Escola Politècnica Superior
de Castelldefels

UNIVERSITAT POLITÈCNICA DE CATALUNYA

TREBALL DE FI DE CARRERA

TÍTOL: Diseño e implementación de un sistema de Telefonía IP sobre una red P2P I

AUTOR: David Martín Nevado

DIRECTOR: Sergio Machado

DATA: 20 de septiembre de 2006

Título: Diseño e implementación de un sistema de Telefonía IP sobre una red P2P I

Autor: David Martín Nevado

Director: Sergio Machado

Data: 20 de Septiembre de 2006

Resumen

La elaboración de este TFC ha tenido como objetivo la implementación de un sistema de Telefonía IP sobre una arquitectura SIP-P2P con una serie de requisitos.

La telefonía tal y como la conocemos ahora está sumamente limitada a la comunicación analógica de voz a 8KHz. Hoy en día, con los avances tecnológicos y la evolución de Internet el servicio de telefonía se nos ha quedado anticuado, por ello nació la Telefonía IP, dando lugar a una serie de posibilidades mucho más amplias, gracias a la red de Internet, todo ello a través de VoIP.

En este TFC no nos hemos centrado en la finalidad del problema, la Telefonía IP, sino que hemos trabajado a fondo en todo su diseño, desde la red P2P Pastry, a la arquitectura de señalización y registro mediante SIP.

Por ello, hemos dividido éste documento de la misma forma que planteamos el problema a la hora de ponernos a trabajar en él y por lo tanto, programar la aplicación, en 2 fases bien diferenciadas, en las que más adelante profundizaremos y daremos motivos por los cuales ha sido necesario trabajar dividiéndolas, para a posteriori, juntarlas hasta completar el diseño final.

En la primera fase, trataremos la red P2P Pastry que hemos diseñado, explicando su funcionalidad, características y uso dentro de nuestro proyecto, siendo la base de todo el sistema, y el primer eslabón de todo el conjunto.

En la segunda fase, profundizaremos sobre la arquitectura SIP, cómo la hemos creado, cómo funciona, y porqué la hemos usado.

Title: Design and implementation of a system of Telephony IP on a net P2P I

Author: David Martín Nevado

Director: Sergio Machado

Date: Sep, 20th 2006

Overview

The elaboration of this TFC has had like objective the implementation of a system of Telephony IP on an architecture SIP-P2P with a series of requirements.

The telephony so and as we know it now extremely is limited the analogical communication of voice to 8KHz. Nowadays, with the technological advances and the evolution of Internet the service of telephony old fashioned we have had left, for that reason Telephony IP was born, giving rise to a series of much more ample possibilities, thanks to the network of Internet, all it through VoIP.

In this TFC we have not been centered in the purpose of the problem, Telephony IP, but that have worked thoroughly in all its design, from network P2P Pastry, to the architecture of signaling and registry by means of SIP.

For that reason, we have divided this one document of the same form that we created the problem at the time of putting to us to work in him and therefore, to program the application, in 2 differentiated phases affluent, in which more ahead we will deepen and give reasons by which it has been necessary to work dividing them, for a then, to join them until completing the final design.

In first stage, we will treat network P2P Pastry that we have designed, explaining its functionality, characteristics and use within our project, being the base of all the system, and the first link of all the set.

In the second phase, we will deepen on architecture SIP, how we have created it, how it works, and because we have used it.

ÍNDEX

INTRODUCCIÓN

El mundo de la Telefonía IP abre un mar de oportunidades muy amplias y distintas de las que ofrece la telefonía analógica que conocemos hoy en día.

Nos abre una comunicación digital, no basada en conexiones analógicas simples, sino mediante paquetes, por la red de Internet, como si de datos de software se tratase.

Al hablar del uso de Internet, estamos hablando de la necesidad de usar una red, para encaminar nuestros paquetes para poder comunicar a los usuarios, de la misma forma que necesitamos señalizar dichos usuarios, saber dónde están, en qué estado están, y poder garantizar una comunicación con éxito y segura.

La proliferación de las redes Peer-to-peer está en expansión, y gran culpa de ello lo tienen los videojuegos online, que nos han hecho entender este tipo de redes, como redes mucho más eficientes y dinámicas.

Para nuestro sistema, hemos hecho uso de un tipo de red P2P, la red Pastry, y al programar en código Java, hemos usado la API de FreePastry.

Pastry tiene la ventaja que se adapta muy bien a nuestro sistema, ya que se basa en una red overlay auto-organizada, lo que significa que tiene un esquema genérico de localización de objetos y encaminamiento P2P. Esto nos ha venido muy bien, para la localización de nodos dentro de nuestra red, y por lo tanto, la comunicación entre ellos.

Como señalización y registro en el sistema, hemos hecho uso del protocolo de señalización SIP (Session Initiation Protocol).

SIP nos aporta un conjunto de funcionalidades de procesamiento de llamadas idóneo como pueda ser la de, llamar a un número, hacer sonar a un teléfono cuando recibe una llamada, avisar de la no disponibilidad de ese número ante una llamada, etc. Pero no nos confundamos, SIP es usado únicamente para iniciar y terminar comunicaciones, para ello necesitamos registrarnos como usuarios e intercambiar una serie de mensajes entre los nodos que Pastry ha creado anteriormente. Una vez establecida la comunicación, está viajará sobre RTP (Real-time Transport Protocol).

Capítulo 1. Pastry

1.1. Introducción: Visión global

El diseño que se ha procurado programar, y recrear bajo lenguaje Java, es complejo y necesita ser explicado por partes, pero para comprender los detalles, mejor explicar primeramente el todo, para a posteriori pasar a los detalles.

El programa, como comentamos en la introducción, consta de 2 partes bien diferenciadas, por un lado la red Pastry P2P y por otro el sistema de señalización SIP.

Dentro de todo el abanico de posibilidades a la hora de elegir una red P2P, elegimos la red Pastry, ya que al programar en Java, hemos usado la Api de FreePastry, que es la que mejor se adaptaba a nuestras necesidades.

La red Pastry, consta de nodos, que simulan cada uno un usuario. Estos nodos, son identificados con un ID, y se comunican buscando su peer mediante su Key, la cual, no deja de ser la ID del peer con el cual, se quieren comunicar.

FreePastry, como comentaremos más en adelante, nos permite simular la red en un solo ordenador, lo que es una gran ventaja, ya que podemos simular una cantidad de nodos enorme sin necesidad de tener que usar varios pc's.

Una vez establecida la red P2P con los nodos, los guardaremos en unas tablas llamadas DHT (Distributed Hash Table) donde cada peer guardará la información que tiene sobre cada usuario y su key.

Para la comunicación entre usuarios, necesitamos de un mecanismo de señalización, donde además podamos registrar a los usuarios, de tal modo, que nos sea fácil comunicarnos con ellos, ver en que estado se encuentran, etc. Para ello, hemos usado SIP (Session Initiaton Protocol).

SIP nos permite que los peer se comuniquen entre ellos, estableciendo sesiones de comunicación. Cabe puntualizar, que SIP no se encarga de las transmisiones, para ello se usan otros protocolos como el RTP, lo que hace SIP es tener un registro de todos los peers, y establecer comunicación entre ellos.

Como todo sistema de registro, donde hayan usuarios que se comunican con otros, necesitamos de una cierta seguridad, como pueda ser una contraseña. SIP no permite el envío de otra información que no sea propia de SIP, es decir, al establecer una comunicación solamente podemos pasar direcciones SIP, y no podemos añadir detrás nada más, por ello, todo el tema de seguridad, lo hemos programado a un nivel inferior, en la red Pastry, mediante funciones de hash, y cifrado asimétrico RSA.

1.2. Redes

Hoy en día existen muchos sistemas de mensajería instantánea, que permiten la comunicación entre usuarios, como puedan ser MSN Messenger, Yahoo Messenger, etc.

La particularidad de estos sistemas, es que tienen una arquitectura centralizada, al que cada uno de los usuarios de conecta con un servidor/es central/es el que les proporciona todos los servicios, como la lista de usuarios conectados, mensajería, comunicación con los usuarios, intercambio de mensajes, etc.

Este tipo de arquitectura centralizada, tiene varias desventajas, ya que dependen del servidor central, que les proporciona el servicio, en el momento que ese servidor falla, automáticamente, los usuarios caen por efecto barrido.

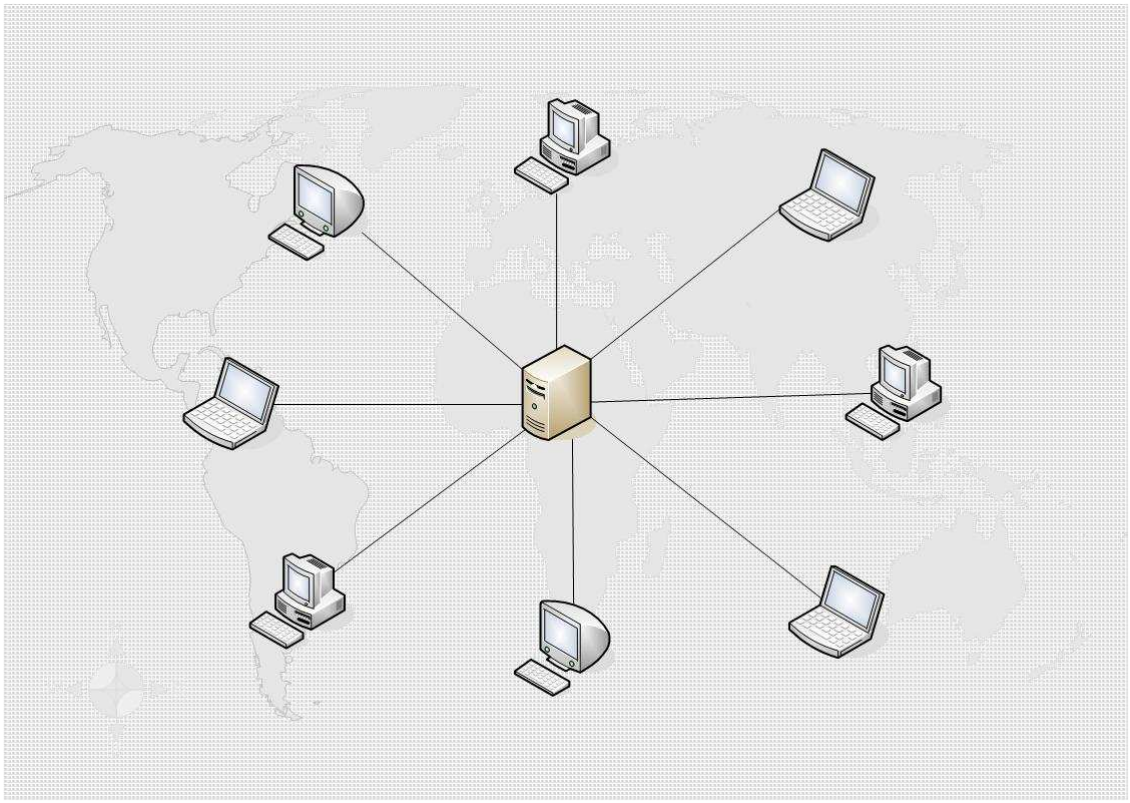

Fig. 1.1. Arquitectura centralizada, cliente – servidor

Otro problema es la confidencialidad de los usuarios, con la arquitectura centralizada los usuarios han de guardar sus datos, su lista de contactos, etc, en dicho servidor central, lo que a nivel de seguridad, puede llegar a ser un problema. Por último, mantener los servidores centrales, suponen un gasto, por lo que mucho de estos sistemas tienen tarifas, lo que no los hacen eficientes, de la misma forma que los usuarios están a expensas de los requisitos que el

servidor exija a los clientes para poder usar su sistema, lo que supe disponer de un hardware adaptado al sistema.

Para solucionar todos estoy problemas, surgieron las redes Peer-to-peer (P2P).

La filosofía de una red Peer-to-peer, es la de la inexistencia de un servidor central, no hay ni clientes ni servidores, solo una serie de nodos que se comportan a la vez como servidores y como clientes de lo demás nodos de la red.

De este modo, el modelo cliente-servidor, del a arquitectura centralizada, queda eliminado, cualquier nodo puede iniciar o completar una comunicación compatible entre ellos.

Los nodos pueden ser diferentes entre ellos, en cuanto a configuración local, velocidades, ancho de banda, etc.

En una red Peer-to-peer, los usuarios han de compartir con todos los usuarios, para que la red funcione y sea eficiente.

