ugr | Universidad de Granada

ESTUDIOS DE INGENIERÍA
DE TELECOMUNICACIÓN

PROYECTO FIN DE CARRERA

CURSO:11/12

JESÚS RECUERDA HUESO

El tribunal constituido para la evaluación del PFC titulado:

CLIENTE VOIP P2P EN DISPOSITIVOS MÓVILES

Realizado por el alumno: **Jesús Recuerda Hueso**.

Y dirigido por el tutor: **Jorge Navarro Ortiz**.

Ha resuelto asignarle la calificación de:

- ☐ SOBRESALIENTE (9 – 10 puntos)
- ☐ NOTABLE (7 – 8.9 puntos
- ☐ APROBADO (5 – 6.9 puntos)
- ☐ SUSPENSO

Con la nota[1]: ☐ puntos.

El Presidente: _____

El Secretario: _____

El Vocal: _____

Granada, a de de 20 .

[1] Solamente con un decimal.

ugr | Universidad de Granada

ESTUDIOS DE INGENIERÍA DE TELECOMUNICACIÓN

CLIENTE VOIP P2P EN DISPOSITIVOS MÓVILES

REALIZADO POR:

Jesús Recuerda Hueso

DIRIGIDO POR:

Jorge Navarro Ortiz

DEPARTAMENTO:

Teoría de la Señal, Telemática y Comunicaciones

Granada, Enero de 2012

CLIENTE VOIP P2P EN DISPOSITIVOS MÓVILES

Jesús Recuerda Hueso

PALABRAS CLAVE

VoIP, P2P, Chord, SIP, Bada, Ad-Hoc

RESUMEN

Ante la expansión de los teléfonos inteligentes y la posibilidad de usar redes Ad-Hoc que ofrecen, surge la posibilidad de dar una solución a la telefonía sobre IP sin infraestructuras. Esta solución sería aplicable en situaciones de falta de infraestructuras, como podrían ser una catástrofe natural o su uso como alternativa al *walkie-talkie*.

Por este motivo, este proyecto se centra en la telefonía VoIP en redes P2P, lo que permite que la arquitectura del sistema no necesite servidores o *registrars*. Además, mediante el uso de conexiones Ad-Hoc el sistema no requiere de una infraestructura de red. Así pues, un usuario puede realizar una llamada VoIP a otro sin necesidad de ninguna infraestructura (servidor, *registrar*, conmutadores, enrutadores…).

El objetivo del proyecto es crear una aplicación para *smartphone* que permita realizar este tipo de llamadas. Para ello, debe crear una red P2P y usar los protocolos SIP y SDP para establecer sesiones multimedia. Además, la aplicación debe codificar la voz y utilizar el protocolo RTP para su transporte.

En concreto, se ha elegido una red P2P de tipo Chord, basada en tablas hash distribuidas. El protocolo SIP ha sido modificado para adaptarlo a una red sin servidores. El códec de voz implementado es G.711 en su variante ley-A.

El resultado del proyecto es una aplicación para Bada, un sistema operativo de reciente aparición que es propiedad de la compañía Samsung. La aplicación permite al usuario elegir varias opciones: por un lado, se puede ejecutar como *peer* y cliente simultáneamente o sólo como cliente; por otro lado, se puede ejecutar usando una red con infraestructura o usando una red Ad-Hoc. Gracias a esta aplicación, cualquier usuario puede realizar una llamada VoIP a otro usuario a través de su nombre, ya que la red Chord se encargará de encontrar su localización.

La aplicación resultante cumple los requisitos planteados de forma satisfactoria, ofreciendo una solución aplicable a las situaciones que lo motivaron.

VOIP P2P (PEER-TO-PEER) CLIENT ON MOBILE DEVICES

Jesus Recuerda Hueso

KEYWORDS

VoIP, P2P, Chord, SIP, Bada, Ad-Hoc

ABSTRACT

The continual growth of the smartphones and the possibility of using Ad-Hoc networks offer a possibility to give a solution for the VoIP telephony without infrastructure. This solution could be useful scenarios such as a catastrophic situation or replacing the walkie-talkie.

This project focuses on the VoIP telephony in peer-to-peer networks. Using this type of network, the system architecture has no servers or registrars. Furthermore, if Ad-Hoc connections have been used for the communications, the system would not need a network infrastructure. Thus, one user could make a VoIP call to other user without needing to use any network infrastructure (server, registrar, switches, routers…).

The aim of the project is to provide a smartphone application that allows a user to make this type of calls. For that purpose, it shall create a P2P network and use the SIP and SDP protocols to establish multimedia sessions. Furthermore, the application shall implement a voice codec and use the RTP protocol for carrying the voice data.

In particular, we have chosen a Chord P2P network, which is based on DHT (Distributed Hash Tables). The SIP protocol has been modified to adapt it to a network without servers. The voice codec implemented is G.711 A-Law.

The result is an application for Bada, a new operating system which is property of Samsung. The application allows the user to choose between several options: on the one hand, the user may execute the application as peer and client or only as client; on the other hand, the user may execute the application using an infrastructure network or using an Ad-Hoc connection. Using this application, any user can make a VoIP call to another user just knowing his/her name, since the Chord network will manage to find the user location.

The resulting application satisfies the requirements and offers an applicable solution for the aforementioned situations.

D. Jorge Navarro Ortiz

Profesor del departamento de Teoría de la Señal, Telemática y Comunicaciones de la universidad de Granada, como director del Proyecto Fin de Carrera de D. Jesús Recuerda Hueso

Informa:

que el presente trabajo, titulado:

Cliente VoIP P2P en dispositivos móviles

Ha sido realizado y redactado por el mencionado alumno bajo nuestra dirección, y con esta fecha autorizo a su presentación.

Granada, a _____ de _____ de 20___

Fdo.:_____

Los abajo firmantes autorizan a que la presente copia de Proyecto Fin de Carrera se ubique en la Biblioteca del Centro y/o departamento para ser libremente consultada por las personas que lo deseen.

Granada, a _____ de _____ de 20___

(Firma y números de DNI del alumno y el tutor)

AGRADECIMIENTOS

Quiero agradecer aquí, no sólo a aquellas personas relacionadas con este proyecto, sino a todos los que en algún momento durante mis estudios lo merecieron.

A todos mis compañeros, pero en especial a aquellos que más han sufrido –y cómo no también disfrutado– conmigo y junto a mí.

A mis padres y a mi hermano, porque siempre han estado ahí.

A todos mis amigos, por muchísimos momentos inolvidables, y en particular a Fernando, porque la mayor parte del camino la hemos hecho juntos.

A Jorge, por el verdadero esfuerzo que ha realizado por mí.

A Laura, porque ha sufrido incluso más que yo y porque siempre me ha apoyado.

Gracias a todos.

Índice

Índice de figuras

Índice de tablas

Glosario

SÍMBOLOS

3G Tercera generación de tecnologías para comunicaciones móviles.

4G Cuarta generación de tecnologías para comunicaciones móviles.

A

ACF *Admision ConFirmation*, mensaje de confirmación empleado en la fase de establecimiento.

API *Application Programming Interface*, interfaz que proporciona la empresa desarrolladora del sistema operativo para la programación de aplicaciones.

ARQ *Admision ReQuest*, mensaje de solicitud de admisión empleado en la fase de establecimiento.

C

C++ Lenguaje de programación orientada a objetos.

CLC *Close Logical Channel*, mensaje empleado para la finalización de llamadas.

CRLF *Carrier Return Line Feed*, combinación de retorno de carro salto de línea.

CSRC *Contributing Source*, identifica las fuentes que contribuyen a la carga útil de ese paquete.

D

DHT *Distributed Hash Table*, tablas hash distribuidas.

E

ESC *End Session Command*, mensaje empleado durante el proceso de desconexión.

G

G.711 Códec de voz muy usado en la telefonía sobre IP.

GPS *Global Positioning System*¸ sistema de localización vía satélite.

H

H.225 Protocolo que forma parte de H.323.

H.245 Protocolo usado para el control del canal, forma parte de H.323.

H.323 Protocolo usado para establecer sesiones de audio o vídeo.

HSDPA *High-Speed Downlink Packet Access*, es la evolución de la tercera generación (3G) en la telefonía móvil.

HTTP *HyperText Transfer Protocol*, protocolo de nivel de aplicación de la pila TCP/IP.

I

ID *IDentifier*, identificador de un nodo.

IETF *Internet Engineering Task Force*, es una organización internacional abierta de normalización.

iOS Sistema operativo usado por algunos dispositivos de la marca Apple.

IP *Internet Protocol,* Protocolo que corresponde con la capa de red de la arquitectura TCP/IP.

ITU *International Telecommunications Union,* organismo especializado encargado de regular las telecomunicaciones a nivel internacional.

L

LAN *Local Area Network*, red de área local.

M

MCU *Multipoint Control Unit*, es uno de los componentes del sistema H.323.

MEGACO *Media Gateway Control Protocol*, protocolo encargado de controlar las pasarelas (*Gateways*).

MOS *Mean Opinion Score*, medida numérica de la calidad percibida por un usuariode un servicio multimedia.

N

NAT *Network Address Translation,* método de traducción de direcciones IP entre redes.

P

P2P *Peer-to-Peer*, tipo de red en la que el trabajo se divide entre pares.

PBX *Private Branch eXanche*, central privada de telefonía.

PCM *Pulse Code Modulation*, método para digitalizar muestras analógicas.

PSTN *Public Switched Telephone Network*, es la red telefónica básica.

Q

QoS *Quality of Service*, parámetro multidimensional que caracteriza lo que demanda la aplicación y lo que ofrece la red.

R

RAM *Random Access Memory*, formato de almacenamiento de datos en los computadores.

RDSI *Red Digital de Servicios Integrados*, evolución de la Red Digital Integrada, que proporciona una amplia gama de servicios.

RTCP *Real-time Transport Control Protocol*, protocolo que proporciona información de control de un flujo de datos de una sesión multimedia RTP.

RTP *Real-time Transport Protocol*, protocolo utilizado para el envío de información en tiempo real, como por ejemplo audio y vídeo.

RTSP *Real-Time Streaming Protocol*, protocolo de nivel de sesión de la pila de protocolos TCP/IP para el control de flujos multimedia en tiempo real.

S

SDK *Software Development Kit*, kit para el desarrollo de aplicaciones.

SDP *Session Description Protocol*, protocolo utilizado para la descripción de sesiones multimedia.

SHA-1 Algoritmo hash muy extendido.

SIP *Session Initiation Protocol*, protocolo usado para el establecimiento de sesiones multimedia.

SO Sistema operativo.

SS7 *Sistema de Señalización nº 7*, es un conjunto de protocolos de señalización telefónica

SSRC *Synchronization Source*, identifica la fuente que está generando el paquete para esa sesión.

T

TCP *Transmission Control Protocol*, protocolo de nivel de transporte de la pila de protocolos TCP/IP orientado a conexión.

U

UA *User Agent*, es una entidad SIP.

UDP *User Datagram Protocol*, protocolo de nivel de transporte de la pila de protocolos TCP/IP orientado a datagrama.

UML *Unified Modeling Language*, lenguaje de modelado de sistemas software.

UP2P *Unstructured P2P*, tipo de red P2P.

URI *Uniform Resource Identifier*, cadena de caracteres que identifica a un recurso.

UTF-8 *8-bit Unicode Tranformation Format*, formato de codificación de caracteres.

V

VoIP *Voice over Internet Protocol*, conjunto de tecnologías que permiten la distribución de servicios de voz sobre redes IP.

W

WAN *Wide Area Network*, red de área amplia.

Capítulo 1. Introducción

1.1 Contexto

En esta sección se hace un análisis del contexto en el que se lleva a cabo el proyecto. Para ello, se describen los terminales conocidos como teléfonos inteligentes (*smartphones*) y el uso de la telefonía sobre IP en estos. También se explica qué son las redes Ad-Hoc, dado que es un tipo de red que será relevante para el desarrollo del proyecto. Finalmente se explica el uso que se hace de la telefonía IP en las *intranets*.

1.1.1 *Smartphones*

Desde al año 2008, año que la empresa *comScore* describe como "el año del *smartphone*" [1], han aparecido en el mercado teléfonos móviles con más funcionalidades que los convencionales. En estos dispositivos, como ocurría con los ordenadores, empieza a ser relevante la capacidad de procesamiento de los mismos.

Estos nuevos terminales tienen dos características que los diferencian del resto: la capacidad de modificar sus funcionalidades mediante la instalación de nuevas aplicaciones y, más importante, una integración y un uso continuo de la red de datos, ya sea a través de WiFi o de las redes 3G o 4G. Este tipo de terminales son los que identificamos como *smartphones* o teléfonos inteligentes.

Uno de estos terminales de gama media/alta actualmente podría contar con características similares a las siguientes:

- Procesador de 1GHz
- 512Mb de memoria RAM
- 4 GB de memoria para almacenamiento
- Cámara fotográfica de 5 Megapíxeles
- GPS
- 802.11 b/g/n
- HSDPA

Estas características los convierten en dispositivos con una alta capacidad de procesamiento y para los que se pueden crear todo tipo de aplicaciones. Este hecho, sumado a la completa integración con las crecientes redes sociales, hace que sus ventas aumenten de forma considerable y que susciten mucha demanda.

En la Figura 1.1 [2] se puede ver la penetración de los teléfonos inteligentes en relación al total de móviles vendidos. Se estima que en un futuro cercano lleguen a suponer un 60% del total de teléfonos móviles. El porcentaje se vería incrementado si sólo se tuvieran en cuenta los países desarrollados para las estadísticas y no todo el mundo.

Las grandes empresas están dedicando muchos recursos y esfuerzo al desarrollo y la mejora de los terminales y de sus aplicaciones, ya que los datos vistos hacen que éste sea un mercado muy interesante. A modo de ejemplo, las ganancias producidas por estos terminales supusieron un 60% de las ganancias de la empresa Samsung en el año 2010 [3].

Pero los smartphones, además de recibir soporte de las grandes empresas, tienen otro pilar fundamental. Este tipo de móviles permite que cualquier usuario pueda crear una aplicación y ponerla a disposición del público de manera sencilla mediante entornos de desarrollo que, en la mayoría de los casos, son libres. Algunos ejemplos son:

- Bada (http://developer.bada.com/devtools/sdk)
- Android (http://developer.android.com/sdk/index.html)
- Symbian (http://www.developer.nokia.com/)

Esta disponibilidad de plataformas de desarrollo hace que éste sea un mercado del que cualquiera puede formar parte de manera sencilla.

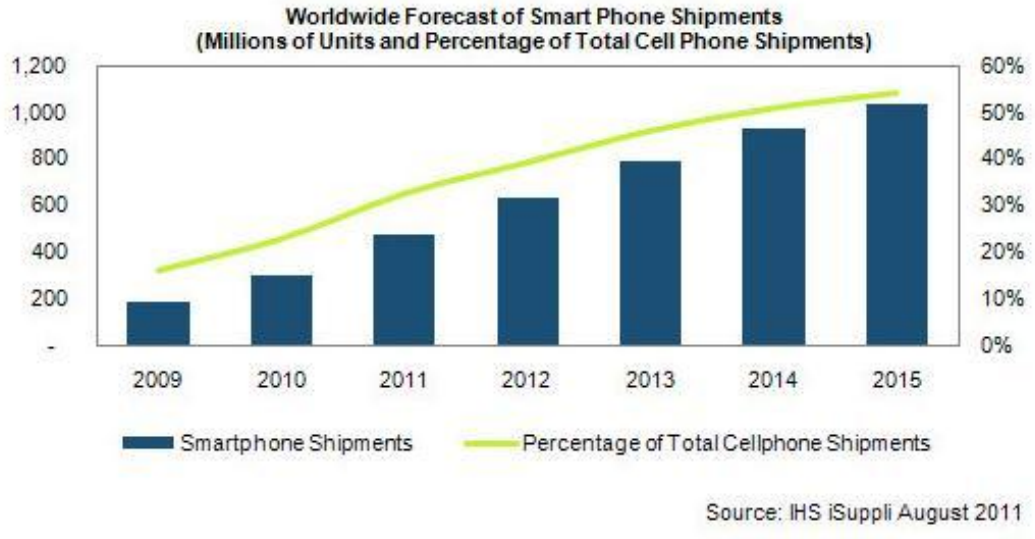


Figura 1.1: Penetración de los smartphones

1.1.2 VoIP en dispositivos móviles

A medida que crecen las conexiones a Internet de banda ancha, crece la relevancia de la telefonía sobre IP. Esto se debe a que ésta ofrece una alternativa a la telefonía convencional mucho más económica, puesto que al utilizar la red de datos no existe facturación por el establecimiento y la duración de la llamada, como sí ocurre con la telefonía convencional.

Como se ha comentado anteriormente, los terminales móviles actuales pueden estar constantemente conectados a la red de datos (usualmente con tarifas planas), además de contar con una gran capacidad de procesamiento y la posibilidad de instalar aplicaciones. Esto lleva a un elevado uso de estas aplicaciones en terminales móviles. Según una encuesta realizada a más de mil usuarios japoneses, más del 50% de ellos han utilizado alguna aplicación VoIP en su *smartphone* [4].

1.1.3 Redes inalámbricas Ad-Hoc

Ad-Hoc es una locución latina que significa literalmente “para esto”. En general, se refiere a algo improvisado y temporal para un fin específico. En concreto, en el ámbito de las comunicaciones inalámbricas el propósito de las redes Ad-Hoc es proporcionar flexibilidad y autonomía aprovechando los principios de auto-organización, permitiendo crear redes de carácter temporal e.g. para una reunión.

Una red inalámbrica Ad-Hoc no tiene ningún elemento que centralice su gestión como podría ser un punto de acceso, sino que consta de nodos móviles que usan una interfaz inalámbrica para enviar paquetes de datos entre ellos. De esta manera, estos nodos tienen idénticas funcionalidades y compiten por el medio inalámbrico en igualdad de condiciones. Los nodos cercanos, es decir, los que están mutuamente en su zona de cobertura, se descubren para formar una red. Estos terminales pueden buscar nodos que están fuera de su área de alcance conectándose a través de otros nodos que sí estén en su área de alcance, incluso a través de múltiples nodos.

Debido a que este tipo de redes no necesita una administración centralizada, son idóneas para entornos en los que no se dispone de infraestructuras o en situaciones en las que sea necesario desplegar una red de una manera rápida y con carácter temporal. Por lo tanto, esta tecnología se puede aplicar, por ejemplo, para usos militares, para el restablecimiento de comunicaciones ante catástrofes naturales o para entornos de trabajo cambiantes, como podrían ser la minería o la construcción.

En la Figura 1.2 se puede ver un escenario de este tipo de redes para uso militar. En este ejemplo se puede apreciar cómo todas las unidades de un destacamento permanecen en contacto sin necesidad de la implantación de ninguna torre de radio u otra infraestructura similar.

1.1.4 Telefonía en la intranet

En toda empresa es indispensable la comunicación entre los distintos componentes de la misma, normalmente mediante llamadas telefónicas que se realizan dentro de la empresa. Para este tipo de llamadas, tradicionalmente se ha hecho uso de centralitas telefónicas PBX situadas dentro de la misma empresa y teléfonos tradicionales conectados a la misma.

Sin embargo, dado que en los puestos de trabajo suele haber una conexión de datos, en la actualidad se tiende a realizar las llamadas internas mediante teléfonos VoIP, como por ejemplo ocurre en la Universidad de Granada [5]. Para realizar llamadas externas mediante la red telefónica conmutada se utiliza una pasarela para convertir y trasladar estas llamadas a través de líneas telefónicas analógicas convencionales.

En la Figura 1.3 se puede observar un escenario como el comentado anteriormente. En éste, distintos terminales con capacidad para realizar llamadas VoIP se conectan a la red de datos mediante la que tienen salida hacia Internet, o hacia la red telefónica convencional mediante una pasarela.

Con este tipo de soluciones se consigue unificar la infraestructura desplegada en una empresa, siendo necesaria sólo una red de datos mediante la cual se tiene disponibilidad para todos los servicios necesarios en la misma, sin necesidad de instalar una alternativa para las llamadas telefónicas.

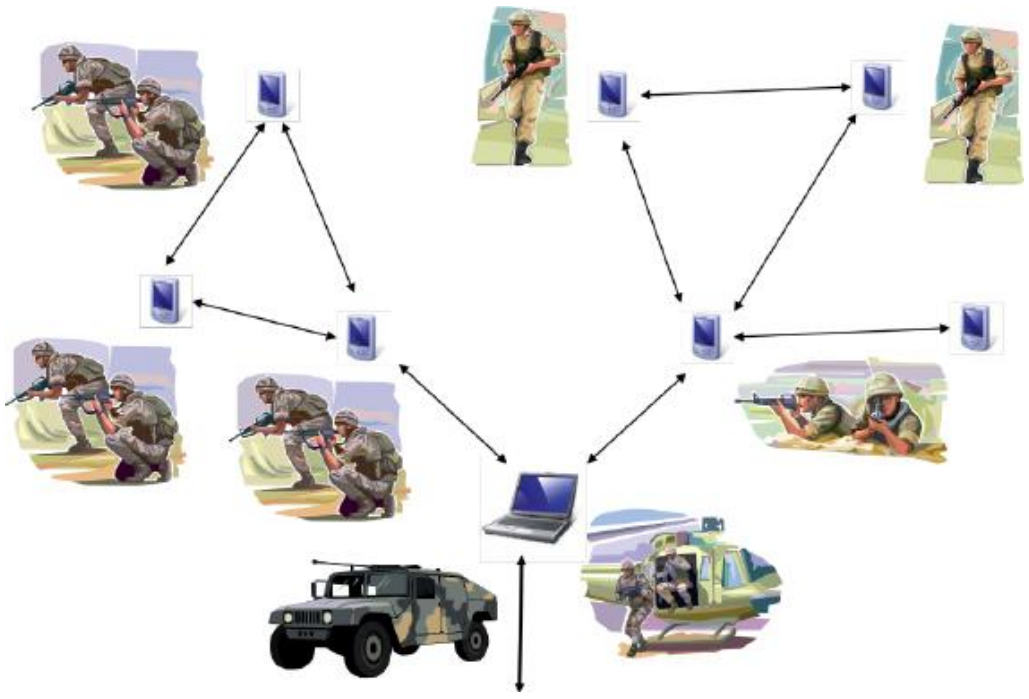


Figura 1.2: Escenario de red Ad-Hoc

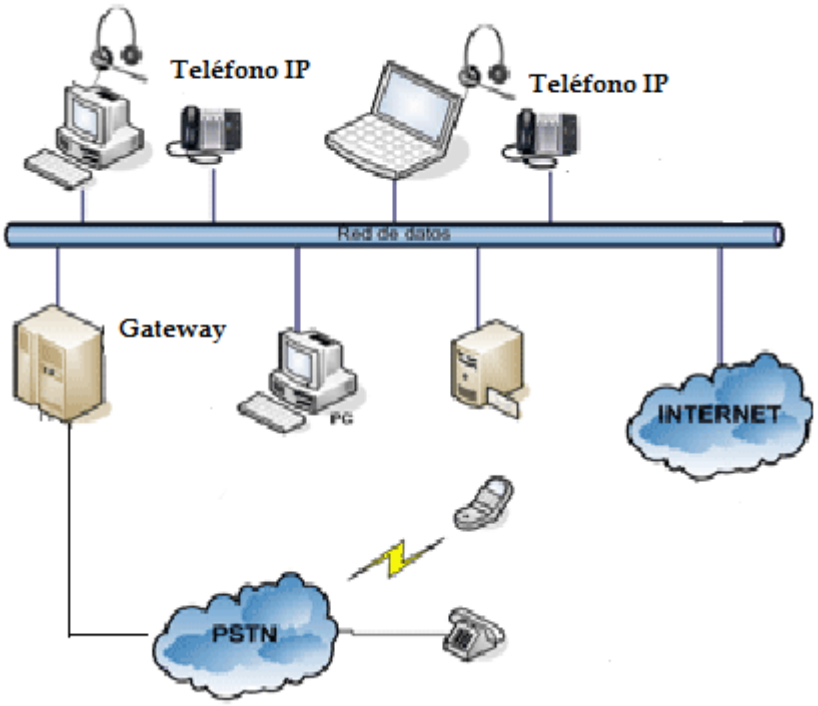


Figura 1.3: Intranet con salida a PSTN

1.2 Motivación

Analizando este contexto de forma global, es importante destacar la disponibilidad de dispositivos móviles con una alta capacidad de procesamiento. Estos terminales permiten la instalación de aplicaciones cuyo desarrollo y distribución a nivel mundial son relativamente sencillos. Como ya se comentó anteriormente, estos terminales están teniendo una alta penetración en la población de teléfonos móviles en el mercado y se espera que siga creciendo hasta que la mayoría de los ciudadanos dispongan de un terminal de este tipo.

