot D_i(M_i) + \sum_{j=1, j\neq i}^{K} \frac{1-\rho_{ii}}{K-1} \cdot [1 + D_j(M_j) + C(\sum_{k=1}^{K} S_k)]$$

Esta relación es valiosa en algunos escenarios como VoIP donde $\rho_{ii} > \rho_{ij}$, lo que implica que las llamadas entre peers del mismo dominio son más frecuentes. En otros servicios donde la probabilidad de buscar en el propio dominio es la misma que en otro ( $\rho_{ii} = \rho_{ij} = \frac{1}{K}$ ), la relación se simplifica a;

$$RP_i = \frac{1}{K} \cdot D_i(M) + \sum_{j=1, j\neq i}^{K} \frac{1}{K} \cdot [1 + D_j(M) + C(\sum_{k=1}^{K} S_k)]$$

Finalmente, si el mismo tipo de overlay es usado en todos los dominios el sumatorio puede ser eliminado quedando:

$$RP_i = \frac{1}{K} \cdot D(M) + \frac{K-1}{K} \cdot [1 + D(M) + C(\sum_{k=1}^{K} S_k)] =$$
$$= D(M) + \frac{K-1}{K} \cdot [1 + C(\sum_{k=1}^{K} S_k)]$$

### 4.1.2 Simulación

Una vez obtenido un modelo analítico el siguiente paso lógico es pasar a validarlo mediante simulación. En este caso podemos ver en [MYBG+08a] las simulaciones que muestran una tendencia similar a la esperada tras ver la forma matemática del modelo.

Estas simulaciones fueron realizadas con el simulador de redes P2P PeerfactSim.KOM e incluían pruebas para 1, 5, 10 y 20 dominios, un número de peers entre 100 y 1000 distribuidos logarítmicamente y para probabilidad de buscar en un propio dominio igual a 0.3, 0.6, 0.9. Para cada uno de estos tipos las pruebas se repitieron 10 veces con el objetivo de aumentar la precisión y reducir el tamaño de los intervalos de confianza.

En estas pruebas se analizó el número de saltos necesarios para llevar a cabo una búsqueda, dando como resultado una relación logarítmica con respecto al número de peers que participaban en el experimento. De igual manera, se observa que el número de saltos disminuye a medida que aumentamos el número de dominios y cuando aumentamos la probabilidad de buscar en nuestro propio dominio.

El otro resultado analizado en estas simulaciones es el número de entradas en la tabla de rutas de los peers. Al igual que en los saltos, se observa que al aumentar el número de dominios, el número de entradas de tabla de rutas disminuye.

### 4.1.3 Emulación

Después de comprobar el modelo analítico mediante simulación vamos a pasar a probarlo con nuestra implementación real del protocolo. Dada la

dificultad de probar una red overlay con cientos de nodos en un entorno real, vamos a realizar las pruebas ejecutando nuestro prototipo sobre un emulador de red.

## 4.2 Creación del escenario pruebas

### 4.2.1 Configuración del sistema

Para el desarrollo y evaluación de nuestras pruebas tenemos un sistema compuesto por 6 máquinas físicas de las cuales 5 son usadas para emular una red real con ModelNet y la sexta es usada para desarrollar el software necesario.

Las máquinas encargadas de emular la red están conectadas entre sí a través de un switch Gigabit Ethernet formando una red privada. Además, la máquina llamada “aphrodite” también está conectada a la red pública del laboratorio para poder comunicarnos con ella desde el exterior. Esta última conexión será la que usará “biogridnet2” para configurar e iniciar las pruebas.

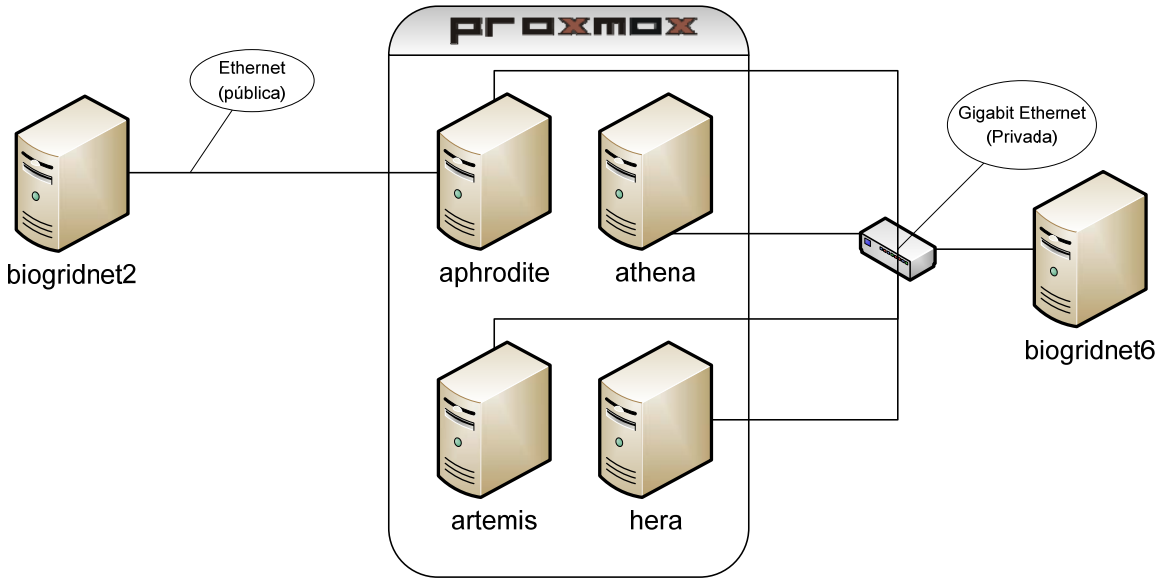


**Figura 32:** Esquema físico del sistema

Como podemos ver en la figura anterior, en 4 de las máquinas cuya función es emular la red se ha instalado Proxmox [Apéndice D] con el fin de crear en ellas máquinas virtuales que nos ayuden a aprovechar todo el potencial

de nuestro equipo. De este modo obtenemos un sistema diferente al físico sobre el que emular nuestra red virtual. En este sistema lógico podemos diferenciar claramente tres tipos de máquinas.

Por un lado tenemos las máquinas que actuarán como edge node de ModelNet en las que podemos ver 8 máquinas virtuales conectadas al bridge virtual creado por Proxmox.

En otro grupo se encuentras las máquinas que actuaran como core node de ModelNet. De ellas, tenemos dos que son máquinas virtuales conectadas al bridge virtual y una última que es una máquina física conectada a la red privada.

El tercer grupo de máquinas son las máquinas de apoyo. En ellas encontramos a “zeus”, que actúa como servidor DHCP y DNS, a control, que es la pasarela hacía el exterior que tiene la red privada además de ser la máquina desde la que se ejecutan las diferentes pruebas de cada experimento, y a biogridnet2 que es la máquina que usaremos para desarrollar nuestro software y ejecutar las pruebas.

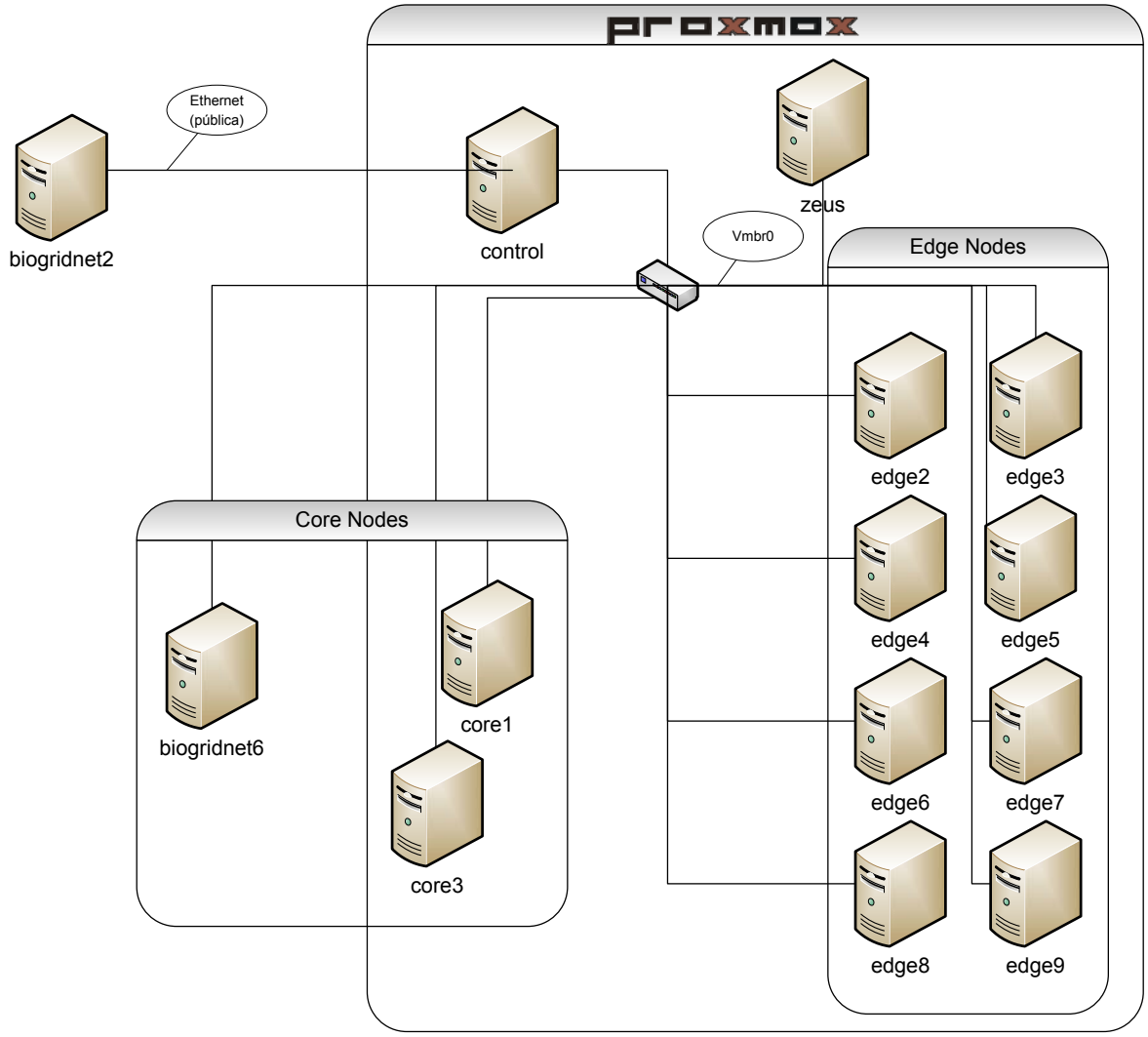


**Figura 33:** Esquema lógico del sistema

Las máquinas virtuales que actúan como edge node están basadas en un sistema operativo Debian 5.0 corriendo en un sistema de tipo OpenVZ, en el que se ha instalado la aplicación de ModelNet según [Apéndice C]. En estas máquinas también se ha instalado Java para facilitar algunas tareas.

En las máquinas que actúan como core node se ha recompilado un FreeBSD 4.8 para ejecutar sobre él los emuladores de ModelNet. Dos de estas máquinas son máquinas virtuales del tipo KVM/Qemu mientras que la tercera, biogridnet6, es una máquina real. En estas máquinas no se ha instalado ninguna otra aplicación, ya que interesa que sean los más simple posibles para poder realizar su labor sin problemas.

En cuanto a las máquinas de apoyo, en “zeus”, una máquina virtual de tipo OpenVZ, se ha configurado un Debian 4.0 para que actúe como servidor DHCP y DNS, “control”, una máquina virtual de tipo KVM/Qemu, es un Ubuntu 8.0 y “biogridnet2” es una máquina real con Debian 5.0 sobre el que se ha instalado el software NetBeans [NETB09] para ayudar a desarrollar el software.

### *4.2.2 Herramientas*

Con el fin de facilitar las tareas a la hora de preparar y ejecutar el experimento hemos desarrollado una serie de aplicaciones en Java que automaticen estas tareas. Las principales de estas aplicaciones son un analizador del XML con los datos de proyecto CAIDA para generar nuestro gráfico de la red, un programa que genera los archivos necesarios para ejecutar un experimento y un programa que genera un archivo en formato CSV a partir de las trazas generadas por nuestra implementación de P2PP para poder importar los datos después con Matlab.

**Analizador del XML de CAIDA**

El proyecto “AnalizadorXML_CAIDA” es una sencilla aplicación desarrollada en Java que se encarga de analizar un archivo con datos del proyecto caída para generar una topología que pueda entender ModelNet.

Para ello la aplicación hace uso de JDOM y va analizando un archivo XML con un formato similar al siguiente:

```
<SummaryReport from="Germany" to="Germany" minimumRtt="6.0" averageRtt="7.68"
delayVariation="2.25" packetLoss="0.0"/>

    <SummaryReport from="Germany" to="Africa" minimumRtt="93.65" averageRtt="102.64"
delayVariation="3.18" packetLoss="0.01"/>

    <SummaryReport from="Germany" to="Balkans" minimumRtt="68.84" averageRtt="84.02"
delayVariation="5.47" packetLoss="0.29"/>

    <SummaryReport from="Germany" to="France" minimumRtt="43.68" averageRtt="48.06"
delayVariation="4.75" packetLoss="0.01"/>

    <SummaryReport from="Germany" to="Australia" minimumRtt="292.06" averageRtt="310.8"
delayVariation="11.57" packetLoss="0.01"/>
```

Durante la ejecución se recorre el archivo en busca de enlaces en los que estén involucrados dos de nuestros países seleccionados. Una vez encontrado un enlace así se calcula el tiempo de transito a partir del Rtt medio y se introduce en el archivo con el gráfico de la topología para ModelNet.

En el caso de encontrar un enlace que vaya desde un país a si mismo se genera una entrada dentro del apartado “specs” de la topología que será usado para marcar el tiempo de tránsito entre los edge node asignados a ese país y su emulador.

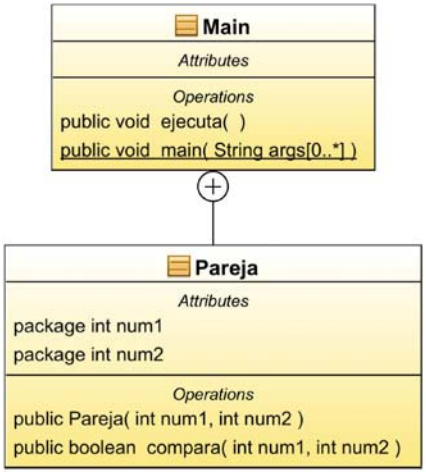


**Figura 34:** Esquema de clases de AnalizadorXML_CAIDA

La clase Main es la encargada de todo el proceso de análisis y generación del archivo de topología. Para ello hace uso de la clase Pareja, que básicamente guarda los números de un enlace con el fin de evitar que introduzcamos más de un enlace entre dos nodos en nuestra topología.

### Generador del experimento

En cada uno de nuestros experimentos lanzamos cientos de pruebas, y en cada una de estas pruebas se ejecuta la implementación cientos de veces y se realizan miles de consultas. Es por ello que hemos creado el proyecto “ExperimentoP2PP” encargado de generar automáticamente todos los archivos necesarios para ejecutar cada experimento.

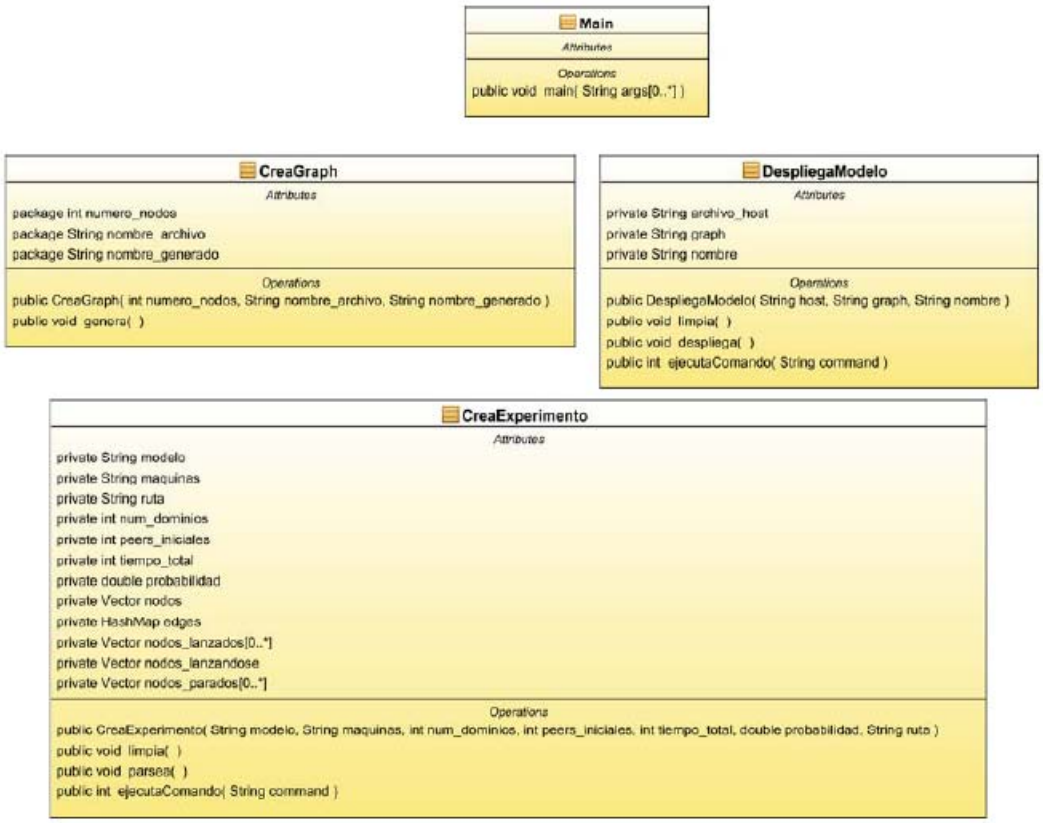


**Figura 35:** Esquema de clases de ExperimentoP2PP

Cuando ejecutamos el generador de experimentos, la clase Main se encarga en un primer momento de usar las clases CreaGraph y DespliegaModelo. Después, va llamando a la clase CreaExperimento tantas veces como pruebas vamos a lanzar y va generando un script que lance luego todas estas pruebas. Una vez acabada la creación de todos los archivos, el programa se encarga de comprimirlos y finalmente manda el archivo comprimido a la máquina virtual que actúa como centro de control.

La clase CreaGraph se encarga de terminar el archivo con la topología de la red. Esta clase recibe un archivo con la topología central de la red y le agrega tantos nodos virtuales como vayamos a necesitar. Para ello hace uso de JDOM, para leer el archivo de entrada, comprobando el número de enlaces y nodos existentes, para a continuación añadir los nodos virtuales.