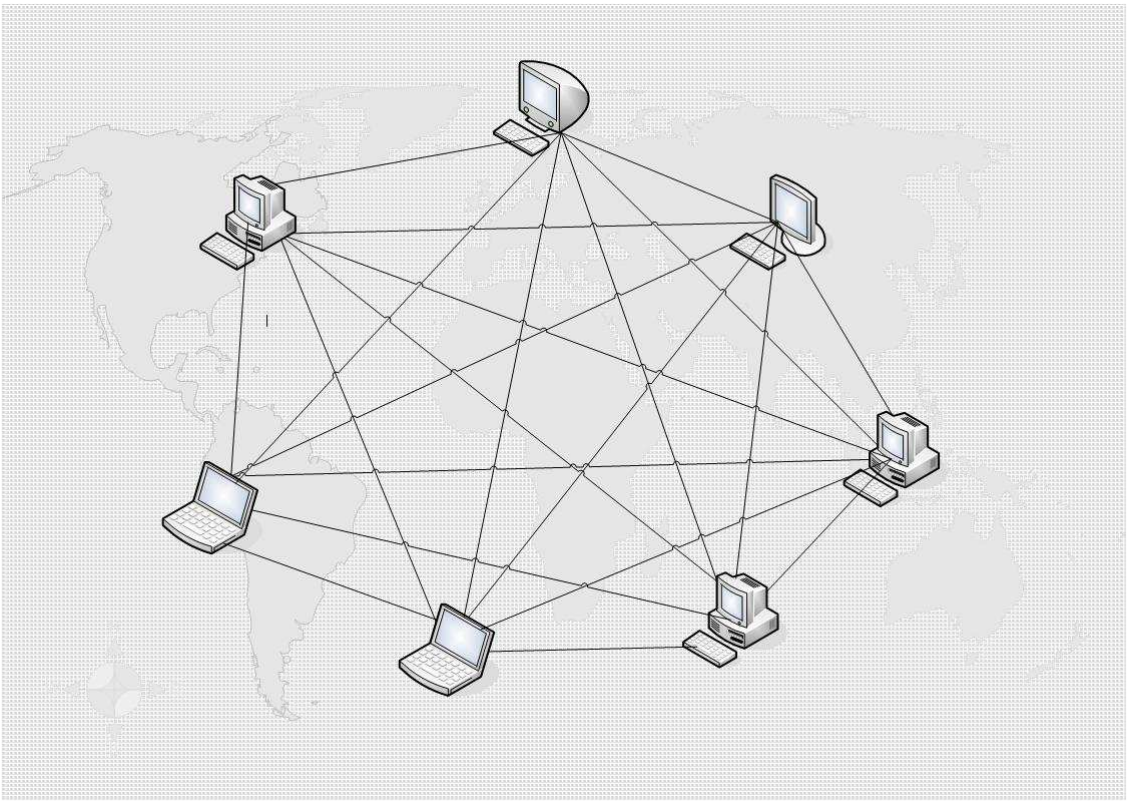

Fig 1.2. Arquitectura Peer-to-peer

Para la elaboración de nuestro sistema, hemos hecho uso de una red P2P, más concretamente, hemos creado una red Pastry.

1.3. Redes Pastry

Pastry es un sistema de redes Peer-to-peer, que funciona como un sistema independiente de posición distribuido a través de muchos terminales individuales, que hacen de nodos, dentro de la red.

Es un sistema distribuido, en el que todos los nodos tienen capacidades idénticas, donde las comunicaciones son simétricas. Pastry apunta para proporcionar una disponibilidad, adaptabilidad y seguridad, altas, gracias a su arquitectura descentralizada.

Pastry es por tanto, un sistema genérico de localización de objetos (nodos), y encaminamiento P2P basado en una red overlay auto-organizada, en la que los nodos se conectan los unos a los otros, mediante Internet.

Así pues, una vez montada la red, añadidos los nodos, que hacen de usuario, sobre Pastry pueden ir diferentes aplicaciones, en nuestro caso, la Telefonía IP, como aplicación de comunicación entre los usuarios, haciendo uso de SIP para establecer las conexiones.

1.3.1. Características de Pastry

La elección de Pastry ha sido por sus características, que se adaptaban muy bien a nuestras necesidades, de encaminar tráfico de Telefonía IP.

Estas son las características más importantes:

- Localización independiente de los usuarios.
- Tolerancia a fallos: si un nodo abandona el sistema, sus responsabilidades inmediatamente son asumidas por sus nodos vecinos.
- Eficaz encaminamiento dinámico de mensajes.
- Descentralización de todas las funciones de red, (sistema peer-to-peer).
- Adaptabilidad alta.
- Económico, no se necesitan servidores de alto coste, el coste lo asume cada usuario con su máquina.
- Seguro.

Cada nodo en una red Pastry posee un identificador único e intransferible, dentro de una sesión, llamado NodeId de 128 bits. Veámoslo en el siguiente apartado.

1.3.2. NodeId

El NodeId es un identificador de 128 bits, único para cada usuario, e intransferible dentro de cada sesión, es decir, en una sesión activa, dos usuarios no pueden tener el mismo NodeId, pero una vez un usuario finaliza su sesión, libera su NodeId y queda disponible para ser asignado a otro usuario.

Los rangos de NodeIds van normalmente desde 0 a $2^{128} - 1$. Un NodeId puede ser generado mediante una función de Hash SHA – 1[1] de la dirección IP del nodo o de su clave pública.

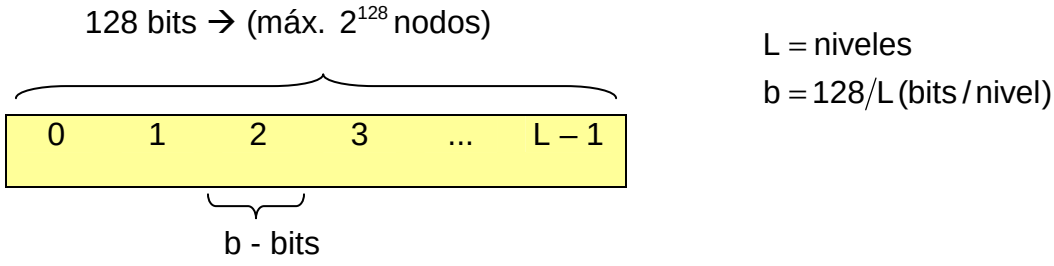

Fig. 1.3 Representación del espacio circular de NodeIds en una red Pastry. Tenemos que el NodeId es una secuencia de L con una base de 2^b (b-bits) dígitos.

En nuestro sistema esto no funciona así exactamente, ya que asignamos manualmente el NodeId de casa usuario, tratándolo como una cuenta de usuario personal, ya la que hablaremos más en adelante.

Por defecto, Pastry asigna dichos NodeId de forma aleatoria, asignándolos de tal forma que la red se distribuya de forma uniforme, ya que Pastry crea la red en un espacio circular de nodos, cada uno con su NodeId como identificador.

Cabe destacar, que por el mero hecho de tener NodeIds semejantes, los nodos no tienen porque estar próximos, sino que normalmente quedarán dispersos por la red, lo que nos lleva a un problema esencial, el encaminamiento para la comunicación entre nodos.

1.3.3. Encaminamiento

Pastry encamina de una forma sencilla, cada nodo posee su NodeId como hemos explicado anteriormente, dicho NodeId a su vez hace funciones de key, para así encaminar mensajes.

Es decir, Pastry encamina sus mensajes pasando por el nodo con un NodeId más cercano numéricamente a la key del destinatario, lo veremos más claro con esta figura.

[1] Hablaremos de SHA – 1 en el punto 1.4.3

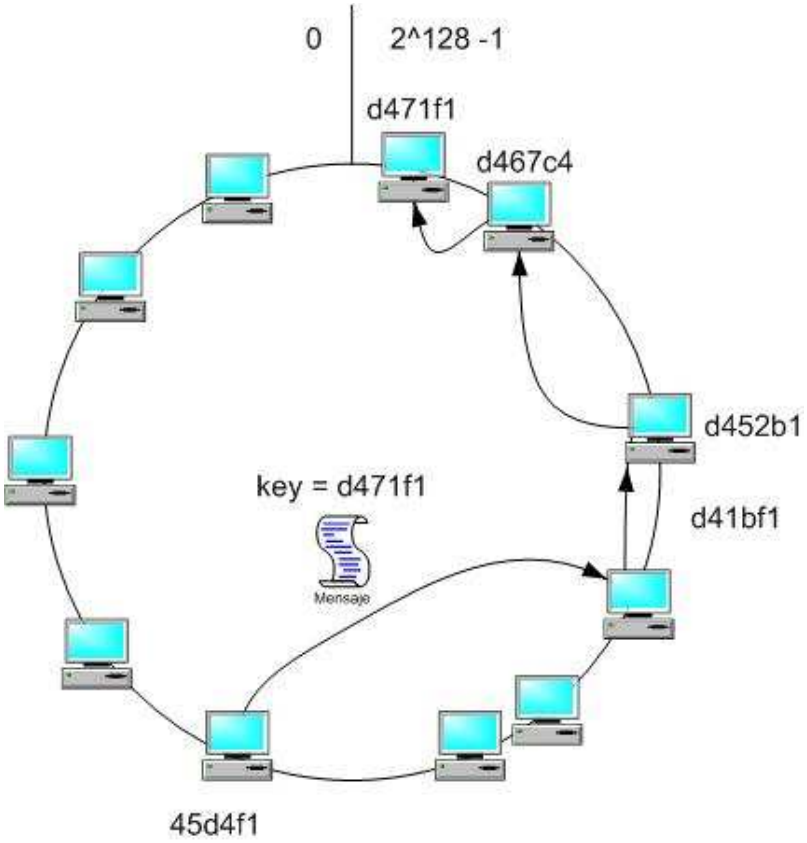

Fig. 1.4 Encaminamiento Pastry

En la figura 1.3, podemos observar como el origen usa un Id llamado Key que es el NodeId del destinatario, vemos como los nodos intermedios van encaminando el mensaje hasta llegar al destino.

El encaminamiento se realiza, a través de los nodos que tienen una key similar numéricamente a la que el origen está deseando llegar.

Un mensaje puede ser encaminado hacia el nodo con NodeId más similar o cercano a la key, en $\log_{2^b} N$ pasos, donde tenemos que:

- b: es un parámetro de configuración que indica los bits destinados a cada digito del NodeId.
- N: es el número de nodos en la red Pastry.

El NodeId y la key tienen la misma base numérica 2^b.

Siempre tendrá éxito el recibo del mensaje a excepción de que $L/2$ nodos con NodeIds similares a la key fallen a la vez, donde L es un parámetro de configuración con valor entre 16 o 32.

1.3.3. Tablas de estado

Pastry hace uso de unas tablas de estado para poder encaminar sus mensajes por la red.

En dichas tablas obtenemos la información sobre los nodos cercanos con sus keys.

Con las tablas de estado locales y el algoritmo de encaminamiento que usa Pastry, se pueden encaminar los mensajes, a partir de la obtención del nodo cuyo NodeId es más similar con la key a la que va destinado el mensaje.

Para que este encaminamiento sea válido, el NodeId ha de ser al menos un digito (o b dígitos) más largo que el prefijo que comparte la key con el nodo actual, es decir, si tenemos una *key = 3445*, y el siguiente nodo es *NodeId = 3450*, no podemos encaminarlo, porque tiene 2 dígitos distintos al de la key, en cambio si el *NodeId = 3447*, sí podríamos, ya que solo se diferencian en 1 digito.

Por lo tanto, podríamos encaminarlo hacia ese NodeId, una vez allí buscar el siguiente, y así hasta llegar al destinatario más probable.

De esta forma, se van rellenando las tablas de estado en cada nodo, que son 2 por nodo: tabla de ruta y leafset.

1.3.3.1. Tabla de ruta

Una tabla de ruta, tiene $\log_{2^b} N$ niveles, cada nivel corresponde a una fila y $2^b - 1$ entradas por nivel. Las entrada de cada fila en particular, tienen una cosa en común, y es que todos tienen el mismo número de dígitos iguales, de izquierda a derecha, es decir, las entradas de la fila 4, tienen los 4 primeros dígitos iguales.

*	Xi	Yi
1	**	Yi
1	2	***

- * → Puede ser cualquier número comprendido entre $\{0,2^b - 1\} - \{1\}$
- ** → Puede ser cualquier número comprendido entre $\{0,2^b - 1\} - \{2\}$
- *** → Puede ser cualquier número comprendido entre $\{0,2^b - 1\} - \{3\}$
- Xi y Yi → Pueden ser cualquier número entre $\{0,2^b - 1\}$

Fig. 1.5 Representación de cómo se distribuye una tabla de ruta Pastry.

1.3.3.2. El leafset

Llamamos leafset al conjunto de nodos más cercanos entre si, numéricamente hablando. Este tratamiento es local, es decir, cada nodo tendrá su propio leafset, ya que cada uno tendrá un conjunto de nodos cercanos, diferentes al resto.

El parámetro leafset (L) tiene valores que van desde 2^b a $2x2^b$, donde L son los nodos más cercanos basados en la proximidad en el espacio de NodeId.

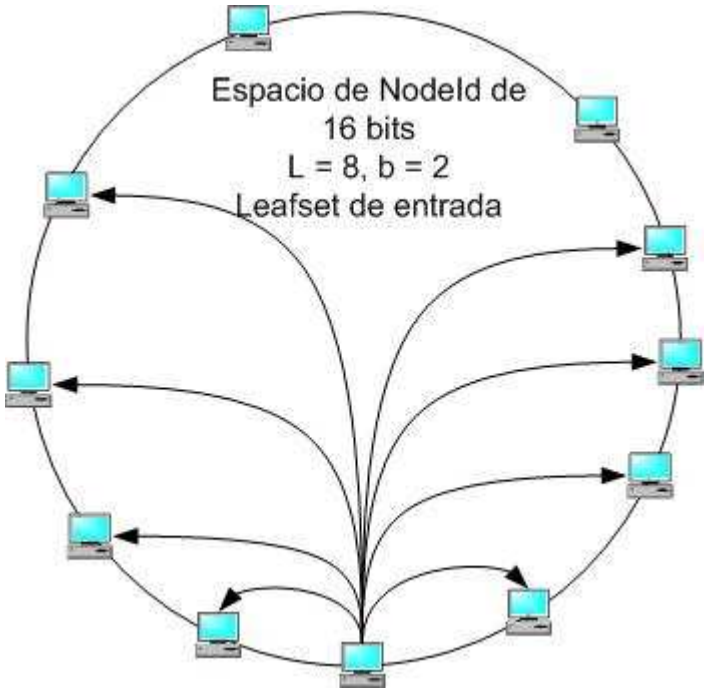

Fig 1.6 Representación de la vecindad mediante el valor de Leafset del nodo emisor.