Si se analizan los ámbitos de aplicación de las redes Ad-Hoc anteriormente mencionados, y teniendo en cuenta la expansión de estos dispositivos móviles, surge la posibilidad de crear una aplicación que aporte una solución a la comunicación en situaciones concretas donde no existe una infraestructura.

Como ejemplos significativos de casos de uso se podrían considerar 1) catástrofes naturales en las que las infraestructuras hayan sido destruidas, en las que esta aplicación brindaría una posibilidad de comunicación de forma instantánea, y 2) entornos similares al de la construcción, donde se podrían realizar llamadas internas a través de la aplicación de forma gratuita (reemplazando al *walkie-talkie*).

De esta forma, el objetivo principal de este proyecto es el desarrollo de una aplicación que, sin necesidad de ningún tipo de infraestructura, permitiera realizar una llamada de voz a cualquier usuario simplemente conociendo su nombre.

Una aplicación de estas características, podría complementarse con una funcionalidad más habitual, como su uso para telefonía IP dentro de empresas. Si la empresa dispone de una infraestructura de red y proporciona cobertura WiFi, simplemente estando dentro del alcance de un punto de acceso se podrían utilizar estos *smartphones* como teléfonos IP móviles. De esta forma, sin necesidad de estar en un puesto fijo, se permitiría la movilidad de los empleados sin perder su disponibilidad para recibir o realizar llamadas y sin que la empresa tuviera que realizar una inversión costosa para este fin.

Si bien existen aplicaciones que proporcionan funcionalidades de VoIP, éstas están disponibles para los sistemas operativos Android, iOS, o BlackBerry, pero no para Bada. Skype, Tango o Viber, son ejemplos de estas aplicaciones. Éstas proporcionan funcionalidades de VoIP, pero no permiten usar esta funcionalidad sin una conexión a Internet.

Tal como se ha comentado, hasta donde llega el conocimiento del autor, no hay ninguna aplicación que ofrezca esta funcionalidad para el sistema operativo Bada. Esto se debe a una prohibición explícita de Samsung (compañía desarrolladora del sistema operativo) de crear aplicaciones que permitan usar VoIP que terminó en Marzo de 2011 [6]. Así pues, el hecho de que no existan aplicaciones con estas características para este sistema operativo supone una motivación adicional para la realización del proyecto.

1.3 Objetivos y alcance del proyecto

Este proyecto tiene como objetivo principal el desarrollo de una aplicación para *smartphone* que permita establecer llamadas de voz entre terminales mediante el protocolo SIP, proporcionando las funcionalidades típicas de un teléfono: realizar llamadas, rechazarlas, terminar las mismas e identificar al remitente.

Para este fin, se dará a elegir entre establecer estas comunicaciones mediante una red Ad-Hoc creada por los usuarios o bien conectándose a una red local mediante la interfaz WiFi.

Para la realización de estas llamadas, el usuario sólo deberá conocer el nombre de usuario de aquel a quien quiere dirigirla, sin necesidad de conocer su dirección IP ni otros datos. Para que esto sea posible sin la necesidad de que haya un *servidor* ni un *registrar*[2] en la red, se implementará una red *peer-to-peer* (redes P2P, véase la sección 2.3) mediante la cual los nodos podrán encontrar a cualquier otro nodo que se halle dentro de su misma red.

Los nodos que se unan a esta red P2P deberán tener capacidad para actualizar sus tablas ante la entrada de otros nodos, e igualmente en caso del abandono de alguno. También deberán proporcionar mecanismos para encontrar a cualquier nodo que se encuentre en ésta conociendo simplemente su nombre de usuario. Además, esta aplicación puede ser usada por usuarios que no serán pares de la red P2P, por lo que los

[2] El *registrar* es un terminal de la red que en una topología SIP se encarga de registrar la localización de los dispositivos.

nodos que forman la red deben registrarlos y darles la posibilidad de buscar a otros usuarios de la aplicación.

En resumen, se busca implementar una aplicación que, sin necesidad de ninguna infraestructura, permita a un grupo de usuarios crear una red P2P y realizar llamadas simplemente conociendo el nombre del usuario a quien se quiere llamar, localizándolo mediante la red creada.

Capítulo 2. Protocolos relacionados

En este capítulo se detallan diversos protocolos que podrían estar presentes en el desarrollo de una aplicación como la planteada en los objetivos del proyecto en la sección 1.3

De esta forma, se describen dos protocolos cuya finalidad es gestionar sesiones multimedia en redes IP: el protocolo H.323 y el protocolo SIP (*Session Initiation Protocol*).

También se detalla qué son las redes P2P, y dos tipos de redes P2P aplicadas a la telefonía: la red desestructurada (UP2P – *Unstructured P2P*) y la red Chord.

Por último se hablará del protocolo RTP (*Real-time Transport Protocol*) y de los diversos códecs de voz empleados en telefonía sobre IP, ambos relacionados con el apartado multimedia.

2.1 Protocolo H.323

El estándar H.323 [7] es un conjunto de normas y protocolos recomendado por el ITU-T diseñado para permitir transmisiones multimedia en redes basadas en IP. Fue rápidamente adoptado por fabricantes de equipos para transmitir voz y videoconferencia sobre IP, ya que define un modelo básico de llamada con servicios suplementarios (convergencia de voz, vídeo y datos en una sola red).

Este estándar forma parte de la serie de protocolos H.32x, los cuales también dirigen las comunicaciones sobre RDSI (H.320), RTC o SS7. Esta familia de protocolos ha ido evolucionando con el tiempo para permitir mejorar las transmisiones de voz y vídeo en LANs (Local Area Networks) y WANs (Wide Area Networks) sobre distintos medios. La versión actual data de 2006 y se conoce como H.323v6.

Sus principales características son:

- No garantiza una calidad de servicio (QoS).

- Es independiente de la topología de la red.
- Admite pasarelas.
- Permite usar más de un canal (voz, vídeo, datos) al mismo tiempo.
- El estándar permite que las empresas añadan funcionalidades, siempre que implementen las funciones de interoperabilidad necesarias.

Los componentes principales del sistema H.323 son:

- **Terminales**: Equipamiento que utilizan directamente los usuarios. Se pueden implementar tanto por *software* como por *hardware*.
- **Guardianes (*Gatekeepers*)**: Son los encargados de gestionar las comunicaciones VoIP, equivalentes a las centralitas privadas o PBXs (*Private Branch eXchange*). Normalmente se implementan por *software*.
- **Pasarelas (*Gateways*)**: Hacen de enlace con la red telefónica conmutada, actuando de forma transparente para el usuario.
- **Unidades de Control Multipunto (MCUs)**: Se encargan de gestionar las multi-conferencias.

Los principales protocolos utilizados en la arquitectura H.323 (véase la Figura 2.1) son:

- **RAS** (Registro, Admisión, Estado): Se utiliza solamente en zonas que tengan un *gatekeeper* para la gestión de la zona de control del mismo.
- **H.225**: Mensajes de establecimiento y finalización de llamada entre terminales o con el guardián.
- **H.245**: Mensajes de control extremo a extremo. Negociación de las capacidades de ancho de banda (mensajes *TerminalCapabilitySet*), de la apertura y cierre de los canales lógicos mediante los mensajes *OpenLogicalChannel, CloseLogicalChannel y EndSessionCComand*, de los códecs y mensajes de control de flujo.
- **RTP/RTCP** (*Real-Time Transport Protocol / Real-Time Transport Control Protocol*): Transporte punto a punto de datos en tiempo real.

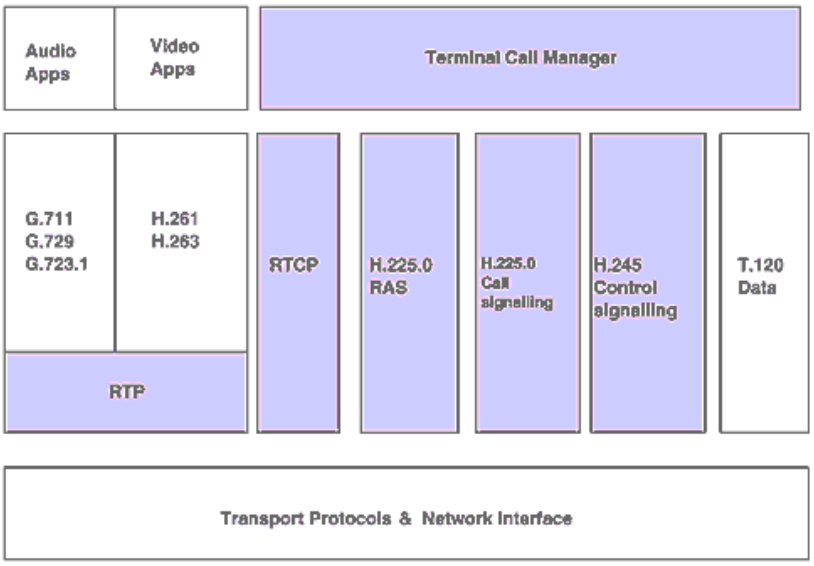


Figura 2.1: Estructura de protocolos de H.323

A continuación se describirá el procedimiento para realizar una llamada (véase la Figura 2.2), teniendo en cuenta cada una de las fases de la misma así como los protocolos involucrados.

1. Establecimiento

- Uno de los terminales se registra en el *gatekeeper* utilizando el protocolo RAS mediante los mensajes de solicitud de admisión ARQ (*Admission Request)* y los mensajes de confirmación de admisión ACF (*Admission Confirmation*).

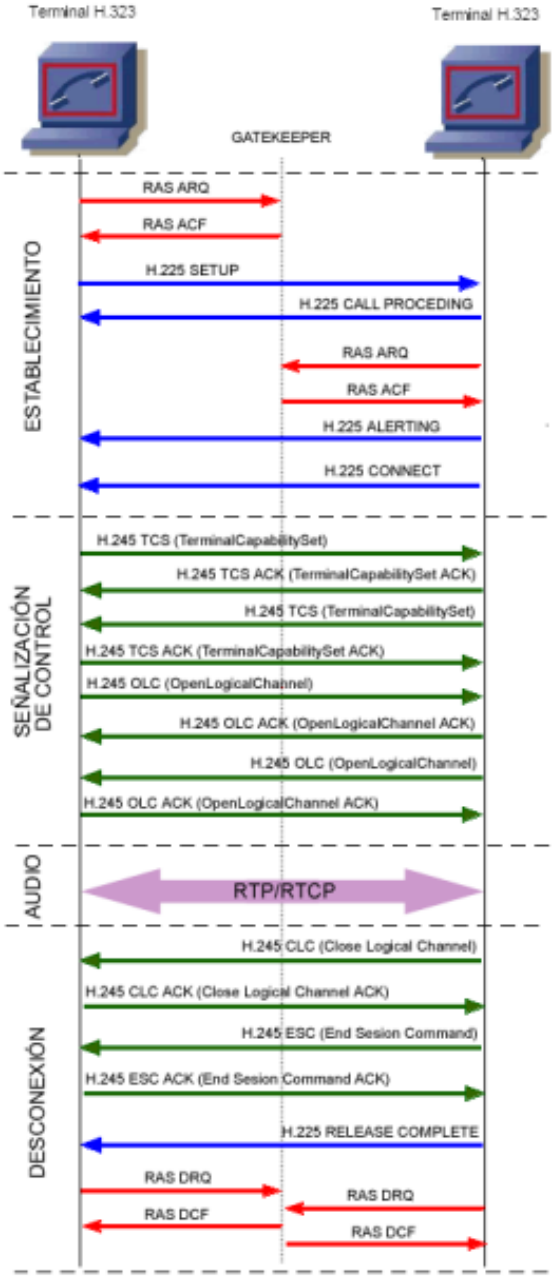


Figura 2.2: Llamada mediante H.323

- Mediante el protocolo H.225 se manda un mensaje de inicio de llamada (*Setup*) con los datos (IP y puerto) de llamante y llamado.
- El terminal llamado contesta con *Call Proceding*.
- El segundo terminal tiene que registrarse con el *gatekeeper* de manera similar al primer terminal.
- *Alerting* indica el inicio de generación de tono.
- *Connect* indica el comienzo de la conexión.

2. Señalización de control

- Se abre una negociación mediante el protocolo H.245 para establecer quién será el primario (*master*) y quién el secundario (*slave*), las capacidades de los participantes y los códecs de audio y vídeo a utilizar. Como punto final de esta negociación se abre el canal de comunicación (con unas direcciones IP y puertos determinados).

3. Audio (más datos y/o vídeo)

- Los terminales inician la comunicación y el intercambio de audio (además de datos y/o vídeo) mediante los protocolos RTP/RTCP.

4. Desconexión

- Cualquiera de los participantes activos puede iniciar el proceso de finalización de llamada mediante mensajes CLC (*Close Logical Cannel*) y ESC (*End Session Command*) de H.245.
- Posteriormente, utilizando H.225, se cierra la conexión con el mensaje *Release Complete*
- Por último, se liberan los registros con el *gatekeeper* utilizando mensajes del protocolo RAS.

2.2 Session Initiation Protocol

SIP [8] es un protocolo de control de la capa de aplicación que permite establecer, modificar y terminar sesiones multimedia (conferencias) así como llamadas telefónicas a través de Internet. SIP también permite invitar a participantes a sesiones ya existentes y a conferencias multimedia. Soporta servicios de redirección de forma transparente, permitiendo así movilidad, por lo que un usuario puede mantener externamente un identificador de su posición en la red sin necesidad de estar en el mismo sitio.

SIP soporta cinco tareas relacionadas con el establecimiento y la terminación de una comunicación multimedia:

- **Localización de usuarios**: Localización de usuarios finales para ser usados en una comunicación.
- **Disponibilidad de usuarios**: Determinación de la disponibilidad de los usuarios para formar parte de una comunicación.

- **Capacidad de los usuarios:** Determinación del tipo de comunicación y de los parámetros involucrados para una llamada.
- **Establecimiento de sesión:** Envío de tono de llamada (*Ringing*) y establecimiento de parámetros de sesión en ambos extremos, llamante y llamado.
- **Administración de la sesión:** Incluyendo la transferencia y terminación de las sesiones, modificación de los parámetros de éstas y administración de servicios.

SIP no es un sistema completo de comunicaciones, sino más bien un componente que puede ser usado con otros protocolos de la IETF para completar una arquitectura multimedia. Típicamente estas arquitecturas incluyen protocolos como *Real-time Transport Protocol* (RTP) [9] para transportar datos de tiempo real y proporcionar realimentación para calidad de servicio (*Qualitiy of Service* – QoS), *Real-Time Streaming Protocol* (RTSP) [10] para controlar el transporte de las sesiones de *streaming*, *Media Gateway Control Protocol* (MEGACO) [11] para controlar los *Gateways* hacia la red telefónica conmutada (*Public Switched Telephone Network* - PSTN) y *Session Description Protocol* (SDP) [12] para la descripción de las sesiones multimedia. Además, SIP debería ser usado en combinación con otros protocolos para proporcionar servicios completos a los usuarios. Sin embargo, la funcionalidad básica y operación de SIP no depende de ninguno de estos protocolos anteriormente mencionados.

Por ello, SIP no proporciona servicios sino que facilita primitivas que pueden usarse para implementar diferentes servicios. Por ejemplo, SIP puede localizar a un usuario y enviarle un objeto, cuyo contenido desconoce, hasta su localización actual. Si esta primitiva se usa para entregar una descripción de sesión escrita con SDP, los puntos terminales pueden negociar los parámetros de una sesión. Si la misma primitiva es usada para entregar una foto del llamante y el nombre del mismo, podría servir para implementar un servicio de identificación del llamante. Este ejemplo muestra como una sola primitiva se usa para proporcionar diferentes servicios.

2.2.1 Sintaxis y tipos de mensajes.

SIP es un protocolo basado en texto que usa caracteres UTF-8 [13]. Los mensajes SIP pueden ser solicitudes de un cliente a un servidor (*request*) o respuestas de un servidor a un cliente (*response*).

Ambos tipos de mensajes, tanto las solicitudes como las respuestas, consisten en una línea de comienzo, uno o más campos de cabecera, una línea en blanco indicando el final del campo de cabeceras y opcionalmente un cuerpo del mensaje.

```
Mensaje genérico =      start-line
                        message-header
                        CRLF
                        [message-body]
```

```
start-line =          Request-Line / Status-Line
```

La línea de comienzo, cada línea de cabecera y la línea en blanco deben terminar con un salto de carro (CRLF). La línea en blanco debe estar presente tanto si el mensaje tiene cuerpo como si no.

La mayoría de los mensajes SIP utilizan una sintaxis idéntica a HTTP/1.1 [14], aunque hay que aclarar que SIP no es una extensión de HTTP.

Solicitudes

Se distinguen por tener una *Request-Line* como línea de comienzo. Ésta contiene el nombre del método, un *Request-URI* y la versión del protocolo separados por un espacio simple (SP).

La *Request-Line* termina con un *CRLF* (línea en blanco), no permitiéndose ni tabulaciones ni espacios en blanco en ninguno de los elementos que la conforman, quedando la estructura de ésta como sigue:

```
Request-Line = Método SP Request-URI SP Versión-SIP CRLF
```

- Se pueden especificar seis tipos de métodos: *REGISTER* para el registro de la información del contacto; *INVITE*, *ACK*, y *CANCEL* para el establecimiento de sesiones; *BYE* para terminar las mismas; y *OPTIONS* para solicitar a los servidores sus funcionalidades.
- El *Request-URI* indica el usuario o el servicio al que se envía esta solicitud.
- La versión de SIP se incluye tanto en solicitudes como en respuestas para conocer la versión utilizada.

Las cabeceras obligatorias para que una solicitud formulada por un cliente sea válida se enumeran y se detallan a continuación:

- **To:** Esta cabecera indica el receptor deseado para la solicitud.
- **From:** Indica la identidad lógica del generador de la solicitud.
- **Call-Id:** Actúa como identificador único para agrupar una serie de mensajes. Debe ser el mismo para todas las solicitudes y respuestas de un diálogo.
- **CSeq:** Sirve para identificar y ordenar transacciones. Consiste en un número de secuencia seguido del método.
- **Max-Forwards:** Sirve para limitar el número de saltos que una solicitud puede dar durante su transmisión.
- **Via:** Indica la localización a la cual la respuesta debe ser enviada.

Otra cabecera relevante pero no obligatoria es **Contact**, que contiene un SIP-URI que puede usarse para contactar con un cliente en siguientes solicitudes.

Respuestas

Las respuestas se distinguen de las solicitudes en que tienen una línea de estado (*Status-Line*) como línea de comienzo. Ésta consiste en la versión del protocolo seguida

de un código numérico y una frase asociada a este código, con cada uno de los elementos separados con un espacio simple. A continuación se muestra esta estructura:

```
Status-Line = Versión-SIP SP Status-Code SP Frase-Razón
```

El código consiste en un entero de tres dígitos que indica la respuesta a una solicitud satisfactoria. La frase asociada tiene la intención de dar una pequeña descripción textual del código. Éste está pensado para ser entendido por un autómata, mientras que la frase está pensada para ser entendida por un humano. El cliente no tiene por qué examinar o mostrar la frase asociada al código.

El primer dígito del código es el que define la clase de respuesta. Siendo esto así, se asume que, por ejemplo, cualquier respuesta con un código entre 200 y 299 corresponde a una respuesta del tipo *2xx response*. En SIP/2.0 se disponen de seis tipos de respuestas:

- **1xx**: Provisional – solicitud recibida, se continúa con el procesado de ésta.
- **2xx**: Satisfactorio – la solicitud fue recibida correctamente, entendida y aceptada.
- **3xx**: Redirección – más acciones tienen que tener lugar para completar la solicitud.
- **4xx**: Error del cliente – la solicitud contenía una mala sintaxis o no se pudo analizar en el servidor.
- **5xx**: Error del servidor – se produjo un fallo en el servidor al analizar una solicitud aparentemente válida.
- **6xx:** Fallo global – se produjo un error general.