La clase DespliegaModelo genera tres scripts. El primero, despliegaModelo.sh, se diseña para ser ejecutado desde la máquina virtual de

control. Ese script se encarga de mandar a un emulador el archivo con el gráfico de la topología y el que contiene las máquinas a usar para generar el archivo con el modelo y el archivo de rutas que usará ModelNet. Después coge estos archivos y los manda a todas las máquinas que participan en el experimento. Para aumentar el rendimiento, también se encarga de terminar cualquier ejecución del experimento que haya en curso. El script creado tiene la siguiente forma (A partir de ahora, los ejemplos de scripts se limitarán a una prueba en la que solo participara el core node “biogridnet6” y el edge node “edge3”):

```
#!/bin/bash

ssh root@biogridnet6.mnuc3m.es "pkill p2pmain"

ssh root@edge3.mnuc3m.es "pkill p2pmain"

scp /home/control/p2pp/experimentoP2PP/probando.graph
root@biogridnet6.mnuc3m.es:/root/modelos/probando.graph

scp /home/control/p2pp/experimentoP2PP/sample.hosts
root@biogridnet6.mnuc3m.es:/root/modelos/sample.hosts

ssh root@biogridnet6.mnuc3m.es "allpairs /root/modelos/probando.graph >
/root/modelos/probando.route"

ssh root@biogridnet6.mnuc3m.es "mkmodel /root/modelos/probando.graph
/root/modelos/sample.hosts > /root/modelos/probando.model"

scp root@biogridnet6.mnuc3m.es:/root/modelos/probando.model
/home/control/p2pp/experimentoP2PP/probando.model

scp root@biogridnet6.mnuc3m.es:/root/modelos/probando.route
/home/control/p2pp/experimentoP2PP/probando.route

scp /home/control/p2pp/experimentoP2PP/probando.model
root@biogridnet6.mnuc3m.es:/root/modelos/probando.model

scp /home/control/p2pp/experimentoP2PP/probando.model
root@edge3.mnuc3m.es:/root/modelos/probando.model

scp /home/control/p2pp/experimentoP2PP/probando.route
root@edge3.mnuc3m.es:/root/modelos/probando.route
```

Además, también crea otro archivo llamado “ejecuta1.sh” que se encarga de mandar a la máquina virtual de control los archivos de ModelNet y de ejecutar el script creado anteriormente. También se encarga de traer a la máquina en la que se está ejecutando el generador de experimentos el archivo con el modelo creado por ModelNet, usado posteriormente por la clase CreaExperimento. Este script tiene la siguiente forma:

```
#!/bin/bash

ssh control@163.117.140.36 "~/p2pp/reiniciaRed.sh"

sleep 3

ssh control@163.117.140.36 "mkdir /home/control/p2pp/experimentoP2PP/"

ssh control@163.117.140.36 "mkdir /home/control/p2pp/experimentoP2PP/logs"

ssh control@163.117.140.36 "rm /home/control/p2pp/experimentoP2PP/*"
```

```
ssh control@163.117.140.36 "rm /home/control/p2pp/experimentoP2PP/logs/*"

ssh control@163.117.140.36 "pkill despliega.sh"

scp despliegaModelnet.sh
control@163.117.140.36:/home/control/p2pp/experimentoP2PP/despliegaModelnet.sh

ssh control@163.117.140.36 "chmod 777
/home/control/p2pp/experimentoP2PP/despliegaModelnet.sh"

scp probando.graph
control@163.117.140.36:/home/control/p2pp/experimentoP2PP/probando.graph

scp sample.hosts control@163.117.140.36:/home/control/p2pp/experimentoP2PP/sample.hosts

ssh control@163.117.140.36 "/home/control/p2pp/experimentoP2PP/despliegaModelnet.sh"

scp control@163.117.140.36:/home/control/p2pp/experimentoP2PP/probando.model
probando.model
```

También crea el archivo ejecutaModelnet.sh que será el encargado de arrancar los emuladores antes de empezar cada prueba. Este script termina primero los procesos lanzados por pruebas anteriores y después ejecuta el comando “deployhost” de ModelNet en todos los emuladores y edge nodes para que se creen las interfaces en las máquinas y ModelNet pueda enrutar bien los paquetes. El script generado tiene la siguiente forma:

```
#!/bin/bash

ssh root@biogridnet6.mnuc3m.es "pkill p2pmain"

ssh root@biogridnet6.mnuc3m.es "pkill checker"

ssh root@edge3.mnuc3m.es "pkill p2pmain"

ssh root@edge3.mnuc3m.es "pkill checker"

sleep 0.5

ssh root@biogridnet6.mnuc3m.es "deployhost /root/modelos/probando.model
/root/modelos/probando.route"

ssh root@edge3.mnuc3m.es "deployhost /root/modelos/probando.model
/root/modelos/probando.route"
```

Por último se ejecuta el método parsea() de la clase CreaExperimento tantas veces como pruebas vallamos a lanzar en el experimento. Esta clase es el centro de esta aplicación y es la encargada de crear todos los archivos que necesita nuestra implementación de P2PP, así como los scripts para lanzarla.

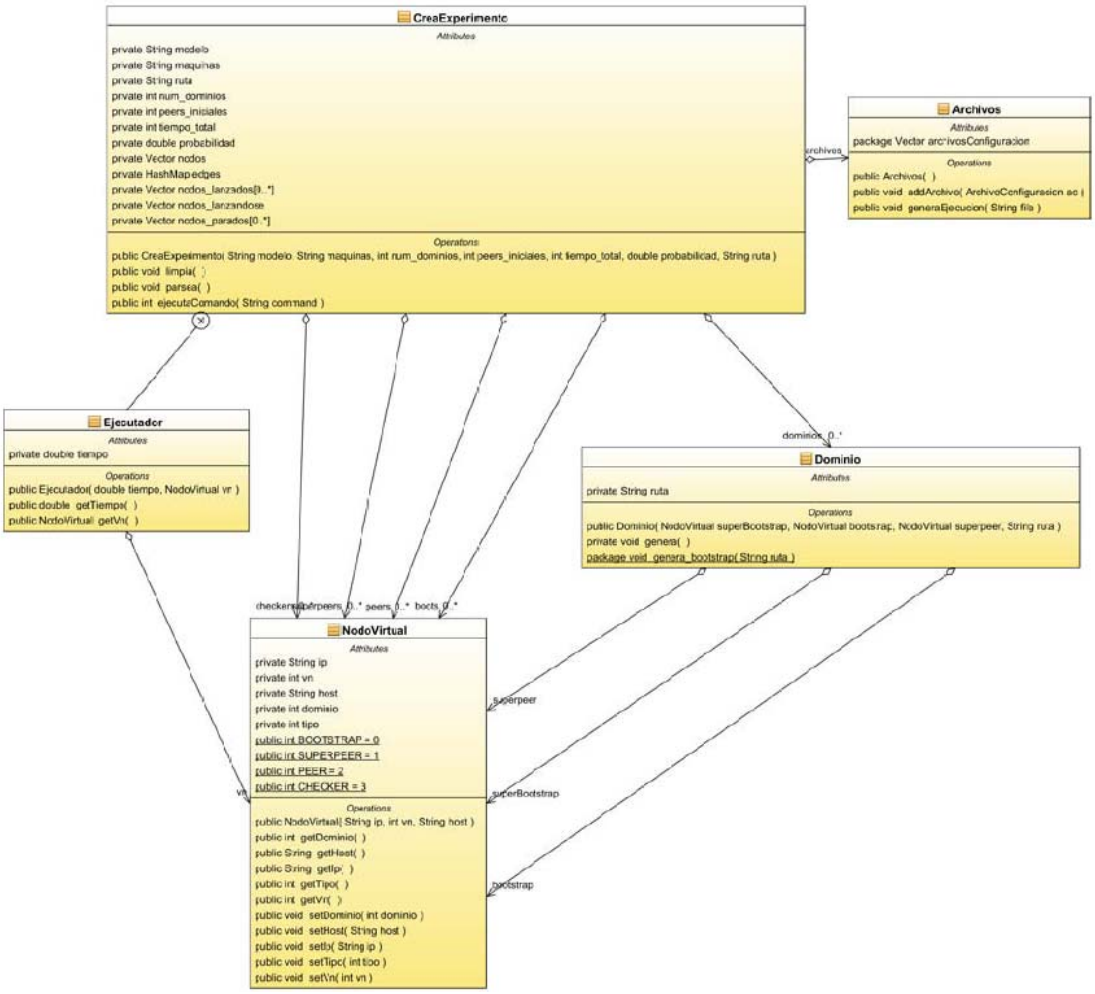


**Figura 36:** Esquema de clases de CreaExperimento

El método parsea() recorre en un primer lugar el archivo con el modelo creado por ModelNet llenando un Vector con los datos de todos los nodos virtuales, para guardar los datos de cada nodo virtual usa a su vez la clase NodoVirtual.

El segundo paso para la generación de los archivos es asignar a cada nodo virtual la función que va a tener en la prueba. Para ello asignamos a los primeros la función de checker, luego la de boostrap, superpeer o peer normal. Aprovechamos este momento además para asignar a cada nodo el dominio en el que va a estar.

Una vez tenemos los datos de todos los nodos participantes en la prueba, procedemos a crear los archivos de configuración necesarios. Para ello hacemos uso de la clase Dominio, la cual a partir de los datos del nodo asignado como

superpeer y el nodo asignado como bootstrap en el dominio, además del nodo que actúa como bootstrap en la red de superpeers genera los archivos de configuración para los peers y los superpeers de cada dominio. También en este momento se generan los archivos de configuración para los bootstraps y los checkers.

Una vez tenemos los archivos de configuración creados pasamos a crear los scripts para ejecutar la prueba. Para ello por cada edge node participante en la prueba se generarán cuatro archivos, uno para ejecutar los nodos que actúan como bootstrap, otro para ejecutar los superpeers, el tercero para ejecutar los peers al inicio del experimento y uno último que se encarga de realizar las querys y aplicar el churn. Los nodos que participan desde el principio en el experimento son elegidos aleatoriamente.

Para la generación de las querys y el churn usamos unas distribuciones estadísticas que nos dicen cuando se producirá el siguiente evento. Para la generación de los tiempos en que se producen las querys usamos una distribución de poisson, mientras que para el de los tiempos de entrada y salida de los peers usamos una distribución binomial con r=17 and p=0.127 como se indica en [SENB07a].

Cuando se produce un evento de salida o entrada de un peer este es elegido aleatoriamente entre todos los peer disponibles en ese momento. Cuando se llega al tiempo en el que debemos hacer una query elegiremos un peer desde el que comenzar la búsqueda, y luego elegimos el peer a buscar dentro o fuera de su dominio dependiendo de la probabilidad marcada.

Los archivos generados tienen un formato similar al siguiente:

```
#!/bin/bash

sleep 0.661

nohup vnrunhost 5 /root/modelos/probando.model /root/p2pp/checker -a 10.0.0.121 -p 7080 -u
nodo961@dominio3 -n 6005 -o nodo960@dominio2 >> /root/p2pp/logs_checker/log_querys2a3.txt
&

sleep 0.9520000000000002

nohup vnrunhost 5 /root/modelos/probando.model /root/p2pp/checker -a 10.0.1.57 -p 7080 -u
nodo539@dominio3 -n 6013 -o nodo444@dominio1 >> /root/p2pp/logs_checker/log_querys1a3.txt
&

sleep 0.8740000000000003
```

```
nohup vnrunhost 5 /root/modelos/probando.model /root/p2pp/checker -a 10.0.1.214 -p 7080 -u
nodo309@dominio1 -n 6021 -o nodo683@dominio3 >> /root/p2pp/logs_checker/log_querys3a1.txt
&

sleep 0.919

nohup vnrunhost 5 /root/modelos/probando.model /root/p2pp/checker -a 10.0.3.244 -p 7080 -u
nodo675@dominio4 -n 6029 -o nodo927@dominio2 >> /root/p2pp/logs_checker/log_querys2a4.txt
&

sleep 0.96

nohup vnrunhost 5 /root/modelos/probando.model /root/p2pp/checker -a 10.0.1.4 -p 7080 -u
nodo756@dominio4 -n 6037 -o nodo29@dominio1 >> /root/p2pp/logs_checker/log_querys1a4.txt &

sleep 0.8319999999999999

nohup vnrunhost 5 /root/modelos/probando.model /root/p2pp/checker -a 10.0.1.156 -p 7080 -u
nodo265@dominio2 -n 6045 -o nodo215@dominio0 >> /root/p2pp/logs_checker/log_querys0a2.txt
&

sleep 0.8740000000000006

nohup vnrunhost 5 /root/modelos/probando.model /root/p2pp/checker -a 10.0.0.66 -p 7080 -u
nodo218@dominio2 -n 6053 -o nodo518@dominio4 >> /root/p2pp/logs_checker/log_querys4a2.txt
&

sleep 0.8629999999999995

nohup vnrunhost 5 /root/modelos/probando.model /root/p2pp/checker -a 10.0.3.159 -p 7080 -u
nodo48@dominio3 -n 6061 -o nodo249@dominio3 >> /root/p2pp/logs_checker/log_querys3a3.txt &

sleep 0.8250000000000002

nohup vnrunhost 5 /root/modelos/probando.model /root/p2pp/checker -a 10.0.2.72 -p 7080 -u
nodo432@dominio2 -n 6069 -o nodo570@dominio0 >> /root/p2pp/logs_checker/log_querys0a2.txt
&

sleep 0.9249999999999998

nohup vnrunhost 5 /root/modelos/probando.model /root/p2pp/checker -a 10.0.3.82 -p 7080 -u
nodo268@dominio3 -n 6077 -o nodo654@dominio0 >> /root/p2pp/logs_checker/log_querys0a3.txt
&
```

Una vez generados los archivos que llevarán a cabo la ejecución de la prueba hay que crear también un script que recupere los resultados obtenidos de los edge node. Para ello generamos el archivo recoge.sh que lleva hasta la máquina virtual de control las trazas obtenidas en la prueba. Este archivo tiene un formato como el siguiente:

```
#!/bin/bash

scp root@edge3.mnuc3m.es:/root/Resultados.txt
/home/control/p2pp/experimentoP2PP/experimentos/dom5_prob0.2_indice0/resultados/temp.txt

cat
/home/control/p2pp/experimentoP2PP/experimentos/dom5_prob0.2_indice0/resultados/temp.txt
>>
/home/control/p2pp/experimentoP2PP/experimentos/dom5_prob0.2_indice0/resultados/Resultados
.txt

scp root@edge3.mnuc3m.es:/root/*.txt
/home/control/p2pp/experimentoP2PP/experimentos/dom5_prob0.2_indice0/recoge/
```

Aparte de todos estos archivos también creamos otro script diseñado para ser ejecutado en la máquina virtual de control que se encarga de copiar los otros archivos a los edge nodes y de ejecutar la prueba. Este archivo tiene el siguiente formato:

```
#!/bin/bash
.sh"
/home/control/p2pp/reiniciaRed.sh
ssh  root@edge3.mnuc3m.es "pkill lanza"
ssh  root@edge3.mnuc3m.es "pkill p2pmain"
ssh  root@edge3.mnuc3m.es "pkill -f checker"
ssh  root@edge3.mnuc3m.es "rm /root/p2pp/*"
ssh  root@edge3.mnuc3m.es "rm /root/*.txt"
ssh  root@edge3.mnuc3m.es "rm /root/nodo*"
ssh  root@edge3.mnuc3m.es "rm /root/p2pp/logs_checker/*"
ssh  root@edge3.mnuc3m.es "mkdir /root/p2pp/logs_checker/"
scp /home/control/p2pp/experimentoP2PP/experimentos/* root@edge3.mnuc3m.es:/root/p2pp/
scp /home/control/p2pp/experimentoP2PP/experimentos/dom5_prob0.2_indice0/archivos/*
root@edge3.mnuc3m.es:/root/p2pp/
ssh  root@edge3.mnuc3m.es "chmod 777 /root/p2pp/*.sh"
ssh  root@edge3.mnuc3m.es "/root/p2pp/lanzabootedge3.sh"
```

Este script, borra los archivos de pruebas anteriores de los edge node y mata todos los procesos relacionados con estas pruebas para dejar las máquinas preparadas. Después copia los archivos generados y por último va ejecutando los scripts creados para lanzar los bootstrap, los superpeers, los peers normales y finalmente las querys.

El último archivo generado por esta herramienta es generado por la clase Main a medida que se van creando las diferentes pruebas con el nombre de ejecutaExperimento.sh. Este archivo es el encargado de ejecutar desde la máquina virtual de control todos los script “ejecutaRemoto.sh” y posteriormente “recoge.sh” para ejecutar el experimento y recoger los resultados. Este script tiene el siguiente formato:

```
#!/bin/bash
pkill ejecutaRemoto.sh
chmod 777 /home/control/p2pp/experimentoP2PP/experimentos/ejecutaModelnet.sh
/home/control/p2pp/experimentoP2PP/experimentos/ejecutaModelnet.sh
sleep 15
echo "Empieza el experimento"
date >> /home/control/p2pp/experimentoP2PP/experimentos/log.txt
echo "EXPERIMENTO: dom5 prob:0.2 indice:0 empezamos." >>
/home/control/p2pp/experimentoP2PP/experimentos/log.txt
chmod 777
/home/control/p2pp/experimentoP2PP/experimentos/dom5_prob0.2_indice0/archivos/ejecutaRemot
o.sh
/home/control/p2pp/experimentoP2PP/experimentos/dom5_prob0.2_indice0/archivos/ejecutaRemot
o.sh
sleep 3060
```

```
date >> /home/control/p2pp/experimentoP2PP/experimentos/log.txt

echo "EXPERIMENTO: dom5 prob:0.2 indice:0 completado, recogemos" >>
/home/control/p2pp/experimentoP2PP/experimentos/log.txt

chmod 777 /home/control/p2pp/experimentoP2PP/experimentos/dom5_prob0.2_indice0/recoge.sh

/home/control/p2pp/experimentoP2PP/experimentos/dom5_prob0.2_indice0/recoge.sh

echo "Información recogida"

sleep 60

pkill ejecutaRemoto.sh

chmod 777 /home/control/p2pp/experimentoP2PP/experimentos/ejecutaModelnet.sh

/home/control/p2pp/experimentoP2PP/experimentos/ejecutaModelnet.sh

sleep 15

echo "Empieza el experimento"

date >> /home/control/p2pp/experimentoP2PP/experimentos/log.txt

echo "EXPERIMENTO: dom5 prob:0.2 indice:1 empezamos." >>
/home/control/p2pp/experimentoP2PP/experimentos/log.txt

chmod 777
/home/control/p2pp/experimentoP2PP/experimentos/dom5_prob0.2_indice1/archivos/ejecutaRemot
o.sh

/home/control/p2pp/experimentoP2PP/experimentos/dom5_prob0.2_indice1/archivos/ejecutaRemot
o.sh

sleep 3060

date >> /home/control/p2pp/experimentoP2PP/experimentos/log.txt

echo "EXPERIMENTO: dom5 prob:0.2 indice:1 completado, recogemos" >>
/home/control/p2pp/experimentoP2PP/experimentos/log.txt

chmod 777 /home/control/p2pp/experimentoP2PP/experimentos/dom5_prob0.2_indice1/recoge.sh

/home/control/p2pp/experimentoP2PP/experimentos/dom5_prob0.2_indice1/recoge.sh

echo "Información recogida"
```

Este último archivo es el único que debemos ejecutar desde la máquina virtual de control para ejecutar el experimento completo.

### Analizador de resultados

El proyecto “AnalizaResultados” se ha creado para obtener los datos deseados a partir de las trazas generadas durante el experimento. Para ello hemos desarrollado dos clases Java que se encargan de parsear las trazas y calcular las medias.


**Figura 37:** Esquema de clases de AnalizadorResultados

La clase Main se encarga de ir recorriendo las diferentes pruebas del experimento usando las otras dos clases para realizar los cálculos. Una vez realizados los cálculos recoge la información y la guarda en un archivo con un formato entendible por Matlab. El formato usado es el siguiente:

Número de dominios, probabilidad, Media saltos, Saltos Máximos, Saltos Mínimos, Media Tiempo, Tiempo Máximo, Tiempo Mínimo, Media tabla Superpeer, Máximo tabla Superpeer, Media tabla Peer, Máximo tabla Peer, Media tabla Super arriba, Máximo tabla Súper arriba, Media tabla Súper abajo, Máximo tabla Súper abajo.