1.3.3.3. Neighborhood Set

El sistema de vecindad (neighborhood set en inglés), se basa en la proximidad métrica de la red con los nodos adyacentes. A diferencia del Leafset, éste mide la proximidad en hops (saltos) de un nodo a otro, y no se usa para encaminar, sino para añadir o recuperar un nodo.

1.3.4. Algoritmo de encaminamiento

Pastry puede enrutar mensajes aleatoriamente y de forma directa indicándole el nodo al que queremos enviar los mensajes.

Para enrutar aleatoriamente, Pastry tiene un método llamado

routeMyMsg(Id rand);

donde le pasaremos un Id aleatorio, generado por la NodeIdFactory[2] de forma aleatoria, pero en nuestro caso queremos enrutar los mensajes directamente, diciendo hacia que nodo queremos apuntar, para ello usaremos el método

routeMyMsgDirect(NodeHandle boothandle);

donde le pasaremos un objeto NodeHandle[3], el cual contiene las direcciones de los nodos con sus keys para poder enrutar los mensajes como explicaremos seguidamente.

El algoritmo se fija primero si la key está dentro del rango de nuestro leafset, si es así reenvía todo el mensaje al siguiente nodo con el leafset más próximo a la key, y así sucesivamente hasta llegar al destinatario.

Si no está dentro de ese leafset, miramos la tabla de ruta, y buscamos una entrada donde ocurra lo que anteriormente hemos explicado, que el NodeId comparta al menos un digito más con la key que con el NodeId local, si la entrada existe, reenviamos el mensaje.

Con este pseudo-código lo veremos más claro:

route (my-id, key-id, mensaje)

if (key-id está en el rango de mi leafset) {

reenvía al nodo numéricamente hablando más cercano al leafset

}
else {

mira la tabla de ruta y buscamos que el NodeId comparta al menos un digito más con la key que con el NodeId local, si la entrada existe, reenviamos el mensaje

}
else {

reenvía al node cuyo NodeId comparta el mismo prefijo de longitud con el key-id que mi my-id

}

[2] Clase de la Api de Pastry que genera NodeIds, tanto aleatorios como designados manualmente por nosotros.

[3] Clase de la Api de Pastry que dada una Id, enruta el mensaje mediante los mecanismos que en este documento explicamos.

1.3.5. Añadiendo nodos en Pastry

Cuando entra un nodo nuevo en Pastry, se hacen un seguido de comprobaciones que listaremos a continuación.

Antes de nada, pondremos una serie de abreviaturas para hacer más entendible la explicación:

- N = nodo añadido
- C = nodo cercano a X (en proximidad de red)
- V = nodo cercano numéricamente a N

Cuando entra el nodo nuevo en la red, éste ha de ser añadido en la tabla de ruta de N, de la siguiente forma:

- leafset (N) = leafset (V)
- neighborhood – set (N) = neighborhood – set (C)
- routing table N, columna i = routing table Xi, columna i (donde Xi es el i-ésimo nodo encontrado a lo largo de la ruta de C a V.

Una vez realizados todos estos cálculos y procesos, N notifica a todos los nodos con leafset (N), que actualizen sus estados, y por lo tanto, que le agreguen como nuevo nodo en la red.

1.3.6. Eliminando nodos en Pastry

Los neighboring nodes (nodos vecinos en inglés), en el espacio de NodeId se envían periódicamente entre si, mensajes de keepalive[4], si un nodo no contesta a estos mensajes, se le eliminará actualizando las tablas.

Si uno nodo actualiza sus tabla, automáticamente ha de notificárselo a sus vecinos.

1.3.7. Distributed Hash Table (DHT)

DHT se trata de una tabla con información sobre los nodos que han ingresado en la red Pastry.

Para poder almacenar toda la información necesaria para poder encaminar mensajes, necesitamos almacenar información de los nodos, donde se encuentran, que identificadores tienen, que key, etc, por ello hacemos uso de las tablas DHT.

Una vez creados los nodos, estos se guardan en la DHT, como objetos de datos asociados a una key, que en nuestro caso, está generada por el hash de los nombres de usuarios de los participantes en la red, y por tanto, del NodeId, ya que así que se ha diseñado.

[4] Mensaje de información de que el nodo sigue activo y por lo tanto no ha de ser eliminado de la tabla de ruta.

Para añadir los objetos en la DHT usamos el método *insert* de la librería de Past[5]. Si queremos consultar identidad de un usuario en la tabla DHT lo haremos mediante el método *lookup(key)*.

Una vez añadido, obtenemos una tabla con una relación de identidad del usuario y key asignada, de esta forma vamos creando las tablas de ruta locales de cada nodo, que usaremos para encaminar mensajes por la red Pastry.

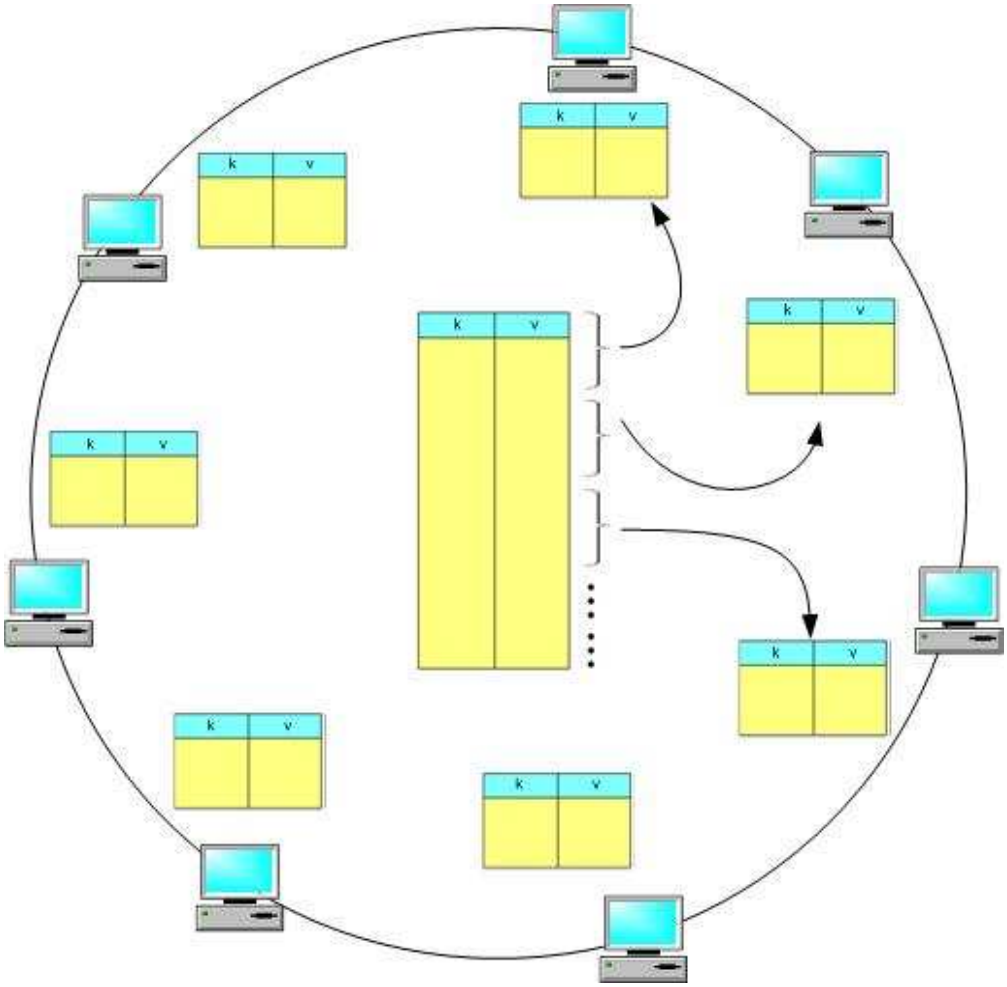

Fig 1.7 Esquema DHT en red Pastry

En la figura 1.4 podemos observar que la DHT a nivel general, está formada por DHTs locales, que posee cada nodo. Todas juntas forman una DHT que visualiza la red entera, pero que es virtual, ya que a modo global nadie la posee, únicamente por partes, a nivel local, como se muestra en la figura.

[5] Sistema de almacenamiento, del que se habla en el punto 1.3.6.

Usaremos los métodos *put* y *get*, para poner y conseguir archivos de usuarios que estén dentro de la DHT. Para alcanzar dichos usuarios, viajaremos usando su key como medio de localización, haciendo una consulta en la DHT local, hallaremos el camino para llegar al nodo, en caso contrario, haremos uso del algoritmo de encaminamiento.

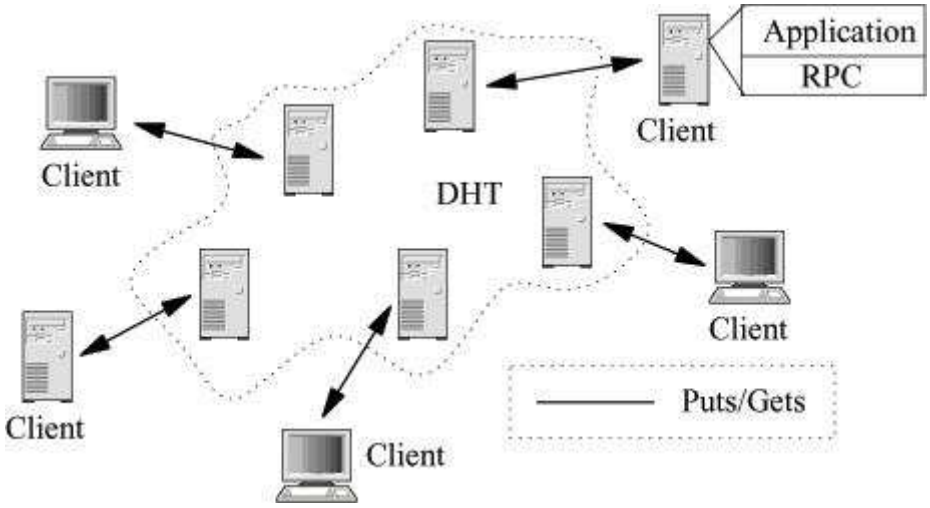

Fig 1.8 Esquema de acceso a archivos

1.3.6. Past

La DHT la hemos creado a partir de la aplicación Past, de la Api de Pastry.

Past, es una aplicación de almacenamiento global peer-to-peer persistente, que nos permite crear la tabla distribuida, añadiendo los nodos con su información, listar nodos, e introducir o recoger datos.

Los nodos entre si, pueden intercambiar información de forma segura y cooperativa con los comandos *put* y *get* que antes hemos comentado en DHT.

Cualquier host conectado a Internet puede actuar como nodo Past instalando el software apropiado. La colección de nodos Past forma una red overlay.

Como mínimo, un nodo Past actúa como punto de acceso para un usuario.

Past se usa para almacenamiento persistente, escalable y seguro. Tiene una gestión de memoria externa, que hace que se repliquen los datos guardados como si de alamacenes de archivos se tratasen.

Se hacen copias de seguridad en los nodos próximos al *fileId* para mantener una redundancia de los archivos en la red, de este modo se descargan de tráfico los nodos con determinada información y se balance mejor la carga de archivos, además de conseguir mayor seguridad ante perdidas.

Pastry asegura en encaminamiento eficiente y seguro, se asegura de que las peticiones del cliente estén encaminadas fiablemente a los nodos apropiados.

El cliente hace una petición para recuperar un archivo y poder encaminarlo, a un nodo que esté cercano en la red[6] al cliente que publicó la petición, entre los nodos vivos que almacenan el archivo solicitado. El número de nodos Past travesados, así como el número de los mensajes intercambiados, es logarítmico en el número total de nodos Past en el sistema.

Para recuperar un archivo en Past, un cliente debe saber su *fileId* y, en caso de necesidad, su llave del desciframiento.

A continuación veremos qué es el *fileId*.

1.3.6.1 El fileId

Mientras que Past ofrece servicio de almacenaje persistente, su semántica se diferencia de la de un archivo de sistema convencional. Los archivos almacenados en Past se asocian a un fileId casi único que se genera a la hora de la inserción de un archivo en Past.

Por lo tanto, los archivos almacenados adentro Past son inmutables puesto que un archivo no se puede insertar varias veces con el mismo fileId, esto hace que el fileId sea único para cada archivo en Past, lo diferencie y por tanto se le localice con facilidad.

Los archivos se pueden compartir distribuyendo el fileId y en caso de necesidad, una llave del desciframiento.

1.3.6.2. Métodos para clientes Past

fileId = Insert(name, owner-credentials, k, file)

- Almacena en un archivo un número especificado usuarios de *k* nodos diversos dentro de la red Past.
- La operación produce un identificador 160-bit (fileId) que se puede utilizar posteriormente para identificar el archivo.

fileId = SHA-1(name, owner-credentials, random number)

- El fileId se calcula como el hash (SHA - 1) del nombre del archivo, de la clave pública del dueño.

file = Lookup(fileId)

[6] Se mide esta proximidad por la métrica de la red en saltos de nodo a nodo.