El formato de las cabeceras viene definido por el nombre de la cabecera. La forma general de estas cabeceras consiste en el valor de la cabecera seguido de un punto y coma y pares que forman el nombre del parámetro y su valor. Se puede observar este formato a continuación:

```
Nombre-cabecera: valor-cabecera *(;nombre-parámetro=valor-parámetro)
```

Se indica con el asterisco que los campos que se encuentran dentro del paréntesis son opcionales, por lo que podrían no encontrarse en algunas cabeceras.

2.2.2 Interacción Cliente-Servidor

Un *User Agent* (UA) es la entidad SIP que interactúa con el usuario y mediante el cual el usuario puede establecer llamadas a través de Internet [15].

Cuando un UA quiere realizar una llamada, se pone en contacto con un servidor para poderla llevar a cabo dado que solamente conoce la URI del mismo. Ante esta consulta el servidor puede actuar de dos formas diferentes, como *proxy server* o como *redirect server*. A continuación se detallan estos dos modos de operación.

En la Figura 2.3 a) se puede ver el modo de operación de un *proxy server*. En el escenario de la figura, la chica quiere invitar a una sesión a Bob, de quien conoce su

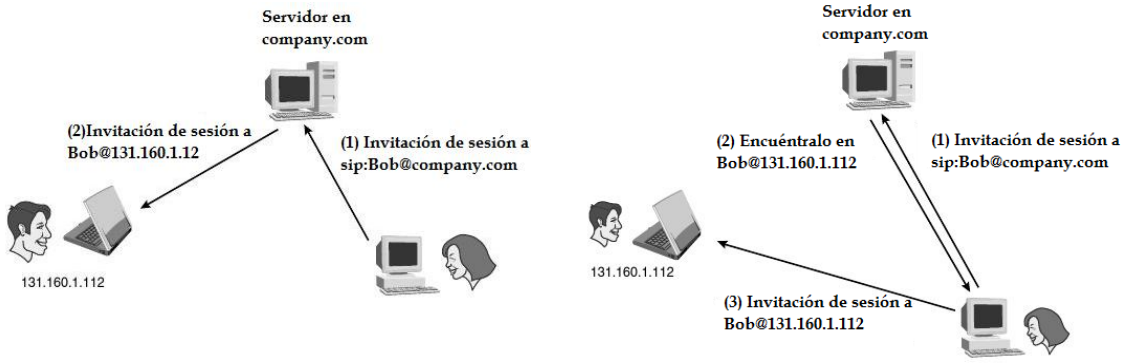
URI (Bob@company.com), por lo que envía un mensaje al servidor. Éste, que conoce la dirección de Bob, reenvía este mensaje hasta el mismo.

En la Figura 2.3 b) la chica envía el mensaje al servidor para iniciar una sesión con Bob. Pero en este caso se trata de un *redirect server*, por lo que el servidor le devuelve un mensaje con la dirección en la que puede encontrar a Bob. Tras estos mensajes, ésta ya puede enviarle el mensaje para el inicio de la sesión.

2.2.3 Operación del protocolo

Para que dos clientes establezcan una llamada satisfactoriamente, deben seguir al menos el siguiente intercambio de mensajes (véase la Figura 2.4):

- En primer lugar, el cliente que quiere realizar la llamada envía un mensaje INVITE.
- El receptor del mensaje INVITE, opcionalmente puede enviar una respuesta *180 Ringing* comunicando que la llamada ha llegado y el teléfono está sonando.
- Cuando el receptor de la llamada acepta la llamada, envía un mensaje *200 OK* indicándolo.
- Para el establecimiento de llamadas se usa un intercambio de tres pasos (*three-way handshake*): *Invite,* 200 *OK* y *ACK.* Así pues, el llamante debe confirmar la recepción del mensaje 200 *OK*. La necesidad de este tipo de intercambio se comentará en capítulos posteriores.
- Tras este intercambio, se establece la comunicación de voz entre ambos.
- Para finalizar la misma, el usuario que desea colgar envía una solicitud *BYE*.
- El otro participante, debe confirmar la recepción mediante un *200 OK*.



a) *Proxy Server* **b)** *Redirect Server*

Figura 2.3: Modos de operación de los servidores

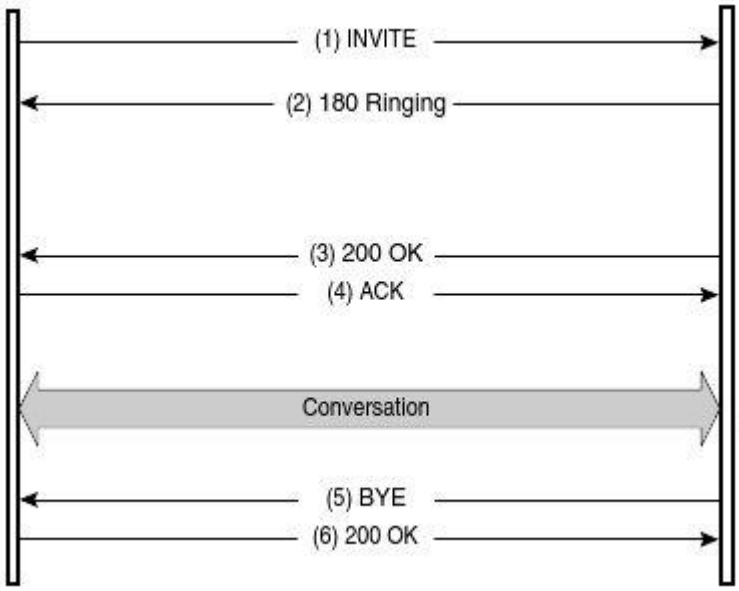


Figura 2.4: Establecimiento y cierre de una llamada mediante SIP

2.3 Redes P2P

La computación o trabajo en red P2P (*Peer-to-Peer*) es una arquitectura de aplicación distribuida que reparte tareas o cargas de trabajo entre los equipos pertenecientes a la misma, conocidos como pares (*peers*). En español, las redes P2P suelen, a menudo, traducirse como *redes entre pares* o *redes entre iguales* puesto que los pares disfrutan de los mismos privilegios en el entorno de dicha aplicación, trabajando de forma equitativa al aplicárseles a todos los mismos roles tanto de servidor como de cliente.

En este tipo de entornos los pares ponen a disposición de los otros participantes de la red una porción de sus recursos, tales como potencia de procesamiento, almacenamiento en disco o ancho de banda, sin la necesidad de una coordinación central realizada por servidores o equipos estables.

A continuación se van a describir los tipos de redes P2P principales aplicadas a SIP.

2.3.1 P2P Desestructurado y SIP (UP2P SIP)

Este tipo de red P2P se propone para telefonía en Internet móvil [16]. La ventaja de construir una red P2P de forma desestructurada es principalmente la robustez frente a la movilidad de los nodos.

Al contrario que en la selección de vecinos aleatoria, en ésta se propone establecer enlaces P2P con los *amigos* que se encuentran en línea, llamados *buddies*. Así pues, UP2P SIP puede construir una red de amigos y nodos solamente teniendo constancia de los nodos en los que se está interesado. La idea de esta topología de red viene dada por la propiedad de que los usuarios normalmente llaman a sus amigos, y no a extraños.

Además las relaciones de amistad de nuestros amigos pueden llevarnos a encontrar extraños en pocos pasos, debido al "fenómeno del mundo pequeño"[3].

Por lo tanto, los nodos SIP podrían encontrar con facilidad tanto a aquellos que son nuestros amigos como a los que no lo son en esta red, y establecer llamadas con estos de forma rápida.

2.3.2 Red Chord

Este tipo de redes tiene tres características que las distinguen de la mayoría de otras redes P2P: simplicidad, exactitud y buen rendimiento.

Chord es un protocolo empleado para realizar búsquedas en tablas hash distribuidas (*Distributed Hash Table* - DHT). Una tabla hash es una estructura de datos asocia claves con valores, siendo la clave, un valor generado como el resultado de una función hash, de los datos que se van a almacenar. Los datos se almacenan en la entrada de la tabla correspondiente al valor obtenido de la realización de la función hash. Las tablas hash distribuidas son tablas hash en las que la responsabilidad de almacenar los datos recae sobre distintos nodos de forma distribuida. Estos nodos (pares) son los que conforman la red y se encargan de mantenerla.

Este tipo de redes [17] se centran en afrontar uno de los problemas fundamentales de las aplicaciones P2P: conocer el nodo que almacena algún dato en particular. Para ello, Chord sólo realiza una operación: dada una clave, Chord traza una ruta hacia un nodo. Este nodo es independiente de quién calcule la ruta, ya que sólo depende de la clave para la que se calcula. De esta manera, la localización de datos se puede realizar fácilmente asociando un dato con su clave y almacenando este par clave-dato en el nodo final de la ruta trazada. Esta topología se adapta eficientemente a la unión, al abandono de nodos y puede resolver búsquedas incluso si el sistema está continuamente cambiando.

Chord utiliza una variante de hash consistente (*consistent hashing*) para asignar las claves a los nodos. El hash consistente tiende a balancear la carga de forma que cada nodo recibe aproximadamente el mismo número de claves y reduce el movimiento de claves cuando un nodo se une o abandona la red.

Normalmente, en el hash consistente se supone que los nodos son conscientes de la mayoría de los nodos en el sistema, haciéndolo poco escalable para un gran número de nodos. Por el contrario, usando Chord cada nodo necesita solamente la información de rutas hacia unos pocos, dado que las tablas de rutas están distribuidas. Así, en un sistema con N nodos, cada nodo almacena solamente la información de $O(log\ N)$ de ellos y resuelve las búsquedas enviando $O(log\ N)$ mensajes. La unión o el abandono de alguno, implicaría aproximadamente el envío de $O(log^2\ N)$ mensajes. [17]

El hash consistente asigna a cada nodo un identificador de *m* bits, usando como base una función hash cualquiera, como podría ser *SHA-1*. El identificador de un nodo se elige realizando la función hash de la dirección IP de ese nodo, mientras que el

[3] Este fenómeno, a veces referido como "seis grados de separación" dice que tendríamos un enlace con cualquier persona de la tierra con seis saltos entre "amigos de amigos". [26]

identificador de una clave se produce haciendo la función hash de la clave. La longitud del identificador (*m*) debe ser suficientemente grande como para que no se produzca colisión en la realización de la función hash.

La asignación de las claves a los nodos se realiza como sigue: los identificadores son ordenados en un círculo, quedando por lo tanto un círculo de identificadores módulo 2^m. Se asigna una clave *K* al primer nodo que tenga un identificador mayor o igual a ésta. Éste se conoce como *nodo sucesor* de la clave *K*, y se denota como *successor(k).* Si se representa este nodo en un círculo desde *0* hasta $2^m - 1$, el *successor(k)* será el primer nodo en el sentido de las agujas del reloj partiendo desde *K*. En la Figura 2.5, se puede ver el ejemplo de un círculo con *m* igual a tres, y con tres nodos en la red: 0, 1, y 3. En la figura, se puede ver, como se asignan los sucesores y el almacenamiento de las claves.

Este tipo de red, como ya se comentó, está diseñada para permitir a los nodos entrar y salir de la misma con un mínimo desorden. Para mantener las rutas del hash consistente, cuando un nodo *n* se une a la red, se le asignan las claves que se encuentran entre él, y su predecesor, claves que antes correspondían al que ahora es su sucesor. Cuando un nodo *n* abandona la red, todas las claves que tenía asignadas son reasignadas a su sucesor. En este tipo de redes no hay necesidad de otros cambios.

Búsqueda de claves

Cada nodo sólo necesita ser consciente de su sucesor en el círculo. Las solicitudes para un identificador dado se pueden enviar a través del círculo, pasando de sucesor en sucesor hasta encontrar al identificador deseado. Una parte del protocolo Chord se asegura de mantener un puntero hacia su sucesor. De esta forma se asegura que todas las búsquedas se resolverán satisfactoriamente.

Sin embargo, este esquema de resolución de búsquedas es ineficiente dado que puede requerir que se atraviesen todos los nodos del anillo para encontrar a un nodo. Para acelerar el proceso, Chord mantiene además información de otras rutas.

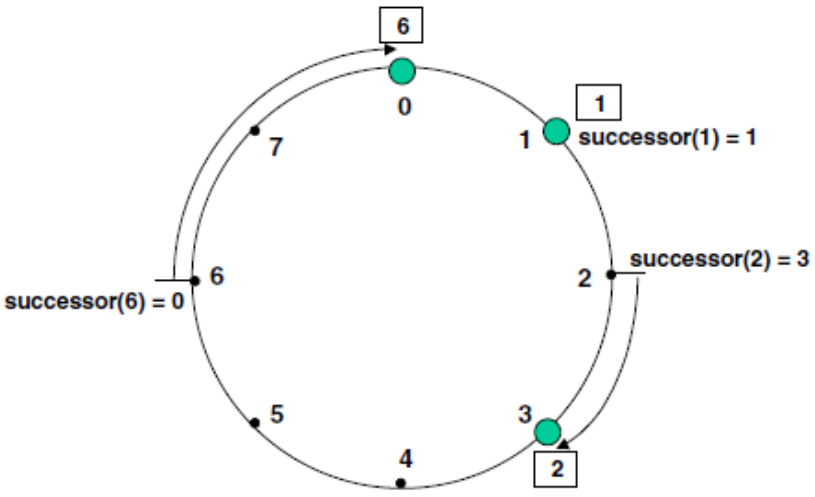


Figura 2.5: Asignación de claves a los nodos

Si *m* es el número de bits permitido para los identificadores, cada nodo *n* mantendrá una tabla de rutas con (como máximo) *m* entradas, llamada *finger table*. La *i-ésima* entrada en la tabla del nodo *n* contiene la identidad del primer nodo (*s*) que sucede a *n* más 2^{i-1} en el anillo, de forma que $s = successor(n + 2^{i-1})$ donde $1 \leq i \leq m$ con aritmética módulo 2^m. Al nodo *s* se le llamará el *i-ésimo finger* del nodo *n*, denotándose como *n.finger[i].node* (véase la Tabla 2.1).

Notación	Definición
finger[i].start	$(n + 2^{i-1}) \bmod 2^m$, $1 \leq i \leq m$
.interval	[*finger[i].start* , *finger[i+1].start*)
.node	Primer nodo mayor o igual que *finger[i].start*
successor	Siguiente nodo (a *n*) en el anillo; *finger[1].node*
predecessor	Nodo previo a *n* en el anillo.

Tabla 2.1: Definición de variables para el nodo *n* con *m* bits de identificación

Una entrada en la *finger table* incluye tanto el identificador Chord como la dirección IP y el puerto del nodo al que corresponda. Nótese que el primer *finger* de *n* es el sucesor del nodo en el anillo. En la Figura 2.6 se puede ver un ejemplo de anillo en el que se han usado *3 bits* para los identificadores por lo que cada nodo tiene tres entradas en su tabla de rutas. En esta red hay tres pares, *0, 1* y *3*.

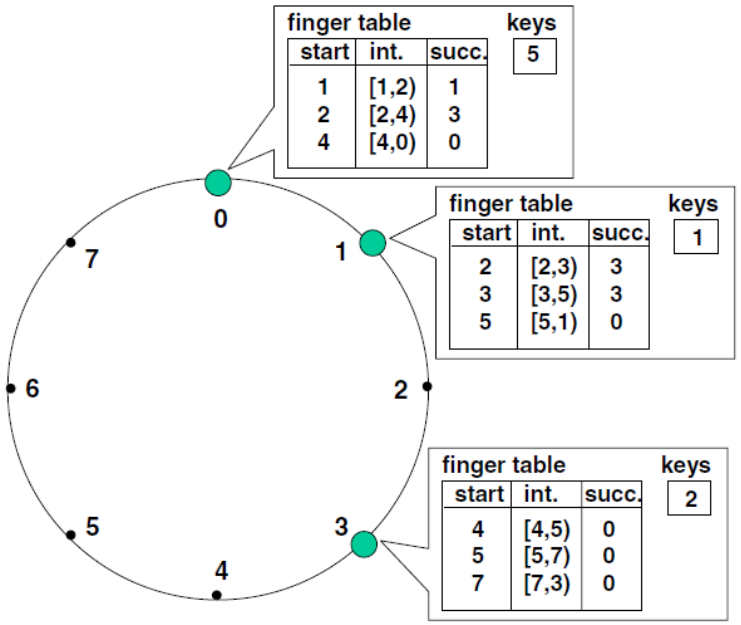


Figura 2.6: Ejemplo de red Chord con *m* = 3

En este ejemplo, la *finger table* del nodo 1 apunta a los nodos sucesores de los identificadores $(1+2^0) \bmod 2^3 = 2$, $(1+2^1) \bmod 2^3 = 3$ y $(1+2^2) \bmod 2^3 = 5$ respectivamente. El sucesor del identificador 2 es el nodo 3, el sucesor del identificador 3 es el nodo 3 y de forma trivial, el sucesor de 5 es el nodo 0.

Este esquema tiene dos características importantes. La primera es que cada nodo almacena información solamente sobre un pequeño número de nodos, y tiene más información sobre nodos siguientes cercanos en el anillo que sobre nodos lejanos. La segunda es que la *finger table* de un de un nodo normalmente no contiene suficiente información para determinar el sucesor de una clave arbitraria *k*. Por ejemplo, el nodo 3 en el ejemplo anterior no conoce al sucesor de 1, ya que éste (el propio nodo 1), no aparece en la *finger table* del nodo 3.

Cuando un nodo *n* no conoce el sucesor de una clave dada *k*, se actúa como sigue. Si el nodo *n* pudiese encontrar un nodo cuya ID esté más cercano a *k* que él mismo, ese nodo sabría más que el propio nodo *n* sobre la región del círculo en la que se encuentra *k*. Así pues, el nodo *n* busca en su *finger table* un nodo *j* tal que su ID sea la más cercana que precede al nodo *k*, y pregunta a ese nodo *j* sobre el nodo que él conoce cuya ID esté más cercana a *k*. Repitiendo este proceso, el nodo *n* va descubriendo nodos cuyas IDs están cada vez más cerca del nodo *k*. Siguiendo este procedimiento, si en el ejemplo anterior el nodo 3 quisiera encontrar el sucesor del identificador 1, miraría en su *finger table* y observaría que pertenece al intervalo [7, 3). El nodo asignado a esta entrada de la tabla es el 0, por lo que el nodo 3 preguntaría al nodo 0 acerca del nodo 1. Cuando éste reciba la pregunta, observará en su tabla que el sucesor del nodo 1 es el propio nodo 1, por lo que devolverá al nodo 3 una respuesta indicando que el sucesor que buscaba es el nodo 1.

Unión de nodos.

En una red dinámica, los nodos pueden unirse y abandonarla en cualquier momento. Sin embargo, los nodos deben preservar la habilidad de localizar cualquier clave en la red. Para conseguirlo, Chord necesita preservar dos condiciones:

- El sucesor de un nodo tiene que mantenerse correctamente.
- Para cada clave k, el sucesor de k es su responsable.

Además, para que las búsquedas se puedan realizar de forma rápida, es conveniente también que las *finger tables* se mantengan de forma correcta.

Por último, para simplificar los mecanismos de unión y abandono, cada nodo Chord mantiene también un puntero a su predecesor. Éste conteniene el identificador del nodo, su dirección IP y puerto, y puede usarse para moverse en el sentido contrario a las agujas del reloj por el anillo de identificadores.

Para mantener las condiciones anteriormente mencionadas, Chord debe realizar tres tareas cuando un nodo *n* se une a la red (se considera que *n'* es un nodo de la red que conoce al nodo *n* mediante algún mecanismo externo):

- **Inicialización de *fingers* y del predecesor:** El nodo *n* aprende quién es su predecesor y cuál es su *finger table* preguntándole al nodo *n'* que ya forma parte de la red.
- **Actualización de los *fingers* en los nodos que ya estaban en la red:** El nodo *n* debe formar parte de las tablas de otros nodos, por lo que los nodos que ya formaban parte de la red deben actualizar sus tablas e incluir el nodo *n* si le correspondiera.
- **Transferencia de las claves:** La última operación necesaria para completar la unión es que el sucesor del nodo *n* le transfiera las claves que ahora le corresponde almacenar.

2.4 Multimedia

2.4.1 Real-time Transport Protocol (RTP)

El protocolo RTP (*Real-time Transport Protocol*) [18] es el protocolo de transporte para flujos multimedia en Internet. Fue diseñado para trabajar con *IP multicast*, aunque se utiliza también de forma *unicast*, para proporcionar información temporal y de sincronización de flujos multimedia. Se trata de un protocolo ligero (*light-weight protocol*) sin mecanismos de control de errores ni de control de flujo. Además, no proporciona ni reserva de recursos ni control de la calidad de servicio. Se trata de un protocolo de transporte independiente de la tecnología de red sobre la que se utilice. En la Figura 2.7 se puede observar cómo encaja RTP en la pila de protocolos TCP/IP, así como otros protocolos mencionados anteriormente en este capítulo.

RTP/RTCP proporcionan todas las características para la transmisión de información de flujos multimedia en tiempo real y ofrecen los mecanismos necesarios para que las aplicaciones reproductoras puedan conseguir una *sincronización local* entre los flujos recibidos (*sincronización inter-flujo*).

Estas transmisiones de información multimedia se realizan utilizando una *sesión RTP*, que consiste en la asociación de un grupo de participantes que se intercambian un mismo flujo mediante RTP en una *sesión multimedia*. Dicho flujo quedará definido por una pareja particular de direcciones de destino a nivel de transporte (una dirección de red más dos puertos, uno para RTP y otro para RTCP). El par de direcciones destino a nivel de transporte podrán ser comunes para todos los participantes, como en el caso de *IP Multicast*, o podrán ser diferentes para cada uno, como en el caso de transmisiones *unicast.*

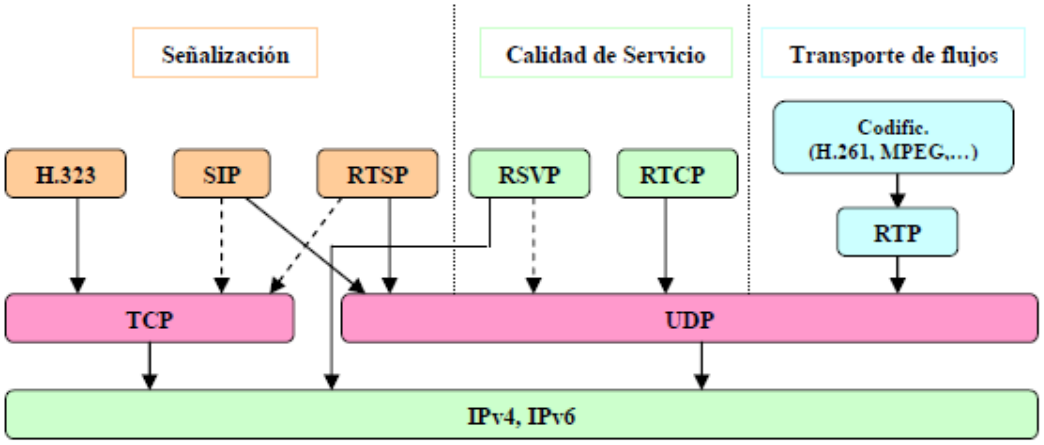


Figura 2.7: Pila de protocolos sobre IP.