La clase AnalizaQuerys se encarga de recorrer el archivo generado por el checker para calcular los datos relacionados con la performance de la red. El archivo a analizar tiene el siguiente formato:

```
118, nodo287@dominio2, 200, 4, nodo209@dominio2, 10.0.2.154:7080, 0.554184,
10.0.2.164:7080

126, nodo644@dominio0, 200, 2, nodo778@dominio0, 10.0.1.98:7080, 0.532778, 10.0.2.81:7080

134, nodo825@dominio3, 200, 3, nodo876@dominio3, 10.0.2.110:7080, 0.547146,
10.0.0.232:7080

142, nodo76@dominio2, 200, 2, nodo167@dominio2, 10.0.1.150:7080, 0.520672, 10.0.3.9:7080
```

```
150, nodo785@dominio0, 200, 3, nodo913@dominio0, 10.0.0.243:7080, 0.578757,
10.0.0.227:7080
94, nodo622@dominio1, 200, 5, nodo294@dominio3, 10.0.2.37:7080, 8.831891, 10.0.3.205:7080
166, nodo935@dominio0, 404, 4, nodo785@dominio0, 10.0.0.227:7080, 0.558492
174, nodo766@dominio2, 200, 2, nodo619@dominio2, 10.0.2.78:7080, 0.525201, 10.0.3.96:7080
```

De este archivo, nos tenemos que fijar en el tercer término de cada línea, que nos dice el código de respuesta que hemos recibido a esa búsqueda, si el código es el 200 quiere decir que hemos recibido una respuesta correcta y la tenemos que analizar. A continuación nos tenemos que fijar en el cuarto término y en el séptimo, que son el número de saltos necesarios y el tiempo tardado respectivamente. El programa va sumando estos datos para calcular finalmente la media a la vez que va guardado el valor máximo y mínimo.

La clase AnalizaTablaRutas es la encargada de buscar la información en los archivos de trazas generados por los diferentes peer. Nuestra implementación de P2PP genera unos archivos con un como el siguiente:

```
NORMALPEER-total->TablaRutas=[28]
NORMALPEER-total->TablaRutas=[28]
NORMALPEER-total->TablaRutas=[30]
NORMALPEER-total->TablaRutas=[32]
NORMALPEER-total->TablaRutas=[34]
```

El método analiza() va recorriendo estos archivos (uno por cada peer que participa en el experimento) y calcula para cada peer su número medio, máximo y mínimo para posteriormente hallar la media del experimento.

### *4.2.3 Configuración de las pruebas*

Para la realización de las pruebas vamos a emplear dos escenarios distintos. En primer lugar realizaremos pruebas con un gráfico que simule la red REDIMadrid para tener una prueba en la que comprobar el comportamiento de nuestra aplicación en una topología real. Después usaremos los datos del proyecto ARK de CAIDA para crear una topología en la que podamos emular los retardos reales que sufren los paquetes al ser transmitidos entre diferentes países.

**REDIMadrid**

REDIMadrid [REDI09] nace en el año 2003 con el objetivo de establecer una infraestructura de comunicaciones avanzada que permita el intercambio de datos a alta velocidad entre las instituciones con actividad investigadora en el ámbito de la Comunidad de Madrid y que, a su vez, proporcione tránsito hacia otras redes de investigación nacionales e internacionales a través de RedIRIS. REDIMadrid se encuentra en la actualidad en pleno rendimiento, ofreciendo un servicio orientado a obtener la máxima funcionalidad, escalabilidad, disponibilidad y prestaciones de la red.

De esta manera, la red telemática abre posibilidades de comunicación sin precedentes a los investigadores de la Comunidad en todos sus campos de actividad, desde el acceso a bases de datos científicas multimedia a la transferencia masiva de información en aplicaciones de computación distribuida o el uso compartido de equipamiento de investigación. Junto al desarrollo progresivo de la investigación en red, la red fomenta la colaboración entre las instituciones, los proyectos de investigación y las redes temáticas, así como el acercamiento e interacción directa y en tiempo real con otros investigadores en centros internacionales.

***Entidades asociadas***

Las entidades que, actualmente tienen alguna relación con el proyecto REDIMadrid, pueden dividirse en dos grupos diferenciados:

MIEMBROS REDIMADRID:

- Universidad de Alcalá
- Universidad Autónoma de Madrid
- Universidad Carlos III de Madrid
- Universidad Complutense de Madrid
- Universidad Politécnica de Madrid
- Universidad Rey Juan Carlos
- Universidad Nacional de Educación a Distancia (UNED)

- Consejo Superior de Investigaciones Científicas (CSIC)
- Instituto Nacional de Técnica Aeroespacial (INTA)

Estas instituciones disponen de una conexión basada en una interfaz Gigabit Ethernet hacia el nodo central. Éste se halla conectado con RedIRIS a través de un enlace POS a 2,5Gbit/seg. La siguiente figura muestra la topología lógica de la red.

OTRAS ENTIDADES ASOCIADAS:

- Universidad Europea de Madrid (UEM)
- Universidad San Pablo CEU (CEU)
- Universidad Alfonso X El Sabio (UAX)

Estos centros obtienen conectividad con el nodo central de la red mediante un enlace punto a punto a través de un proveedor de servicios de Internet comercial.

Estas instituciones disponen de una conexión basada en una interfaz Gigabit Ethernet hacia el nodo central. Éste se halla conectado con RedIRIS a través de un enlace POS a 2,5Gbit/seg. Los centros asociados reciben una conectividad diferente mediante accesos propios: en el caso de la Universidad Europea de Madrid y la Universidad San Pablo CEU, el enlace es de 100 Mbps, mientras que la Universidad Alfonso X el Sabio el enlace es de 10 Mbps.

***Topología de la red***

La infraestructura de red de REDIMadrid dispone de tres anillos de doble fibra que conectan a las instituciones afiliadas con el nodo central de la red, situado en dependencias del CTI-CSIC, desde donde se proporciona conectividad con RedIRIS, la Red Académica Nacional.

Cada institución conectada a la topología en anillo dispone de una portadora óptica protegida a 1 Gb/s hasta el CTI-CSIC, de modo que la información se transmite desde cada institución hasta el nodo central de REDIMadrid simultáneamente por los dos sentidos del anillo. Desde el nodo

central de REDIMadrid se provee una conexión redundante con el nodo regional de RedIRIS a 2.5 Gb/s.

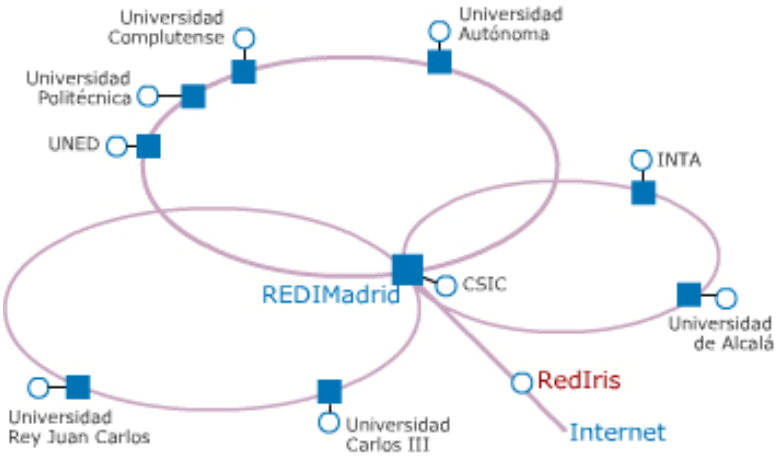


**Figura 38:** Esquema de red de REDIMadrid [7]

***Nuestra topología***

A partir de la topología física de la red hemos generado un archivo de gráficos para usar con ModelNet. El archivo generado es el siguiente:

```
<?xml version="1.0" encoding="UTF-8"?>
<topology>
  <vertices>
    <vertex dbl_y="2" dbl_x="4" int_idx="0" role="gateway" /> <!-- CSIC -->
    <vertex dbl_y="3" dbl_x="4" int_idx="1" role="gateway" /> <!-- Autonoma -->
    <vertex dbl_y="3" dbl_x="3" int_idx="2" role="gateway" /> <!-- Complutense -->
    <vertex dbl_y="3" dbl_x="2" int_idx="3" role="gateway" /> <!-- Politecnica -->
    <vertex dbl_y="3" dbl_x="1" int_idx="4" role="gateway" /> <!-- UNED -->
    <vertex dbl_y="1" dbl_x="1" int_idx="5" role="gateway" /> <!-- Rey Juan Carlos -->
    <vertex dbl_y="1" dbl_x="3" int_idx="6" role="gateway" /> <!-- Carlos III -->
    <vertex dbl_y="2" dbl_x="5" int_idx="7" role="gateway" /> <!-- INTA -->
    <vertex dbl_y="2" dbl_x="6" int_idx="8" role="gateway" /> <!-- Alcala -->

  </vertices>
  <specs xmloutbug="workaround">
    <client-stub int_delayms="1" dbl_plr="0" int_qlen="10" dbl_kbps="10000" />
    <stub-transit int_delayms="1" dbl_plr="0" int_qlen="10" dbl_kbps="1000000" />
  </specs>
```

[7] Figura obtenida de [REDI09]

```
  <edges>
  <!-- Anillo norte -->
    <edge specs="stub-transit" int_idx="0" int_src="0" int_dst="1" int_len="1" />
    <edge specs="stub-transit" int_idx="1" int_src="1" int_dst="0" int_len="1" />
    <edge specs="stub-transit" int_idx="2" int_src="0" int_dst="4" int_len="1" />
    <edge specs="stub-transit" int_idx="3" int_src="4" int_dst="0" int_len="1" />
    <edge specs="stub-transit" int_idx="4" int_src="2" int_dst="1" int_len="1" />
    <edge specs="stub-transit" int_idx="5" int_src="1" int_dst="2" int_len="1" />
    <edge specs="stub-transit" int_idx="6" int_src="3" int_dst="4" int_len="1" />
    <edge specs="stub-transit" int_idx="7" int_src="4" int_dst="3" int_len="1" />
    <edge specs="stub-transit" int_idx="8" int_src="3" int_dst="2" int_len="1" />
    <edge specs="stub-transit" int_idx="9" int_src="2" int_dst="3" int_len="1" />
  <!-- Anillo sur -->
    <edge specs="stub-transit" int_idx="10" int_src="0" int_dst="5" int_len="1" />
    <edge specs="stub-transit" int_idx="11" int_src="5" int_dst="0" int_len="1" />
    <edge specs="stub-transit" int_idx="12" int_src="0" int_dst="6" int_len="1" />
    <edge specs="stub-transit" int_idx="13" int_src="6" int_dst="0" int_len="1" />
    <edge specs="stub-transit" int_idx="14" int_src="5" int_dst="6" int_len="1" />
    <edge specs="stub-transit" int_idx="15" int_src="6" int_dst="5" int_len="1" />
  <!-- Anillo este -->
    <edge specs="stub-transit" int_idx="16" int_src="0" int_dst="7" int_len="1" />
    <edge specs="stub-transit" int_idx="17" int_src="7" int_dst="0" int_len="1" />
    <edge specs="stub-transit" int_idx="18" int_src="0" int_dst="8" int_len="1" />
    <edge specs="stub-transit" int_idx="19" int_src="8" int_dst="0" int_len="1" />
    <edge specs="stub-transit" int_idx="20" int_src="7" int_dst="8" int_len="1" />
    <edge specs="stub-transit" int_idx="21" int_src="8" int_dst="7" int_len="1" />
<!-- Enlaces con los edge node -->

  </edges>
</topology>
```

En el vemos como están generados los 3 anillos de REDIMadrid y las conexiones entre ellos. A partir de esta base generaremos una topología completa añadiendo los vértices en los que correrá nuestra aplicación. Estos vértices harán las veces de equipos conectados en las diferentes entidades asociadas a REDIMadrid.

**Proyecto CAIDA**

La Asociación Cooperativa para el Análisis de Datos en Internet CAIDA [CAID09] es una tarea en la que colaboran organizaciones comerciales, gobiernos y los sectores de investigación encaminadas a promover una mayor

cooperación en la ingeniería y mantenimiento de una infraestructura robusta, escalable y global de Internet.

- CAIDA investiga práctica y teóricamente aspectos de Internet para:
- Dar una idea de la función macroscópica de la infraestructura de Internet, el comportamiento, el uso y la evolución.
- Promover un entorno de colaboración en el que pueden adquirirse los datos, analizarlos y (según corresponda) compartirlos.
- Mejorar la integridad del campo científico en Internet.
- Informar a la ciencia, la tecnología y comunicar las políticas públicas.

Dentro de CAIDA se engloban muchos proyectos entre los que se encuentran *Macroscopic Topology Project* y *Archipelago Measurement Infrastructure (Ark)*.

*Macroscopic Topology Project* es un proyecto que se inicio en 1998. En él se monitorizaba la conectividad global de Internet mandando paquetes de prueba desde un conjunto de monitores origen hasta cientos o miles de destinos distribuidos por todo el planeta.

Los datos recogidos caracterizaban la conectividad global y el rendimiento de Internet, permitía varias representaciones topológicas y geográficas con diferentes grados de granularidad, además de proporcionarnos una base empírica muy importante para modelar el comportamiento y las propiedades de Internet.

Tras una década recogiendo datos, en febrero de 2008, el *Macroscopic Topology Project* dejó paso al proyecto *Archipelago Measurement Infrastructure (Ark)*. Este nuevo proyecto usa una evolución de las herramientas usadas por su antecesor de modo que consigue reducir el esfuerzo destinado a desarrollar y desplegar las sofisticadas medidas a gran escala, además de proveer una infraestructura a la comunidad de medida que permite a los colaboradores realizar sus propias medidas a través de una plataforma distribuida segura.

*Ark* está diseñado específicamente para la medida en redes activas. Esto hace que *Ark* sea más simple que otras plataformas distribuidas de propósito

general lo cual nos permite concentrarnos en proveer las utilidades necesarias para ayudar a la investigación de la red. En particular, ofrece un servicio de comunicación y coordinación que hace que sea más fácil desarrollar medidas distribuidas que deben trabajar juntos para lograr una meta.

***Nuestra topología***

A partir de los datos de *Ark*, y con ayuda de la herramienta "AnalizadorXML_CAIDA" hemos generado una topología base eligiendo los datos de los países que más tráfico Kademlia generan en el mundo según [SENB07c]. Los países elegidos han sido China, Francia, Italia, Dinamarca, Brasil, Alemania, Estados Unidos, Taiwán, Corea, Argentina, Portugal, Reino Unido, Japón, India, Tailandia, Méjico, Suiza, Sudáfrica, Rusia y Australia. El archivo generado usando solo los 5 primeros países tendría la siguiente forma:

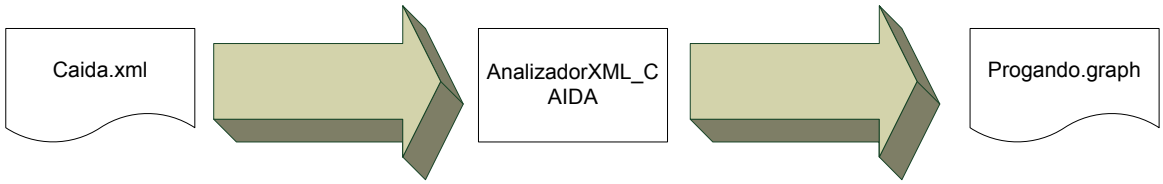


**Figura 39:** Diagrama de creación de una topología

```
<?xml version="1.0" encoding="UTF-8"?>
<topology>
  <vertices>
    <vertex dbl_y="1" dbl_x="1" int_idx="0" role="gateway" />
    <vertex dbl_y="2" dbl_x="2" int_idx="1" role="gateway" />
    <vertex dbl_y="3" dbl_x="3" int_idx="2" role="gateway" />
    <vertex dbl_y="4" dbl_x="4" int_idx="3" role="gateway" />
    <vertex dbl_y="5" dbl_x="5" int_idx="4" role="gateway" />
  </vertices>
  <specs xmloutbug="workaround">
    <client-stub int_delayms="1" dbl_plr="0" int_qlen="100" dbl_kbps="1000" />
    <stub-transit int_delayms="1" dbl_plr="0" int_qlen="100" dbl_kbps="100000" />
    <pais0 int_delayms="1" dbl_plr="0" int_qlen="100" dbl_kbps="100000" />
    <pais1 int_delayms="5" dbl_plr="0" int_qlen="100" dbl_kbps="100000" />
    <pais2 int_delayms="4" dbl_plr="0" int_qlen="100" dbl_kbps="100000" />
    <pais3 int_delayms="5" dbl_plr="0" int_qlen="100" dbl_kbps="100000" />
    <pais4 int_delayms="17" dbl_plr="0" int_qlen="100" dbl_kbps="100000" />
  </specs>
  <edges>
    <edge specs="stub-transit" int_idx="22" int_src="13" int_dst="5" int_delayms="104" />
```

```
    <edge specs="stub-transit" int_idx="23" int_src="5" int_dst="13" int_delayms="104" />
    <edge specs="stub-transit" int_idx="24" int_src="13" int_dst="8" int_delayms="0" />
    <edge specs="stub-transit" int_idx="25" int_src="8" int_dst="13" int_delayms="0" />
    <edge specs="stub-transit" int_idx="26" int_src="13" int_dst="1" int_delayms="96" />
    <edge specs="stub-transit" int_idx="27" int_src="1" int_dst="13" int_delayms="96" />
    <edge specs="stub-transit" int_idx="28" int_src="13" int_dst="17" int_delayms="307" />
    <edge specs="stub-transit" int_idx="29" int_src="17" int_dst="13" int_delayms="307" />
    <edge specs="stub-transit" int_idx="30" int_src="13" int_dst="4" int_delayms="183" />
    <edge specs="stub-transit" int_idx="31" int_src="4" int_dst="13" int_delayms="183" />
    <edge specs="stub-transit" int_idx="32" int_src="13" int_dst="19" int_delayms="231" />
    <edge specs="stub-transit" int_idx="33" int_src="19" int_dst="13" int_delayms="231" />
    <edge specs="stub-transit" int_idx="44" int_src="13" int_dst="3" int_delayms="107" />
    <edge specs="stub-transit" int_idx="45" int_src="3" int_dst="13" int_delayms="107" />
    <edge specs="stub-transit" int_idx="46" int_src="13" int_dst="9" int_delayms="281" />
    <edge specs="stub-transit" int_idx="47" int_src="9" int_dst="13" int_delayms="281" />
    <edge specs="stub-transit" int_idx="48" int_src="13" int_dst="7" int_delayms="190" />
    <edge specs="stub-transit" int_idx="49" int_src="7" int_dst="13" int_delayms="190" />
    <edge specs="stub-transit" int_idx="50" int_src="13" int_dst="0" int_delayms="212" />
    <edge specs="stub-transit" int_idx="51" int_src="0" int_dst="13" int_delayms="212" />
    <edge specs="stub-transit" int_idx="52" int_src="13" int_dst="14" int_delayms="132" />
    <edge specs="stub-transit" int_idx="53" int_src="14" int_dst="13" int_delayms="132" />
    <edge specs="stub-transit" int_idx="54" int_src="13" int_dst="12" int_delayms="185" />
    <edge specs="stub-transit" int_idx="55" int_src="12" int_dst="13" int_delayms="185" />
    <edge specs="stub-transit" int_idx="56" int_src="7" int_dst="8" int_delayms="43" />
    <edge specs="stub-transit" int_idx="57" int_src="8" int_dst="7" int_delayms="43" />
  </edges>
</topology>
```

## *4.3 Resultados*

Una vez acabadas las pruebas analizamos nuestros resultados ayudándonos para ello de Matlab. Para ello hemos creado un script que recorre el archivo creado por la herramienta “AnalizaResultados” calculando para cada número de dominios y cada probabilidad la media y el intervalo de confianza de los datos obtenidos, y dibujando gráficas a partir de ellos.