- Recupera una copia del archivo, normalmente de un nodo próximo.

Reclaim(fileId, owner-credentials)

- Reclama el almacenaje ocupado por las copias del archivo identificado por el fileId.

1.3.7. Ejemplo de funcionamiento de Pastry

Pero, ¿cómo traza las keys con los NodeIds Pastry?

Veamos un ejemplo, de cómo funciona una red Pastry.

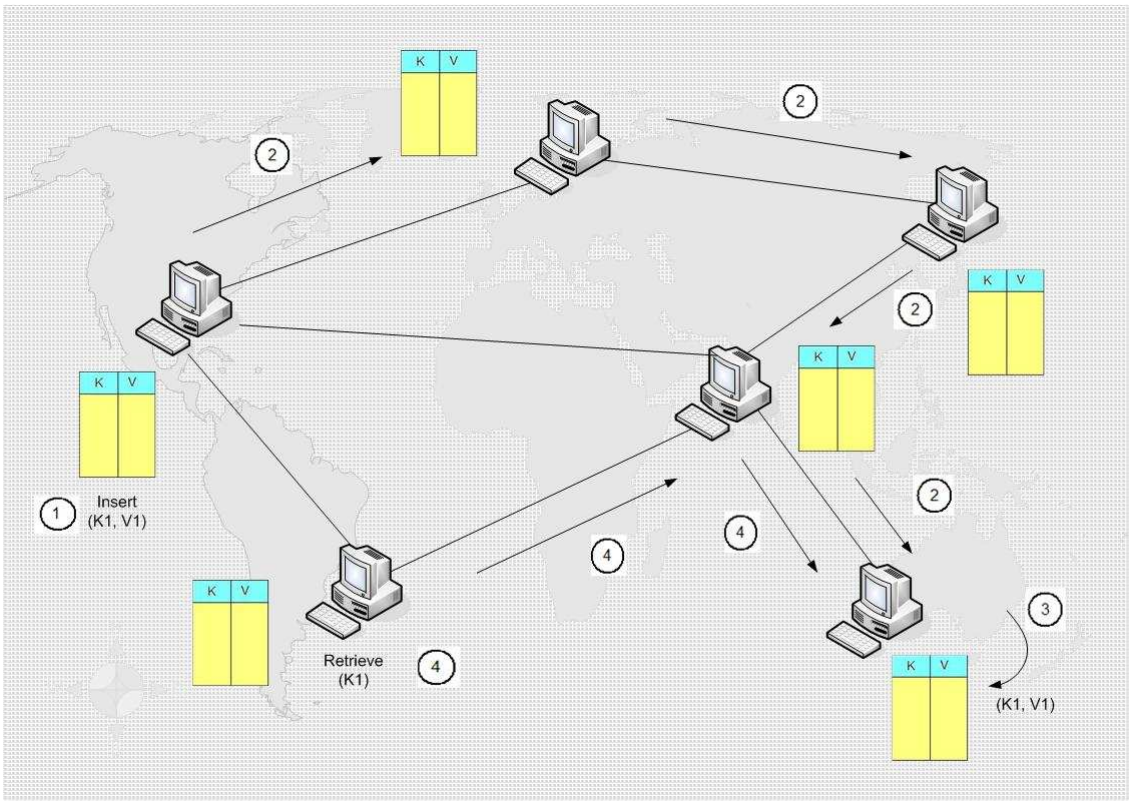

Fig. 1.9 Aqui observamos un ejemplo donde vemos por pasos como se guardan las keys referentes a los nodos.

En la figura 1.9 apreciamos el proceso que se sigue para que un nodo pueda encaminar sus mensajes a partir de la key del destinatario.

En el punto 1, vemos como un nodo inserta su key y su valor en un nodo destino, pasando por un total de 3 nodos intermedios.

Una vez llegado al destino, vemos como en el punto 3, el nodo destino complementa en su DHT el valor de la key del nodo origen, y la almacena.

Por ultimo, en el punto 4, otro nodo, quiere encaminar hacia el nodo del inicio del ejemplo, para ello recoge la key, del nodo que la ha guardado en su DHT en el punto 3, de esta forma, consigue la key, y puede encaminar hacia el nodo 1.

1.4. Seguridad en Pastry

1.4.1. Introducción

Como todo sistema que se precie, el nuestro necesita de una seguridad. Hay muchos datos que necesitan ser protegidos, para garantizar una privacidad y una integridad del sistema.

Hemos hecho uso de varios mecanismos de seguridad, tanto a nivel de robustez para conseguir integridad en los datos, como en privacidad.

Antes, explicaremos un par de pinceladas sobre términos de seguridad para poder comprender mejor qué hemos hecho y como.

1.4.2. Funciones de Hash.

Podemos imaginarnos una Función de Hash como una caja negra con una entrada y una salida. Por la entrada, introducimos un mensaje de longitud variable y por la salida obtenemos un código (hash) de longitud fija, equivalente a un número de unos 50 dígitos decimales.

Dos mensajes diferentes generan *"hashes"* diferentes, a priori, pero accidentalmente dos mensajes pueden tener el mismo hash, con una probabilidad de 10-50, pero lo que debe garantizar el algoritmo es la imposibilidad de encontrar un segundo mensaje con igual hash que otro.

Además, dado un mensaje, es fácil y rápido calcular su *"hash"*, pero es imposible reconstruir el mensaje original a partir del código resumen (hash). Es decir, el paso inverso es computacionalmente imposible de calcular, por lo que se considera un sistema seguro.

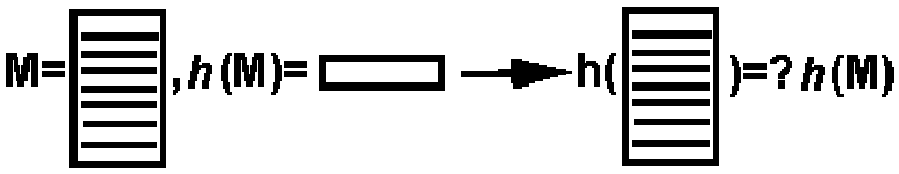

Fig. 1.10 Hash del fichero M

1.4.3. SHA (Secure Hash Algorithm)

SHA es un conjunto de algoritmos criptográficos, creados por la Agencia Nacional de Seguridad, y publicados por el NIST (Nacional Institue of Standards and Technology).

Existen diferentes tipos de algoritmos SHA, pero nosotros nos centraremos en el SHA-1.

SHA-1 produce una salida resumen de 160 bits, es decir, cada mensaje "hasheado" genera un número de 160 bits.

Teniendo en cuenta que existen infinito número de mensajes que se resumen en cada valor posible, hay infinito número de posibles colisiones. Pero debido a que el número de posibles salidas resumen es también muy grande, la posibilidad de encontrar uno por azar es increíblemente pequeña (uno en 2^80, para ser exactos).

1.4.4. Cifrado de Identificadores

Requerimos de seguridad para hacer nuestro sistema fiable y a la vez robusto, para ello, hacemos gala de varias técnicas, entre ellas el cifrado de los identificadores de cada usuario.
Básicamente, se trata de la primera medida de seguridad que usamos en el arranque del sistema, y sirve para proteger la naturaleza del identificador del usuario enmascarándolo, permite autenticar un mensaje y garantizar su integridad.

La criptografía asimétrica permite identificar al emisor y al receptor del mensaje. Para identificar el mensaje propiamente dicho, se utilizan las llamadas funciones resumen (en inglés, hash).

Para ello, ciframos los identificadores, mediante una función de hash SHA-1, antes de nada, comentaremos, que es un hash, en qué consiste el SHA-1.

1.4.5. Nuestra aplicación de cifrado.

Hemos creado un sistema, para cifrar mediante una función resumen SHA-1, los identificadores (ID) de cada usuario, que acceda a la red Pastry, y se le asigne un nodo.

Pastry tiene la capacidad de auto-asignar ID una vez nos conectamos a la red, esta asignación la realiza de forma aleatoria, generando un número de 160 bits en hexadecimal.

Nosotros no queríamos generar ID aleatorios, ya que cada ID diferenciará a cada usuario, siendo su "nickname", para así, poder ser identificado dentro de la red.

FreePastry ya esta preparado para asignar ID manualmente, mediante el método newNode.

newNode(NodeHandle bootstrap, NodeId nodeId);

Este método abstracto, permite tanto auto-generar ID como asignarlas manualmente, dependiendo de los parámetros que le pasemos, si dejamos en blanco el campo de NodeId, él solo generara aleatoriamente un ID para ese usuario, en cambio, si le damos uno, lo cogerá y lo asignará manualmente, esta ultima opción ha sido la que hemos utilizado.

El proceso es bastante sencillo, consiste en hacer un hash del nombre de usuario, que el propio usuario elija para identificarse dentro de la red Pastry. Este nombre de usuario, lo hemos pasado a través de un método, que hace el hashing en SHA-1, y lo convierte en una cadena de 160 bits.

Una vez obtenido dicho hashing, se lo pasamos al NodeId, y creamos un nodo nuevo para ese usuario.

Cada usuario, es único e intransferible, así que antes de hacer el hashing y por lo tanto, antes de añadirlo a la red, comprobamos si ya está en uso dicho ID.

1.4.6. Cifrado de mensajes

Otro aspecto en la seguridad de la aplicación, es garantizar la integridad, y la confidencialidad de los mensajes que se intercambien los usuarios.

Para ello, hemos usado un cifrado asimétrico RSA, el cual usa claves públicas.

Para comprenderlo un poco mejor, hablaremos por encima en que consiste un sistema de clave pública, y más concretamente, como funciona el RSA.

1.4.7. Sistema de clave pública

El cifrado asimétrico, o de clave pública, está basado en la adulteración del mensaje que vamos a enviar cifrado, mediante una clave privada, que generaremos a partir de una clave pública, dicho de otra forma, la criptografía de clave pública esta basada en el uso de un par de claves que cumplen, entre otros requisitos, que lo que somos capaces de cifrar con una de ellas, somos capaces de descifrarlo con la otra y sólo con ella.

Una de las claves solo está en poder del propietario, que debe conservarla de forma segura, y se denomina *clave privada.*

La otra clave es publicada para que la conozcan todos aquellos que quieran comunicarse de modo seguro con el propietario mencionado, a esta última se la denomina *clave pública.*

Para enviar un mensaje cifrado, sólo tendremos que coger la clave pública del receptor, cifrar el mensaje con dicha clave, transmitirle el mensaje, y el receptor lo descifrará con su clave privada.

Solamente él será capaz de descifrarlo, ya que es el único poseedor de la clave privada que es capaz de descifrar mensajes, cifrador con su clave pública.

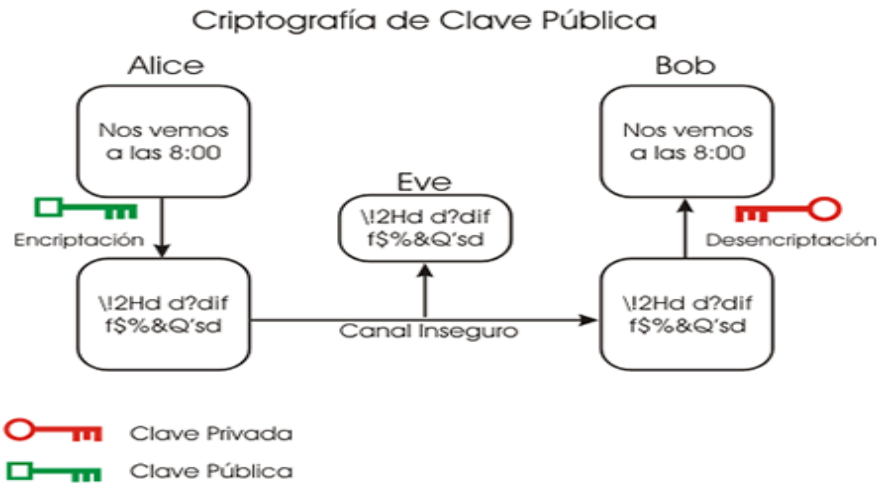

Fig.1.11 Ejemplo de cifrado de clave pública

1.4.8. RSA

Nosotros hemos usado RSA como sistema de clave pública, su funcionamiento consiste en los siguientes pasos:

1. Seleccionar dos números primos *p* y *q* de manera que $p \neq q$.
2. Calcular n = p·q.
3. Calcular $\phi(n) = (p-1)\cdot(q-1)$.
4. Seleccionar un entero positivo *e* tal que el $1 < e < \phi(n)$, tales que *e* y φ(*n*) sean primos entre si.
5. Calcular *d* tal que $d \equiv 1 (\mathrm{mod}\, \phi(n))$.
6. La clave privada será *d* y la clave pública será *e*.

1.4.9. Nuestra aplicación de clave pública.

Para poder aplicar RSA, necesitábamos generar los valores *p* y *q*, para cada usuario que entrase en la red, así que creamos un método, que auto-genera dichos valores aleatoriamente, así como un valor *e,* y los guarda en un fichero de texto, con el nombre del usuario creado.

Cuando se es preciso hacer uso del RSA para enviar mensajes, leemos dichos valores, y hacemos correr el algoritmo para que cifre y posteriormente descifre.