Para identificar la procedencia de un flujo de paquetes RTP se utilizan los denominados *Identificadores de Fuente* o *SSRC* (*Synchronization Source*), únicos en cada sesión e incluidos en la cabecera RTP. Los SSRC son necesarios para que los receptores puedan identificar los paquetes del flujo provenientes de cada fuente involucrada en la sesión multimedia. Todos los paquetes procedentes de la misma fuente formarán parte del mismo espacio de temporización y de números de secuencia. Un ejemplo de fuente RTP puede ser un transmisor de un flujo de paquetes generado a partir de una fuente de señal tal como un micrófono o una cámara de vídeo.

La cabecera de un paquete RTP tiene el formato mostrado en la Figura 2.8. Los primeros veinte octetos aparecen en todos los paquetes RTP, mientras que la lista de los identificadores CSRC se presentan sólo cuando los inserta un dispositivo *mezclador*.

Los veinte octetos obligatorios en cada paquete RTP corresponden a los siguientes campos:

- **V:** Versión de RTP.
- **P:** Relleno (*padding*). El paquete contendrá al menos un octeto de relleno.
- **X:** Bit de extensión, para indicar si hay cabeceras de extensión.
- **CSRC:** Número de identificadores CSRC que hay en la cabecera.
- **M:** Marcador (*marker*), definible en cada perfil.
- **PT:** Tipo de carga (*payload type*). Identifica el formato útil de la carga, para su interpretación por parte de otras aplicaciones.
- **Número de secuencia**
- **RTP Timestamp:** Marca temporal que refleja el instante de muestreo del primer octeto del paquete de datos RTP.
- **SSRC:** Identificador de la fuente elegido aleatoriamente.
- **CSRC:** Identifica las fuentes que contribuyen a la carga útil de este paquete.

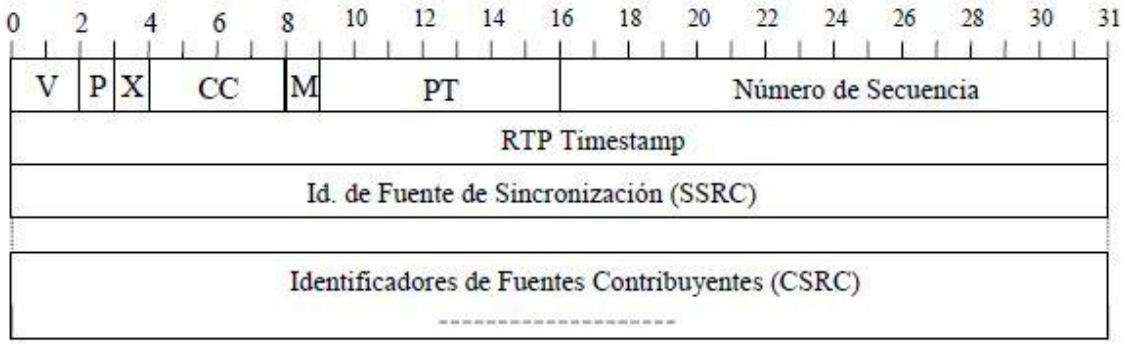


Figura 2.8: Cabecera RTP

2.4.2 Códecs VoIP

Existen una gran variedad de códecs para telefonía sobre IP. Muchos de estos códecs están estandarizados (e.g. por la ITU-T en forma de recomendación) y otros son propietarios.

Nombre	Bit rate (kbps)	Tiempo entre muestras (ms)	Tamaño típico de paquete (Bytes)	MOS
G.711	64	10	240/160/80	4.1
G.729	8	10	40/20	3.92
G.723.1	6.3	30	48/24	3.9
G.723.1	5.3	30	40/20	3.8
G.726	32	5	120/60	3.85
G.726	24	5	80/40	3.7
G.728	16	5	80/40	3.61
G.722	64	10	160	4.13
iLBC_20	15.2	20	38	~4
iLBC_30	13.33	30	50	~4
SILK(Skype)	6 - 40	20	20/40/60/80/100	~4.4
Speex	2 - 44	10 - 30	20-80	--

Tabla 2.2: Principales códecs de la ITU-T

En la Tabla 2.2 [19] [20] [21] se muestran algunos de los códecs más comunes. En esta tabla se especifican la tasa binaria (*bit rate*) que generan, el tiempo transcurrido entre la captura de muestras, el tamaño del paquete típico en el que se envían las muestras codificadas, el número de paquetes por segundo que se envían y la máxima puntuación media de opinión (*Mean Opinion Score* - MOS)[4].

[4] El MOS es una valoración de la calidad de la voz subjetiva que varía de una evaluación a otra dependiendo de una gran variedad de factores, como por ejemplo pueden ser el entorno acústico o la metodología. Los valores que se muestran en la tabla van desde cinco (excelente) hasta uno (malo).

Capítulo 3. Especificación de requisitos

En la sección 1.3 se describieron los objetivos y el alcance del proyecto. En este capítulo, primero se hará una descripción de los requisitos generales que se precisan para llevar a cabo los objetivos propuestos. En segundo lugar se describirán con más detalle qué requisitos funcionales específicos se considerarán para el desarrollo del proyecto.

3.1 Requisitos no funcionales

La aplicación constará de dos bloques claramente diferenciados: cliente y *peer*. El cliente será el bloque encargado de gestión de las llamadas, mientras que el bloque *peer*, será el encargado de la gestión de la red. Tanto para uno como para otro hay una serie de requisitos comunes:

- **Sistema operativo:** La aplicación requiere de un sistema operativo que le permita comunicarse con el hardware involucrado en su ejecución. Este sistema operativo será Bada, para el que la aplicación ha sido desarrollada.
- **Interfaz de usuario:** Debe ofrecerse al usuario una interfaz gráfica mediante la que poder recibir e introducir la información necesaria para interactuar con la aplicación. Teniendo en cuenta el tipo de terminales hacia los que se destina, esta interactuación se realizará de forma táctil.
- **Hardware:** Es necesario que la aplicación tenga acceso, a través de la API correspondiente, a diferentes recursos hardware del terminal. Así, debe tener acceso al detector de gestos de la pantalla táctil, de forma que el programa tenga información de, por ejemplo, si se está arrastrando el dedo por la pantalla para desplazar ésta en algún sentido, se está pulsando algún botón, o se está escribiendo algo en el teclado virtual que el sistema operativo proporciona. También es imprescindible el uso de la interfaz WiFi.

- **Interfaces de comunicaciones:** Debe poder hacerse uso de las interfaces de comunicación requeridos en una arquitectura de red UDP/IP, teniendo acceso a la información necesaria (e.g. la dirección IP), la posibilidad de modificar los principales parámetros de estas interfaces (e.g. el tamaño del buffer de recepción) y elegir el tipo de recepción (bloqueante o no bloqueante).

En cuanto a la exigencia de conocimientos que se requiere al usuario, sólo se necesita que esté habituado al uso de terminales con características similares. Para que esto sea posible, se plantea otro requisito que consiste en que la aplicación sea intuitiva y sencilla de usar y comprender para cualquier usuario, sin que este tenga que tener ningún conocimiento de los mecanismos y protocolos sobre los que se sustenta el proyecto.

Además, dado que esta aplicación está destinada para funcionar con el sistema operativo Bada como ya se comentó anteriormente, es necesario que se disponga de algún terminal que opere con el mismo. Dado que Bada es propietario de la compañía SAMSUNG, es necesario contar con alguno de los siguientes dispositivos fabricados por esta compañía: GT-S5250, GT-S5330, GT-S7230E, GT-S5780, GT-S8500, GT-S8530, GT-S5380, GT-S7250 y GT-S8600.

3.2 Requisitos funcionales

Como ya se ha comentado anteriormente el sistema consta de dos partes bien diferenciadas. Por un lado, los clientes actúan como teléfonos VoIP convencionales. Además, será necesaria alguna funcionalidad específica con respecto a la red *peer-to-peer* mediante la que se interconectan. Por otro lado, los *peers* son los encargados de crear y administrar esta red. Por lo tanto, la funcionalidad que se requiere se puede dividir en dos secciones: una referente a la telefonía VoIP, y otra referente a la red P2P.

3.2.1 Telefonía VoIP

Desde este punto de vista, las partes implicadas en una llamada son siempre dos clientes. La funcionalidad requerida sería la siguiente:

- **Establecer una llamada:** Un cliente deberá poder enviar una solicitud de llamada enviando una invitación a otro cliente, conociendo la dirección IP de éste. También debe poder aceptar una invitación recibida.
- **Mantener una conversación:** Debe cumplir con la funcionalidad necesaria para el mantenimiento de una conversación, que consiste en simultáneamente:
  - Capturar muestras del micrófono del terminal, codificarlas, formar un paquete agrupando varias de ellas y enviarlas a través de la red.
  - Recibir paquetes a través de la red, decodificar las muestras presentes en éste, y reproducirlas por el altavoz del dispositivo.
- **Finalización de una llamada:** Cualquiera de los participantes en una llamada debe poder finalizar la misma en el momento en el que lo desee.

- **Rechazar una llamada:** Un cliente que esté recibiendo una llamada no tiene por qué aceptar la misma, pudiendo rechazarla y comunicándoselo al cliente que la solicitó.
- **Cancelación de una llamada no establecida:** Si un cliente invita a otro a una llamada, tendrá la posibilidad de cancelar esta invitación antes de que la misma sea aceptada.

En la Figura 3.1, se puede ver un diagrama con la interacción entre dos clientes. En uno de ellos se desglosan las funciones que debe disponer para cumplir con los objetivos del proyecto. Para la correcta interpretación de la figura, se debe tener en cuenta que todos los bloques encerrados por la zona sombreada componen un cliente, estando también presentes en el segundo cliente (círculo). No se desglosan los dos clientes por simplicidad.

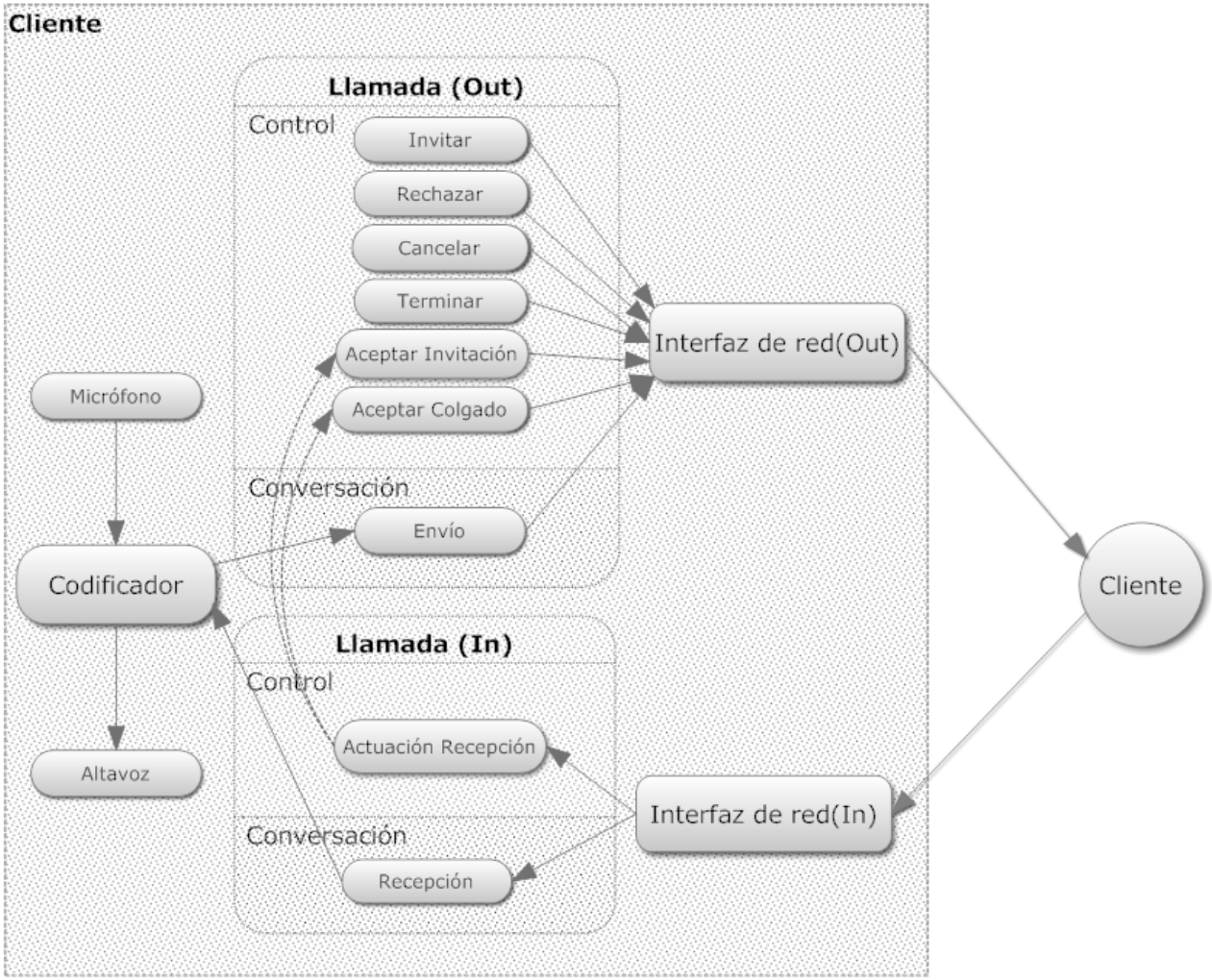


Figura 3.1: Interacción entre clientes

3.2.2 Red P2P

Desde el punto de vista de la red P2P, se exigen funcionalidades en las que se ven implicadas tanto los clientes como los *peers*. Se incluyen en esta sección, por lo tanto, las interacciones que se realizan entre un cliente y un *peer*, y las que se realizan entre dos *peers*.

La interacción entre un cliente y un *peer* se realiza a través de las siguientes actividades:

- **Descubrir:** El cliente debe tener mecanismos que le permitan localizar algún *peer* en su arranque, cuando aún no conoce a nadie.
- **Registrarse:** Un cliente puede registrar su dirección y ubicación en un *peer* mediante algún mensaje destinado para ello.
- **Darse de baja:** De igual modo que el cliente se puede registrar, debe tener la posibilidad de eliminar este registro.
- **Pedir una búsqueda:** Encomendar al *peer* que le corresponda, la búsqueda de la dirección y ubicación de un cliente a través del nombre del mismo.
- **Almacenar clientes:** Un *peer* tiene que almacenar la dirección y la ubicación de aquellos clientes que quieran registrarse y le pertenezca a él almacenar.

La interacción que se debe llevar a cabo entre *peers* se realiza a través de las siguientes actividades:

- **Descubrir:** Mecanismo mediante el cual un *peer*, cuando comienza su ejecución, puede localizar a otros *peers* que ya se están ejecutando y que forman parte de la red en la que está interesado.
- **Crear una red:** Si cuando un *peer* comienza su ejecución no es capaz de descubrir ningún otro *peer*, éste puede crear una nueva red en la que podrá ser descubierto por otros.
- **Unión a una red:** Un *peer*, conociendo otro *peer* que forme parte de la red a la que se quiere unir, puede unirse a ella a través de dicho *peer*. Se conecta a través de un *peer* puesto que es el enlace que tiene con la red, y será este quien se encargue de facilitarle su unión.
- **Salir de una red:** De igual modo que puede entrar a formar parte de una red, un *peer* podrá abandonar la misma en cualquier momento, comunicándoselo a alguno de sus compañeros.
- **Estabilizar:** Debe haber algún mecanismo para la estabilización de la red, de forma que ésta converja a una situación estacionaria mientras no entren ni salgan *peers*.
- **Entrada de un *peer*:** Si algún *peer* en su fase de inicio se pone en contacto con un *peer* que ya forma parte de la red con el fin de unirse a ésta, éste debe facilitar y administrar la entrada del nuevo *peer* a la red, trasladando esta solicitud al nodo de la red al que corresponda ubicarlo.

- **Salida de un *peer*:** Si un participante de la red desea abandonar la misma, se lo comunicará a algún compañero. El compañero, debe almacenar aquellos datos almacenados por el *peer* que desea abandonar la red, de forma que no haya una pérdida de información ante la salida de uno de los *peer*.
- **Búsqueda en la red:** Es necesario que cualquier nodo pueda localizar a cualquier otro de la red conociendo su dirección de correo, mediante una búsqueda que sólo requiera de esa información.

En la Figura 3.2 se muestran estos requisitos mediante un diagrama. En ésta se pueden ver dos clientes y dos pares, pero al igual que en la Figura 3.1 no se detallan todos. Solamente se detallan un cliente y un *peer* por simplicidad, dado que los otros serían iguales. Además, cabe destacar que las funciones del *peer* han sido divididas en dos secciones, la administración de la red por un lado y las relativas a la gestión de tablas hash por otro. Dos de los puntos anteriores, *Entrada de un peer*, y *Salida de un peer*, están agrupados en el bloque *Modificar Tablas*.

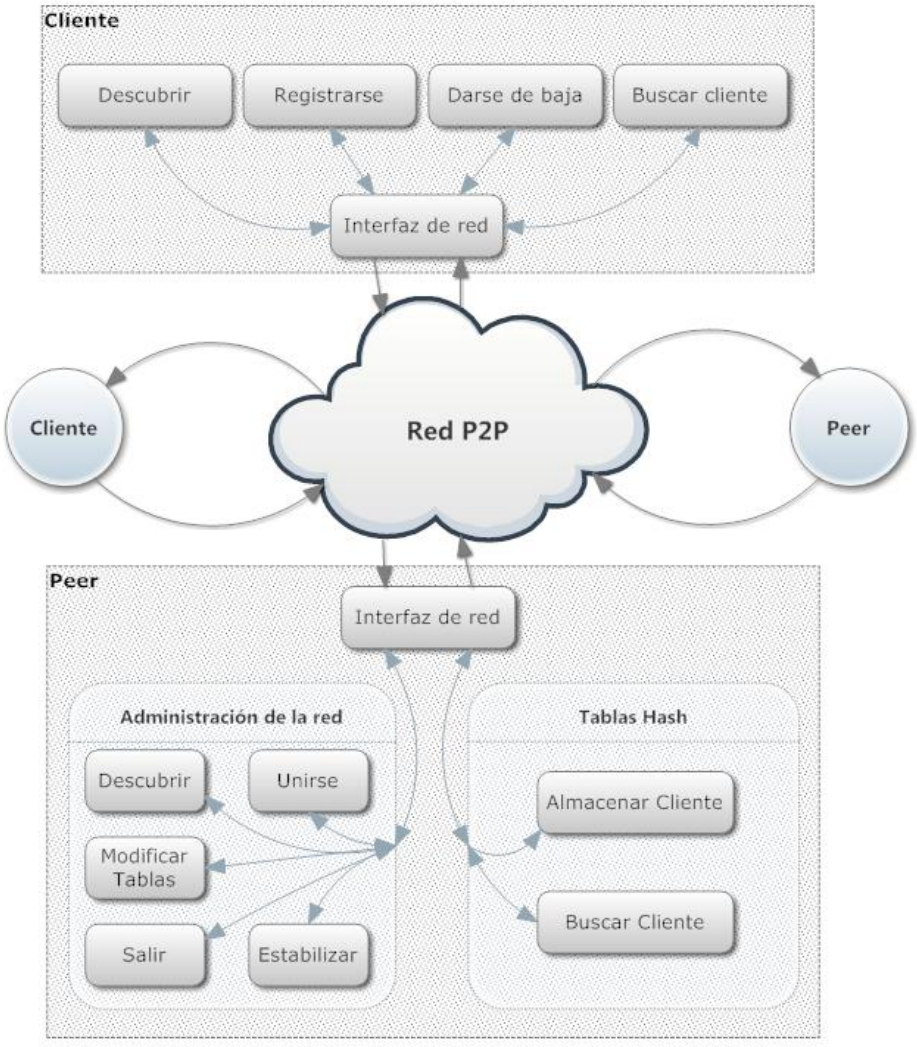


Figura 3.2: Interacción para la red P2P

Capítulo 4. Planificación y estimación de costes

4.1 Recursos

En este apartado se detallan los recursos utilizados para el desarrollo del proyecto, separados en tres categorías: recursos humanos, recursos hardware y recursos software.

4.1.1 Recursos Humanos

- Jorge Navarro Ortiz, profesor ayudante doctor del Departamento de Teoría de la Señal, Telemática y Comunicaciones de la Universidad de Granada, como tutor del proyecto.

- Jesús Recuerda Hueso, alumno de la Escuela Técnica Superior de Ingeniería Informática y de Telecomunicaciones de la Universidad de Granada, como autor del proyecto.

4.1.2 Recursos Hardware

- Varios ordenadores personales.

- Teléfono Samsung GT-8500.

4.1.3 Recursos Software

- Sistema operativo Windows 7.

- Entorno de desarrollo para Bada, SDK-1.2.

- OpenOffice para la elaboración de la memoria.

- Visual Paradigm 8.0.3 Trial Version para la elaboración de las máquinas de estados.
- ArgoUML para el diseño de los diagramas de clases.

4.2 Fases de desarrollo

El desarrollo del proyecto se reparte en diferentes fases que se especifican en este apartado. En la Figura 4.1 se puede ver un diagrama con las fases de desarrollo del proyecto ubicadas en el tiempo.

4.2.1 Revisión del estado del arte

Se trata de una revisión de aplicaciones similares ya existentes tanto para este sistema operativo como para otros. También se realiza un análisis de los diversos protocolos que se pueden utilizar para este tipo de aplicaciones.

4.2.2 Especificación de requisitos

Tras analizar el estado del arte y los protocolos utilizables, se establecen las bases de los objetivos del proyecto, detallando para qué situaciones se plantea y cuáles son las funcionalidades que se desean conseguir.

4.2.3 Diseño

Se detalla una solución concreta a los requisitos planteados, especificando los protocolos a utilizar y cómo se hará uso de éstos.