En nuestra prueba final hemos trabajado con 1000 nodos, de los cuales, una media de 800 estarán activos en el experimento en cada momento. Este número de nodos es pequeño en comparación con las decenas o centenas de

miles que participan en una red real, aunque son suficientes para obtener unos resultados validos.

Para mejorar la precisión, cada escenario se ha repetido diez veces. Con estas medidas, obtenemos una desviación típica muy baja y un intervalo de confianza al 95% pequeño. Finalizado el proceso nos encontramos con 2 conjuntos de gráficas, una para cada topología.

### 4.3.1 Resultados REDIMadrid

Entre los resultados recuperados de nuestro experimento el que podemos considerar más importante es el número de saltos necesarios para encontrar un peer.

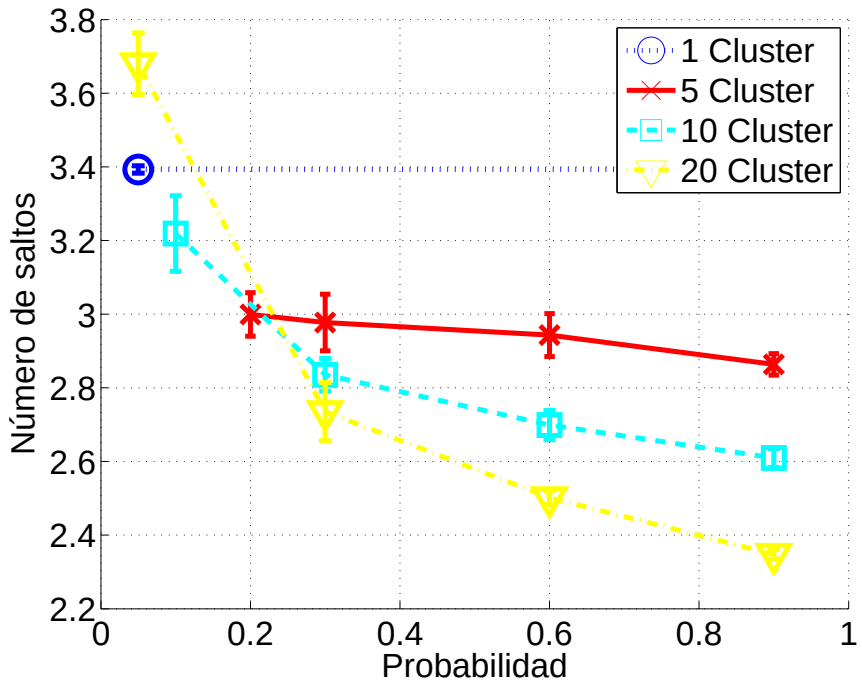


**Figura 40:** Número medio de saltos en REDIMadrid(1)

Como podemos ver en la gráfica, a medida que aumenta la probabilidad de buscar por un peer de nuestro propio dominio, más clara está la ventaja de tener varios dominios. Este resultado es coherente, ya que a medida que aumentamos el número de dominios, cada vez se hacen más pequeñas cada una de las subredes en el nivel de peers normales. De este modo, para altas probabilidades de buscar en tu propio dominio, la mayoría de las búsquedas se hacen en la pequeña red de nuestro cluster, mientras que para probabilidades

más bajas se usa muchas veces la red de interconexión que es mayor cuanto mayor sea el número de dominios.

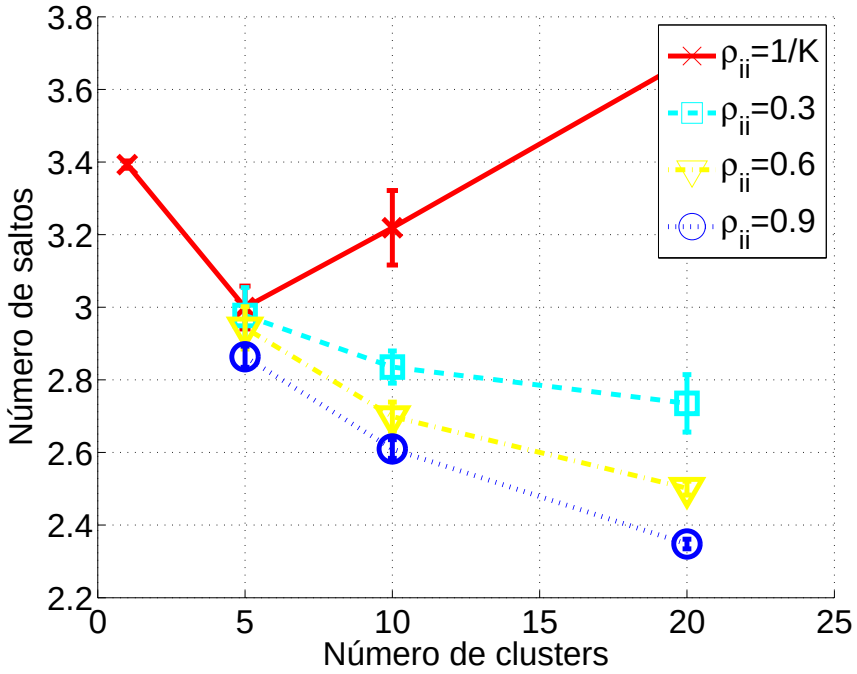


**Figura 41:** Número medio de saltos en REDIMadrid(2)

Si observamos la gráfica anterior poniendo como eje el número de dominios en lugar de la probabilidad de buscar en nuestro propio dominio podemos observar de una manera más clara la tendencia a la disminución del número de saltos cuando aumentamos el número de dominios. El caso en el que la probabilidad es la inversa del número de dominios podemos ver que esta tendencia se invierte precisamente por esta dependencia de la probabilidad con el valor del eje de abcisas.

En cuanto al número máximo de saltos la cosa es bastante diferente.

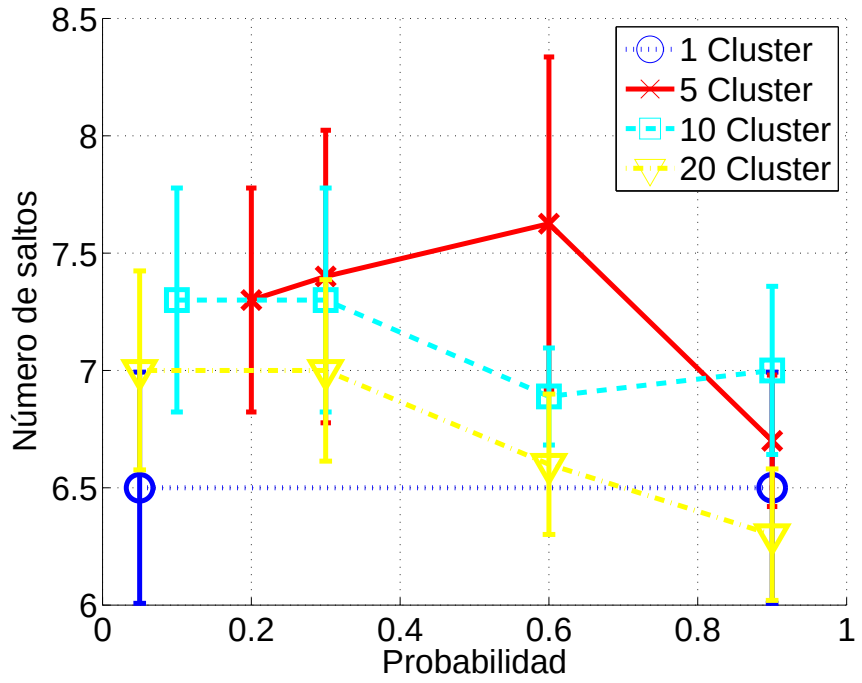


**Figura 42:** Número máximo de saltos en REDIMadrid

En este caso, podemos ver que el número máximo de saltos no tiene una relación tan clara como en el caso anterior con en el caso anterior. Sin embargo, podemos ver como para mayores probabilidades de buscar en el propio dominio el número de saltos disminuye. Esta tendencia tiene lógica si pensamos que el número máximo de saltos debe darse cuando se busque a un peer de otro dominio. En este caso el número de saltos será la suma de los saltos necesarios para buscar la información en el cluster de destino, más los saltos necesarios para encontrar el dominio en la red de superpeers, más el salto necesario para llegar al responsable de nuestro dominio.

En el caso del tiempo medio necesario para realizar una búsqueda, tenemos una relación similar a la del número medio de saltos.

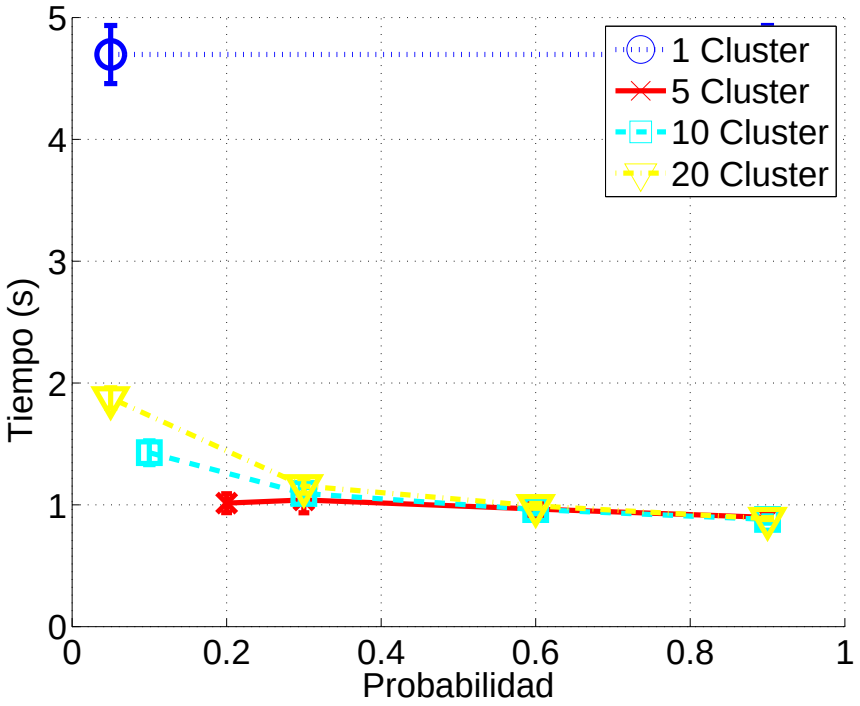


**Figura 43:** Tiempo medio en REDIMadrid

Seguimos observando la tendencia anterior, en la que a medida que crece el número de dominios y la probabilidad desciende el tiempo empleado en realizar una búsqueda. En este caso vemos que el tiempo empleado para un solo cluster es mucho mayor, en relación al número de saltos. Esto es debido a que en esta prueba el sistema tenía una carga mayor, lo que aumenta el tiempo de cada salto. Esto se puede apreciar en la siguiente gráfica.

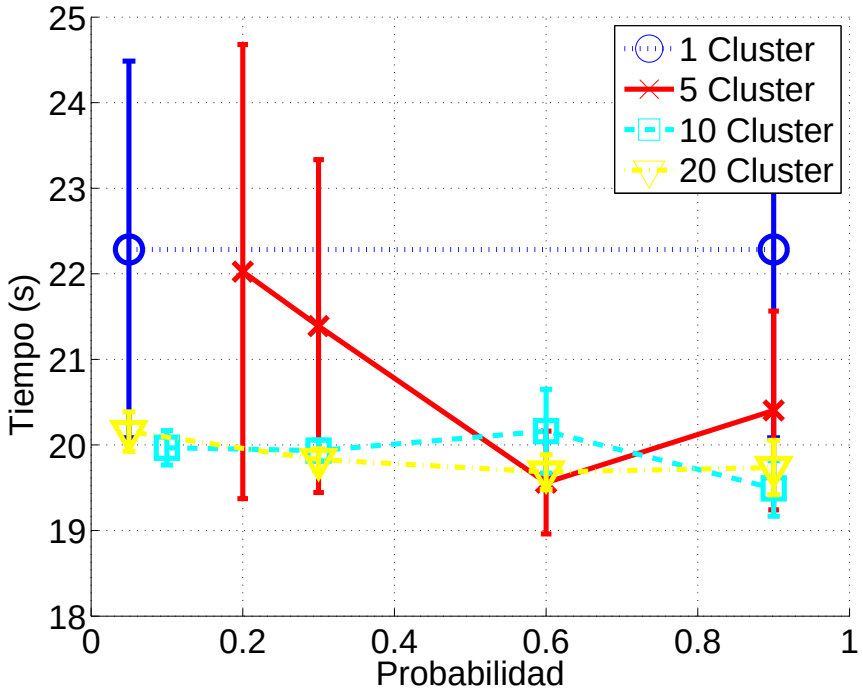


**Figura 44:** Tiempo máximo en REDIMadrid

En la gráfica anterior vemos como en algunos casos el tiempo máximo es mayor de 20 segundos. Esto solo se produce cuando la carga del sistema es muy grande, ya que el tiempo máximo de espera para una query está marcado en 20 segundos en nuestra implementación del protocolo P2PP. Vemos como en el caso de un solo cluster el tiempo es bastante mayor de 20 segundos, lo que indica la gran carga de esa prueba.

Además del tiempo y el número de saltos es importante fijarse también en cuanto esfuerzo le estamos requiriendo al sistema para poder manejar nuestra red jerárquica. Una buena aproximación de la cantidad de memoria usada por el programa será la cantidad de entradas en la tabla de rutas que tiene cada peers en cada momento.

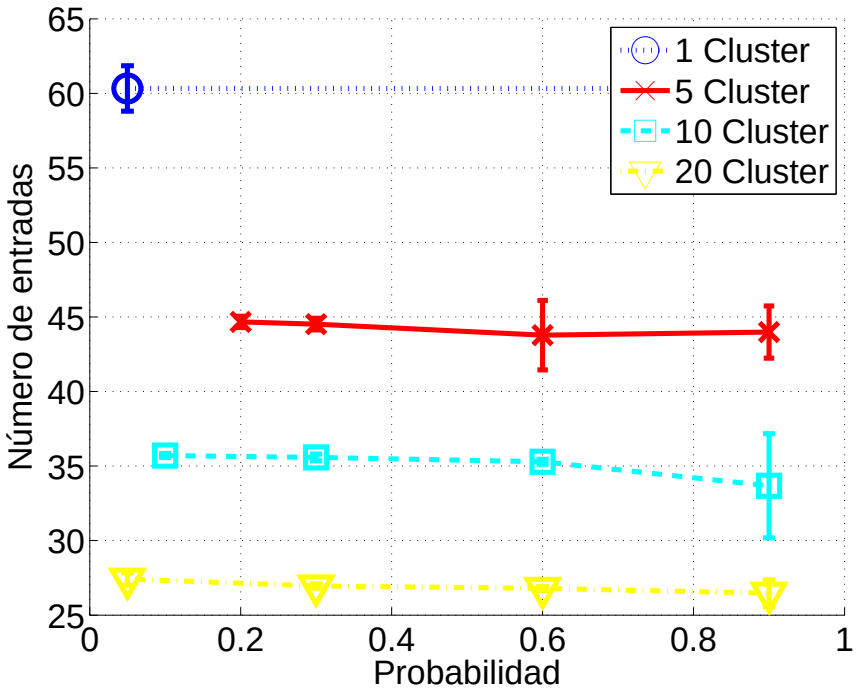


**Figura 45:** Número medio de entradas en REDIMadrid

Podemos ver que el número medio de entradas que tiene un peer en la tabla de rutas disminuye según aumenta el número de dominios. Es lógico pensar que si los peers participan en una red con menos nodos, necesitarán menos entradas en la tabla de rutas.

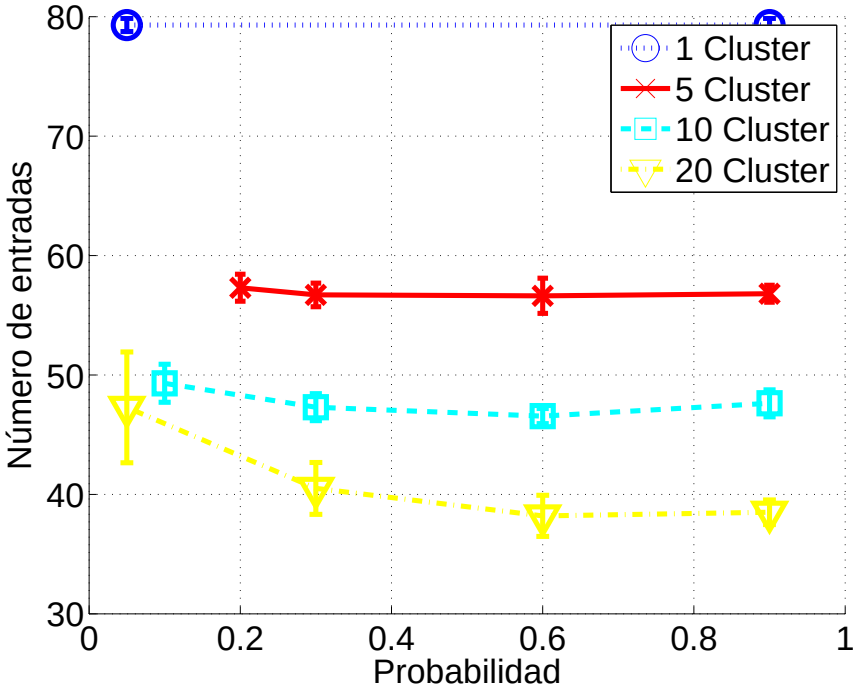


**Figura 46:** Número máximo de entradas en REDIMadrid

El número máximo de entradas en la tabla de rutas también disminuye al aumentar el número de clusters tal como pasaba con el número medio.

Sin embargo, aunque el número de entradas descienda en general es lógico pensar que en los superpeers sucederá lo contrario al tener que participar en dos redes diferentes.

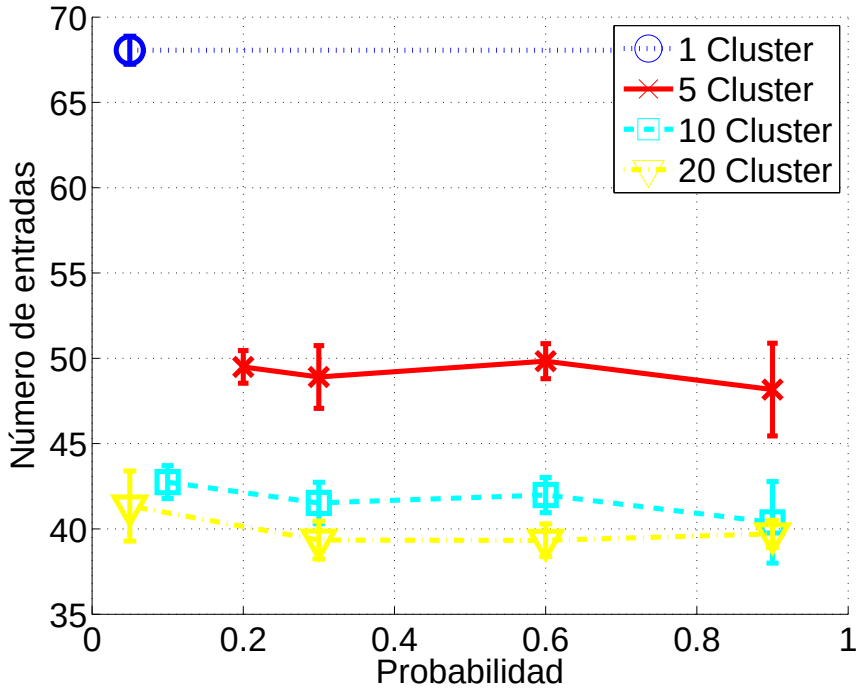


**Figura 47:** Número medio de entradas en los superpeers, REDIMadrid

Vemos en la figura anterior, que como era de esperar, el número medio de entradas en las tablas de rutas de los superpeers es en general superior al de los peers normales. Aun así el número de entradas decrece a medida que aumentamos el número de dominios.