Al arrancar la aplicación, se comprueba si ya existen los ficheros correspondientes a los ID que se introducen en la red, de ser así, se leerá directamente del fichero, sino se creará como hemos explicado arriba.

Capítulo 2. SIP

2.1. Introducción

Session Initiation Protocol (SIP o Protocolo de Inicialización de Sesiones) es el nombre de un protocolo de comunicaciones, desarrollado por el IETF[7] con la intención de ser el estándar para la iniciación, modificación y finalización de sesiones interactivas de usuario donde intervienen elementos multimedia como el video, voz, mensajería instantánea, juegos online, etc.

SIP se usa para la señalización de la VoIP, por ello hacemos uso de él en este proyecto.

2.2. Características de SIP

Los clientes SIP, pueden trabajar sobre TCP y UDP, dependiendo de las necesidades que tengan. En ambos protocolos, dichos clientes usan el puerto 5060 para conectar con los servidores SIP, ya que necesitan establecer comunicación con un proxy SIP para establecer las comunicaciones entre clientes.

En nuestra aplicación eso no ha sido necesario, ya que Pastry se ha encargado de dichas comunicaciones, pero de esto hablaremos más adelante.

Todas las comunicaciones de voz/video van sobre RTP (Real Time Protocol), ya que SIP no se encarga de transportar el flujo de datos de las comunicaciones, únicamente establece sesiones, conecta usuarios y mantiene información de presencia de los mismos.

Un objetivo de SIP fue aportar un conjunto de las funciones de procesamiento de llamadas y capacidades presentes en la red pública conmutada de telefonía. Así, implementó funciones típicas que permite un teléfono común como son: llamar a un número, provocar que un teléfono suene al ser llamado, escuchar la señal de tono o de ocupado. La implementación y terminología en SIP son diferentes.

SIP funciona en colaboración con otros muchos protocolos pero solo interviene en la parte de señalización al establecer la sesión de comunicación. SIP actúa como envoltura al SDP[8], que describe el contenido multimedia de la sesión, por ejemplo qué puerto IP y códec se usarán durante la comunicación, etc. En un uso normal, las sesiones SIP son simplemente flujos de paquetes de RTP. RTP es el portador para el actual contenido de voz y video.

[7] (**I**nternet **E**ngineering **T**ask **F**orce) es una organización internacional, creada en EEUU en 1986, que tiene como objetivos el contribuir a la ingeniería de Internet, actuando en diversas áreas, tales como transporte, encaminamiento, seguridad.

[8] Session Description Protocol, es un protocolo para describir los parámetros de inicialización de los flujos multimedia.

2.3. Arquitectura y funcionamiento del SIP

SIP sigue un modelo cliente-servidor, donde un Terminal cualquiera puede actuar como cliente y servidor simultáneamente, por un lado generando peticiones SIP *request* (cliente) y por otra procesando peticiones SIP *request* y generando respuestas SIP *response*.

También entran en juego los llamados Agentes de Usuario (UA), aplicaciones que interactúan con el usuario final. Estos UA, están compuestos por un Cliente Agente de Usuario (UAC) y un Servidor Agente de Usuario (UAS).

- El UAC inicia las peticiones SIP actuando como agente de usuario llamante, dicho UAC inicia pues la llamada.

- El UAS actúa como agente de usuario llamado.

Los clientes SIP se direccionan a partir de las SIP – URL, direcciones parecidas a las URL de HTML.

Dichas direcciones son únicas por cada usuario e independientes de su localización, aquí vemos su formato:

sip: [user:passwd@] hostname | ipv4addr | ipv6addr [:port; params]

Por ejemplo:

sip: alice@80.65.113.243
sip: alice:password@ 80.65.113.243:5060

2.3.1. Mensajes SIP

Los mensajes SIP los forman los clientes SIP mediante peticiones de distinto calibre, y devueltos por los servidores en forma de respuesta.

Dichos mensajes tienen una estructura parecida a la de http, con una primera línea, una cabecera y finalmente un cuerpo del mensaje opcional.

Los mensajes se usan para el intercambio de información necesaria entre clientes, para inicializar llamadas, agregar usuario a la lista de contactos, eliminar usuarios, mostrar la presencia de los mismos, finalizar llamadas, enviar confirmaciones, etc.

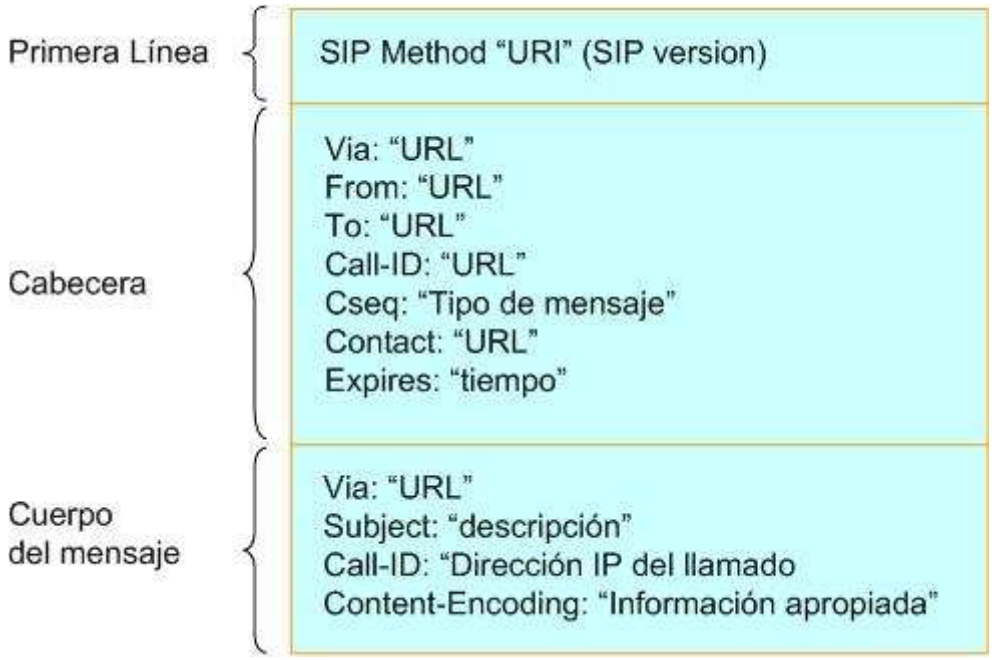

Fig 2.1 Estructura de los mensajes SIP

Aquí tenemos la descripción de todos los parámetros de la estructura del mensaje SIP:

- SIP Method: URL del destino
- Via: Indica la ruta tomada por la petición
- From: Origen de la petición
- To: Receptor de la petición
- Call – ID: Identifica los mensajes de una conversación
- Cseq: Contiene la petición del mensaje de solicitud y el número de secuencia.
- Contact: URL de contacto para comunicaciones adicionales.
- Expires: Identifica la fecha y hora que expira el mensaje.
- Subject: Naturaleza de la llamada.
- Content – Encoding: Indica el mecanismo de decodificación que debe ser aplicado.

2.4. Nuestra aplicación con SIP

Una vez tenemos los usuarios en sus nodos, identificados, activos y listos gracias a Pastry y todo lo explicado hasta ahora, necesitamos establecer comunicación entre ellos, iniciar una sesión de flujo RTP con Telefonía IP.

Hemos programado un módulo de SIP, que se adecua a nuestras necesidades. Sabemos donde están los usuarios gracias a Pastry, los tenemos localizados, y

tenemos un algoritmo de encaminamiento que nos permite enviar mensajes de un nodo a otro, de una forma segura gracias al RSA.

Estos mensajes de los que tanto hemos hablado hasta ahora, son mensajes SIP.

Mensajes de inicialización de sesiones, de descubrimiento de usuarios, de presencia, de finalización de comunicaciones…así funciona nuestro sistema.

2.4.1. ¿Para qué usamos SIP?

SIP es un protocolo de inicialización de sesiones, la finalidad de nuestro sistema es la de comunicar usuarios mediante una conversación en Telefonía IP y para esto SIP encaja a la perfección.

Con nuestro sistema podremos observar qué usuarios tenemos en nuestra lista de contactos, en qué estado están, y si podemos inicializar una comunicación.

Gracias a SIP, podremos realizar llamadas a los usuarios, ver su estado de conexión, finalizar llamadas, etc.

Pero SIP solo se encarga de iniciar de señalizar, el encaminamiento, y transporte lo hemos realizado por debajo, con la arquitectura de la red Pastry, y las tablas de memoria DHT.

Tenemos los usuarios, cada uno en su nodo, los tenemos identificados en la DHT, autenticados, son capaces de enviarse mensajes de forma segura gracias al RSA, solo nos faltaba establecer la comunicación entre ellos, necesitábamos un mecanismo que nos estableciese una sesión con un usuario, que nos informase de los eventos que sufre cada usuario, y SIP nos proporciona todo esto, por todo ello, hemos usado SIP.

2.4.2. Mensajes SIP en nuestra aplicación

Como ya hemos explicado anteriormente, SIP hace uso de un listado de mensajes para hacer funcionar todo su sistema. En nuestra aplicación hemos hecho uso de los siguientes mensajes, para conseguir nuestro propósito.

Encaminándolos sobre Pastry, estos mensajes hacen posible que SIP funcione en nuestro sistema.

2.4.2.1. Register

Este mensaje permite que un usuario registre su dirección en un servidor SIP.

Los clientes se pueden registrar desde cualquier localización. El registro sólo es posible dentro de un dominio administrativo.

En nuestro sistema, el usuario que entra en la red, envía un SIP Register, con la intención de darse de alta en la red, si ese nombre de usuario ya está activo se le deniega el Register y ha de registrarse con otro nombre de usuario.

2.4.2.2. Subscribe

El mensaje Subscribe permite modificar o consultar el estado de un usuario.

Para controlar la presencia va muy bien, ya que la respuesta a este mensaje es el estado en el que se encuentran los usuarios, y con ello poder confeccionar las listas de usuarios *"buddy lists".*

2.4.2.3. Notify

La respuesta a un mensaje Subscribe normalmente es un mensaje Notify.

Este tipo de mensaje, notifica el estado de presencia del usuario, y transporta el estado en el que se encuentra el usuario.

Con la respuesta Notify se confecciona una lista de usuarios *"notify list".*

2.4.2.4. Invite

Mensaje que se envía para iniciar una comunicación.

Indica que un usuario o servidor está siendo invitado a participar en una sesión. Dentro de un mensaje Invite, se describe las sesiones utilizando SDP, para indicar que características tendrá la comunicación:

- Describe el media.
- Direcciones IP del llamante y del llamado.
- Localización del usuario y los códecs a utilizar.

2.4.2.5. Ack

Mensaje de confirmación de las solicitudes Invite.

2.4.2.6. Bye

Mensaje de cancelación de sesión, que envía el agente de usuario al servidor, y con ello liberar la llamada.

2.4.2.7. Ok

Mensaje de confirmación positiva, lleva el código de estado 200.

2.4.3. La Buddy List

El concepto de buddy list, es el de tener una “lista de amigos” diferente en cada usuario y personal, con el que poder tener un control en tiempo real del estado de todos ellos, para así poder comunicarte con ellos si están disponibles, o no, si no lo están, etc.

Mediante los mensajes Subscribe y las respuestas Notify podremos ir confeccionando nuestra buddy list dinámicamente, a partir de la información que iremos recibiendo de los usuarios de la red.

Fig 2.2 Aquí vemos el ejemplo de una Buddy List, en un programa de mensajería instantánea.

2.4.4. La Subscribe List

La subscribe list, es prácticamente igual que la buddy list, pero tiene la particularidad de que solamente registra y guarda a los usuarios que me te tienen añadido y no están conectados.

Hacemos uso de la buddy list, para observar qué usuarios están conectados, y podemos establecer comunicación en ese momento, pero para saber qué usuarios tenemos sin conexión necesitamos de la subscribe list para poder ser avisados de cuándo dichos usuarios estarán conectados, para así poder establecer comunicación con ellos si lo desearamos.

Para diseñar el programa, hemos hecho uso de ambas listas, en la subscribe list tenemos los usuarios añadidos y no conectados, y en la buddy list, los usuarios añadidos y conectados en este momento.

2.4.5. Funcionamiento del SIP en nuestra aplicación

El funcionamiento de nuestra aplicación en el apartado de SIP es el siguiente.

Cuando un usuario entra en la red, como hemos explicado en el Capitulo 1, lo primero que hará será identificarse con su nombre de usuario, una vez Pastry lo identifique y genere sus claves para RSA y compruebe que ese usuario está libre, le asignará un nodo dentro del espacio de claves de la red Pastry, como explicamos anteriormente y en ese momento entrará en juego el SIP.

Para explicarlo todo, pondremos el ejemplo donde Alice entra en la red, donde están Charlie y Bob. Cabe destacar, que todos los mensajes SIP se encaminan sobre Pastry.

Lo primero que hará Alice, será enviar un mensaje SIP Register al nodo donde se va a conectar. Este nodo consultará su DHT, y verá que Alice obviamente está desconectada, por lo que aceptará su entrada, y le enviará un mensaje de Ok.