4.2.4 Implementación

En esta fase se implementa la solución planteada en la fase de diseño, haciendo uso del lenguaje y del entorno de desarrollo propuesto.

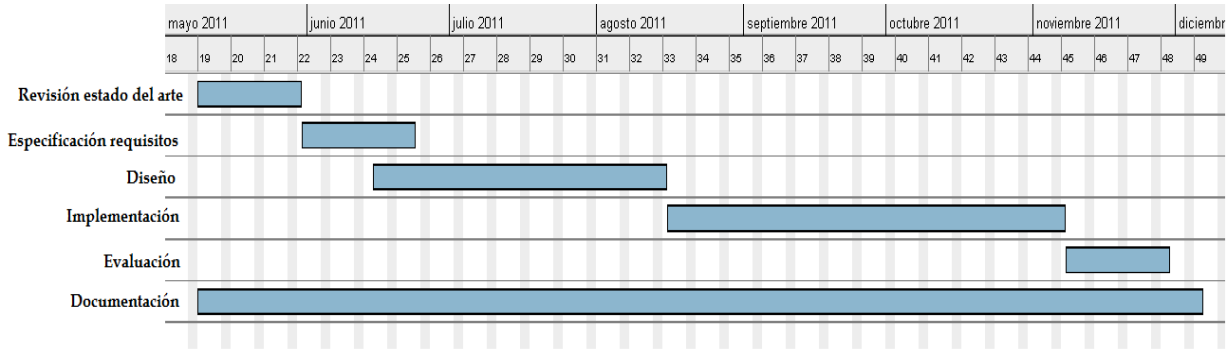


Figura 4.1: Fases de desarrollo

4.2.5 Evaluación y pruebas

Tras terminar la fase de implementación, se analizan los escenarios típicos, así como escenarios específicos susceptibles de error con el fin de evaluar la solución obtenida.

4.2.6 Documentación

La fase de documentación se lleva a cabo durante todo el proyecto, documentando las diferentes fases del mismo.

4.3 Estimación de costes

4.3.1 Recursos humanos

El coste de los recursos humanos se calcula en base a los días empleados para el desarrollo del proyecto. El coste en días, dividido en las diferentes fases de desarrollo, se puede ver en la Tabla 4.1. En el cálculo de los días no se tiene en cuenta ni sábados ni domingos.

Se estima un coste de 20€/hora, teniendo 8 horas de duración una jornada. El cálculo del coste en euros dividido en las diferentes fases de desarrollo del proyecto se puede ver en la

Figura 4.2.

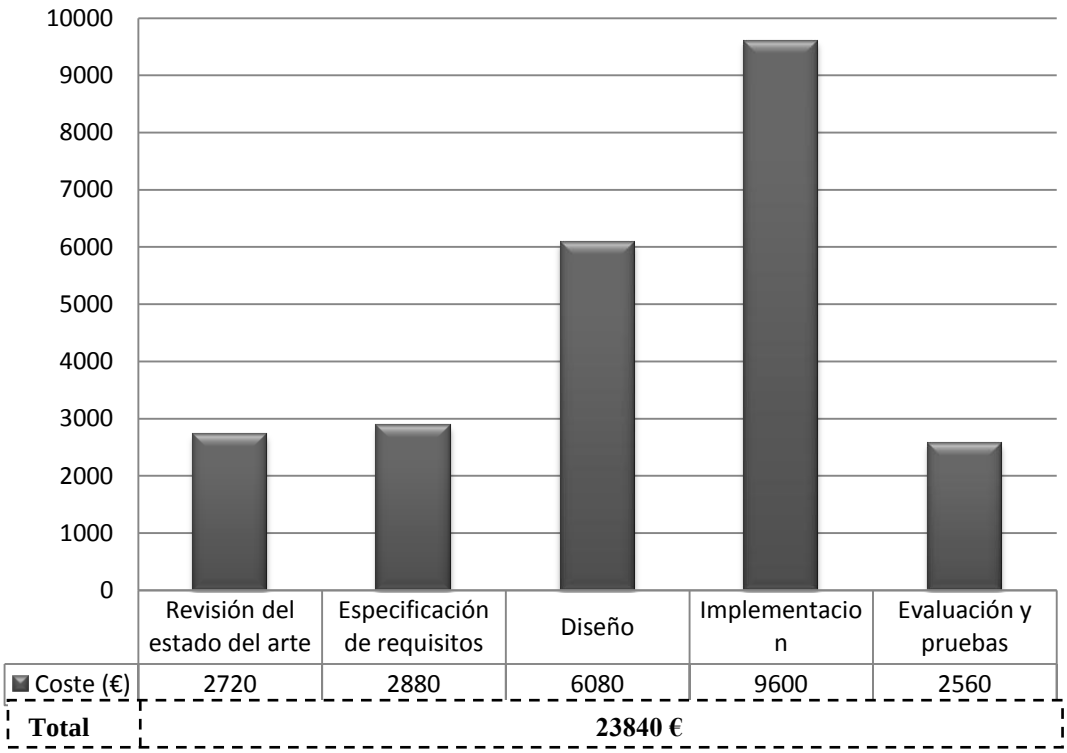


Figura 4.2: Coste estimado de los RR.HH.

Fase	Duración (días)
Revisión del estado del arte	17
Especificación de requisitos	18
Diseño	38
Implementación	60
Evaluación y pruebas	16

Tabla 4.1: Temporización del proyecto

4.3.2 Hardware

El coste de los elementos hardware, así como la vida media de estos, se detalla en la Tabla 4.2. El tiempo de vida medio que se presenta en la tabla es una estimación aproximada.

En el transcurso del proyecto se han usado hasta 4 ordenadores personales en períodos de tiempo de diferente duración. El coste hardware total se puede ver en la Tabla 4.3.

Elemento	Vida media	Coste (€)
Ordenador personal	36 meses	800
Teléfono móvil	24 meses	350

Tabla 4.2: Coste de cada elemento hardware

Elemento	Unidades	Tiempo de uso	Coste (coste/unidad x unidades x tiempo_uso/vida_media)
Ordenador personal	1	5.5 meses	800 x 1 x 5.5/36 = 122 €
Ordenador personal	1	1 mes	800 x 1 x 1/36 = 22.20 €
Ordenador personal	2	0.5 meses	800 x 2 x 0.5/36 = 22.20 €
Teléfono móvil	1	5.5 meses	350 x 1 x 5.5/36 = 53.5 €
Total			219.9 €

Tabla 4.3: Coste total del hardware

4.3.3 Software

En la Tabla 4.4 se puede apreciar el coste del software utilizado. Cabe destacar que la mayoría del software empleado es gratuito o se ha utilizado durante el período de prueba que permite la empresa desarrolladora.

4.3.4 Presupuesto general

El presupuesto general se puede observar en la Tabla 4.5.

Elemento	Coste
Windows 7	119 €
Procesador de textos	Gratuito
Bada-SDK	Gratuito
Virtual Paradigm 8.0.3	Período de prueba
ArgoUML	Gratuito
Total	119 €

Tabla 4.4: Coste de cada elemento software

Recursos	Coste
Humanos	23840 €
Hardware	219.9
Software	119 €
Total	24178 €

Tabla 4.5: Presupuesto general

Capítulo 5. Diseño

En este capítulo se tratará el diseño realizado para la aplicación que se desea conseguir, cumpliendo con los requisitos especificados en el 0

Para explicar el diseño realizado, este capítulo se divide en cinco secciones. En la primera se detallan algunas decisiones que requieren ser tomadas previamente. En segundo lugar se describe la estructura y el diseño de la red P2P. En la tercera sección se expone en detalle la estructura de bloques del sistema y cómo se comportan. En cuarto lugar se explica la interacción entre las dos principales partes en las que se divide el sistema. En la última sección se comentan algunas otras consideraciones de diseño que no han sido tratadas a lo largo del capítulo.

5.1 Decisiones previas

5.1.1 Modificaciones en la arquitectura cliente-servidor

La finalidad de este proyecto consiste en eliminar las infraestructuras necesarias para el funcionamiento de una aplicación VoIP tradicional. Se propone la utilización de una red P2P donde los nodos conectados se repartan el trabajo que se requiere en una arquitectura tradicional. En este trabajo se incluyen las labores de registro de clientes, localización de los mismos y, dependiendo de la implementación del servidor, la actuación o no como *proxy* para los mensajes generados desde unos clientes hacia otros.

Por lo tanto, son relevantes las modificaciones que deben llevarse a cabo en esta arquitectura para que el sistema siga funcionando de forma fiable.

Con el uso de una topología cliente-servidor, el servidor se encuentra en una ubicación concreta. De esta forma cualquier usuario que se una al sistema puede contactar con él, ya sea teniendo su dirección asignada de forma que no tenga que realizar ninguna acción previa o mediante una consulta al servidor de nombres dominio, preguntando por la dirección IP del dominio asignado al servidor SIP. Al modificar esta arquitectura, no existe un servidor específico. Por tanto, ni se tiene una dirección fija con la que ponerse en contacto, ni un dominio por el que realizar una consulta. Por esta

razón, los clientes del sistema deben tener un medio alternativo para localizar un nodo con el que ponerse en contacto.

Por otro lado, la misión del *registrar* en la arquitectura cliente-servidor consiste en almacenar la ubicación de los usuarios, haciendo referencia a un cliente mediante su *URI*. De esta forma se les permite movilidad, ya que haciendo una consulta al *registrar* respecto a la *URI* de un cliente, éste devolverá la dirección IP en la que podemos localizarlo ya que la tendrá almacenada y actualizada. Tras modificar la topología, esta misión se divide entre los pares que formarán la red. Por lo tanto, es necesario que los nodos tengan consciencia de qué les corresponde almacenar y qué no, así como que las búsquedas y registros puedan alcanzar el nodo destinado a almacenar esa información.

Si bien son necesarias modificaciones en la arquitectura como se ha comentado, siguen siendo dos los principales componentes del sistema, tradicionalmente cliente-servidor y ahora cliente-redP2P. Es por ello que se podría hacer transparente para el cliente, o casi transparente, la topología empleada en la red. De este modo, la implementación de un cliente SIP tradicional con mínimas modificaciones sería válida. Tras esta afirmación, se toma la decisión de implementar un cliente SIP que tendrá el menor número de modificaciones posibles respecto a un cliente SIP tradicional (en la sección 5.3.1 se detalla su diseño). Estas modificaciones principalmente consisten en la forma en la que se utilizan los mensajes *REGISTER*.

5.1.2 Sintaxis de los mensajes en la red P2P

Debido a sus características, la red P2P requiere mantenimiento. En este sentido, los nodos pertenecientes a la red tienen que intercambiar mensajes con referencia a la misma. Es por lo tanto importante decidir qué tipo de sintaxis o protocolo se utilizará para esos mensajes.

Dado que los pares recibirán mensajes SIP de los clientes y que, como se comentó en la sección anterior, se realizará un diseño en el que la topología será casi transparente para el cliente, los pares deben tener capacidad para entender y generar este tipo de mensajes. Así pues, deberán poder contestar a las solicitudes que se realicen. Se plantea, por lo tanto, la posibilidad de mantener la red P2P mediante mensajes SIP de forma que se simplifica el diseño de los mismos al sólo utilizar un tipo de mensajes.

De esta forma, el mantenimiento de la red P2P se realizará mediante mensajes SIP. Estos serán del tipo *REGISTER* únicamente y se usarán para dos finalidades según las cabeceras que contengan, a parte de la misión de registro de usuarios tradicional.

El mensaje de registro de usuarios sigue siendo similar al utilizado en la arquitectura basada en servidor. Éste contiene la cabecera *To*, representando el identificador del usuario, y la cabecera *Contact*, que contiene el identificador de usuario y la localización del mismo.

Los dos contextos en los que se usa el mensaje *REGISTER* para el mantenimiento de la red son *actualización* y *solicitud*. Si la cabecera *Contact* está presente se trata de un mensaje de actualización, en el cual se indica que se quiere actualizar la vinculación con

el nodo presente en la cabecera *To*. En caso de que la cabecera *Contact* no esté presente se trata de una solicitud, en la que se requiere la información del usuario presente en la cabecera *To.*

5.1.3 Quién es *peer*

La aplicación que se desea implementar está destinada a ejecutarse en dispositivos móviles con una capacidad de procesamiento relativamente alta, pero no todos los terminales tendrán la misma capacidad. Es por ello que surge la pregunta de quiénes formarán parte de la red y quiénes no, así como cuál será el criterio para decidirlo.

La solución que se plantea a esta cuestión consiste en que el usuario tenga la posibilidad de decidir si quiere ser cliente y *peer*, o si sólo quiere ser cliente. Si se elige la primera opción, la aplicación ejecutará los dos modos de operación. Si se elige ser cliente, únicamente se ejecutará la parte correspondiente al mismo.

La decisión por parte del usuario de elegir una u otra opción se puede basar en el siguiente razonamiento. Si el terminal en el que se está ejecutando la aplicación es un terminal con prestaciones bajas, elegir actuar sólo como cliente requerirá menos carga computacional. Esto se debe a que no tendrá interacción con la red en términos de mantenimiento y solicitudes de otros usuarios. Sin embargo, un usuario que ejecute la aplicación eligiendo actuar además como *peer* tendrá más carga computacional, pero sus acciones como cliente serán más rápidas y fiables. Esto se debe a que el *peer* que se ejecuta en el mismo terminal realiza las operaciones necesarias por el cliente en la red P2P, mientras que si no fuese *peer* el acceso a la red lo realizaría poniéndose en contacto con otro terminal.

5.1.4 Servidores *proxy* o *redirect*

En la sección 2.2.2 se explicaron ambos modos de funcionamiento de forma general. En este apartado se analizarán en el contexto del proyecto.

Se ha tomado la decisión de que la interacción entre los clientes y los pares se realice en modo *redirect*. Esta decisión se toma en base a la carga computacional que exige el modo *proxy*. En el caso del modo *redirect*, un cliente que quiere establecer una llamada sigue el siguiente proceso: solicita a la red la dirección IP de la *URI* con la que se desea establecer la llamada; el nodo al que corresponda lo buscará en sus tablas y responderá con la dirección. Tras este intercambio, los nodos pertenecientes a la red P2P no tendrán que realizar más operaciones con motivo de esta llamada, dado que el cliente enviará directamente la solicitud de invitación al cliente en cuestión. Sin embargo, en el caso del modo *proxy*, cuando un cliente solicita una llamada a la red P2P se sigue el siguiente proceso: el nodo de la red al que llega esta solicitud tendría que, en primer lugar, buscar al nodo en cuestión a través de la red; en segundo lugar, tendría que transferir la llamada hasta el cliente de destino. Por lo tanto, en este modo se requeriría que ese nodo, mediante el cual se establece la llamada, almacene los estados de cada uno de los participantes en la misma. Esto se debe a que necesita saber qué mensajes irán dirigidos a quién, puesto que podría haber diversas llamadas establecidas a través de él. Además,

los recursos necesarios requeridos por los clientes que intervienen en la llamada al nodo de la red P2P no podrán ser liberados hasta que la misma finalice.

Analizando los dos formatos de operación, y puesto que es conveniente liberar en la medida de lo posible de carga de procesamiento a los terminales en los que se ejecuta la aplicación, se ha optado por utilizar el modo *redirect* en la solución propuesta.

5.2 Red P2P

En la sección 2.3 se comentaron dos posibles tipos de redes P2P aplicables al contexto de la telefonía sobre IP mediante SIP. Comparando ambas, se puede ver que la red desestructurada es más rápida en la realización de búsquedas que la red Chord, pero esta diferencia se hace relevante sólo ante un elevado número de nodos en la red (véase la Figura 5.1 [16])

Sin embargo, al mismo tiempo que la red UP2P es más rápida para un elevado número de nodos, también requiere más carga computacional. Por este motivo, y considerando que el escenario de aplicación tendrá un número reducido de nodos, se ha optado por implementar una red Chord como red P2P para el desarrollo del proyecto.

En la descripción de la red Chord que se puede encontrar en [17] (véase la sección 2.3.2) se especifica que, por un lado, se hace la función hash de la dirección IP de los nodos pertenecientes a la red y, por otro lado, la función hash de los datos a almacenar. Así pues, el nodo encargado de almacenar los datos es aquel cuya identificación corresponda al primer nodo que sucede al resultado de la función hash de los datos, en sentido horario.

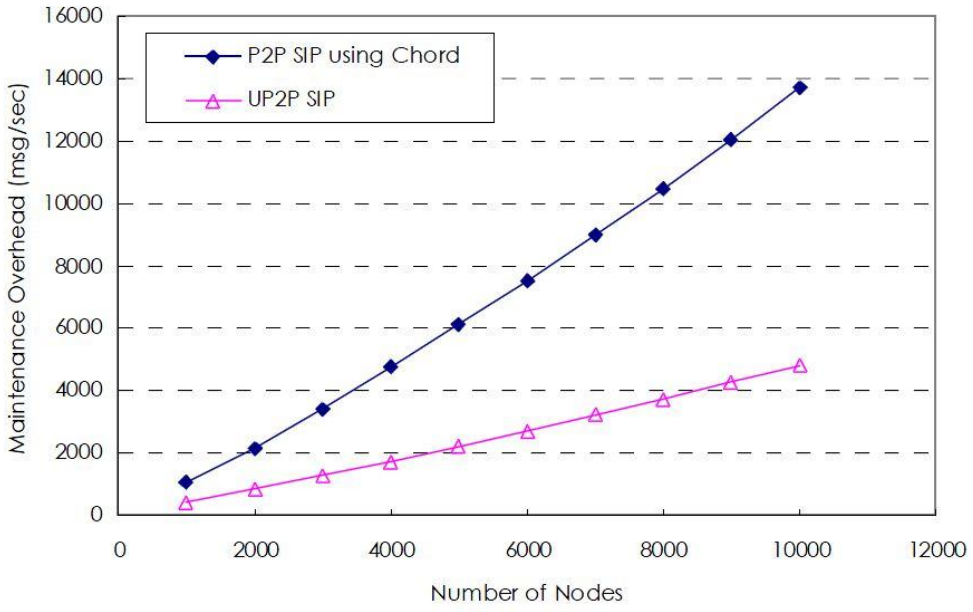


Figura 5.1: Comparativa respecto a la sobrecarga por mensajes de mantenimiento entre redes Chord y redes UP2P frente al número de nodos

En el contexto del proyecto, los datos a almacenar son las *URIs* de los clientes acompañada de sus direcciones IP. Estos datos quedarían indexados por el resultado de realizar la función hash a la *URI*. Así pues, para la realización del proyecto, se modifica la directriz que especifica que los nodos se identifican mediante la función hash de su dirección IP. En su lugar, los nodos quedarán identificados por el resultado de la función hash de su *URI*. Se podría pensar que un nodo no tiene por qué tener una URI, pero no es así. Esto se debe al contexto en el que se desarrolla el proyecto donde, como mínimo, un usuario ejecutará un cliente (pudiendo ejecutar también un *peer*). Por consiguiente, cada terminal siempre dispondrá de un *URI*.

De esta forma, cliente y *peer* se pueden vincular fácilmente si ambos se ejecutan en un terminal. Esto se debe a que tendrán la misma identificación, lo que les permitirá interactuar sin la necesidad de que un cliente ejecutado en un terminal tenga asociado un *peer* que se está ejecutando en otro. Así se asegura que el cliente siempre pertenece al *peer* que se ejecuta en el mismo dispositivo.

Además, si se está ejecutando un *peer* en el terminal, los usuarios que se encuentren en la misma red pueden encontrarse más rápido lo que supone una ventaja adicional. Esto se debe a que las búsquedas de clientes no implican una búsqueda en las tablas de clientes, puesto que un nodo conocido también implica un cliente conocido.

A continuación se expone el diseño de la red P2P en cuanto a su comportamiento, mientras que en la sección 5.3.2 se detallan las clases involucradas para su implementación en detalle.

5.2.1 Descubrimiento de redes

Al comenzar la ejecución del sistema, si el usuario decide ser *peer* su nodo debe buscar si existe alguna red ya creada (véase la Figura 5.2). Esta búsqueda se realiza mediante el envío de un mensaje a la dirección multicast 224.0.1.75. Este mensaje consiste en un mensaje SIP de tipo *REGISTER*, en el que la cabecera *To* contiene su propio identificador y la cabecera *From* contiene también el propio identificador.

Al enviar este mensaje se lanzan dos temporizadores[5]. Tras enviarlo se pasa a esperar una respuesta, pudiendo darse tres situaciones diferentes que se detallan a continuación:

- Que expire el temporizador E, por lo que se vuelve al estado anterior y se volverá a enviar el mensaje multicast. Este temporizador se volverá a lanzar, pero el F seguirá corriendo, por lo que habrá menos diferencia entre ambos, es decir, cuando se relance este temporizador la duración del temporizador F será menor.
- Que tras expirar varias veces el temporizador E, y por lo tanto enviarse varias veces mensaje multicast, expire el temporizador F. Esto hará al nodo entender que no existe ninguna red creada, por lo que pasará a crear una él mismo para que otros puedan unirse.

[5] Denominados E y F, siendo la duración de F varias veces la de E. Se explican con más detalle en la sección 5.3.2 "Diagrama de estados de la red P2P" y en [8].

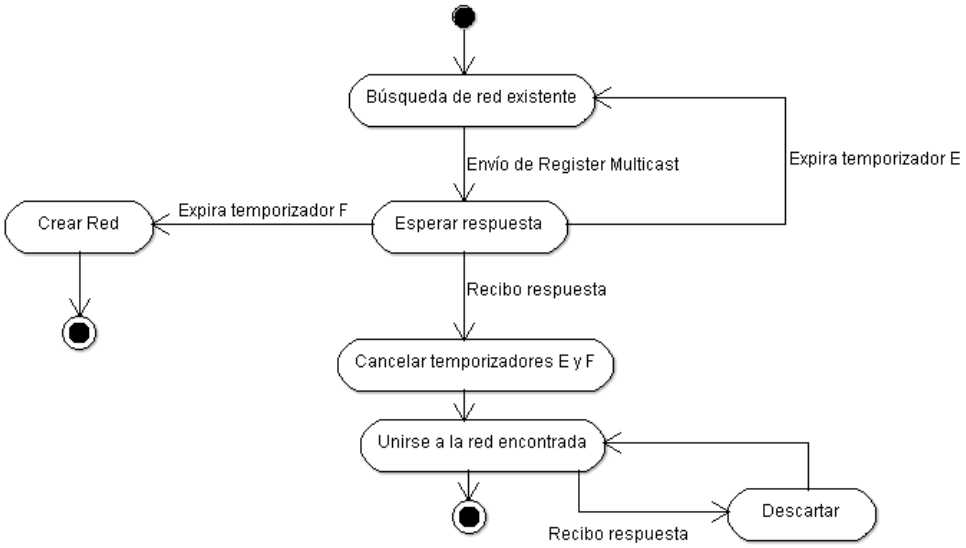


Figura 5.2: Descubrimiento de red existente

- Que se reciba una respuesta al mensaje enviado. Ésta consiste en un *302 Redirecting*, en la cual se indica la dirección donde se puede encontrar al nodo que contestó. En ese momento se cancelan los dos temporizadores y el nodo pasa a unirse a la red encontrada, descartando cualquier otra respuesta que se recibiera al mensaje multicast enviado.