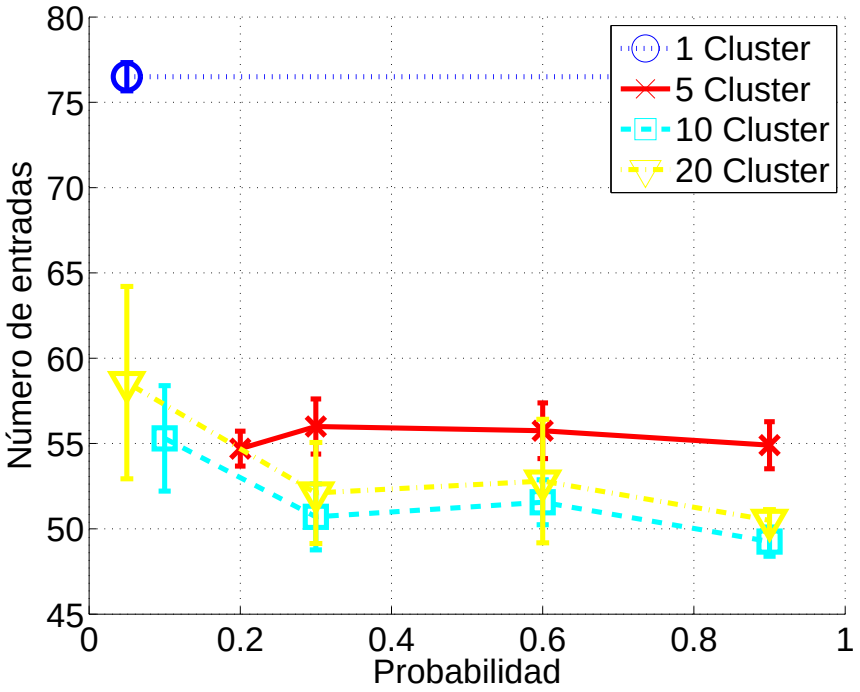


**Figura 48:** Número máximo de entradas en los superpeers, REDIMadrid

El número máximo de entradas también disminuye al aumentar el número de dominios, aunque en este caso los valores para 10 y 20 cluster son muy similares, estando el número de entradas para 20 dominios por encima del de 10, esto no es extraño, ya que el número máximo de entradas en la tabla de rutas tiene una gran variación entre los diferentes experimentos.

### *4.3.2 Resultados CAIDA*

La principal diferencia entre el escenario creado con los datos de REDIMadrid y los de CAIDA, es que además de la limitación en ancho de banda de los enlaces, en el caso de la nueva topología también tenemos retardos en los enlaces, con lo que el tiempo que se tarde en realizar una búsqueda debe ser mayor que en caso anterior, mientras que el resto de medidas no debe cambiar significativamente.

Como en el caso anterior, la medida que mejor define el rendimiento de la red es el número medio de saltos.

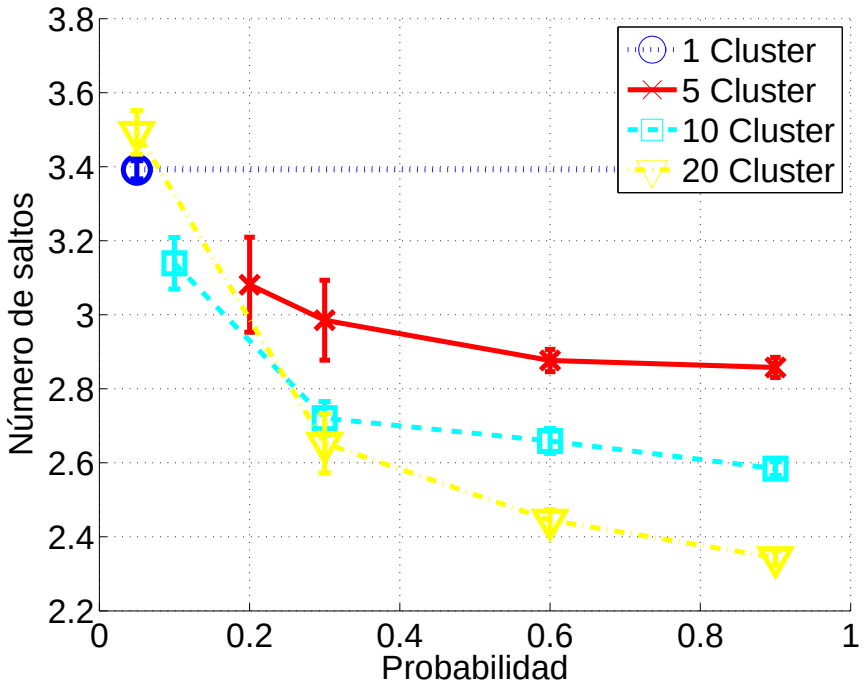


**Figura 49:** Número medio de saltos en CAIDA (1)

En la gráfica anterior podemos observar una gran similitud con el caso de REDIMadrid. Al igual que en el caso anterior, en este escenario se observa como a medida que crece el número de saltos y la probabilidad de buscar en el mismo dominio, va disminuyendo el número de saltos.

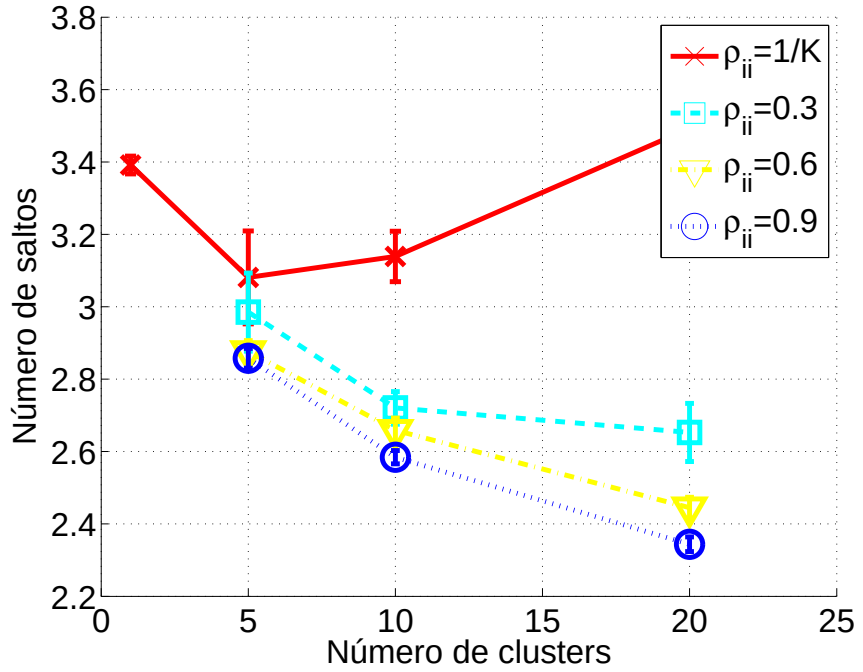


**Figura 50:** Número medio de saltos en CAIDA (2)

Como en el caso de REDIMadrid, observamos mejor la tendencia a disminuir el número de saltos a medida que crece el número de dominios colocando este valor en el eje de abscisas.

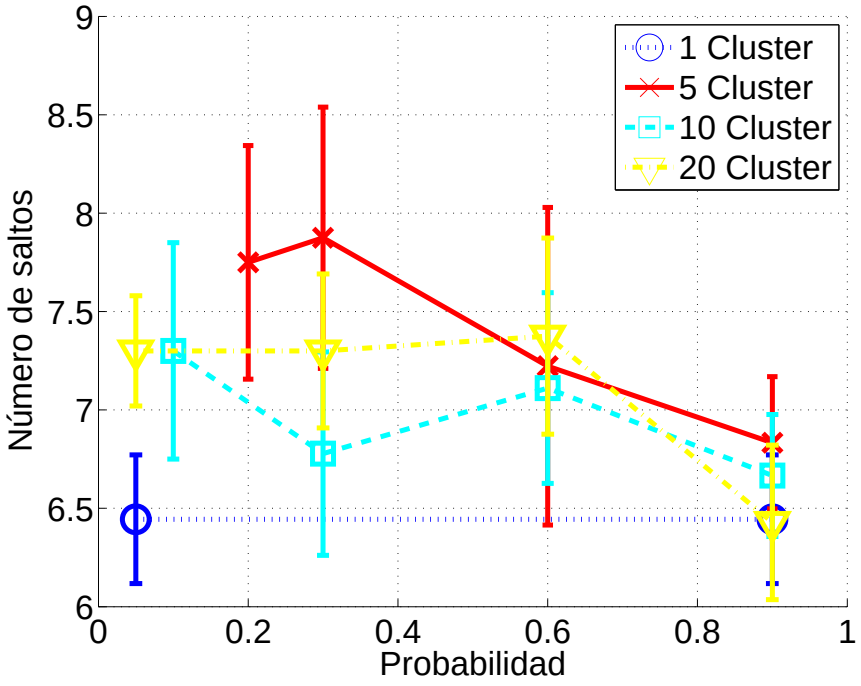


**Figura 51:** Número máximo de saltos en CAIDA

De nuevo, en este escenario se confirma que no hay una relación clara entre el número de dominios y el número máximo de saltos. Sin embargo, sigue habiendo una tendencia hacia la disminución de los saltos máximos cuando aumentamos la probabilidad de buscar en nuestro propio dominio.

Visto que el número de saltos no cambia significativamente con el cambio de la topología, vamos a ver a continuación que es lo que sucede con el tiempo empleado en realizar una búsqueda.

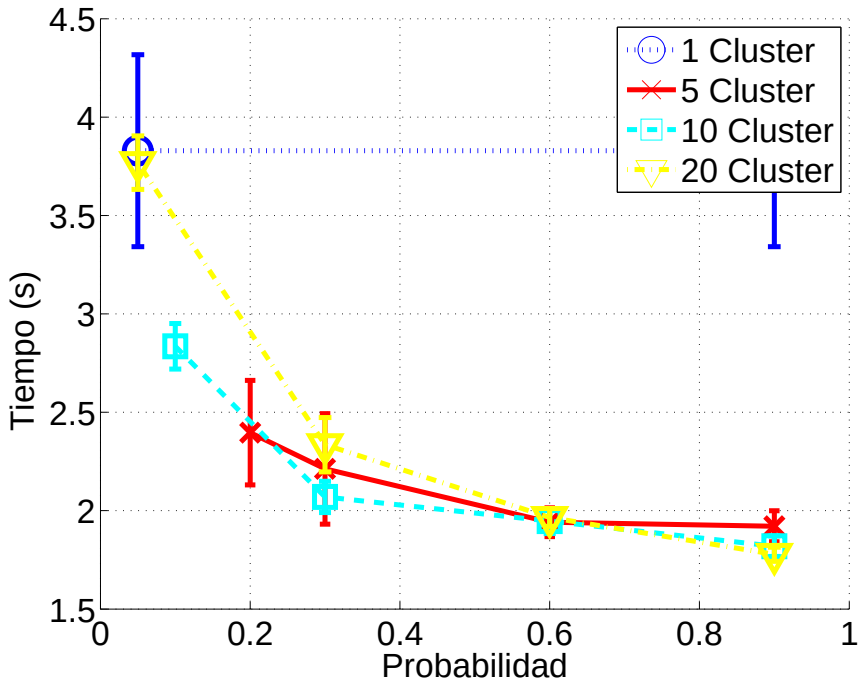


**Figura 52:** Tiempo medio en CAIDA

Como era de esperar, observamos que el tiempo tardado en realizar una búsqueda ha aumentado significativamente con respecto al escenario anterior. En este caso, observamos que el tiempo que se tarda en realizar una búsqueda para probabilidades bajas es mayor para un mayor número de dominios, pero esta tendencia se invierte a medida que crece la probabilidad de buscar en un propio dominio. Podemos suponer que la poca diferencia en los tiempos se debe a que trabajamos con un número pequeño de nodos y que cuando este número aumente los tiempos estarán más distanciados.

Al igual que en el caso anterior podemos tomar el tiempo máximo como una medida de la carga que sufrió el sistema durante las pruebas.

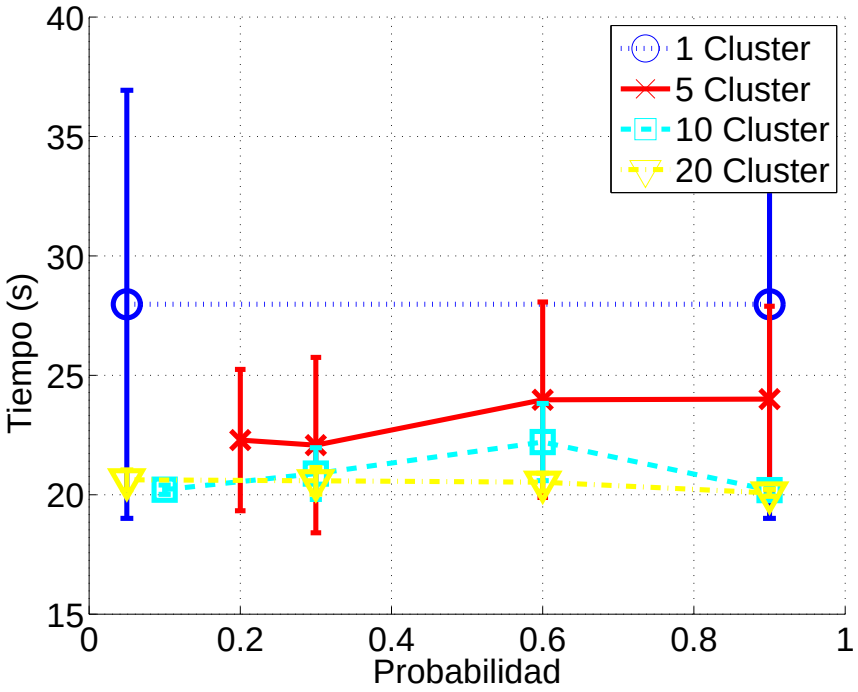


**Figura 53:** Tiempo máximo en CAIDA

Al igual que en el caso anterior se observa que la carga ha sido mayor en las pruebas con un menor número de dominios. Esto nos da una idea de la mejora en el rendimiento que nos supone la inclusión de la jerarquía en la red.

Tras el análisis de los datos relacionados con las querys, pasamos a fijarnos en las tablas de rutas de los nodos para comprobar si las medidas obtenidas son parecidas al escenario de REDIMadrid.

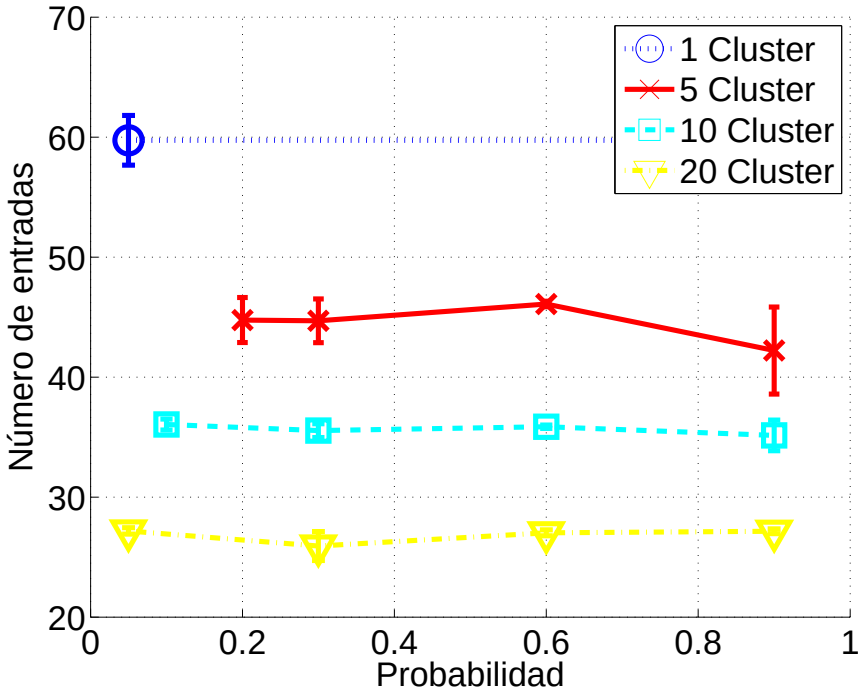


**Figura 54:** Número medio de entradas en CAIDA

En este caso, el número medio de entradas es parecido al del escenario de REDIMadrid. De este modo, podemos ver que el número medio de entradas disminuye a medida que crece el número de nodos.

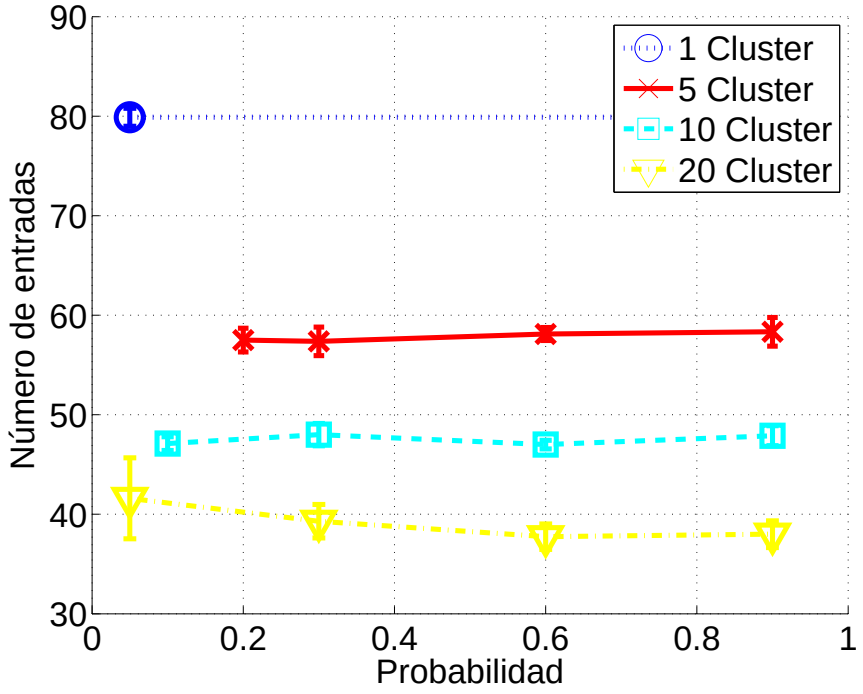


**Figura 55:** Número máximo de entradas en CAIDA

De nuevo observamos que esta gráfica es prácticamente igual que en el caso del escenario del REDIMadrid. El número máximo de entradas en los superpeers desciende de forma muy clara a medida que aumentamos el número de clusters.