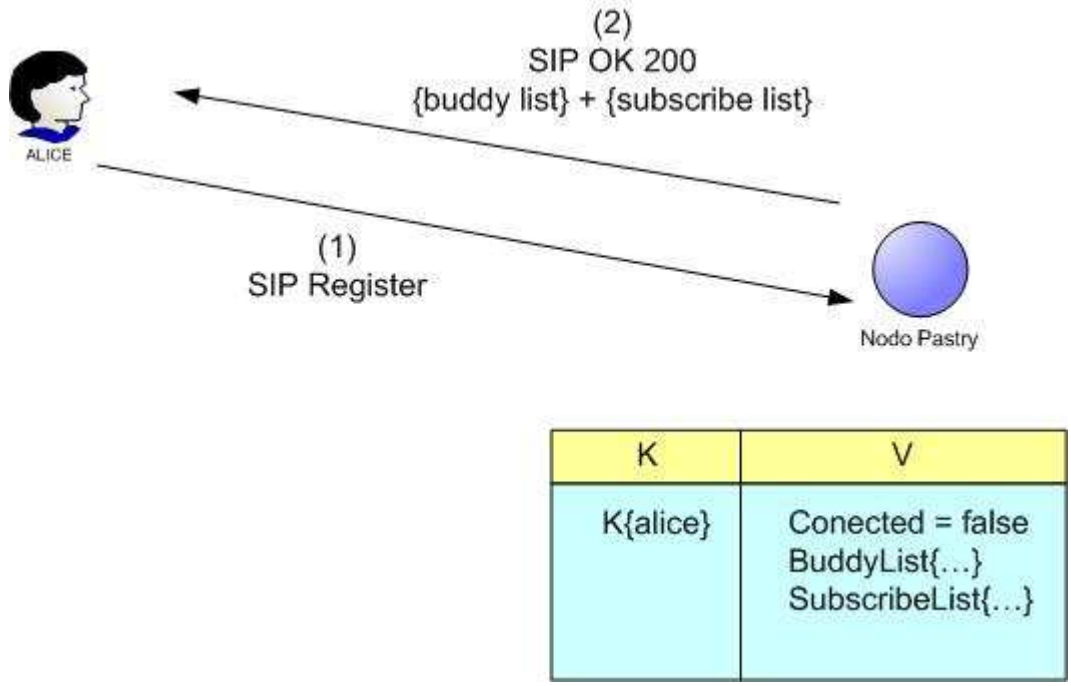

Fig. 2.3 Intercambio de mensajes SIP

Junto con este mensaje de Ok, se le enviará a Alice la buddy list y la subscribe list. En nuestro caso, Charlie y Bob estan desconectados, así que el estado de las tablas será el siguiente:

Subscribe List
Charlie
Bob

Buddy List
Alice

Fig 2.4 Estado de las listas

Alicie ya tiene las listas, y está dentro de la red activa, ahora envia un mensaje Subscribe a todos los usuarios de su subscribe list, en nuestro caso, se los envia a Charlie y a Bob.

Cuando Charlie o Bob se conecten, revisarán su subscribe list, verán que Alice está agregada, y le enviarán un notify a Alice comunicándola de que se han conectado. Si Alice estubiese desconectada, le enviarian un subscribe, se añadirian a la subscribe list de Alice, y en este caso, Alice seria la encargada de enviarles el notify a Charlie o a Bob cuando se conectase.

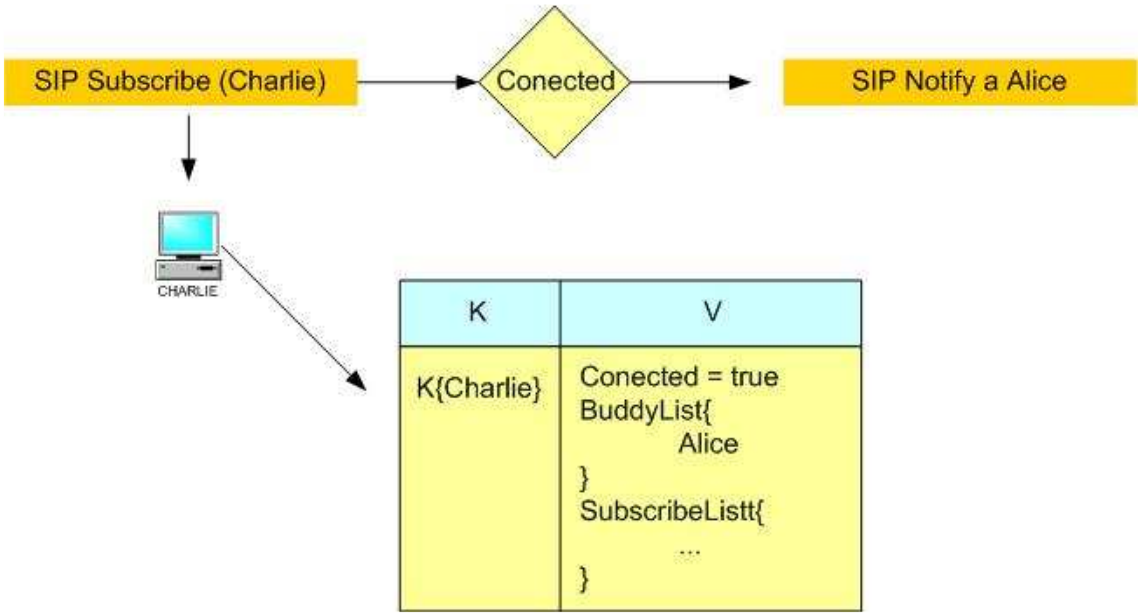

Fig. 2.5 Podemos observar la DHT de Charlie en el momento que recibe el Subscribe de Alice.

Si Alice quiere establecer comunicación con algún usuario, habrá de consultar su buddy list, y ver que usuario está conectado, y enviarle un mensaje Invite. Si este usuario lo acepta, se establecerá la comunicación.

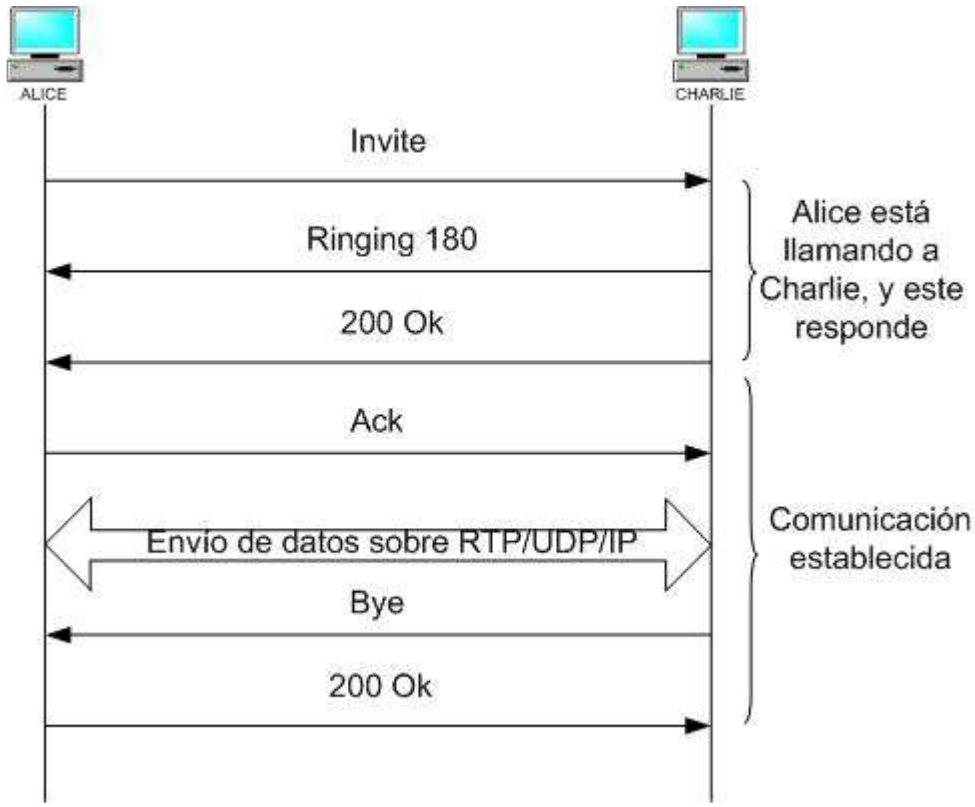

Fig 2.6 Esquema de inició de sesión SIP

Capítulo 3. JPS (JavaPastrySip)

3.1. Introducción. ¿Qué es JPS?

Para el estudio y desarrollo de este proyecto, hemos realizado el diseño y la implementación de un sistema que emule todo lo descrito en este documento.

Para ello, hemos hecho uso de Java, ya que nos proporciona una API muy bien documentada de Pastry, llamada FreePastry, de la misma forma que de SIP, gracias a la API de Jain Nist SIP.

Con ello y con los conocimientos adquiridos de Java durante el bloque de intensividad de sistemas telemáticos, hemos visto oportuna y correcta la elección de todas estas herramientas para el desarrollo de nuestra aplicación.

Pero, que es JPS?

JPS son las siglas de Java Pastry SIP, nuestra aplicación, la cual pasaremos a explicar con más detalle a lo largo de este tercer capítulo.

Como epílego, decir que JPS es un programa en Java, que simula una red Pastry de 4 nodos, dentro de un espacio circular de nodos, donde se asienta SIP, y permite la comunicación, establecimiento, presencia y manejo de los usuarios entre si, para poder estudiar el comportamiento de SIP sobre una red P2P como es la de Pastry.

3.2. Funcionamiento

Cuando arrancamos la aplicación vemos la siguiente pantalla:

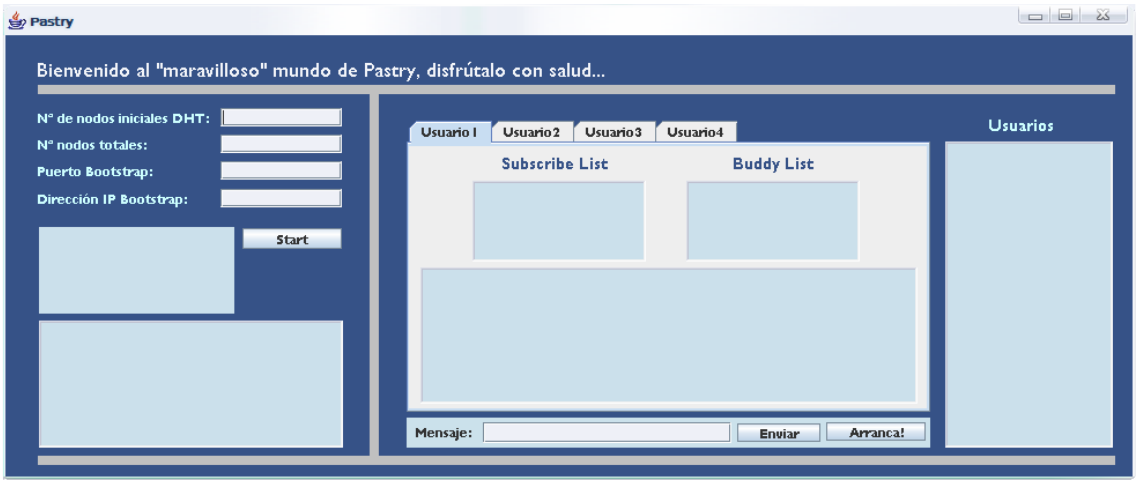

Fig 3.1 Imagen de JPS antes de arrancar los nodos.

Podemos observar que existen un total de 4 campos a la izquierda vacíos donde recogeremos información de arranque, abajo 2 cuadros de texto donde iremos viendo las incidencias y sucesos.

En el centro tenemos un conjunto de pestañas, una por usuario, junto con 2 listas por usuario (subscribe list y buddy list) junto con un cuadro de texto donde iremos viendo información de lo acontecido en tiempo real.

Por último, a la derecha vemos una lista también vacía, donde veremos en tiempo real el estado de todos los usuarios de la red.

3.2.1. El arranque

En la zona superior izquierda, podemos observar un conjunto de 4 áreas de texto en blanco, que serán de vital importancia para poder configurar el programa y obtener su arranque con éxito.

Los 4 campos requeridos son:

- Nº de nodos iniciales DHT: Número de usuarios en el sistema.
- Nº de nodos totales: Número de usuarios on-line del sistema.
- Puerto Bootstrap: Puerto del nodo origen.
- Dirección IP Bootstrap: Dirección IP del nodo origen.

Debajo, veremos un botón de Start, que una vez introducidos todos los campos correctamente, procederá al arranque del sistema.

Aquí vemos una imagen del proceso.

Fig 3.2 Imagen del arranque de JPS.

Una vez cargados los datos, el programa accederá a la base de datos SQL, mediante las llamadas JDBC de Java, y se descargará la información de los usuarios que formarán parte del sistema.

Una vez descargada la información mediante la clase SQL.java, mapearemos la dicha información a la tabla DHT asociada al sistema, guardando así todos los datos de cada usuario, siguiendo el patrón anteriormente descrito de “clave-valor”.

Con la información guardada en memoria de cada usuario, el sistema procederá a insertar cada usuario, dentro del espacio circular de memoria de la red Pastry, asignándole a cada uno un IdNode de 128 bits.

Dicho IdNode, se hasheará como anteriormente hemos explicado, mediante una función de hash SHA-1, pasándo dicho identificador a una longitud de 160 bits.

Con todo ello, introduciremos a cada usuario en la red Pastry, una vez allí, Pastry se encargará de realizar los enrutamientos mediante los mecanismos anteriormente descritos, en el capítulo 1 de este documento.