5.2.2 Unión a una red

Una vez descubierta una red (recepción del mensaje *302 Redirecting*) como se comentó en la sección anterior, tiene lugar el proceso de unión a la misma (véase la Figura 5.3).

Este proceso comienza enviando de nuevo el mismo mensaje que se envió para el descubrimiento, pero con una diferencia. Anteriormente se envió de forma multicast y en esta ocasión se envía de forma unicast a la dirección que se recibió en el mensaje *302 Redirecting*. A continuación, se espera hasta recibir una respuesta al mensaje enviado. En esta respuesta se encuentra la información del nodo que le precede y del nodo que le sucede en la red.

Conocido el sucesor, el nodo actualiza su tabla de *fingers*. Si $N_{succ} \geq N_{Id} + 2^{m-1}$, toda la tabla de *fingers* quedará actualizada con $finger[i].node=N_{scucc}$ para todo i desde uno hasta m. De no ser así, se actualizan sólo aquellos para los que $finger[i].start \leq N_{succ}$.

Si se ha actualizado toda la tabla, la unión ya está completada. En el proceso de estabilización, el resto de nodos que tengan que modificar sus tablas con motivo de la incorporación de éste se darán cuenta.

Si no se ha actualizado toda la tabla, se realiza una búsqueda en la red preguntando por el sucesor del identificador *finger[i].start* que no cumplió con la condición para ser actualizado. Esta pregunta consiste en un mensaje *REGISTER* que se envía al nodo predecesor más cercano que se conoce a aquel que se busca. El mensaje contiene la cabecera *To* indicando $N_{buscado}$*@desconocido*. Tras recibir la respuesta a este mensaje se vuelve a actuar de forma similar a la anterior, con la diferencia de que se continúa actualizando la tabla desde la entrada por la que se preguntó. Además, en lugar de realizar el proceso con N_{succ}, se realiza con $N_{recibido}$. Este proceso se repite hasta que la tabla de *fingers* se haya actualizado en su totalidad.

5.2.3 Mantenimiento de la red

Una vez que el nodo se ha unido a la red, el nodo realizará un procedimiento que tiene como misión el mantenimiento de la red (véase la Figura 5.4), al igual que el resto de nodos de la red.

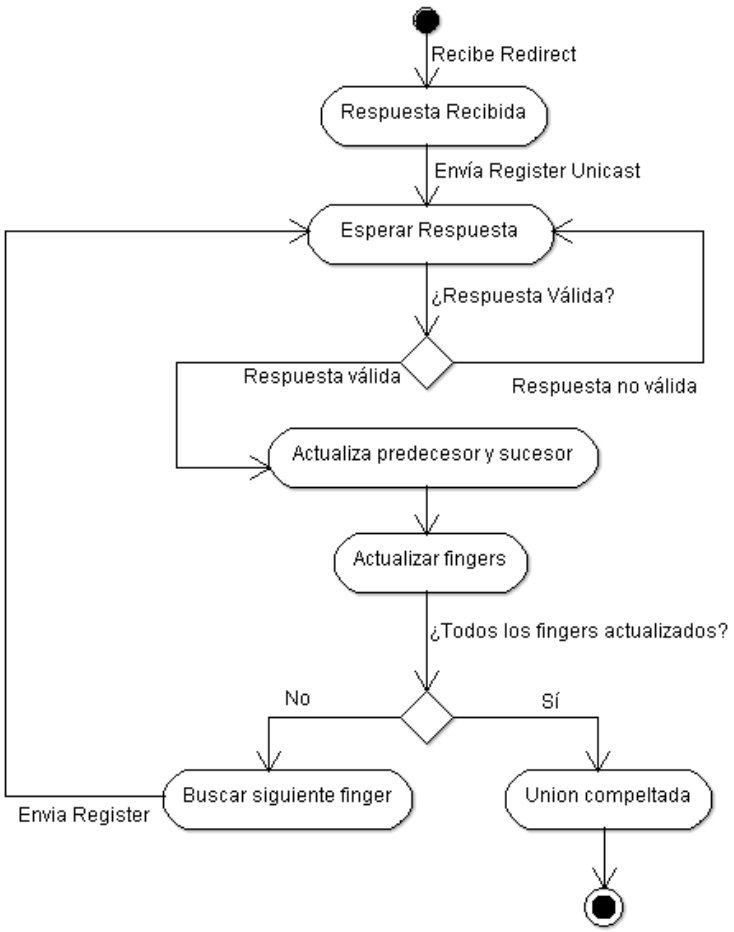


Figura 5.3: Unión a una red

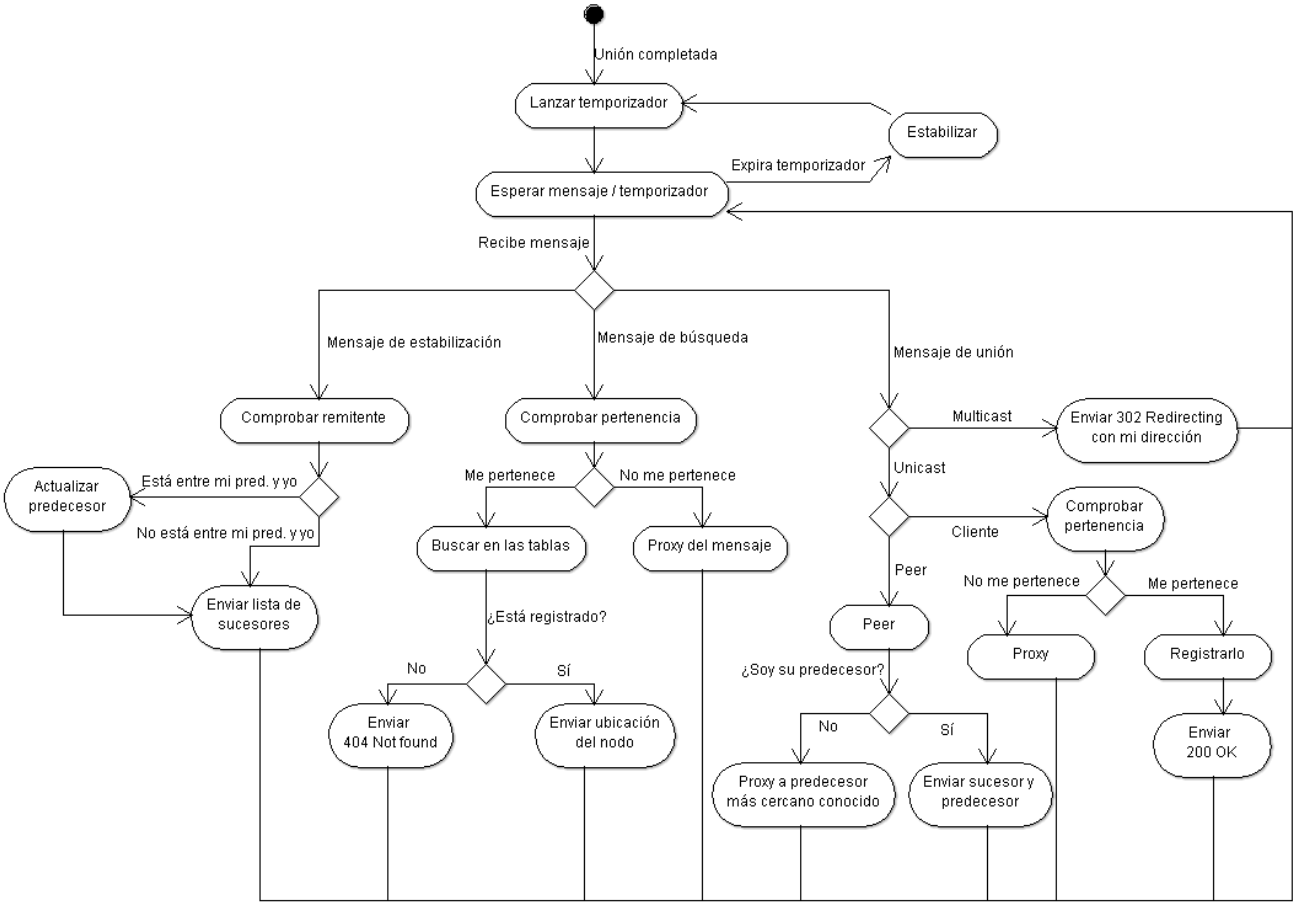


Figura 5.4: Mantenimiento de la red

Para ello, tras completar la unión, el nodo lanza un temporizador y pasa a un estado de espera que se verá alterado cuando el temporizador expire o cuando se reciba algún mensaje.

La misión del temporizador consiste en que la red se estabilice de forma periódica. Cuando éste expira, el nodo comienza el proceso de estabilización y, tras terminarlo, lanza de nuevo el temporizador volviendo al estado de espera.

Los mensajes que el nodo puede recibir son de tres tipos y se describen a continuación.

Mensaje de estabilización

Se considera como mensaje de estabilización aquel mensaje *REGISTER* que contiene una cabecera *Contact*. El nodo que recibe el mensaje comprueba si el remitente del mismo se encuentra ubicado entre su predecesor y él. De ser así, significa que un nodo se ha incorporado a la red en esa posición y debe modificar su predecesor con los datos del remitente del mensaje. Tras esta modificación, el nodo envía su lista de sucesores al nuevo predecesor. Si el nodo que envió el mensaje no es su predecesor directo, sólo le envía la lista de sucesores.

Mediante este mecanismo un nodo que se incorpora a la red comunica al resto de los presentes su adhesión a la misma. En la sección 5.2.2 se especificó cómo se lleva a cabo la unión de un nodo a la red, pero durante ese proceso los nodos que le facilitan información para su unión no modifican sus datos respecto a la red. Es cuando reciben este mensaje, enviado en el proceso de estabilización (sección 5.2.4), cuando los componentes de la red modifican sus datos.

Mensaje de búsqueda

Un mensaje de búsqueda consiste en un mensaje *REGISTER* en el que la cabecera *To* contiene *N@desconocido*, indicando así que se trata de una búsqueda del nodo *N*.

Si el nodo receptor es el encargado de almacenar esa información, realizará una búsqueda en sus tablas. Si el cliente buscado está en el sistema, enviará un mensaje *200 OK* al remitente del mensaje en el que la cabecera *Contact* contendrá la dirección donde puede localizarlo. En caso de que el cliente que se busca no se encuentre en la red (por lo que su información no habría sido registrada), se le envía al nodo que generó el mensaje una respuesta *404 NOT FOUND* para notificárselo.

Si el nodo receptor no es el encargado de almacenar esa información, debe actuar como *proxy* y enviar el mensaje al nodo conocido más cercano al nodo que se busca. Actuando de esta forma, es posible que un mensaje tenga que ser reenviado varias veces hasta localizar el nodo al que corresponde almacenar esta información.

Mensaje de unión

Un mensaje de unión puede ser un mensaje multicast para el descubrimiento de la red o un mensaje unicast. Este mensaje unicast puede ser de un *peer* que quiere formar parte de la red o bien de un cliente que quiere ser registrado en la misma.

Si se trata de un mensaje multicast, el nodo enviará una respuesta *302 Redirecting* con su dirección en la cabecera *Contact* para poder ser localizado.

Si es un *peer* que quiere formar parte de la red, comprobará si el remitente del mensaje se encuentra entre su identificación y la de su sucesor. De ser así, quiere decir que él es el predecesor del nodo en cuestión. Eso significa que el sucesor del nodo que envió el mensaje será el que era su sucesor y su predecesor será el propio nodo. En este caso se enviará un mensaje *200 OK* informando en la cabecera *Contact* de quiénes son su sucesor y su predecesor.

Si es un cliente que quiere unirse a la red, el nodo comprobará si él es el encargado de registrarlo. Si es el encargado, almacena su información y envía un mensaje *200 OK* al cliente confirmándole que se ha registrado correctamente. Si no le corresponde almacenarlo, actúa como *proxy* reenviando la solicitud al nodo conocido más cercano al nodo que lo solicitó.

5.2.4 Estabilización

El proceso de estabilización consiste en el envío de un mensaje a los nodos sucesor y predecesor en el que la cabecera *Contact* contendrá la identificación del nodo que realiza

la estabilización y la de su predecesor. Cuando el nodo al que se envía el mensaje lo recibe, éste actualizará su predecesor en caso de que sea necesario como se comentó en la sección 5.2.3

Tras enviar el mensaje, el nodo recibe una respuesta con la lista de sucesores de aquel a quien se le envió y actualiza su lista de sucesores con la lista recibida en caso de ser necesario. Además, actualiza su tabla de *fingers* a partir de la entrada que corresponda, cuyo índice se extrae de la expresión 5.1 (véase la Figura 5.5).

$$i = \lceil \log_2(N_{succ} - N_{Id}) \rceil \qquad (5.1)$$

De este modo, cuando un nodo se incorpora a la red tras realizar su primera estabilización, provocará que su sucesor lo incorpore como su nuevo predecesor. Cuando el nodo que ha modificado su predecesor ejecute la estabilización, provocará que otro nodo modifique sus tablas si fuera necesario, derivando en una actualización en cadena. Tras la ejecución del proceso de estabilización en los pares involucrados varias veces, se llegará a un estado estacionario en el que las tablas de todos los pares quedarán completas de forma adecuada.

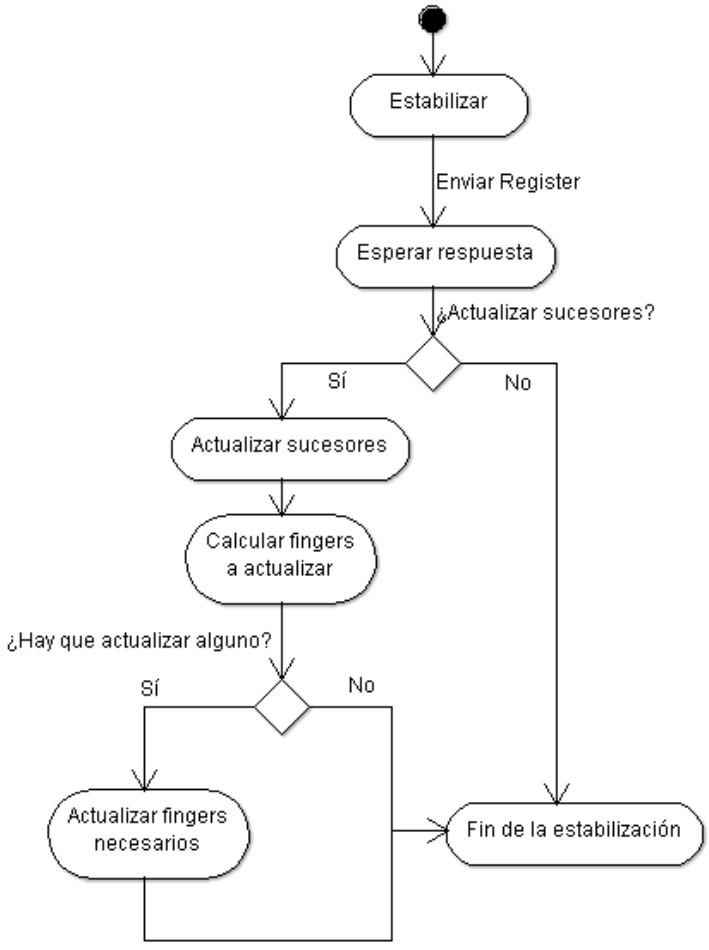


Figura 5.5: Estabilización

Tras la estabilización, si ha habido alguna modificación en el predecesor, el nodo debe comprobar si había clientes registrados que ahora pertenezcan al nuevo predecesor. De ser así, se registrarán los clientes que lo requieran en el nuevo predecesor.

5.2.5 Abandono de la red

Cuando un nodo quiere abandonar la red debe comunicarlo a su predecesor y a su sucesor para que éstos puedan actualizar sus tablas (véase la Figura 5.6). Tras comunicar su abandono, si el nodo había registrado algún cliente (identificadores entre su predecesor y él) deberá registrar a éstos en su sucesor, que pasará a ser el encargado de los mismos tras su abandono de la red.

5.3 Estructura de clases

En este apartado se detalla el diseño realizado para la aplicación desde el punto de vista de las clases necesarias. En éste se comenta cada clase, así como sus principales métodos y atributos.

La aplicación tiene como clase principal la clase `Forma1`, que es la encargada de la interfaz gráfica y de crear los demás objetos involucrados en la ejecución según el usuario requiera. Los objetos que puede generar esta clase son dos, correspondientes a los dos principales bloques de la aplicación que ya se han comentado anteriormente, `cliente` y `peer`.

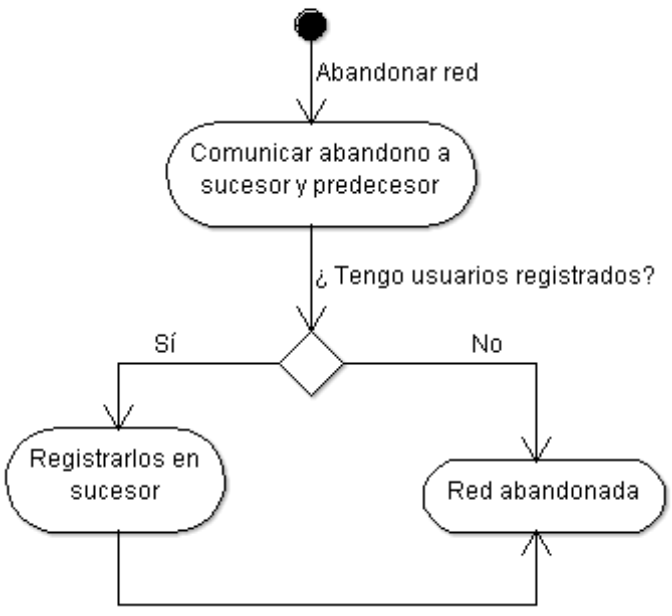


Figura 5.6: Abandono de la red

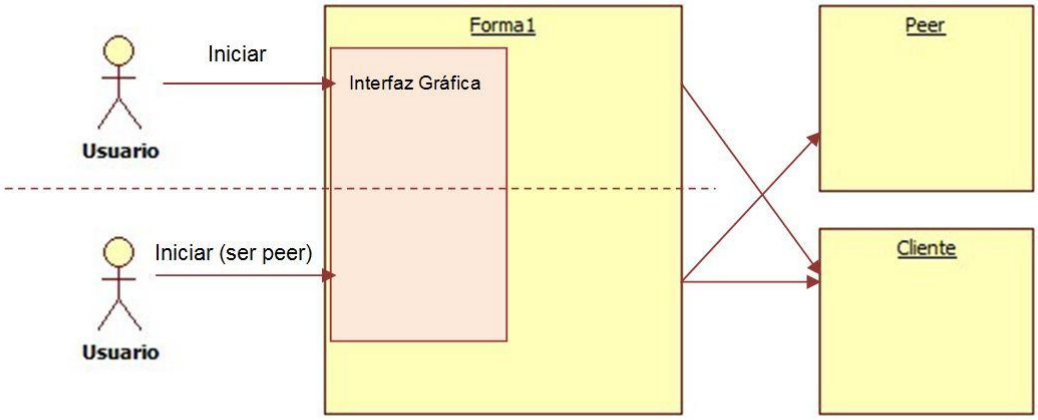


Figura 5.7: Clases principales

Puesto que la clase `Forma1` es la encargada de la interfaz gráfica, el usuario interactúa directamente con ésta. El usuario, en una elección al principio de la ejecución, indicará a la clase `Forma1` a través de la interfaz si se creará un objeto de las clases `Cliente` y `Peer` o solamente un objeto `Cliente`, como se puede ver en la Figura 5.7.

5.3.1 Bloque cliente

Este bloque contiene a su vez dos partes bien diferenciadas. Por un lado, la parte encargada del control de las llamadas, y por otro lado, las sesiones multimedia que se establecen en una llamada. El bloque cliente se explicará diferenciando estas dos partes con motivo de que sea más simple y de mejorar su comprensión.

Clases para control

Las clases involucradas en el control son: `Cliente`, `SipCliente`, `SDP` y `Dialogo` (véase el diagrama de la Figura 5.8). Éstas se encargan de las funciones de control de las llamadas, incluyendo funciones tales como: enviar peticiones de llamadas, finalizar llamadas, o rechazar llamadas. Además de estas funciones, también se encarga del registro y las búsquedas en la red P2P en caso de que no se ejecute un *peer* en el terminal. A continuación se detallan los principales atributos y métodos de las clases implicadas.

`Cliente`

Es la clase encargada de realizar las funciones del cliente de telefonía IP. Gestiona los `sockets` para enviar y recibir los paquetes (Figura 5.8). Esta case no tiene conocimiento de los protocolos implementados, es decir, actúa como actuaría un usuario con un teléfono, encargando las tareas de construcción de mensajes y de comprensión de éstos a clases auxiliares. Este procedimiento se explica con más detalle en la sección 5.5.5.