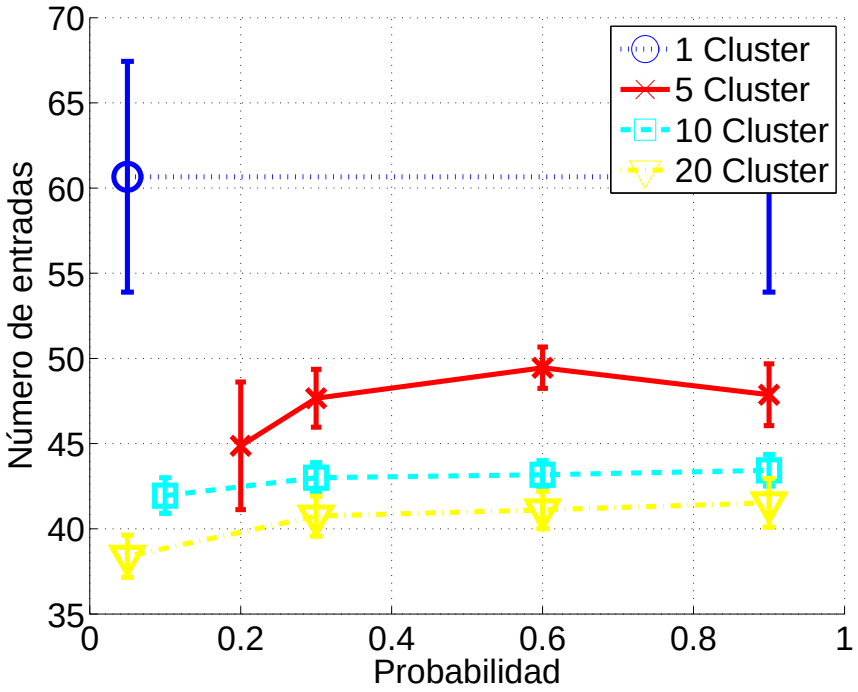


**Figura 56:** Número medio de entradas en los superpeers, CAIDA

En los superpeers vemos claramente que el número de entradas disminuye a medida que aumenta el número de clusters, aunque para los casos de 10 y 20 dominios ya no está tan claro.

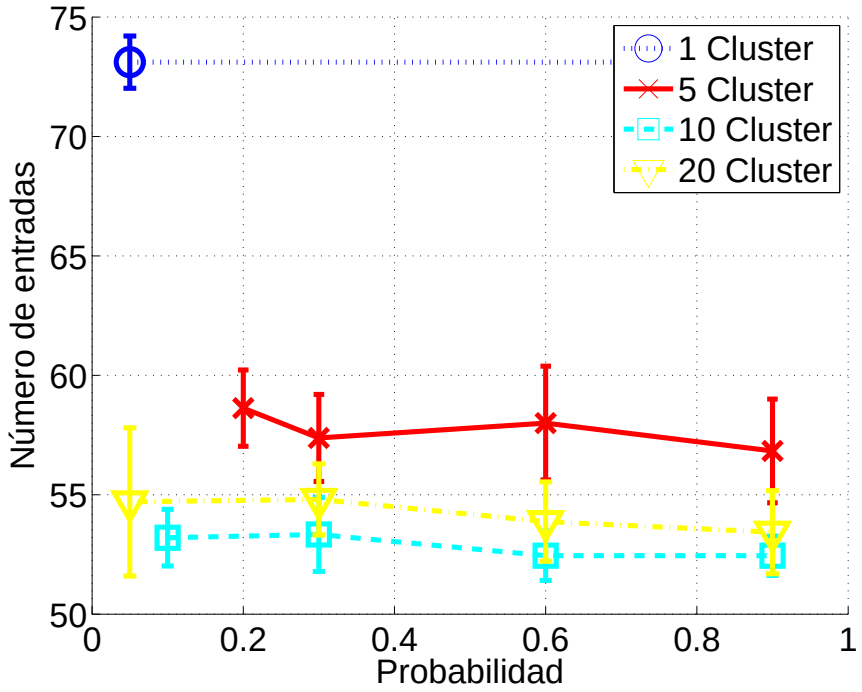


**Figura 57:** Número máximo de entradas en los superpeers, CAIDA

En este caso se observa claramente como hay un número máximo de entradas mayor para 20 cluster que para 10.Esto se debe a que para el número de nodos que manejamos el tener 20 dominios puede ser demasiado.

### *4.3.3 Comparación con las simulaciones*

Tras recuperar y analizar los resultados, es importante compararlos con las simulaciones realizadas en [MYBG+08a] para saber si son parecidos. Para ello es importante comprender que las pruebas realizadas en las simulaciones eran diferentes, en su base, a las realizadas en este proyecto, y que estas simulaciones trataban de encontrar una relación entre el número de saltos necesarios para realizar una búsqueda o de entradas en la tabla de rutas en función del número de nodos participantes en la prueba, mientras que nosotros solo tenemos datos para el caso en el que hay 800 nodos y nuestra relación se establece en función del número de dominios o de la probabilidad de buscar en el propio dominio.

No obstante, podemos ver que la tendencia a reducirse el número de saltos a medida que aumentamos el número de dominios o la probabilidad de

buscar en un mismo dominio también está presente en las simulaciones, y que en ellas esta tendencia de incrementa según crece el número de nodos.

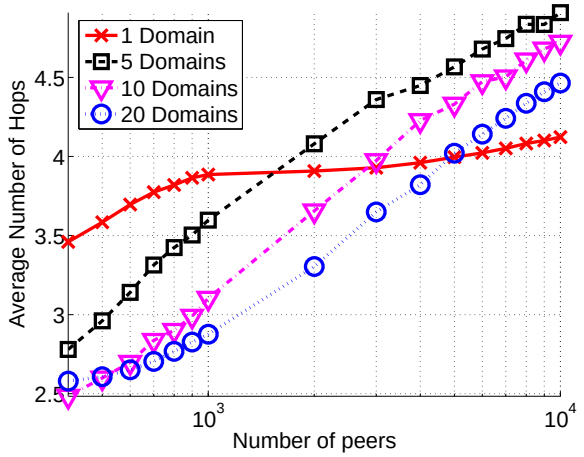


**Figura 58:** Número medio de saltos en la simulación para diferentes números de dominios

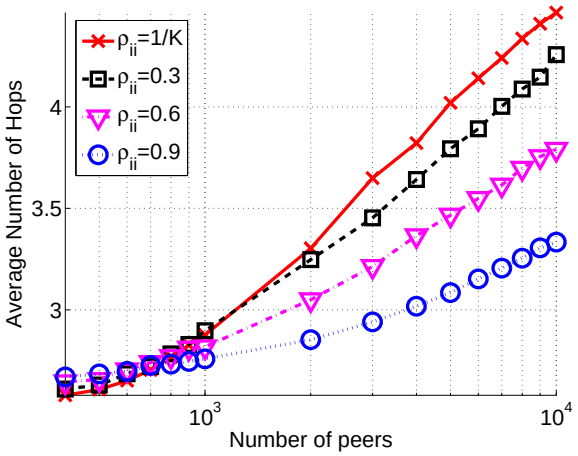


**Figura 59:** Número medio de saltos en la simulación con 20 dominios

En las simulaciones también se analiza el número de entradas en las tablas de rutas que tienen los diferentes nodos.

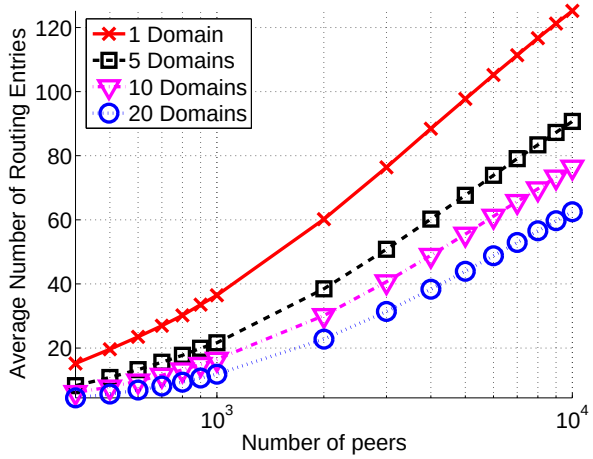


**Figura 60:** Número medio de entradas en la tabla de rutas en la simulación para diferentes números de dominios

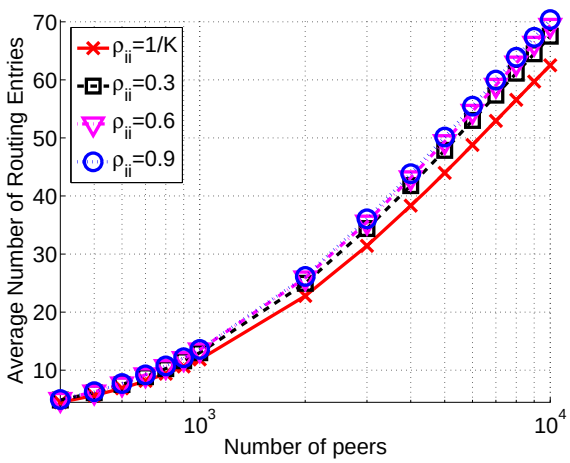


**Figura 61:** Número medio de entradas en la tabla de rutas en la simulación con 20 dominios

De igual manera que en nuestros resultados, las simulaciones nos mostraban que el número de entradas en la tabla de rutas disminuye al aumentar el número de dominios, mientras que la probabilidad de buscar en un mismo dominio apenas influye, y mucho menos para un bajo número de nodos.

# CAPÍTULO V

# CONCLUSIONES

# 5. CONCLUSIONES

Los resultados obtenidos nos permiten concluir que la inclusión de una jerarquía en una red overlay mejora de manera significativa el rendimiento de estas redes, tanto en la facilidad para encontrar la información buscada, como en los recursos consumidos en estas búsquedas.

Parece claro que tener un conjunto de redes pequeñas conectadas entre sí disminuye el número de saltos necesarios para encontrar la información en la overlay. Además de esta importante mejora, también hemos visto que los nodos, en general, necesitan guardar una menor cantidad de información para formar parte de la red lo que conlleva una menor necesidad de mensajes para el mantenimiento de la red.

Este tipo de redes incremente la carga de algunos terminales de la overlay que se tienen que encargar de la comunicación entre dominios, sin embargo, este sobreesfuerzo no es excesivamente grande y es compensado por la mejora de rendimiento del resto de los terminales.

Los resultados obtenidos para los dos escenarios probados nos han dado unos resultados muy similares en cuanto a número de saltos y entradas lo que nos indica que el rendimiento del protocolo en ese sentido es independiente de la topología de red sobre la que se esté ejecutando. No obstante, el tiempo empleado en llevar a cabo una búsqueda es diferente para ambos escenarios, por lo que podemos pensar que una distribución diferente de los nodos en la topología podría mejorar aún más estos resultados.

También debemos destacar que los protocolos jerárquicos no son adecuados para cualquier aplicación de nivel superior. Sin embargo, muchos tipos de aplicaciones, como pueden ser las de VoIP, cuya popularidad está creciendo mucho últimamente, encajan a la perfección sobre este tipo de

protocolos al ser muy importante en ellas la proximidad geográfica de los nodos participantes en la red.

Además de las conclusiones relacionadas con el protocolo usado y la implementación realizada, es importante destacar lo aprendido de las herramientas usadas para obtener estos resultados.

Para nuestro trabajo ha sido fundamental el uso de un emulador de red. En nuestro caso hemos usado ModelNet. Tras multitud de pruebas con este emulador hemos descubierto que no es una solución muy escalable, y que el comportamiento del mismo se degrada a medida que se aumenta el número de nodos emulados sin que exista una posibilidad real de aumentar el número de nodos aumentando los recursos disponibles.

Por otro lado, hemos comprobado la importancia de los sistemas de virtualización para poder aprovechar adecuadamente los recursos disponibles. Aunque, por otro lado, también hemos visto que esta virtualización no está exenta de problemas.

En el ámbito personal este proyecto ha supuesto una experiencia enriquecedora profesionalmente ya que me ha acercado al mundo de la investigación y sus maneras de trabajar. A lo largo de este trabajo he tenido que realizar multitud de tareas, aprendiendo de ese modo cosas tan diversas como instalar un cluster de máquinas virtuales, analizar y modificar un código creado por otras personas o a emular una red con más de mil nodos con apenas cinco ordenadores.

En cuanto al objetivo del proyecto, podemos darlo por alcanzado completamente al haber terminado el ciclo planeado con la modificación de la implementación, las pruebas en la red emulada y el análisis de los resultados.

A través de la realización de este trabajo he podido aprender la dificultad de analizar y modificar un código realizado por otra persona, y la importancia de la documentación para facilitar la tarea a la gente que quiera continuar nuestro trabajo.

## *5.1 Líneas futuras de trabajo*

Durante el desarrollo de este trabajo han ido surgiendo ideas y mejoras que se pueden plantear como posibles trabajos futuros, como continuación de la línea de este proyecto. A continuación se identifican algunos de ellos:

- Realizar pruebas con **peers localizados**. Parece obvio pensar, que ya que en multitud de aplicaciones, como pueden ser las llamadas de VoIP, es mucho más probable entablar una sesión con alguien geográficamente próximo, el siguiente paso será realizar pruebas en unos escenarios parecidos a los elegidos para este proyecto pero en los que los dominios se definan por un criterio de proximidad en el gráfico de red.
- Elegir los **superpeers dinámicamente**. En la actualidad los superpeers son fijos, y si uno de ellos se cae o sufre algún problema dejaría aislada a su overlay. Además, todos los nodos pertenecientes a un dominio deben conocer a su superpeer de antemano, lo que impide que este pueda cambiar entre sesiones. En este marco, parece un objetivo clave, el diseñar un algoritmo que permita elegir que nodo actuará como superpeer en función de su ancho de banda y capacidad de computo. Además, se debe diseñar el modo en el que el resto de peers de la overlay se enterarán de los cambios de superpeers y un método de recuperación en caso de que el superpeer que hay en cada momento se caiga. Una vez realizado este diseño teórico se debería modificar la implementación actual para probar su funcionamiento.
- Implementar **diferentes protocolos**. Pese a que en nuestras pruebas solo necesitábamos el protocolo Kademlia, sería interesante añadir a la implementación la versión jerárquica de otros protocolos como Chord o Bamboo. Con esta modificación podríamos ver la manera en la que interactúan estos protocolos y comprobar de una manera real la independencia entre los diferentes dominios.
- Realizar una **prueba en un entorno real**. Para finalizar, estaría bien el poder probar la aplicación en un entorno real como PlanetLab. Como se vio en las

simulaciones, el rendimiento del protocolo jerárquico mejora con respecto al plano a medida que aumenta el número de nodos participantes en la red por lo que sería interesante poder probar entornos mayores.

# APÉNDICES

## A. Tareas y memoria económica del proyecto

### A.1. Tareas

- Recuperación de los requisitos. (1 semana).
- Recopilación del estado del arte y evaluación de las diferentes implementaciones. (3 semanas).
- Análisis del código original. (3 semanas).
- Modificación de los peer normales para adaptarlos a la red jerárquica. (4 semanas).
- Modificación de los superpeers para adaptarlos a la red jerárquica. (6 semanas).
- Modificación del código para guardar la información deseada. (1 semana).
- Realización de las pruebas y recuperación de resultados. (4 semanas).

Tareas paralelas:

- Instalación del software en los equipos. (2 semanas).
- Instalación y verificación de ModelNet. (4 semanas).
- Pruebas de esfuerzo de ModelNet. (2 semanas).
- Creación de las herramientas de apoyo. (2 semanas).
- Documentación. (2 meses)

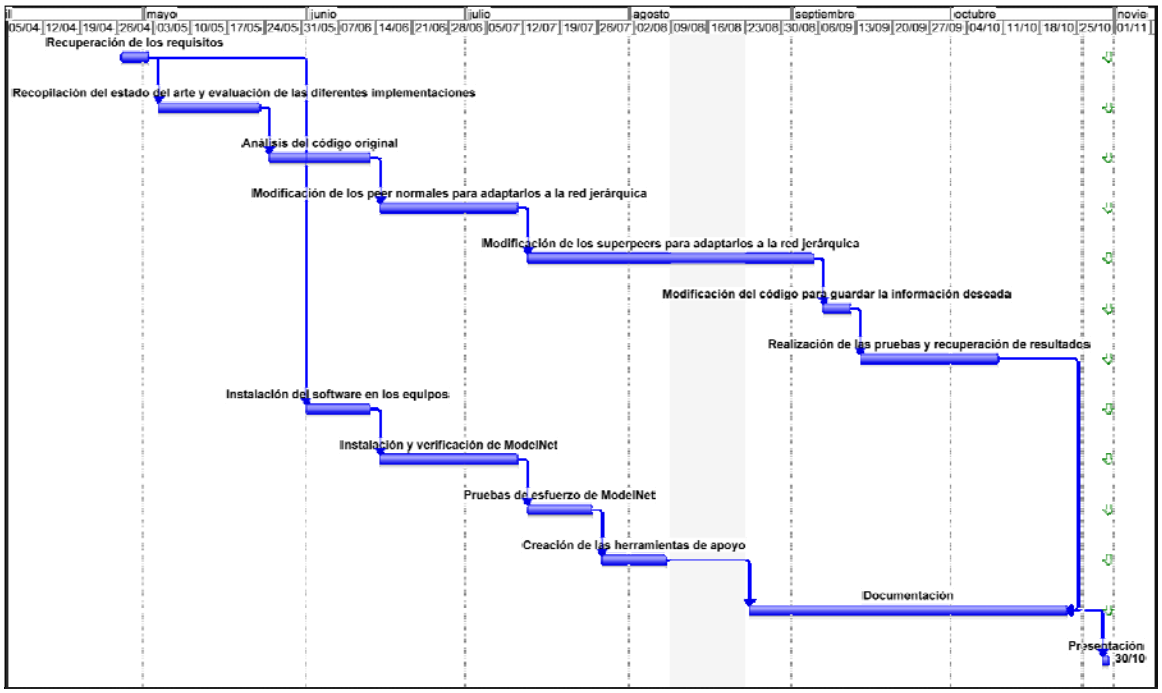


**Figura 62:** Diagrama de Gantt

## *A.2. Memoria económica*

| Concepto | Número | Precio/Unidad | Total |
|---|---|---|---|
| Host A | 4 | 2000 € | 8000 € |
| Host B | 2 | 1000 € | 2000 € |
| Pantalla LCD | 3 | 200 € | 600 € |
| Switch | 1 | 200 € | 200 € |
| KVM (4 puertos) | 1 | 100 € | 100 € |
| IVA | | 16% | 12644 € |
| Salario de ingeniero | 22x20=440 horas | 72 € | 31680 € |
| **Total** | | | **44324 €** |

**Figura 63:** Memoria económica del proyecto

# B. ModelNet

## Introducción

ModelNet [MODE09] es un emulador de red de gran escala que nos permite evaluar sistemas distribuidos en un entorno realista similar a Internet. ModelNet permite probar prototipos sin necesidad de modificarlos sobre sistemas operativos normales en diferentes escenarios de red. Podemos decir, que combina la repetitividad de la simulación con el realismo de un desarrollo real. ModelNet ha sido desarrollado para ayudar en el diseño y prueba de las nuevas redes de distribución de contenido, sistemas P2P, protocolos de la capa de transporte, procesadores de flujos distribuidos, sistemas de archivos distribuidos y herramientas de medición de red.

Una vez desplegado ModelNet cada instancia de nuestra aplicación corre en un nodo virtual; ModelNet multiplexa los nodos virtuales entre el conjunto de nuestras máquinas física a las que llamaremos “edge nodes”. El sistema configura los edge nodes para enrutar sus paquetes a través del core de ModelNet (compuesto por uno o más máquinas físicas). Este core adapta cada paquete a su ancho de banda, tiempo de transito, perdidas… en la topología elegida. ModelNet soporta la emulación hop-by-hop, capturando los efectos del trafico cruzado y la congestión en la red.