Con los nodos en la red, pasamos a arrancar los módulos SIPListener del sistema, entra en el juego SIP.

Fig 3.3 Imagen del arranque de los módulos SIPListener en JPS.

Dependiendo de qué usuarios estén on-line y cuales no, se procederá a rellenar las subscribe list y buddy list de cada usuario, correspondientemente.

En el cuadro de texto inferior izquierdo, se mostrarán todos los eventos genéricos de la aplicación, y uno de ellos es el intercambio de mensajes SIP, como hemos podido apreciar en la figura 3.3.

En el centro de la aplicación, tenemos las pestañas de los usuarios, tantas como usuarios estén en la red, ya estén on-line, o off-line.

Para nuestra simulación, hemos visto suficiente fijar el número de usuarios totales de la red en 4, ya que todas las pruebas hechas con dicho número de usuarios, nos han parecido suficientes y validas.

Por usuario tenemos una pestaña, y por pestaña tenemos la siguiente información:

- Subscribe List: Muestra todos los usuarios de la subscribe list.

- Buddy List: Muestra todos los usuarios de la buddy list.

- Cuadro de texto: Irán apareciendo diferentes mensajes propios e independientes de cada usuario, como mensajes enviados, recibidos, llamadas enviadas, recibidas, etc.

- Cuadro de mensaje: Es própio, e independiente en cada usuario, allí el usuario escribirá un mensaje al usuario que haya decidido, clicando anteriormente el usuario destino, en la buddy list.

- Botón de enviar mensaje.

- Botón de realizar llamada: Se enviará un mensaje Invite de SIP, al usuario destino preseleccionado en la buddy list, intentando establecer una comunicación RTP.

- Botón de arrancar: Si el usuario de la pestaña, está off-line, se conectará a la aplicación, una vez pulsado el botón de arranque, mirará su subscribe list, y les enviará mensajes Notify de SIP a cada uno de los usuarios de su subscribe list, avisandoles de que su estado ahora es on-line. Si un usuario ya conectado pulsa el botón, aparecerá un mensaje avisando de que ya se encuentra on-line.

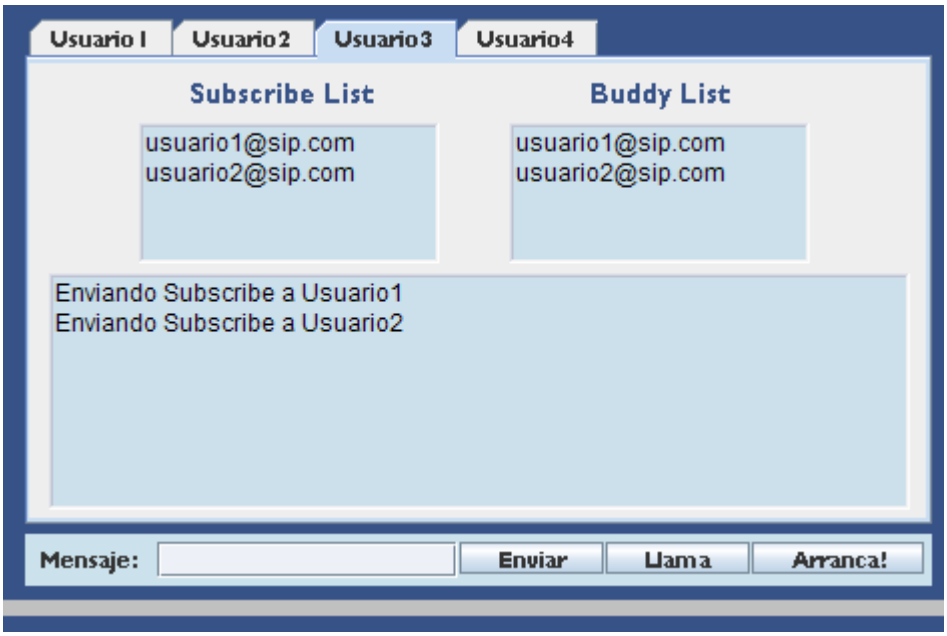

Fig 3.4 Imagen de la pestaña del usuario 3, enviándole subscribes a los usuarios de su subscribe list.

Por último tenemos a la derecha, la lista genérica de usuarios en el sistema, donde podemos apreciar en tiempo real, el estado de todos y cada uno de los usuarios.

Fig 3.5 Imagen de la lista de usuarios del sistema.

3.3. Diagrama de clases

3.3.1. Funcionamiento

El sistema que hemos creado tiene una serie de clases, clasificadas en un total de 5 paquetes, cada uno englobando a un numero de clases que tienen relación directa entre ellas.

Todo junto forma nuestro sistema, el que se compone de los siguientes paquetes:

- Aplicación
- Algoritmos
- Gráficos
- Mensajes
- SIP

El main de arranque está situado en el paquete Aplicación, más concretamente en la clase Inicio.

Cuando la clase Inicio arranca su main, llama a una serie de clases que van inicializando los distintos módulos de los que se compone el programa, y son necesarios para realizar las distintas tareas de las que se compone el sistema.

La primera labor que realiza una vez recogidos los parámetros de inicio de los que anteriormente hemos hablado, es llamar a un método de la clase MetodosInicio, alojada en el mismo paquete Aplicación. Dicha clase se encarga de conectar con la base de datos, descargar la información y arrancar los módulos de SIP.

Posteriormente, se llama a la clase de Pastry, y se conectan a la red los usuarios. Seguidamente llamamos a la clase Presencia, del paquete Algoritmos y inicializamos todo el sistema de SIP.

Para comprender todo un poco mejor, analizaremos paquete a paquete, todas sus clases, mediante la ayuda de diagramas de clases.

3.3.2. Aplicación

El paquete Aplicación, contiene un total de 4 clases:

- Inicio
- Pastry
- MetodosInicio
- Usuario

A continuación explicaremos con cierto detalle el uso y contenido de cada una de ellas.

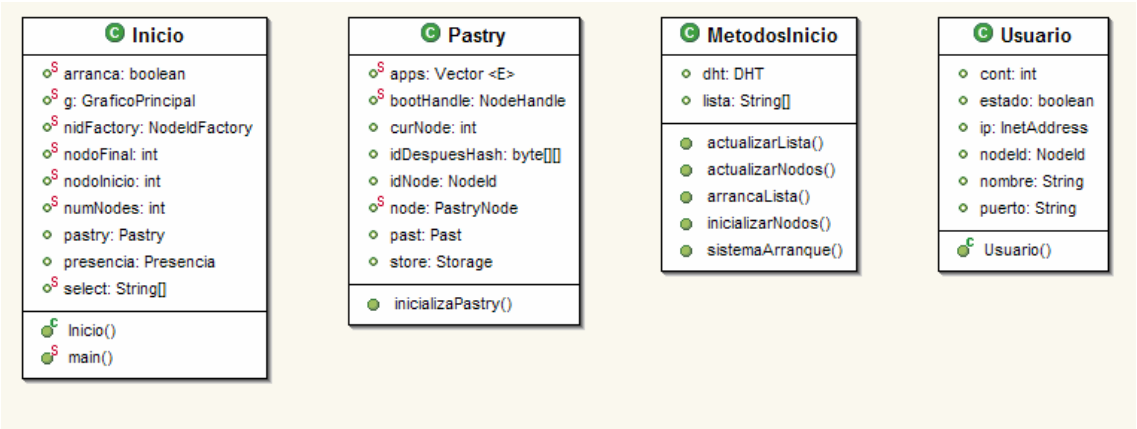

Fig 3.5 Diagrama de clases del paquete Aplicación.

En la clase Inicio tenemos dos métodos esenciales, Inicio y Main.

Inicio simplemente engloba todas las acciones que ha de hacer el main al arrancar, que son las que anteriormente hemos comentado en líneas anteriores (arrancar parámetros de inicio, inicializar Pastry y arrancar modulos de SIP).

Cuando el main llama al metodo de inicializaPastry, está instanciandolo de la clase Pastry, que es la siguiente que pasamos a comentar.

Esta clase Pastry, tiene todos los parámetros necesarios para poder coger los usuarios que se han conectado al sistema, y poder asignarles nodos dentro del espacio de claves de la red Pastry.

Dentro del método, hacemos uso de la función de hash SHA-1 para cifrar los usuarios, de la que hablaremos posteriormente.

La siguiente clase a analizar es la de MetodosInicio, clase que sólo se usa en el arranque, de ahí su nombre de “inicio”, y que se encarga de descargar la información de la base de datos y realizar todas las tareas que Pastry necesita sobre sus nodos.

Por último tenemos la clase Usuario, que simplemente se encarga de inicializar un usuario con todos sus parámetros y características.

3.3.3. Algoritmos

El paquete Algoritmos tiene un total de 4 clases:

- Presencia
- SQL
- DHT
- SHA

Este paquete alberga las clases que tienen funcionalidad independiente las unas con las otras. Es decir, hemos recogido en un paquete las 4 clases que se encargan de 4 tareas distintas, de diversas funcionalidades del sistema.

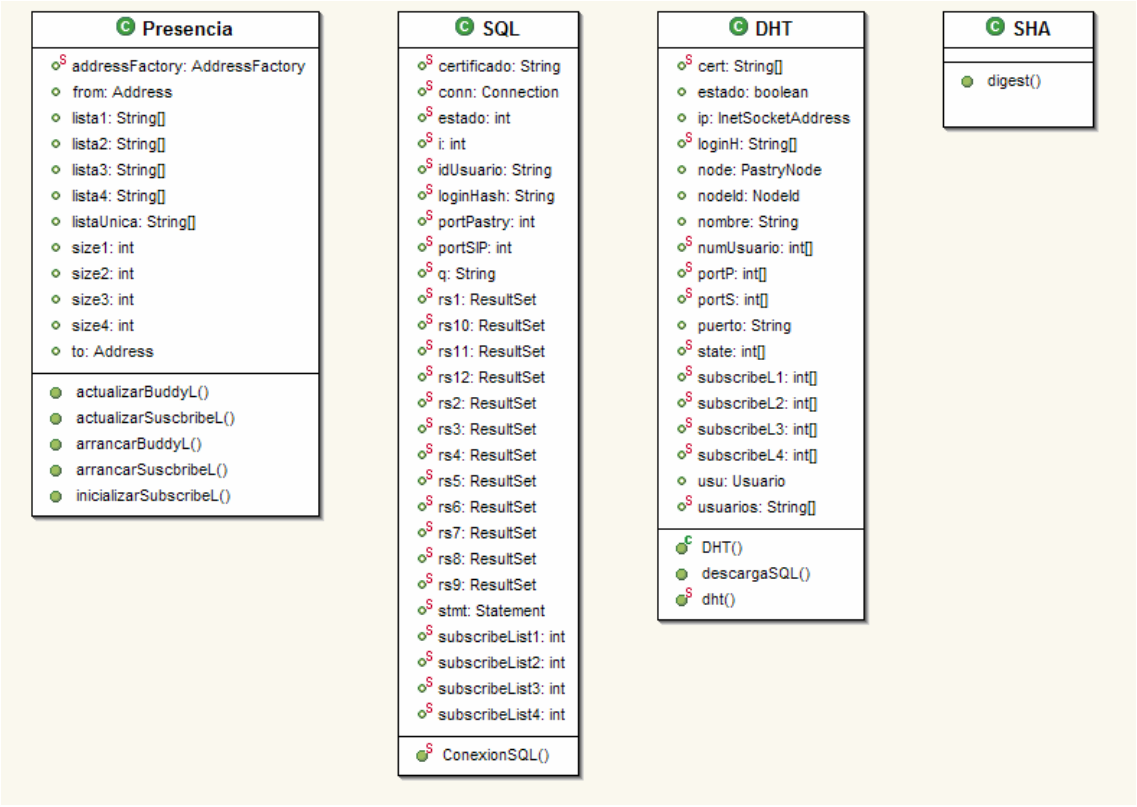

Fig 3.6 Diagrama de clases del paquete Algoritmos.

En la clase Presencia, tenemos todos los métodos que usamos conjuntamente con SIP, para detectar la actividad y el estado de todos los usuarios del sistema. Tenemos métodos de arranque e inicialización de listas de usuarios, y métodos que permiten el descubrimiento dentro del sistema, hacia el resto de usuarios, como puedan ser eventos de conexión al sistema, envío de mensajes SIP, etc.

En la clase SQL, tenemos todo lo necesario para establecer conexión con la base de datos, y descargar toda su información.

Una vez descargada dicha información, hemos de mapearla en memoria como en líneas anteriores hemos descrito, y eso lo logramos mediante la clase DHT, que lee de SQL y mapea la información a la Distributed Hash Table.

Finalmente encontramos la clase SHA, que es un singleton[9], que se encarga de hacer el hash SHA-1 de los ID de usuario, como ya explicamos en el capítulo 1.

[9] El patrón de diseño singleton (instancia única) está diseñado para restringir la creación de objetos pertenecientes a una clase o el valor de un tipo a un único objeto. Su intención consiste en garantizar que una clase sólo tenga una instancia y proporcionar un punto de acceso global a ella.

3.3.4. Gráficos

El paquete Gráficos, consta de una única clase GraficoPrincipal, que lo engloba todo, tanto a nivel visual (botones, pestañas, listas, etc), como a nivel lógico (mensajes, botones, etc).