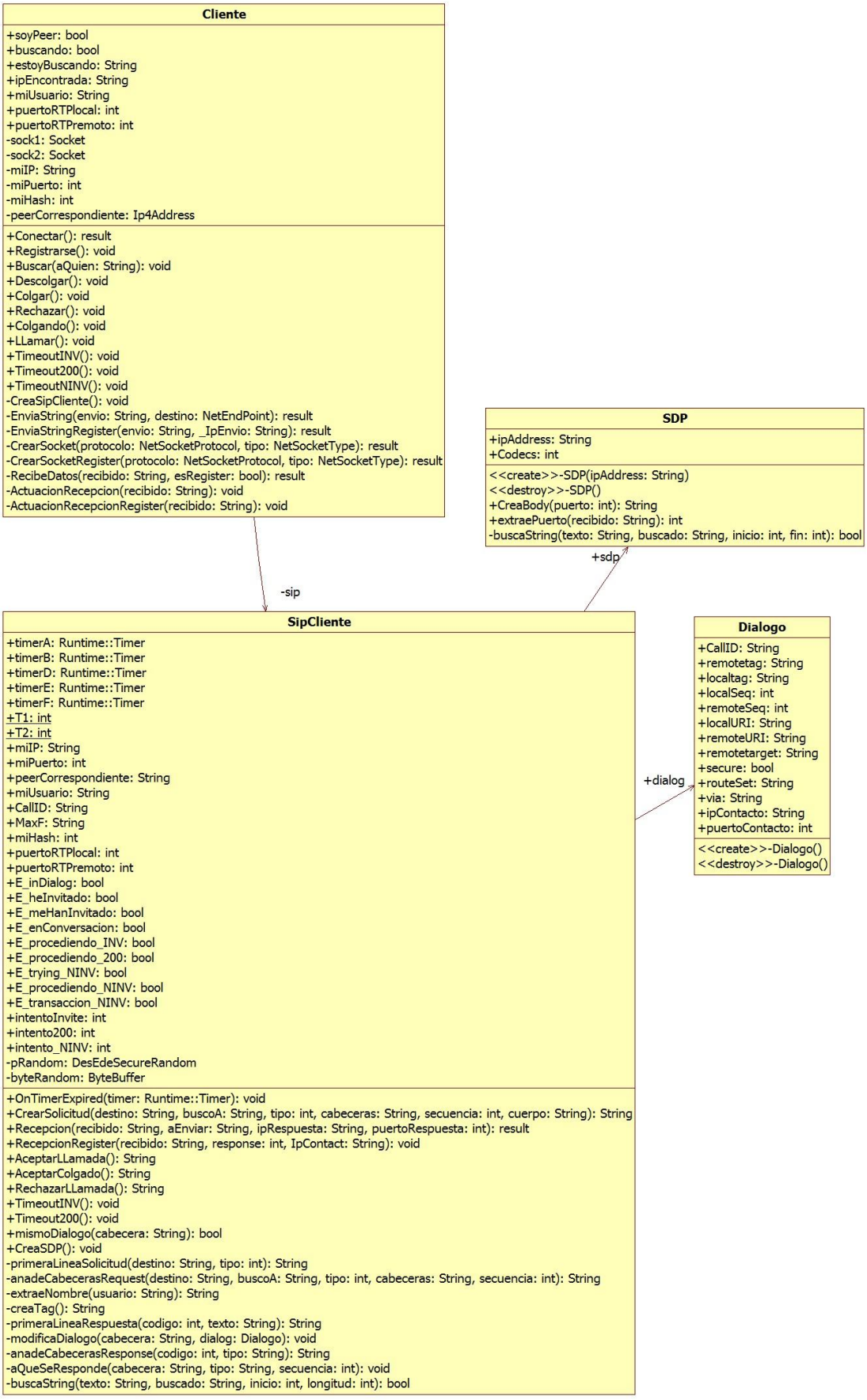


Figura 5.8: Cliente (Control)

Atributos

`soyPeer` Información sobre si se está o no ejecutando un *peer* además de un cliente.

`puertoRTPlocal` Dato introducido por el usuario en la interfaz, necesario para establecer llamadas.

`puertoRTPremoto` Dato recibido de otro cliente para establecer una llamada. Será el puerto RTP que éste utilizará.

`sock1` `Socket` encargado de la gestión de las llamadas. Por defecto vinculado al puerto 5060.

`sock2` `Socket` encargado de la gestión de la red. Sólo se utilizará en caso de que no se ejecute un *peer* en el terminal. Por defecto vinculado al puerto 5061.

`peerCorrespondiente` Dirección IP del *peer* que nos ha registrado, de nuevo, sólo es necesario su uso en caso de que no se ejecute *peer*.

Métodos

`Registrarse()` Se encarga de iniciar el mecanismo de búsqueda de una red y de registro en la misma (sólo si no se ejecuta *peer*).

`Buscar()` Método encargado de realizar la búsqueda de un cliente dado en la red (sólo si no se ejecuta *peer*).

`Descolgar()` La llamada a este método se realiza desde la interfaz gráfica, a través de la clase Forma1, para aceptar una llamada que se haya recibido.

`Colgar()` Igual que el anterior, la llamada a este método se produce desde la interfaz gráfica para finalizar una conversación que esté teniendo lugar.

`Rechazar()` En caso de no querer aceptar una llamada, se encarga de enviar el mensaje oportuno para rechazarla.

`Colgando()` Puesto que la finalización de una llamada consiste en el envío de un mensaje y la confirmación de éste, si el cliente con el que se está hablando cuelga, este método se encarga de confirmar el colgado.

`Llamar()` Se encarga de invitar a otro cliente, a través de su dirección IP.

`TimeoutINV()` Se ejecuta cuando se pierde una solicitud INVITE.

`Timeout200()` Se ejecuta cuando se pierde un mensaje 200 OK al aceptar una llamada.

`TimeoutNINV()` Se ejecuta cuando se pierde alguna solicitud que no sea INVITE.

`EnviaString()` Método encargado de enviar un `String` a través de `socket1`.

`EnviaStringRegister()` Método encargado de enviar un `String` en relación a la red P2P, a través del `socket2`. Es el encargado de los mensajes REGISTER.

`AcutacionRecepción()` Define cómo actúa el cliente tras la recepción de un mensaje, ya sea una solicitud o una respuesta.

`AcutacionRecepciónRegister()` Define cómo actúa el cliente tras la recepción de una respuesta a un mensaje REGISTER.

`SipCliente`

Ésta es la clase que entiende la sintaxis SIP, por lo tanto, es la encargada de crear los mensajes SIP para enviarlos y de analizar los recibidos. También es la encargada de manejar los estados del cliente SIP. Para realizar estas funciones se apoya en otras dos clases que se especifican más adelante: `SDP`, y `Dialogo` (Figura 5.8).

Atributos[6]

`miUsuario` Consiste en el nombre de usuario del cliente, que debe ser conocido por esta clase para la creación de los mensajes SIP.

`miPuerto` Al igual que el atributo anterior, debe ser conocido para la creación de los mensajes.

`MaxF` Variable saltos máximos (*Max Forwards*), la cual debe ser incluida en los mensajes SIP con motivo de evitar bucles.

`byteRandom` Este buffer de bytes tiene como misión almacenar cadenas de bytes, generadas aleatoriamente para la creación de campos que lo requieren en los mensajes SIP.

Métodos

`CrearSolicitud()` Método que devuelve un `String` con una solicitud con los parámetros requeridos.

`Recepción()` Es el encargado de analizar un mensaje SIP recibido. Además, determina como hay que actuar en función del estado en el que se encuentre el sistema.

`RecepciónRegister()` Es el encargado de analizar las respuestas REGISTER recibidas. Éstas provienen de la red P2P.

`AceptarLLamada()` Método encargado de generar la respuesta afirmativa a una invitación recibida.

`AceptarColgado()` Es el encargado de generar la respuesta a una solicitud de colgado.

[6] Los atributos `Timer A,B,D,E,F,T1,T2`, son variables involucradas en la recuperación de tramas perdidas. Se explican, al igual que los atributos cuyo nombre comienza con "E_", en el apartado **Diagrama de estados del cliente** dedicado a la máquina de estados del cliente.

`RechazarLLamada()` Este método se encarga de generar una respuesta negativa a un cliente que solicitó una llamada. En esta respuesta se comunica que la llamada ha sido rechazada por el cliente.

`OnTimerExpired()` Función que se ejecuta cuando expira algún temporizador. Como argumento recibe la identificación del temporizador que expiró.

`SDP`

Esta clase se encarga de generar el cuerpo de los mensajes SIP, mensajes SDP. Además, es la encargada de extraer la información necesaria de los mensajes recibidos para establecer la sesión (Figura 5.8).

Atributos

`ipAddress` Contiene la dirección IP del cliente para incluirla en los mensajes SDP.

`Codecs` Codecs de audio que se incorporan en los mensajes SDP.

Métodos

`CreaBody()` Método encargado de generar un `String` con un cuerpo con el formato requerido.

`extrePuerto()` Se encarga de extraer la información de un mensaje SDP recibido.

`Dialogo`

Cuando el cliente establece un *dialogo* con otro cliente, esta clase es la encargada de almacenar toda la información referente a éste (Figura 5.8).

Atributos

`CallID` `String` que contiene el *Call-Id* del diálogo establecido.

`remotetag` Variable con el que el otro cliente se está identificando en este diálogo.

`localtag` Variable que el cliente está utilizando para identificarse en este diálogo.

`localSeq` Número de secuencia actual en el diálogo con el cliente.

`remoteSeq` Número de secuencia actual del otro cliente.

`localURI` URI del cliente.

`remoteURI` URI del cliente con el que se establece el diálogo.

`via` No es una variable perteneciente al diálogo en sí, pero es relevante para la construcción de las respuestas.

Interacción entre clases para control

En la Figura 5.9 se puede observar un ejemplo de cómo se relacionan estas clases entre ellas. En el ejemplo el usuario desea realizar una llamada, por lo que a través de la

interfaz gráfica y de la clase `Forma1` el usuario provoca que el cliente ejecute el método `LLamar()`, tal como se comentó en el comienzo de la sección. Cuando el cliente quiere realizar una llamada, solicita al objeto de la clase `SipCliente` que cree una solicitud con ese fin, llamando a la función `CrearSolicitud()` de esta clase. Para crear la solicitud, el objeto de la clase `SipCliente` llama a otras funciones de su propia clase que modificarán los datos del diálogo necesarios en un objeto de la clase `Dialogo` y le ayudarán a componer la solicitud. Además, para crear la solicitud el sip-cliente debe pedir a la clase `SDP` que cree un cuerpo para el mensaje. Tras generar el mensaje, la función `CrearSolicitud()` devuelve al cliente el mensaje y se procede a su envío.

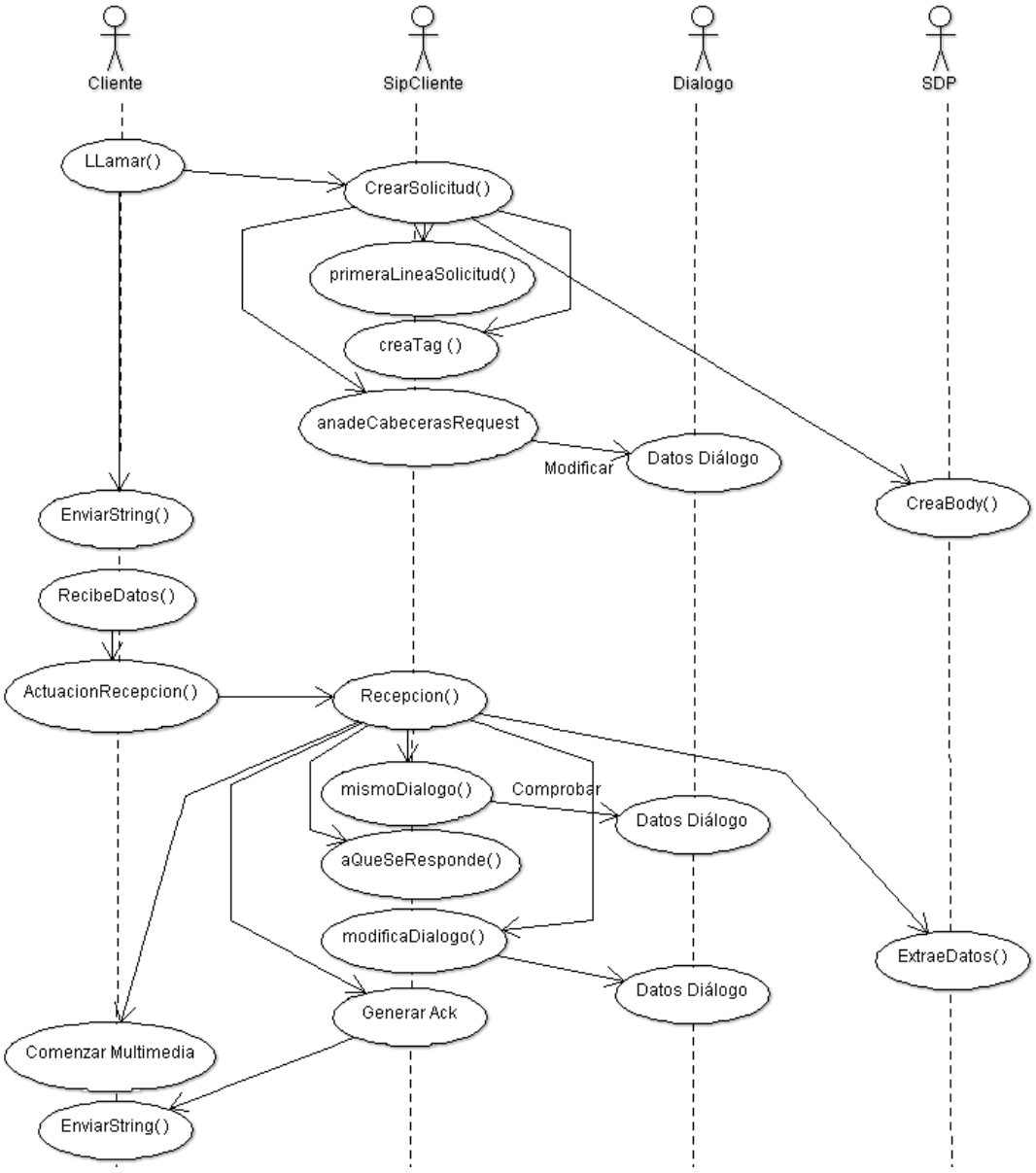


Figura 5.9: Interacción entre clases, Cliente (Control)

En el ejemplo, el cliente recibe una respuesta satisfactoria a la solicitud de llamada (200 OK). Se escenifica esta situación con el propósito de aclarar la interacción entre clases ante el establecimiento de una llamada que transcurre de forma habitual.

Cuando el cliente recibe la respuesta, llama a la función `ActuacionRecepción()` que entregará el mensaje al método `Recepcion()` de la clase `SipCliente`, dado que es esta clase la que conoce la sintaxis SIP. Además de diversas comprobaciones y modificaciones en el objeto de la clase `Dialogo`, la clase `SipCliente` debe pedir a la clase `SDP` que analice el cuerpo del mensaje. Tras analizar el mensaje, el método `Recepcion()` hará que el cliente comience la sesión multimedia, además de construir un mensaje ACK que hará que éste envíe.

Clases para sesión multimedia

Para la sesión multimedia se utilizan las clases `LLamada`, `LLamadaOut`, `ALaw`, y `RTP`. Este bloque es el encargado de enviar y recibir las tramas que contienen la información de voz. También se encarga de reproducir la información recibida por el altavoz y capturar las muestras del micrófono del dispositivo (véase la Figura 5.10). A continuación, se detallan los principales atributos y métodos de las clases implicadas.

`LLamada`

Esta clase es la encargada de controlar las sesiones multimedia, enviando y recibiendo los datos a través de un `socket`. También se encarga de manejar la parte hardware del micrófono para la obtención de muestras de voz (Figura 5.10).

Atributos

`bufferIn` Estos tres buffers de bytes son los encargados de almacenar las muestras capturadas desde el micrófono, a la espera de ser tratadas.

`bufferCodificado` Buffer en el que se almacenan las muestras de voz una vez codificadas por el codificador.

`pAudioIn` Instancia referente al micrófono.

`sock` `Socket` creado para el envío y la recepción de los mensajes.

`puerto` Puerto local, en el cual se reciben datos.

`puertoDestino` Puerto remoto, al que se envían datos.

`tamanoPaquetes` Definición del tamaño de los paquetes, por defecto 240 bytes.

`recibidos` Número de paquetes recibidos hasta el momento.

`enviado` Número de paquete enviado.

`perdidos` Variable en la que se almacena el número de paquetes perdidos.

`primeraVez` Esta variable controla que no se envíen tramas antes de haber llenado un paquete completo.

Métodos

`OnAudioBufferIsFilled()` Método que se ejecuta cuando un buffer se llena de muestras recogidas desde el micrófono. Una vez que se llena un buffer, las muestras se codifican según corresponda y se les añade la cabecera RTP. Después de construir el paquete se envía.

`Empezar()` Este método se encarga de configurar el micrófono (número de bits por muestra, frecuencia de muestreo y formato de las muestras) y crear el objeto que controlará el altavoz.

`EnviarBuffer()` Función que envía a través del `socket` un buffer dado.

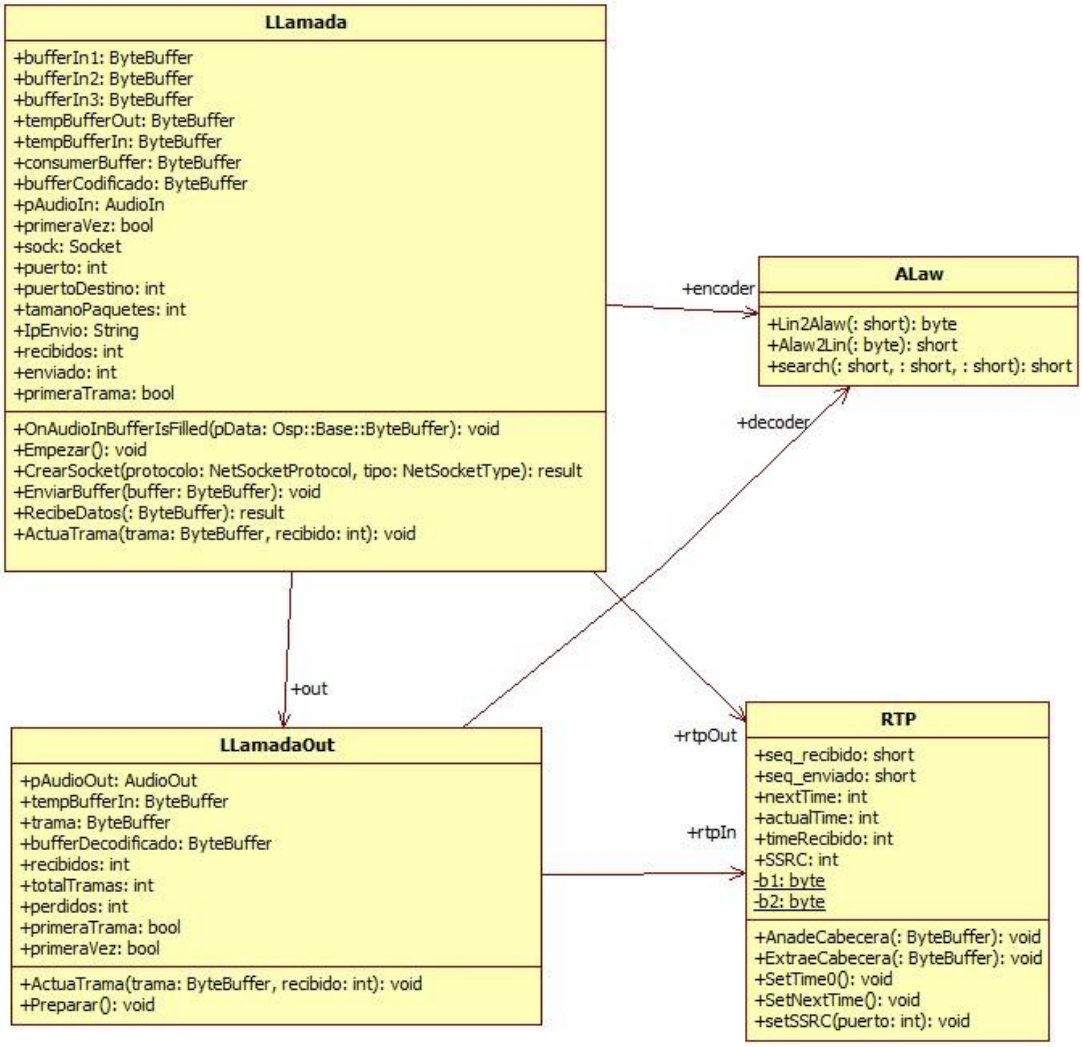


Figura 5.10: Cliente (Sesión Multimedia)

`ActuaTrama()` Al recibir una trama, ésta debe ser entregada al objeto encargado de procesarla.

`LLamadaOut`

Se encarga de la gestión de las tramas recibidas. La gestión de las llamadas se divide con motivo de administrar el micrófono y el altavoz desde clases distintas. Esto se debe a la forma en la que se accede a estos en el sistema operativo Bada, que hace más eficiente su gestión en clases diferentes (Figura 5.10).

Atributos

`pAudioOut` Instancia referente al altavoz.

`bufferDecodificado` Buffer que se obtiene de pasar los datos recibidos por el decodificador.

`recibidos` Número de paquetes recibidos.

`perdidos` Variable en la que se almacena el número de paquetes perdidos en el receptor hasta el momento.

`primeraVez` Variable encargada de que no se empiece a reproducir hasta que se haya recibido información del otro cliente.

`totalTramas` Contiene el número total de tramas que deberían haberse recibido, en función de los números de secuencia recibidos.

Métodos

`ActuaTrama()` Actuación tras la recepción de una trama. Ésta debe ser desencapsulada, decodificada y reproducida.

`Preparar()` Función que prepara el altavoz para ser usado en una reproducción (número de bits por muestra, frecuencia de muestreo y formato de las muestras).

`RTP`

Clase encargada de encapsular los datos a enviar en un paquete RTP y de desencapsular paquetes RTP recibidos. También es la encargada de gestionar los números de secuencia y las marcas te tiempo (*timestamps*) de la cabecera del protocolo RTP (Figura 5.10).

Atributos

`SeqRecibido` Último número de secuencia recibido.

`SeqEnviado` Último número de secuencia enviado.

`nextTime` Marca de tiempo que se incluirá en el próximo paquete que se genere.

`actualTime` Marca de tiempo incorporada en el último paquete generado.

`timeRecibido` Marca de tiempo recibida en el último paquete.

`SSRC` SSRC de la cabecera del protocolo RTP.

Métodos

`AnadeCabecera()` Esta función es la encargada de encapsular un buffer de bytes en un mensaje RTP.

`ExtraeCabecera()` Este método desencapsula un paquete RTP y extrae los datos de la cabecera.

`SetTime0()` Recoge el tiempo del sistema para generar la referencia de tiempo desde la que se generan los *timestamps.*

`SetNextTime()` A esta función se la llama cuando se captura la primera muestra de voz del próximo paquete que se envíe, y genera la próxima marca de tiempo a partir de una referencia.