Usar ModelNet es relativamente fácil y requiere los siguientes pasos:

- Especificación de la topología. Esta topología puede ser cualquiera, desde una red full-mesh hasta una topología real completa.
- Vinculación. ModelNet mapea los nodos virtuales en los edge nodes, dándole a cada uno una IP en el rango 10.0.X.X. En esta fase también se construye la “tabla de rutas” que contiene el conjunto de túneles a atravesar para cada par origen-destino.
- Despliegue. Por último debemos desplegar los archivos generados en los pasos anteriores a todas las máquinas del cluster. ModelNet

necesita un mínimo de 2 máquinas para funcionar, pero lo puede hacer con más de 100.

## ***Instalación***

La instalación de ModelNet se basa en dos pasos claramente diferenciados: la instalación de los core nodes y la instalación de los edge nodes, completamente diferentes entre sí.

### Instalación de los edge nodes

Para la instalación de los edge node en nuestro sistema necesitamos tener instalada previamente una distribución de Linux. Nosotros hemos elegido una Debian Lenny. Una vez instalado el sistema operativo solo debemos ejecutar el siguiente script:

```
#!/bin/bash
apt-get update
#apt-get install linux-headers-2.6.18-6-686
apt-get install openssh-server
apt-get install libxml-simple-perl
apt-get install libboost-graph-dev
apt-get install sudo
apt-get install libxerces27-dev
apt-get install make
apt-get install g++
apt-get install iperf
apt-get install rsync
#wget http://bittella.googlecode.com/files/modelnet-linux-0.99.tar.gz
wget http://bittella.googlecode.com/files/modelnet-0.99.tar.gz
wget http://bittella.googlecode.com/files/Graph-0.20105.tar.gz
wget http://bittella.googlecode.com/files/Heap-0.80.tar.gz
#wget http://bittella.googlecode.com/files/sstv_7_08_linux.patch
tar -xvzf Heap-0.80.tar.gz
cd Heap-0.80
perl Makefile.PL
make
make install
cd ..
rm -rf Heap-0.80
tar -xvzf Graph-0.20105.tar.gz
```

```
cd Graph-0.20105
perl Makefile.PL
make
make install
cd ..
rm -rf Graph-0.20105
tar -xvzf modelnet-0.99.tar.gz
cd modelnet-0.99
mkdir linux
cd linux
../configure
make
make install
cd ../../
```

Para ello podemos seguir los siguientes pasos:

1- Crear máquina en el vbmr1.
2- Cambiar /etc/dhc3p/dhclient.conf para añadir 'send host-name "hostname".
3- Ejecutar "ifconfig eth0 up".
4- ejecutar dhclient.
5- wget enjambre.it.uc3m.es/~rsanchez/ModelNet/configura.sh.
6- chmod 777 configura.sh.
7- ./configura.sh.

Dar a todo "Y".

8- Cambiar máquina a vbmr0 y reiniciar.

**Instalación de los core node**

La instalación de los core nodes es un poco más complicada que la de los edge nodes ya que en un primer paso debemos recompilar el kernel de un sistema operativo FreeBSD. Nosotros hemos elegido un FreeBSD 4.8 para nuestro sistema.

Para facilitar estos pasos hemos usado los siguientes scripts.

Intall_frebsd.sh

```
#!/bin/csh
setenv PACKAGESITE ftp://ftp-archive.freebsd.org/pub/FreeBSD-Archive/ports/i386/
packages-4.8-release/Latest/
pkg_add -r wget gmake bash
```

```
pkg_add -r perl sudo
pkg_add -r p5-XML-Simple linuxthreads
pkg_add -r libgnugetopt
pkg_add -r boost
pkg_add -r xerces-c2
```

Intall_frebsd2.sh

```
#!/bin/csh
wget http://bittella.googlecode.com/files/Graph-0.20105.tar.gz
wget http://bittella.googlecode.com/files/Heap-0.80.tar.gz
wget http://bittella.googlecode.com/files/modelnet-0.99.tar.gz
tar -xvzf Heap-0.80.tar.gz
cd Heap-0.80
perl Makefile.PL
gmake
gmake install
cd ..
rm -rf Heap-0.80
tar -xvzf Graph-0.20105.tar.gz
cd Graph-0.20105
perl Makefile.PL
gmake
gmake install
cd ..
rm -rf Graph-0.20105
tar -xvzf modelnet-0.99.tar.gz
cd modelnet-0.99
mkdir freebsd
cd freebsd
../configure --with-fbsdsys=/sys/
gmake
gmake install
```

Para ello podemos seguir los siguientes pasos:

1- Iniciar KVM(Qemu) con el Cd del FreeBSD 4.8 metido en vbmr1
2- Instalar para kern-development
3- Que sea Gateway y no compatible con los binarios de Linux.
4- Terminada la instalación reiniciar
5- recompilar kernel:

cd /sys/i386/conf/

ee GENERIC (añadir línea options HZ=10000)

config GENERIC

cd ../../compile/GENERIC

make (tarda un poco)

cp kernel /

6- reiniciar.
7- copiar archivos de instalación install e install2
8- chmod 777 instal*
9- ./install (2 veces por si acaso)
10- ./install2
11- ee /etc/rc.conf (cambiar nombre de dominio)
12- ee /etc/dhclient.conf (añadir send host-name "nombre";)
13- apagar y cambiar el vmbr1 por el vmbr0
14- encender
15- visudo

quitar el # de delante de root y wheel.

salir (Q salen los dos puntos y pones wq!)

16- ee /etc/inetd.conf (Quitar el comentario de login, finger y exec)
17- ee /etc/ssh/sshd_config

PermitRootLogin yes

RSAAuthentication yes

PubkeyAuthentication yes

AuthorizedKeysFile /root/.ssh/authorized_keys

18- reiniciar y debería estar

### ***Construcción de una topología***

Para ejecutar una red ModelNet hay que crear varios archivos XML:

- Gráfico (.graph) - las listas de los nodos y enlaces de la red virtual.
- Rutas (.route) - contiene la información para enrutar los paquetes a través de la red virtual.
- Máquinas (.hosts) - lista de las máquinas que actuarán como edge nodes y como core nodes.
- Modelo (.model) – lista de los nodos que se encuentran en cada máquina y de los enlaces entre ellos.

ModelNet para actuar requiere el archivo de rutas y el archivo del modelo. El archivo con el gráfico y el que nos indica las máquinas disponibles solo se usa

en la generación de los dos anteriores. ModelNet incluye herramientas en perl que pueden ser usadas para generar los archives requeridos a partir del .graph y el .hosts.

### ***Gráfico de la topología.***

El archivo .graph que debemos generar debe tener un formato similar al siguiente ejemplo:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
     <topology>
       <vertices>
         <vertex int_idx="0" role="gateway" />
         <vertex int_idx="1" role="gateway" />
         <vertex int_idx="2" role="virtnode" int_vn="0" />
         <vertex int_idx="3" role="virtnode" int_vn="1" />
         <vertex int_idx="4" role="virtnode" int_vn="2" />
       </vertices>
       <edges>
         <edge int_dst="1" int_src="2" int_idx="0" specs="client-stub" int_delayms="1" />
         <edge int_dst="2" int_src="1" int_idx="1" specs="client-stub" dbl_kbps="768" />
         <edge int_dst="1" int_src="3" int_idx="2" specs="client-stub" />
         <edge int_dst="3" int_src="1" int_idx="3" specs="client-stub" />
         <edge int_dst="0" int_src="4" int_idx="4" specs="client-stub" />
         <edge int_dst="4" int_src="0" int_idx="5" specs="client-stub" />
         <edge int_dst="1" dbl_len="1" int_src="0" int_idx="0" specs="stub-stub" />
         <edge int_dst="0" dbl_len="1" int_src="1" int_idx="1" specs="stub-stub" />
       </edges>
       <specs >
         <client-stub dbl_plr="0" dbl_kbps="64" int_delayms="100" int_qlen="10" />
         <stub-stub dbl_plr="0" dbl_kbps="1000" int_delayms="20" int_qlen="10" />
       </specs>
     </topology>
```

En el vemos como hay 3 partes claramente diferenciadas:

- Vertices. En esta sección debemos añadir todos los nodos virtuales que va a tener nuestra topología, tanto los nodos que formarán el núcleo de la red como los que actuaran como nodos virtuales.
- Edges. En esta sección definiremos los enlaces de nuestra topología.

- Specs. Por último definiremos las especificaciones para cada tipo de enlace. Si un enlace específico tiene información sobre su ancho de banda, retardo o pérdidas, esta información sobrescribirá a la que se encuentre en su tipo.

Para la generación de este archivo podemos usar herramientas como inet2xml o podemos generarlo nosotros a mano.

### ***Archivo con las máquinas disponibles***

El archivo .hosts que debemos generar tiene un formato similar al siguiente ejemplo:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<hardware>
        <emul hostname="larry"/>
        <host hostname="curly"/>
        <host hostname="moe"/>
</hardware>
```

Como vemos en este sencillo archivo debemos listar el nombre de las máquinas que vamos a usar. Las que definamos como “emul” serán usadas como core nodes y las que definamos como “host” serán las que hagan la labor de edge node.

### ***Generación del archivo de rutas y del modelo***

Para la generación del archivo de rutas y del modelo usaremos las herramientas que nos proporciona ModelNet.

La primera de ellas se encarga de generar la ruta más corta entre cada par de nodos a partir del gráfico de la red, para ejecutarla usaremos el siguiente comando:

```
allpairs archivo.graph > archivo.route
```

Esta herramienta genera un camino para cada par de nodos origen-destino. Esto quiere decir que para un experimento en el que usemos 1.000 nodos, este archivo tendrá 1.000.000 de entradas. En este punto se encuentra el mayor fallo de ModelNet, ya que al no haber agregación de rutas este archivo se

hará inmanejable para los core node cuando tratemos de emular redes medianamente grandes.

Por último, generaremos el archivo .model a partir del gráfico de la red y las máquinas disponibles mediante el siguiente comando:

```
mkmodel archivo.graph archivo.hosts > archivo.model
```

En este archivo se mapeará cada nodo virtual a una máquina que actuará como edge node y cada emulador a una de las máquinas que actúa como core node.

### ***Despliegue de la red***

Una vez generados los archivos con las rutas y el modelo es necesario enviarlos a todas las máquinas que formarán parte de la red ModelNet. Una vez enviados los archivos debemos ejecutar el siguiente comando en cada máquina.

```
deployhost archivo.model archivo.route
```

De esta manera se construyen las tablas de rutas en cada nodo y emulador con el fin de encaminar los paquetes de forma correcta por la red.

### ***Ejecución de nuestro programa***

Una vez que la topología de red se ha desplegado en el hardware de emulación, el sistema está activo y se puede probar una aplicación distribuida. Todos los paquetes que un edge node mande a la red 10.0.0.0/8 serán enviados hacia su emulador.

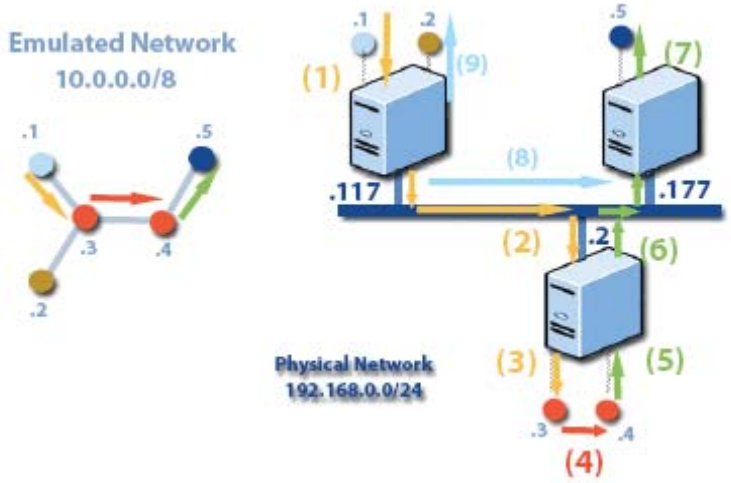


**Figura 64:** Red de ejemplo en ModelNet

Para poder diferenciar los paquetes que provienen de un emulador y de un nodo virtual ModelNet cambia el bit número 9 de la dirección IP al mandar un paquete en un nodo virtual. De modo que convierte las IP 10.0.X.X en 10.128.X.X. Una vez el paquete ha sido enrutado a través de los emuladores se vuelve a cambiar el noveno bit y se envía a su destino con la dirección IP original.

Para ejecutar nuestra aplicación en ModelNet debemos ejecutar el siguiente comando.

```
vnrun [-d] < VN# | all > <archivo.model> <comando>
```

Donde sí queremos que el comando se ejecute en todos los nodos virtuales de la máquina elegiremos la opción “all” y si queremos que solo se ejecute en uno de los nodos pondremos su número de nodo.

# C. Proxmox Virtual Environment

## Introducción

Proxmox Virtual Environment (Proxmox VE) [PROX09] es una plataforma de virtualización de código libre desarrollado y mantenido por Proxmox Server Solutions GmbH y financiado por la Internet Foundation Austria. Proxmox VE permite correr aplicaciones virtuales y máquinas virtuales de una manera muy sencilla.

La última versión estable de Proxmox VE está basada en un Debian Lenny de 64 bits. Usa una partición del tipo LVM2 para el disco duro e incluye el Kernel de Proxmox VE (la última versión es la 2.6.24), en el cuál están incluidos parches para soportar los sistemas OpenVZ y KVM.

Una de las características más importantes de Proxmox VE es que permite crear una estructura de servidores completa a partir de un hardware sin ninguna instalación previa en menos de una hora incluyendo proxys de email y web, wikis, intranets… Contando además con la ventaja de que tenemos un sistema de copias de seguridad integrado y la posibilidad de migrar las máquinas virtuales entre distintas máquinas físicas sin la necesidad de pararlas. Esta facilidad y rapidez en la instalación es debida a que Proxmox VE proporciona aplicaciones virtuales prefabricadas y una interfaz, vía web, que nos permite realizar la gestión de una manera muy sencilla.

## Ventajas e inconvenientes de la virtualización

Entre las ventajas más importantes que encontramos al usar una plataforma de virtualización encontramos que de este modo podemos aprovechar al máximo nuestro hardware. En los últimos años, con la incorporación al mercado de los procesadores de 64 bits, nos hemos encontrado con la posibilidad de usar cantidades de memoria RAM muy superiores al máximo de 4Gb que marcaban los procesadores de 32 bits. Sin embargo, también nos hemos encontrado con la paradoja de que tras invertir grandes cantidades de dinero en equipos con mayor memoria y mejor procesador, muchas aplicaciones no utilizan todo este potencial al haber sido diseñadas para

trabajar únicamente con direccionamiento de 32 bits. En este caso, plataformas como Proxmox (que trabaja sobre 64 bits) nos permiten aprovechar todos los recursos de nuestras máquinas al permitirnos repartir la memoria real de la máquina física entre diferentes máquinas virtuales.

Otra ventaja importantísima que obtenemos al usar una plataforma de virtualización es la reducción de costes. El usar de forma más óptima nuestro hardware nos permite tener menos máquinas físicas lo cual se traduce en un menor gasto de energía (permitiendo por tanto reducir el $CO_2$ generado, muy importante en estos momentos) y además permite tener menos personal de mantenimiento.

Dejando a un lado los beneficios económicos, también nos encontramos con que plataformas como Proxmox VE nos permiten ahorrar tiempo en la instalación, ya que nos proveen de aplicaciones prefabricadas listas para ser usadas en cuestión de minutos, ahorrándonos el proceso de instalación y configuración de las mismas.

Por último, encontramos otra gran ventaja en Proxmox, que es la capacidad de realizar copias de seguridad y restaurarlas de forma sencilla. De esta forma podemos tener un backup de nuestro sistema de una manera muchos más sencilla que en un equipo físico.

Sin embargo, no todo son ventajas, y en la virtualización nos encontramos algunas desventajas claras respecto al trabajo directamente sobre máquinas físicas.

La desventaja más importante de usar virtualización según [DUBI08] es que magnifica los fallos de hardware. Parece obvio pensar que si tenemos 4 máquinas virtuales en la misma máquina física y ésta se desconecta o avería, nosotros tendremos averiadas 4 máquinas en lugar de una. En un caso de fallo normal, si un servidor falla podemos tener servidores de backup que se ocupen de sus tareas, si fallan 4 servidores a la vez será más difícil que el sistema completo siga funcionando.

Tampoco nos podemos olvidar que la inclusión de un sistema de virtualización gasta recursos, por lo que una parte de nuestro hardware será

usado para el mantenimiento de las máquinas virtuales, perdiendo de este modo parte de nuestra capacidad.

También es importante resaltar que del mismo modo que esta solución nos ofrece la posibilidad de descargar máquinas virtuales preinstaladas y configuradas, también es posible que muchas aplicaciones no funcionen sobre un sistema virtualizado y algunas no lo hagan de una forma óptima. Además, al añadir una nueva capa de complejidad al sistema es mucho más difícil el encontrar donde se producen los fallos del sistema.

Por último cabe destacar que cuando tenemos un sistema virtualizado hay partes de la máquina física, como la tarjeta de red o la velocidad del disco duro, que generalmente no limitan al sistema. Pero si la aplicación que corre en el nivel superior es muy exigente con ellas puede que en este caso si sean limitantes. No hay que olvidar que ahora tenemos solo un disco duro (o raid) y una tarjeta de red para varias máquinas virtuales.

### ***Instalación***

La instalación del sistema Proxmox VE sobre una máquina física no difiere demasiado de la instalación de una distribución Debian normal. Tras arrancar el sistema desde el CD de instalación debemos seguir sus instrucciones. La instalación requerirá nuestra atención en varias ocasiones a lo largo de su desarrollo. En un primer lugar nos pedirá que aceptemos la licencia y aceptemos el disco duro en el que Proxmox VE quiere instalarse. Posteriormente le daremos al sistema información sobre nuestra zona geográfica, nos pedirá la clave y el mail de administrador y después los datos de configuración de la red. A partir de este momento, la instalación ya no requerirá más de nuestra atención, y tras un breve periodo el sistema se encontrará instalado.

Una vez finalizada la copia de archivos a nuestro equipo debemos iniciar la configuración del sistema. Para ello accederemos al interfaz web mediante un navegador web, en la actualidad Proxmox soporta Microsoft Internet Explorer a partir de la versión 6, y Mozilla Firefox a partir de la versión 2. Una vez aquí nos encontramos con el siguiente interfaz.