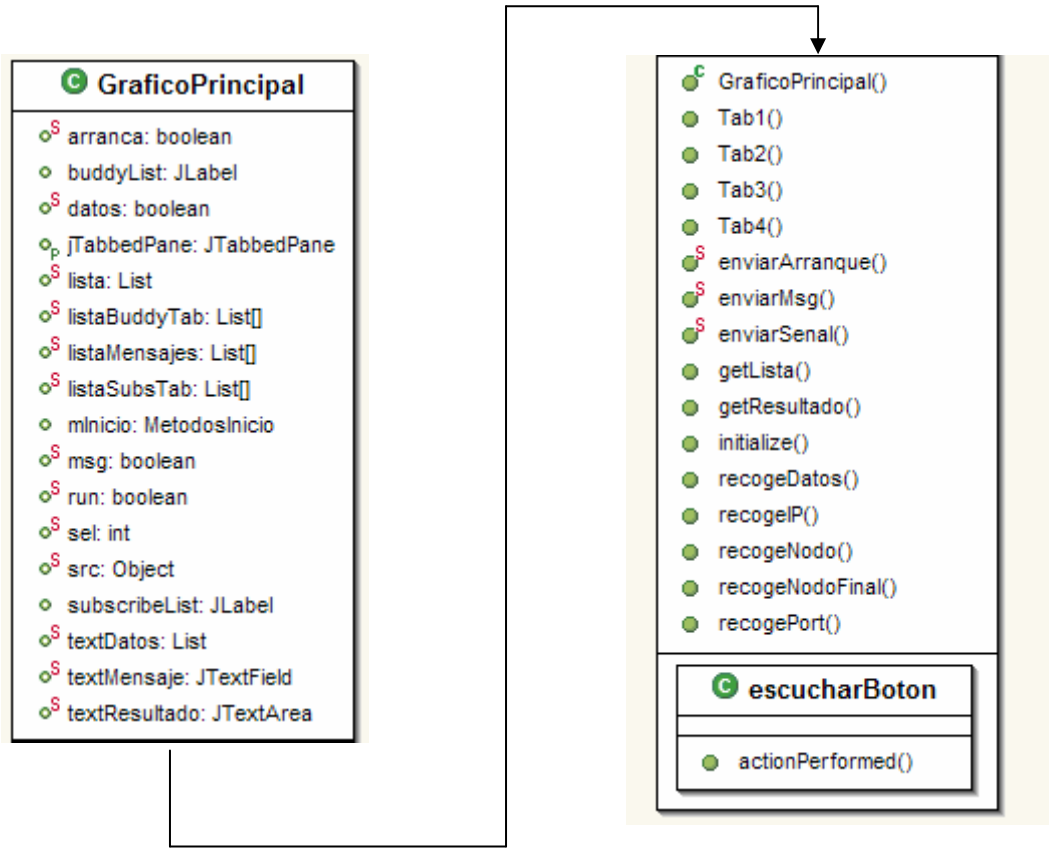

Fig 3.7 Diagrama de clases del paquete Graficos.

3.3.5. Mensajes

El paquete Mensajes, consta de 3 clases:

- RSA
- Mensajes
- Mensaje

Fig 3.8 Diagrama de clases del paquete Mensajes.

La clase RSA, podría ser propia de la clase Algoritmos, ya que realiza una tarea única e independiente del resto del sistema, pero al estar estrechamente relacionada con los mensajes que se intercambian los usuarios, hemos visto pertinente ubicarla en este paquete.

Dicha clase, tiene los métodos necesarios para poder encriptar y desencriptar un mensaje que se intercambie un usuario con otro, mediante el algoritmo de cifrado RSA, como explicamos en el primer capítulo de este documento.

Por otro lado tenemos las clases Mensajes y Mensaje. La primera clase, Mensajes, se encarga de recibir el mensaje cifrado de la clase RSA, enrutarlo hacia su destinatario, y entregarselo satisfactoriamente.

Por otro lado, la clase Mensajes, simplemente tiene la estructura básica de un mensaje en Pastry.

3.3.6. SIP

Por último, el paquete SIP, contiene las siguientes clases:

- ListenerSIPenvio
- ListenerSIPescucha
- ArranqueListenerEnvio
- ArranqueListenerEscucha
- DatosSIP

En este paquete están todos los métodos necesarios para que junto con la API de Jain Nist SIP, se pueda establecer comunicación y presencia con el protocolo SIP.

SIP necesita de uno que envíe y otro que escuche, como todo mecanismo cliente-servidor, solo con la diferencia de que cualquiera puede ser cliente o puede ser servidor.

Así pues, si por ejemplo Alice quisiera comunicarse con Blas, y más en adelante Blas quisiera comunicarse con Alice, ambos deberían de poseer un punto de envío y otro de escucha, ocupando así los puertos 5060 y 5061 respectivamente en exclusividad.

Si tienes solo 2 clientes no hay problema, pero cuando tienes más, empiezan los conflictos de duplicados de sesión, puertos ya ocupados, sobrecarga del sistema, etc.

Por ese motivo, hemos diseñado el sistema de otra forma. Hemos creado 2 puntos o stacks de conexión, al arrancar el sistema entero, uno de envío, y otro de escucha.

Cuando nosotros iniciamos nuestro programa, vemos como además de agregar usuarios a nodos Pastry, arrancamos el SIPListenerEnvio y el SIPListenerEscucha.
Son dos puntos de conexión, para todos los usuarios del sistema, que están levantados y a la espera de peticiones constantemente.

Ahora si Alice quiere comunicarse con Blas, bastará con que Alice instancie un objeto de la clase SIPListenerEnvio, y Blas instancie un objeto de SIPListenerEscucha.

De ahí las clases SIPListener, se arrancan únicamente en el inicio del sistema, pero se instancian cada vez que un usuario lo requiere.

Para poder hacer de esto un servicio multitudinario, hemos creado dos clases que crean un Thread de cada uno de los SIPListener, así cada vez que un

usuario instancie un SIPListener, instanciará un hilo nuevo, siguiendo una arquitectura multi-threading.

Por último la clase DatosSIP, alberga la información que necesita SIP para enviar sus mensajes de un usuario a otro.

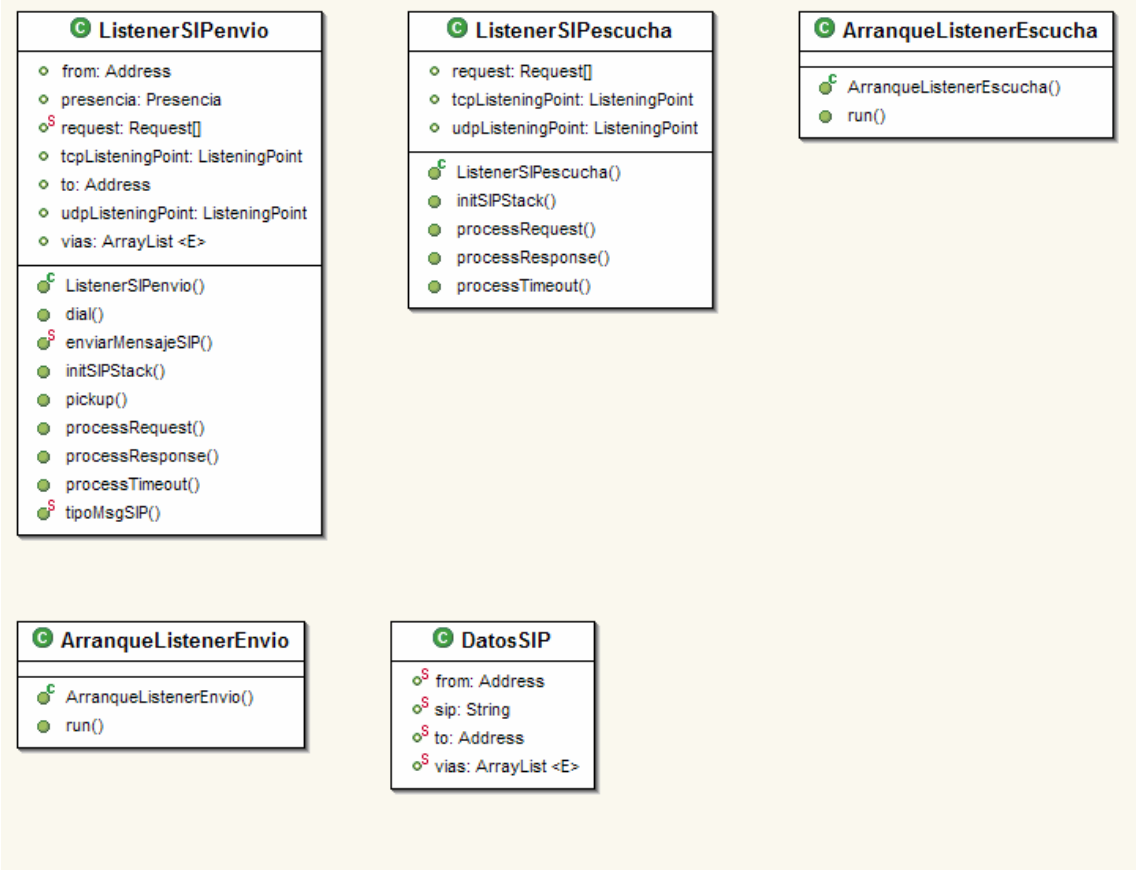

Fig 3.9 Diagrama de clases del paquete SIP.

Capítulo 4. Conclusión

En este trabajo hemos explicado el funcionamiento de las redes P2P, en concreto la red Pastry, que es en la que nos hemos centrado, así como el protocolo de inicio de sesiones SIP, del que también hemos hecho uso.

Para entender todo ello, el autor de este documento ha realizado la programación en el lenguaje orientado a objetos Java, de un sistema que emula una serie de usuarios que se comunican gracias a estos mecanismos.

Ha servido, y mucho, para entender con más precisión y alcance la programación orientada a objetos, la metodología de trabajo, tanto a nivel de algorítmica como a nivel de lenguaje puro de Java.

Hemos visto el uso de una red P2P que desconocía absolutamente. Gracias a este profundo análisis, hemos llegado a la conclusión de que un sistema de estas características, bien implementado y con una cierta aceptación, podría llegar a ser un buen sistema de comunicaciones en ámbito local, ya que la comunicación entre nodos de usuarios, es efectiva, rápida, segura y fácil de enrutar. Gracias a SIP se puede programar con relativa sencillez un sistema de presencia de usuarios, así como de inicio de comunicaciones mediante texto o Telefonía IP, lo que conllevaría a una gran aceptación ya que seria de tarificación cero al viajar todo mediante VoIP.

No obstante, implementar un sistema así dentro de un ámbito superior al local, podría verse seriamente limitado, ya que el espacio de claves de la red Pastry es grande, pero limitado, y conforme más grande se hace la red, más complicados se hacen sus algoritmos de encaminamiento, lo que conllevaría a mucha congestión, al enrutarse por identificadores cifrados.

Medioambientalmente un sistema así no repercutiría en el ambiente, al menos no más de lo que pueda estarlo ya, ya que para hacer funcionar un sistema así, basta con la tecnología y la infraestructura habilitada hoy en día.

No supondría ningún esfuerzo extra, adaptar los sistemas ya existentes para hacerlos funcionar con esta aplicación, ya que todo es software y al estar programado en Java, es multiplataforma, lo que haría posible hacerlo funcionar en cualquier tipo de máquina que tuviese una Java Virtual Machine (JVM) instalada.

Finalmente descartar que si se continua este trabajo en el futuro posiblemente se podrán añadir nuevas y mejoradas funcionalidades. Su nuevo autor tendría la labor de simular el mismo sistema, pero para un número N de usuarios, variable en tiempo real, con la posibilidad de agregarse al sistema en cualquier momento, o mejorar las simulaciones del presente trabajo. Una vez realizada dicha labor, se podría hacer más hincapié en el sistema de comunicación mediante VoIP, creando un programa nuevo que tratase las comunicaciones que en este sistema ya hemos conseguido establecer.

Capítulo 5. Bibliografía

[1] JDK 5.0 Documentation, *J2SE Platform at a Glance*, Sun Microsistems, http://java.sun.com/j2se/1.5.0/docs/

[2] Rice Pastry API, http://www.freepastry.org/FreePastry/javadoc/index.html

[3] Hoye, J, "Pastry, *A substrate for peer-to-peer applications*", http://www.freepastry.org/

[4] KTH & SCS Inc, "*JDHT: Java Distributed Hash Table*", http://dks.sics.se/jdht/

[5] Programación en Castellano, "*Seguridad en la Plataforma Java 2 JDK 1.2*", Sun Microsistems, http://www.programacion.com/java/tutorial/security1dot2/

[6] Hoye, J, "*The FreePastry Tutorial*", http://www.freepastry.org/FreePastry/tutorial/

[7] Orlin Grabbe, J, "*Java Program for RSA Encryption*", http://www.orlingrabbe.com/rsa_java.htm

[8] Nist SIP, "*About the IP telephony project*", http://snad.ncsl.nist.gov/proj/iptel/

[9] Wikipedia, The free Encyclopedia, "*Distributed Hash Table*", http://en.wikipedia.org/wiki/Distributed_hash_table

[10] Wikipedia, The free Encyclopedia, "*Session Initiation Protocol*", http://es.wikipedia.org/wiki/Session_Initiation_Protocol

[11] Tutorial Intermediate Java Swing, http://home.cogeco.ca/~ve3ll/jatutora.htm#bo