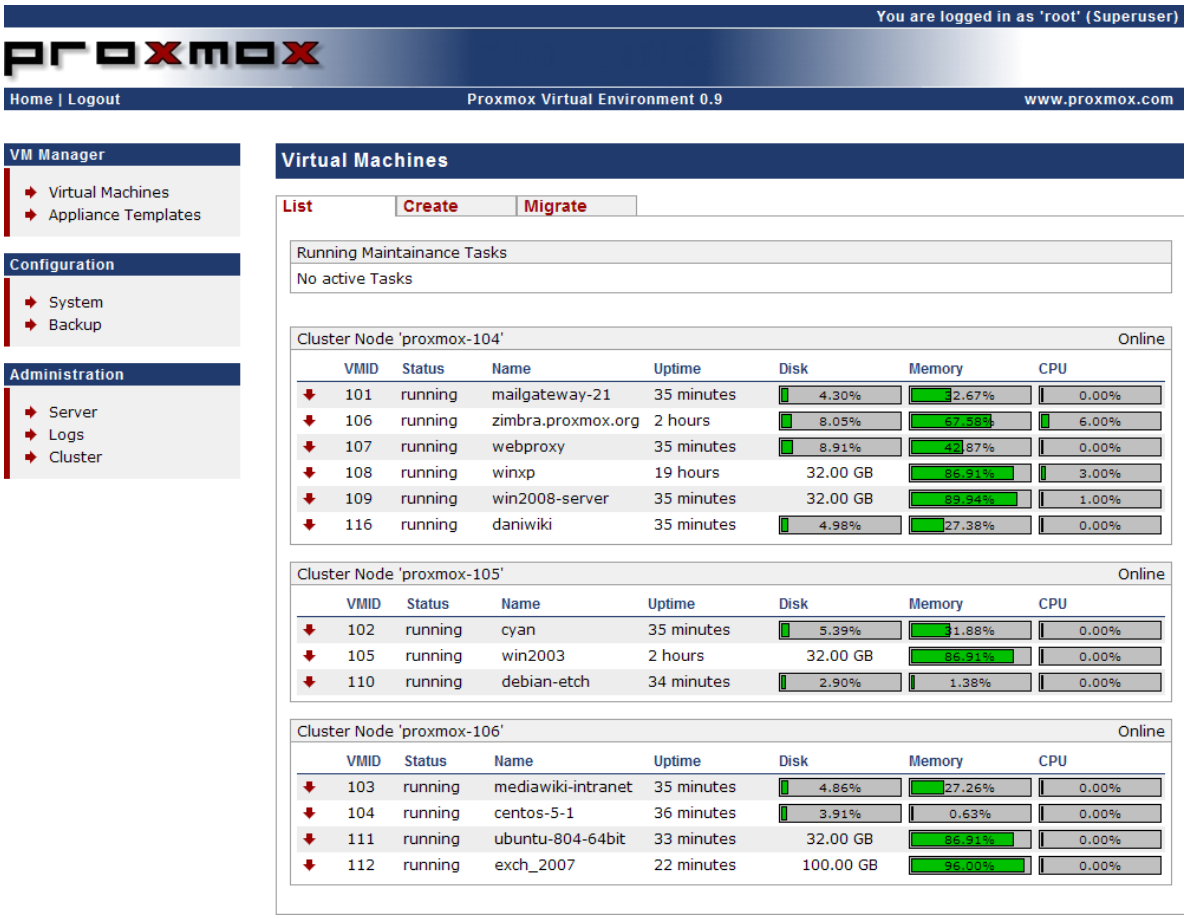


**Figura 65:** Captura de pantalla de Proxmox (1)

Y dentro del apartado System podremos configurar las interfaces de red del sistema o el servidor DNS a utilizar.

## ***Configuración del cluster***

Proxmox VE nos permite tener varias máquinas físicas unidas dentro de un cluster, para de ese modo, poder manejar todas las máquinas físicas desde una sola, que será la máquina principal.

Para añadir una máquina a un cluster debemos teclear el siguiente comando desde la máquina que deseamos añadir:

```
pveca -a -h IP-ADDRESS-MASTER
```

## ***Creación de las máquinas virtuales***

Una vez instalado el sistema Proxmox VE sobre nuestro equipo podremos empezar a crear nuestras máquinas virtuales.

El sistema Proxmox VE nos proporciona dos tipos de máquinas virtuales, los OpenVZ y los KVM cuyo comportamiento es diferente en el sistema. Para la creación de ambos tipos iremos al apartado “Create” dentro del apartado “Virtual Machines”.

### Creando un OpenVZ

Los OpenVZ son contenedores en los que se instala una plantilla prefabricada de las que podemos descargar en el apartado “Virtual Templates” o que podemos subir desde nuestro ordenador. Para poder acceder a estas plantillas debemos seleccionar como tipo de máquina virtual el tipo “Container (OpenVZ)”. Y veremos los siguientes apartados de configuración:

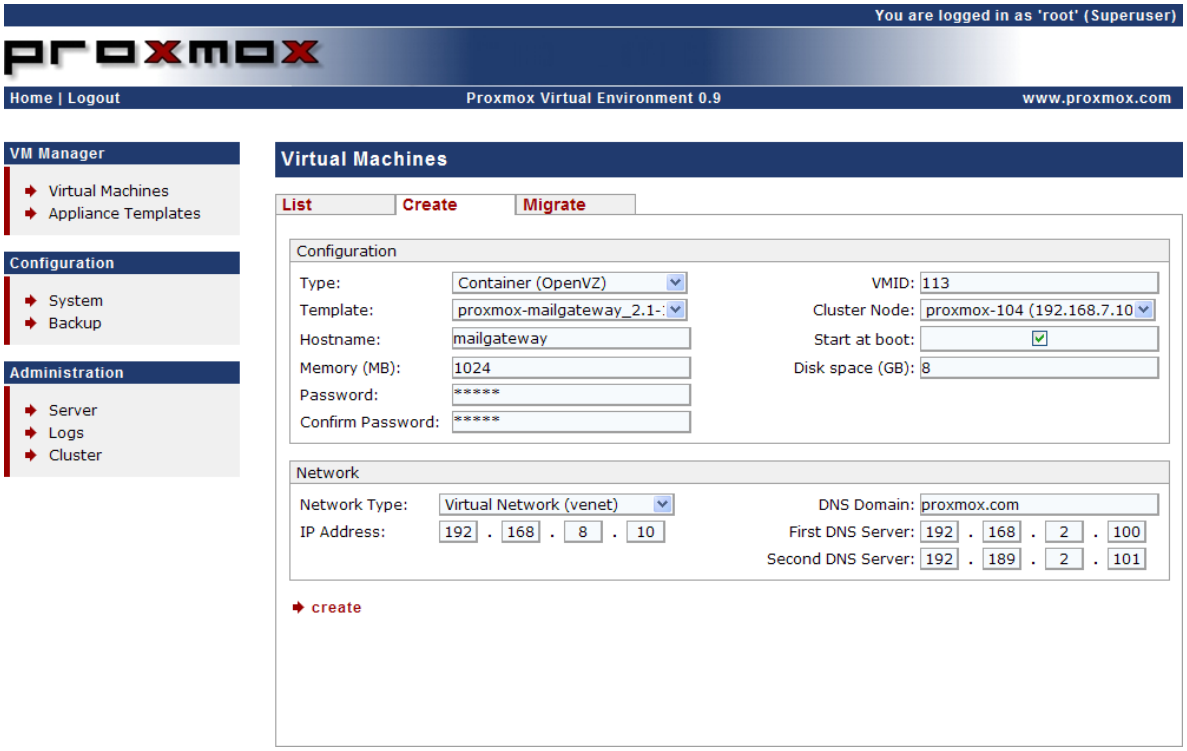


**Figura 66:** Captura de pantalla de Proxmox (2)

- Type: seleccionamos “Container (OpenVZ)”.
- Template: en este apartado tendremos disponibles todas las plantillas que añadamos añadido previamente.
- Hostname: un nombre único para esta máquina virtual.
- Memory (MB): memoria RAM asignada a esta máquina virtual.

- Swap (MB): memoria Swap asignada a esta máquina virtual. En la versión actual de Proxmox esta cantidad de memoria es sumada directamente a la memoria RAM.
- Password: la contraseña de administrador.
- VMID: el identificador de la máquina virtual. Este debe ser único, y si lo cambiamos por uno ya existente se sobrescribirá la máquina virtual con el otro identificador.
- Cluster Node: en una lista veremos los nodos disponibles y podremos elegir en que nodo queremos crear la máquina virtual.
- Start at boot: si activamos este apartado, la máquina virtual se encenderá automáticamente cuando se inicie el sistema.
- Disk Space (GB): Marca el espacio máximo que puede ocupar en disco la máquina virtual. Sin embargo, en el caso de los OpenVZ este espacio no es reservado de ante mano en el disco duro físico, sino que se va llenando según se necesita.

Por último podemos configurar el apartado de red (“Network”) asignando a la máquina virtual una o varias interfaces de red que previamente habremos creado en el sistema.

**Creando un KVM**

Los KVM son máquinas completamente virtualizadas. Al contrario que en el caso de los OpenVZ ahora debemos realizar una instalación y configuración completa del sistema instalado. En este caso el tipo a elegir será "Fully virtualized (KVM)". La creación es muy similar al caso de un OpenVZ por lo que me limito a poner los apartados que cambian.

- Type: elegimos “Fully virtualized (KVM)”.
- Installation Media: aquí podemos elegir “cd rom” si queremos realizar la instalación desde la unidad de CD física el equipo, o una imagen de CD que previamente hayamos subido al sistema.
- Disk Space (GB): En este caso el espacio en disco elegido es reservado completamente al crear la máquina virtual.
- Disk type: debemos seleccionar el tipo de disco duro deseado.

- Guest Type: en este apartado debemos elegir el tipo de sistema que se va a instalar (32 o 64 bits están permitidos).

Una vez creada la máquina virtual, nos limitaremos a arrancarla y procederemos a instalar el sistema deseado.

# BIBLIOGRAFÍA Y REFERENCIAS


[ALAS05] M.S. Artigas, P.G. Lopez, J.P. Ahullo, and A.F.G. Skarmeta. *Cyclone: a novel design schema for hierarchical dhts.* In Peer-to-Peer Computing, 2005. P2P 2005. Fifth IEEE International Conference en, páginas 49–56, Ago.-2 Sept. 2005.

[CAID09] Proyecto CAIDA. 2009. < http://www.caida.org>

[DUBI08] Denise Dubie. 7 side effects of sloppy virtualization. En Nework World. 2008. < http://www.networkworld.com/news/2008/062408-sloppy-virtualization.html?page=1>

[GEBR+03] Luis Garces-Erice, Ernst W. Biersack, Keith W. Ross, Pascal A. Felber, and Guillaume Urvoy-Keller. *Hierarchical p2p systems*. En Proceedings of ACM/IFIP International Conference on Parallel and Distributed Computing (Euro-Par), 2003.

[GEFB+04] L. Garcs-Erice, P. A. Felber, E. W. Biersack, G. Urvoy-Keller, and K. W. Ross. *Data indexing in peer-to-peer dht networks*. Distributed Computing Systems, International Conference on, 0:200–208, 2004.

[CCGC09] José Manuel Camacho Camacho, Carmen Guerrero y Rubén Cuevas. *Evaluation platform for large-scale distributed applications*. Ene, 2009.

[GGGM04] Prasanna Ganesan, Krishna Gummadi, and Hector Garcia-Molina. *Canon en g major: Designing dhts with hierarchical structure*. icdcs, 00:263–272, 2004.

[KKL+07] Aleksandra Kovacevic, Sebastian Kaune, Nicolas Liebau, Ralf Steinmetz, and Patrick Mukherjee. *Benchmarking platform for peer-to-peer systems (benchmarking plattformfür peer-to-peer systeme*). it – Information Technology, 49(5):312–319, 2007.

[MM02] P. Maymounkov and D. Mazieres. Peer-to-Peer Systems: First InternationalWorkshop, IPTPS 2002 Cambridge, MA, USA, March 7-8, 2002. Revised Papers, volume 2429/2002 of Lecture Notes in Computer Science, chapter Kademlia: A peer-to-peer information system based on the XOR metric, páginas 53–65. Springer, 2002.

[MODE09] ModelNet, Systems and Networking, 2009. <https://modelnet.sysnet.ucsd.edu/>

[MGMS+06] J. P. Muñoz Gea, J. Malgosa Sanahuja, A. M. Guirado Puerta, J. C. Sánchez Aarnoutse y J. García Haro. *Arquitectura de Topología Hibrida para Sistemas P2P de compartición de archivos*. XXI Simposium Nacional de la Unión Científica Internacional de Radio, páginas 246-249. Sep. 2006. < http://w3.iec.csic.es/ursi/articulos_modernos/articulos_oviedo_2006/articulos/sesionTEL-I-13small.pdf>

[MS03a] J. Mischke and B. Stiller. *Peer-to-peer overlay network management through agile*. In Integrated Network Management, 2003. IFIP/IEEE Eighth International Symposium en, páginas 337–350, 2003.

[MS03b] J. Mischke and B. *Stiller. Rich and scalable peer-to-peer search with shark.* En Autonomic Computing Workshop, 2003, páginas 112–121, 2003.

[MYBC+09] Isaías Martínez-Yelmo, Alex Bikfalvi, Rubén Cuevas, Carmen Guerrero, y Jaime García. *H-p2psip: Interconnection of p2psip domains for global multimedia services based on a hierarchical dht overlay network*. Computer Networks (Special -Issue on Content Distribution Infrastructures for Community Networks), 53(4), Mar. 2009.

[MYBG+08a] Isaías Martínez-Yelmo, Alex Bikfalvi, Carmen Guerrero, Rubén Cuevas y Andreas Mauthe. *Enabling global multimedia distributed services based on hierarchical dht overlay networks*. International Journal of Internet Protocol Technology (Special Issue on Future Multimedia Networking-), 3(4), Dic. 2008.

[MYBG+08b] Isaías Martínez-Yelmo, Alex Bikfalvi, Carmen Guerrero, Rubén Cuevas y Andreas Mauthe. *Enabling global multimedia distributed services based on hierarchical dht overlay networks*. In IEEE Conference on Next Generation Mobile Applications Services and Technologies 2008. Future Multimedia Networking Workshop, páginas 543–549. IEEE Computer Society, Sep. 2008.

[MYBG09] Isaías Martinez-Yelmo, Alex Bikfalvi, and Carmen Guerrero. *Benets on using h-p2psip in mobile environments*. En JITEL 2009, Sep. 2009.

[MYCGM08] Isaías Martínez-Yelmo, Rubén Cuevas, Carmen Guerrero y Andreas Mauthe. *Routing performance in hierarchical dht-based overlay networks*. En Proceedings on 16th Euromicro International Conference on Parallel, Distributed and network-based Processing, Feb. 2008.

[MYGCM09] Isaías Martínez-Yelmo, Carmen Guerrero, Rubén Cuevas y Andreas Mauthe. *A hierarchical p2psip architecture to support skype-like services*. En Parallel, Distributed and Network-based Processing, 2009.

[NETB09] NetBeans. 2009. www.netbeans.org/

[OLYO09] Olyo. 2009. <http://sourceforge.net/projects/olyo/>

[P2PN09] P2PNS. 2009. <http://www.p2pns.org/>

[P2PS08] REsource LOcation And Discovery (RELOAD), 2008: http://tools.ietf.org/html/draft-bryan-p2psip-reload-03

[P2PS09] P2PSIP WG. 2009. < http://www.p2psip.org/>

[P2PP09] Peer to Peer Protocol. 2009. <http://www1.cs.columbia.edu/~salman/peer/>

[PROX09] Proxmox, Virtual Environment, 2009. <http://www.proxmox.com/>

[REDI09] REDIMadrid. 2009 < http://www.madrimasd.org/queesmadrimasd/RedTelematicaMadrid/CAM/REDIMadrid/default.asp>

[SENB07a] Moritz Steiner, Taoufik En Najjary, and Ernst W Biersack. Analyzing peer behavior in KAD. Technical Report EURECOM+2358, Institut Eurecom, France, Oct 2007.

[SENB07b] Moritz Steiner, Taoufik En-Najjary, and Ernst W. Biersack. Exploiting kad: possible uses and misuses. SIGCOMM Comput. Commun. Rev., 37(5):65– 70, 2007.

[SENB07c] Moritz Steiner, Taoufik En-Najjary, and Ernst W. Biersack. A global view of kad. In IMC '07: Proceedings of the 7th ACM SIGCOMM conference on Internet measurement, páginas 117–122, New York, NY, USA, 2007. ACM.France, Oct 2007.

[SIPC09] Sip-Communicator. <http://sourceforge.net/projects/sip-communicato/>

[SMLN+03] I. Stoica, R. Morris, D. Liben-Nowell, DR Karger, MF Kaashoek, F. Dabek, and H. Balakrishnan. *Chord: a scalable peer-to-peer lookup protocol for internet applications*. IEEE/ACM Transactions on networking, 11(1):17–32, 2003.

[SMSH05] Sameh El-Ansary y Seif Haridi. *An Overview of Structured Overlay Networks*. En Theoretical and Algorithmic Aspects of Sensor, Ad Hoc Wireless and Peer-to-Peer Networks. 2005. < http://eprints.sics.se/237/01/elansary-singlespaced.pdf>

[SOUL07] Juan Soulié. *C++ La*nguage Tutorial. 2007. < http://www.cplusplus.com/files/tutorial.pdf>

[VIAP09] ViaSIP_NG. 2009. <http://sourceforge.net/projects/viasip/>

[WIKI09] Wikipedia, 2009. <http://www.wikipedia.org/>

[XMH03] Zhiyong Xu, Rui Min, and Yiming Hu. *Hieras: a dht based hierarchical p2p routing algorithm*. In Parallel Processing, 2003. Proceedings. 2003 International Conference en, 2003.

# GLOSARIO

**Bootstrap**

Servicio que permite el arranque del sistema. En nuestro caso, un elemento que actúe como servidor de Bootstrap estará encargado de proporcionar a los nuevos peers información sobre los nodos que ya forman parte de la red.

**Chord**

Protocolo de búsqueda distribuida. Es el más sencillo de todos los que describen una red overlay estructurada.

**DHCP**

Dynamic Host Configuration Protocol. Protocolo de red que permite a los dispositivos participantes en una red IP obtener sus parámetros de configuración automáticamente.

**DHT**

Distributed hash table, describe una clase de servicios descentralizados que proporcionan una capacidad de búsqueda parecida a la de una tabla de hash.

**DNS**

Domain Name System. Sistema de nomenclatura jerárquica para computadoras, servicios o cualquier recurso conectado al internet o a una red privada.

**Gateway**

Dispositivo que permite interconectar redes con diferentes protocolos o arquitecturas.

**Hash**

Una función de hash proporciona una clave de manera casi unívoca que representa al contenido deseado. De ese modo se pueden tener claves de un tamaño prefijado.

**H-P2PSIP**

Hierarchical Peer-to-peer SIP. Es una versión jerárquica del protocolo P2PSIP

**ID**

Identificador.

**IETF**

Internet Engineering Task Force. Es una comunidad internacional de diseñadores, operados, comerciantes de red e investigadores centra en la evolución de la arquitectura de Internet.

**Kademlia**

Protocolo de búsqueda distribuida. Se trata de un protocolo que describe una red overlay estructurada.

**Key**

Cada una de las posibles claves que representan a un peer o a algún tipo de contenido dentro de la red.

**MAC**

Media Access Control. Una dirección MAC es un identificador de 48 bits que corresponde de forma univoca a una Ethernet de red.

**ModelNet**

Emulador de red usado en nuestro experimento.

**Napster**

Primera aplicación P2P que apareció. Desarrollada en 1996 pretendía ser una plataforma de distribución de archivos.

**NAT**

Network Address Translation. Es un mecanismo utilizado por routers IP que permite la comunicación de redes que tienen direcciones incompatibles. En la práctica se usa de manera generalizada para la traducción de direcciones privadas en públicas.

**Node**

Cada uno de los elementos que participantes en una red overlay.

**P2PSIP**

Peer-to-peer SIP es un protocolo que permite la creación de redes overlay. Actualmente está siendo estandarizado por el IETF.

**Peer**

Cada uno de los elementos participantes en una red overlay que participa en el encaminamiento de mensajes.

**Query**

Búsqueda en una red overlay.

**RELOAD**

REsource LOcation And Discovery. Es el protocolo de señalización usado por P2PSIP.

**Rtt**

Round-Trip delay Time. Tiempo que tarda un paquete enviado por un emisor en volver a este mismo tras pasar por el receptor de destino.

**Skype**

Software usado para realizar llamadas de VoIP. Es de código cerrado. Utiliza una red overlay para comunicar a los diferentes clientes.

**URI**

Uniform Resource Identifier. Identificador uniforme de recurso.

**VoIP**

Voz sobre IP. Grupo de recursos que hacen posible que la voz viaje a través de internet encapsulada en paquetes IP.

**WG**

Working group. Grupo de trabajo del IETF.