`SetSSRC()` Genera el SSRC que se utilizará para la conversación.

`ALaw`

Consiste en el codificador y decodificador de muestras PCM usando la ley-A (Figura 5.10).

Métodos

`Lin2Alaw()` Función que tiene como entrada una muestra lineal, y devuelve la muestra codificada con ley-A.

`Alaw2Lin()` Función que tiene como entrada una muestra codificada con ley-A, y devuelve una muestra lineal.

Interacción entre clases para sesión multimedia

En la Figura 5.11 se muestra un ejemplo de un cliente al iniciar un intercambio multimedia. En éste se muestran los casos de completado de un buffer y de recepción de un paquete.

Para comenzar la sesión multimedia, el objeto de la clase `LLamada` ejecuta el método `Empezar()` que hará a su vez que se ejecute la función `Preparar()` de la clase `LLamadaOut`. Como ya se comentó, estos métodos tienen como misión configurar el hardware necesario. También se guarda una referencia de tiempo en la clase `RTP` para generar más tarde las marcas de tiempo.

Cuando un buffer se llena con muestras procedentes del micrófono, se empieza a grabar en otro buffer dado que el que se llenó debe ser procesado y liberado para su reutilización. Se genera también el *timestamp* del próximo paquete RTP, puesto que debe generarse cuando se recoge la primera muestra del paquete. El procesamiento del buffer consiste en su codificación, ejecutando el método `Lin2Alaw()` de la clase

`ALaw`, su empaquetamiento mediante el método `AnadeCabeceras()` de la clase `RTP` y por último su envío.

Cuando se recibe un paquete RTP, se entrega a la clase `LLamadaOut`, que es la encargada del procesamiento y de la reproducción del audio. El procesamiento consiste en el análisis de la cabecera RTP y la extracción de los datos por parte del método `ExtraeCabecera()` de la clase `RTP`. Después, los datos extraídos son decodificados en el método `Alaw2Lin()` de la clase `ALaw()`, y se reproducen.

Diagrama de estados del cliente

En la Figura 5.12 se muestra el diagrama de estados que sigue el cliente. Los estados en los que se puede encontrar el cliente son los siguientes:

- **Estado de espera**: No hay ninguna variable que corresponda con este estado en el diseño. Se encuentra en este estado cuándo ninguno de los demás posibles está activo. Éste se corresponde con el cliente sin realizar ningún tipo de acción. Se cambia de estado ante la recepción de una invitación o ante el envío de un mensaje INVITE o REGISTER.
- **`E_heInvitado`**: Cuando el cliente envía un mensaje INVITE a otro cliente, pasa a este estado.

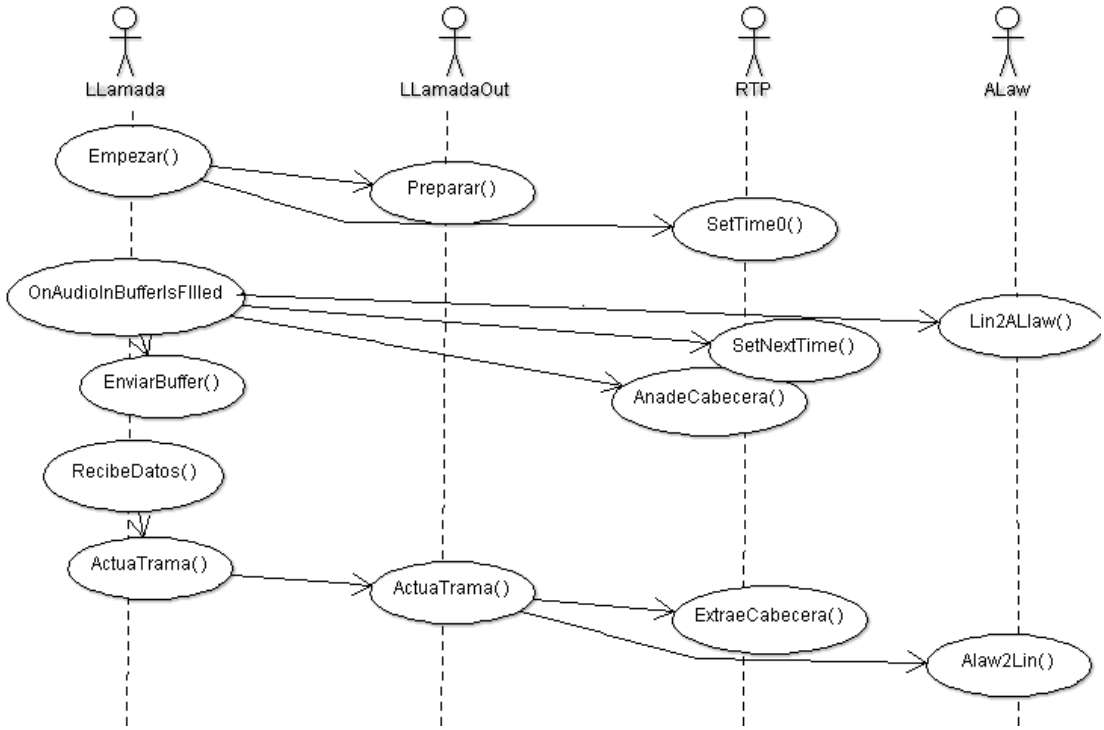


Figura 5.11: Interacción entre clases, Cliente (Multimedia)

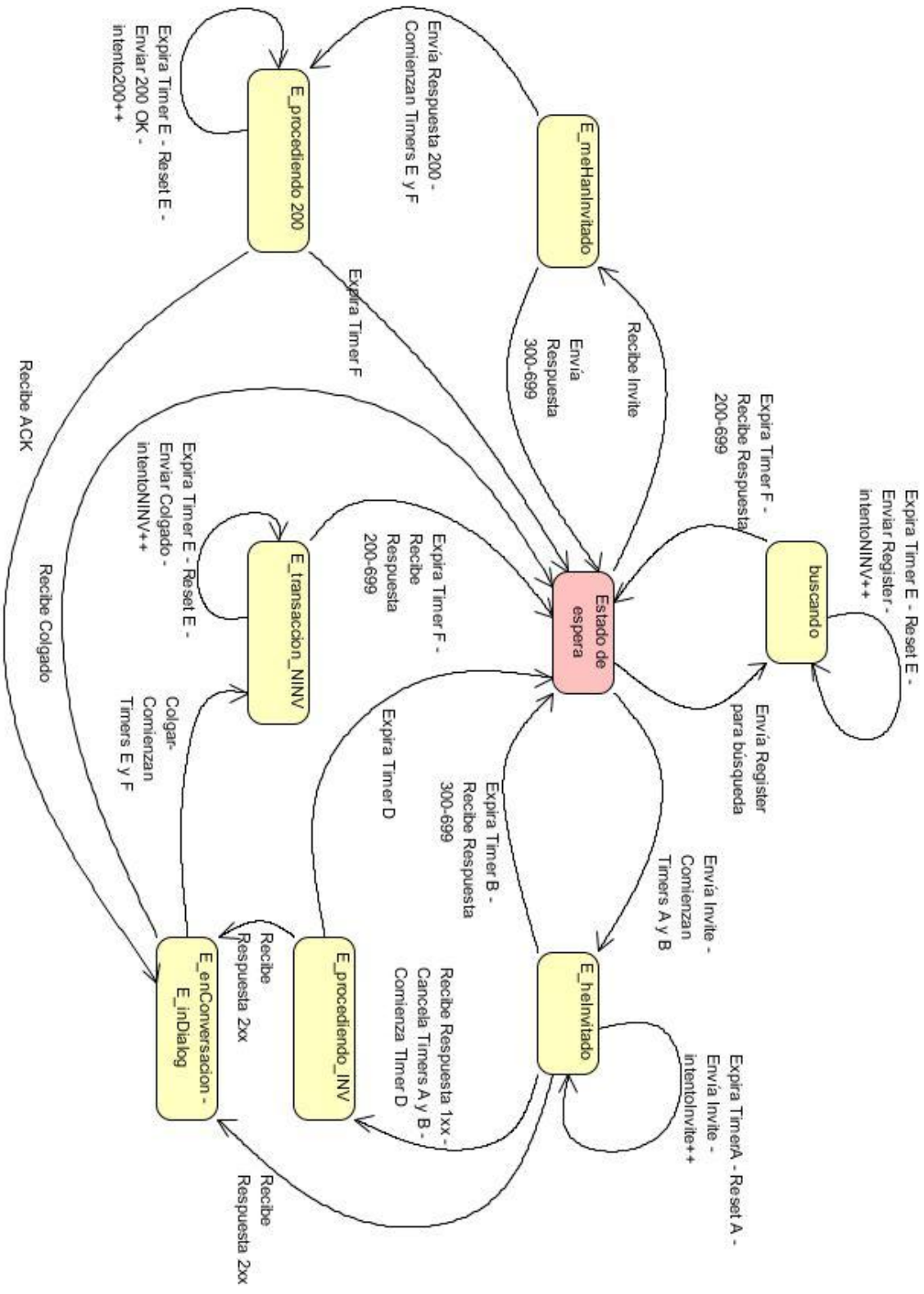


Figura 5.12: Diagrama de estados del cliente

- **E_procediendo_INV:** Este estado es un estado de transición entre el envío de una invitación y la recepción de una respuesta definitiva. Pasa a este estado si, por ejemplo, recibe una respuesta 180 RINGING al envío de un mensaje INVITE. El cliente sabe que el mensaje llegó, pero se espera una respuesta definitiva.
- **E_inDialog:** Implica que ya se conocen todos los datos necesarios para identificar un diálogo con un cliente. Para que esto ocurra, debe recibirse una respuesta 200 OK en caso de ser quien inició el diálogo, o un mensaje ACK cuando se aceptó una llamada.
- **E_enConversación:** Cuando el cliente comienza una sesión multimedia, pasa a este estado.
- **E_transaccion_NINV:** Entra en este estado cuando un cliente envía una solicitud que no es INVITE, por ejemplo BYE para finalizar una llamada, con el fin de confirmar la correcta recepción del mensaje.
- **E_meHanInvitado:** Al recibir una solicitud INVITE, el cliente entra en este estado. Según la respuesta que genere, afirmativa o negativa, cambiará a un estado u otro.
- **E_procediendo_200:** El cliente ha enviado una respuesta afirmativa a una solicitud de llamada. Debe esperar un mensaje ACK para comenzarla.
- **E_buscando:** Cuando el cliente realiza una búsqueda en la red P2P, entra en este estado, que es similar al estado `E_transaccion_NINV`.

En la Figura 5.12 se pueden observar los diversos métodos empleados para la recuperación de tramas perdidas o para evitar que un abandono inesperado de un cliente bloquee a otro.

Cuando un cliente envía un mensaje INVITE a otro lanza dos temporizadores, `Timer A` y `Timer B`. El primero tiene una duración de `T1` (500 milisegundos por defecto), y el segundo una duración de 64*`T1` segundos. Si expira el temporizador `A`, el mensaje se reenvía, la variable `intentoInvite` (que comenzó el proceso con valor uno) se incrementa en uno y el `Timer A` se vuelve a lanzar con una duración de 2*`T1`*`intentoInvite`. Este proceso se repite mientras no se reciba ninguna respuesta o expire el `Timer B`. Si expira el `Timer B` quiere decir que el cliente es inalcanzable, dado que ya se enviaron diversos mensajes sin obtener respuesta. Por lo tanto, se vuelve al estado de espera. El diseño de los temporizadores se ha llevado a cabo en base a [8].

Las respuestas que pueden hacer que `Timer A` y `Timer B` se cancelen son de tres tipos: 1xx, 2xx, o entre 300 y 699:

- **Respuesta del tipo 1xx:** Quiere decir que el mensaje que se envió llegó al cliente, pero que éste todavía no ha determinado si aceptar o rechazar la llamada. Se lanza entonces el `Timer D` con una duración por defecto de 16 segundos. Este temporizador cubre dos posibilidades que harían que el cliente

se bloqueara, que el cliente no conteste a la llamada ni la rechace o que tras enviar una respuesta provisional 1xx desapareciera o se desconectara involuntariamente. De esta forma, si el temporizador `D` expira se vuelve al estado de espera.

- **Respuesta del tipo 2xx:** Se envía el mensaje ACK y se pasa al estado `E_enConversación-E_inDialog`.
- **Respuesta entre 300 y 699:** Si recibe una respuesta de este tipo, el cliente volverá al estado de espera dado que la llamada no se va a establecer.

Por otro lado están las transacciones que no se hacen mediante un mensaje INVITE, es decir, el envío de *requets* como BYE o REGISTER. Este diseño, al igual que el anterior, se ha realizado en base a [8]. Cuando se envía uno de estos mensajes, se pasa al estado `E_transaccion_NINV` y se lanzan dos temporizadores: `Timer E` y `Timer F`. El primero se lanza con una duración de `T1` (es la misma variable que en el `Timer A`) y el segundo con una duración de 64*`T1`.

Si expira el `Timer E`, la variable `intentoNINV` se incrementa (ésta comenzó con un valor de uno), se reenvía el mensaje, y se reinicia el temporizador con una duración de MIN(`intentoNINV`*`T1`,`T2`). `T2` tiene una duración de 4 segundos, de forma que los mensajes se enviarán transcurridos los siguientes intervalos de tiempo: 500ms, 1s, 2s, 4s, 4s, 4s,...

Si expira el temporizador `Timer F`, quiere decir que no se pudo establecer contacto con el nodo al que se dirige le mensaje y se vuelve al estado de espera. Si se recibe alguna respuesta, se cancelan ambos temporizadores.

Esta solución, con `Timer E` y `Timer F`, se utiliza también en el envío de *responses* 200 OK para llamadas. En el caso de aceptar una llamada surge la posibilidad de que la respuesta 200 OK se pierda, por lo que habrá que reenviarla. Pero también cabe la posibilidad de que el nodo que realizó la llamada desaparezca tras nuestra confirmación. Así pues, es necesario utilizar este mecanismo en esta situación. Dado que no se utiliza en el mismo contexto (no se trata del envío de una solicitud, sino de una respuesta), en lugar de utilizar el estado `E_transaccion_NINV`, se utiliza el estado `E_procediendo_200`, y en lugar de utilizar la variable `intentoNINV`, se utiliza la variable `intento200`.

5.3.2 Bloque *peer*

En esta sección se especifican las clases involucradas en el bloque del *peer* de la aplicación, detallando sus principales atributos y métodos. También se especifica la interacción entre clases y el diagrama de estados con el que se rige.

Clases relacionadas con la red P2P

En la Figura 5.13 se pueden ver las clases involucradas en la parte de la aplicación que corresponde a la red P2P: `Peer`, `SipPeer`, `Vecino` y `Finger`.

`Peer`

Esta clase es la encargada de actuar como un nodo de la red P2P. Para ello se apoya en otras clases. Dado que esta clase no entiende la sintaxis de los mensajes SIP, pide a la clase `SipPeer` que cree los mensajes para enviarlos y que interprete los mensajes recibidos. Además, se apoya en la clase `Finger` y en la clase `Vecino` para el mantenimiento de las tablas y para controlar las relaciones con otros nodos de la red. La interacción que se realiza entre las clases se especifica en la sección *Interacción entre clases red P2P* de este apartado.

Atributos

`m` Número de bits utilizado para los identificadores de la red P2P.

`estab` Temporizador que determina que se debe ejecutar la función de estabilización.

`T_estab` Milisegundos que habrá de tiempo entre estabilizaciones.

`ipEncontrada` Dirección IP que se ha obtenido tras realizar un búsqueda en la red.

`nodoPerdido` En el caso de que no se encuentre un nodo, éste se declara como *non-alive*, y se almacena en esta variable.

`joinComplete` Variable que almacena si se ha completado la unión a una red o no.

`N_address` Dirección IP del nodo.

`N_port` Puerto que se utilizará para las transmisiones a través del `socket`.

`N_Id` Identificador del nodo en la red P2P.

`N_pred` Predecesor en la red P2P.

`N_succ` Sucesor en la red P2P.

`listaSucesores` Lista en la que se almacenan los siguientes `m` sucesores.

`soc` `Socket` a través del cual se realizan las comunicaciones con la red.

`multicastPort` Puerto que se utilizará para los mensajes multicast.

`user` Nombre de usuario del cliente vinculado al *peer*.

Métodos

`OnTimerExpired()` Método que detecta que un temporizador ha expirado.

`Tablas()` Esta función genera un `String` con las tablas de la red P2P, con propósito de ser mostradas al usuario.

`ObtenerHash()` Realiza la función hash de un `String` dado y devuelve el resultado.

`Buscar()` Función para realizar la búsqueda de un cliente en la red P2P.

`Join()` Este método comienza el proceso de unión a una red.

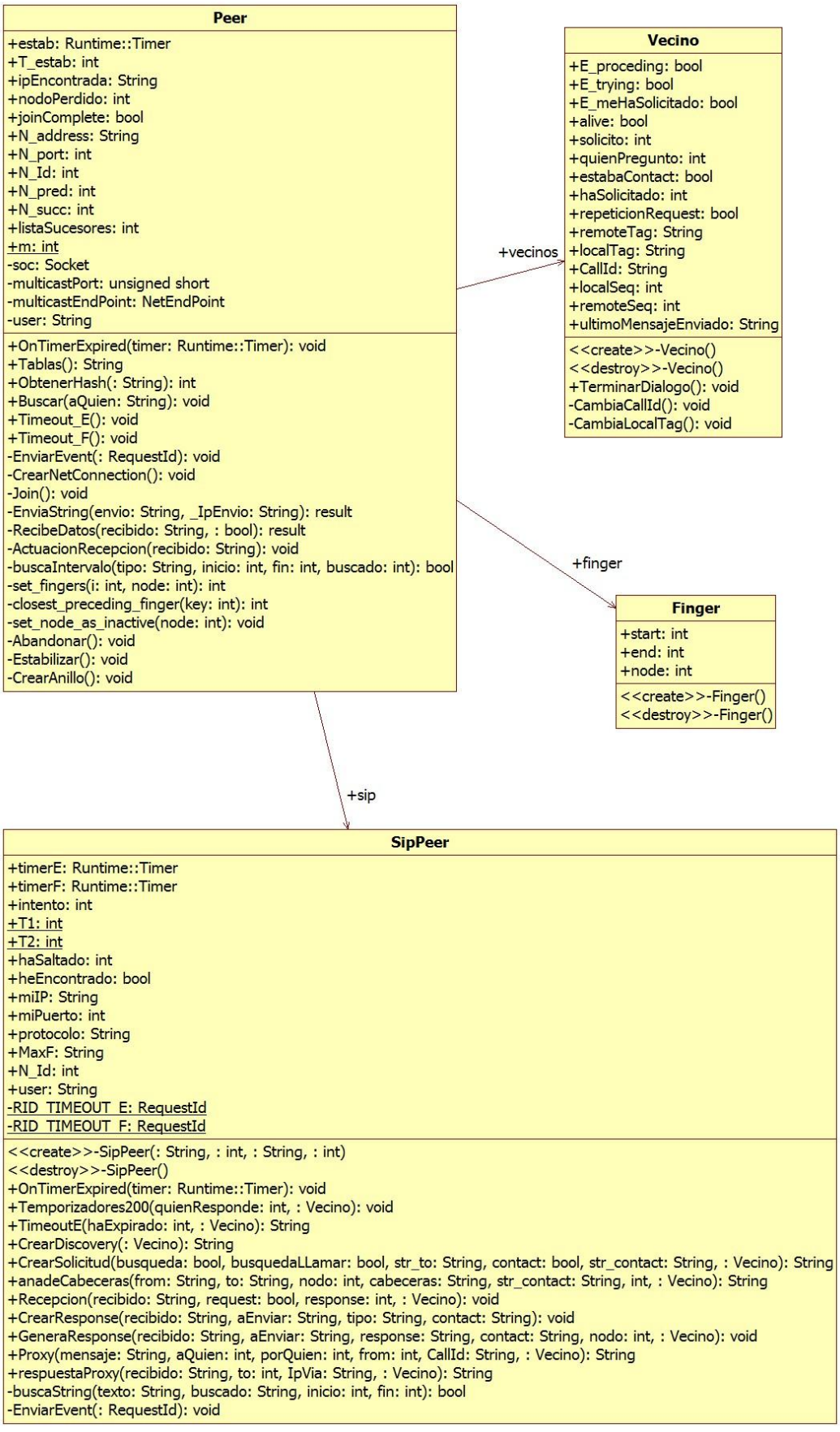


Figura 5.13: Clases *peer*

`EnviaString()` Función para enviar un `String` dado a una IP que recibe como argumento.

`RecibeDatos()` Método encargado de extraer la información recibida en el `socket`.

`ActuacionRecepcion()` Define cómo debe actuar el nodo ante la recepción de algún mensaje dependiendo de cuál sea éste.

`buscaIntervalo()` Función auxiliar para determinar si un identificador se encuentra dentro de un intervalo dado.

`set_fingers()` Método encargado de actualizar las tablas de *fingers* a partir de una posición determinada.

`closest_preceding_finger()` Localiza el nodo predecesor conocido más cercano a un identificador determinado.

`set_node_as_inactive()` Cuando se deja de tener constancia de un nodo, éste debe declararse como inactivo y realizar ajustes en la red.

`Abandonar()` Función mediante la cual el cliente comunica su abandono de la red a los otros nodos pertenecientes a la misma.

`Estabilizar()` Método que realiza los procedimientos necesarios para la estabilización de la red P2P.

`CrearAnillo()` Cuando la búsqueda de una red no es fructífera, el nodo pasa a crear una red mediante esta función.

`SipPeer`

Esta clase, como ya se ha comentado, es la encargada de introducir la sintaxis de los mensajes SIP en la red P2P. Para ello permite crear, siguiendo está sintaxis, mensajes con diferentes propósitos. También permite analizar un mensaje recibido, devolviendo los diferentes parámetros que en éste se encuentren.

Atributos

`timerE` Temporizador empleado para la recuperación de tramas perdidas.

`timerF` Temporizador empleado para determinar que un nodo es inalcanzable.

`intento` Variable que controla el tiempo de expiración de `timerE`.

`T1` Variable que define la duración de los temporizadores.

`haSaltado` Determina a qué identificador pertenecía un temporizador que ha expirado.

`heEncontrado` Especifica si una búsqueda fue fructífera.

`miIP` IP del nodo que ejecuta la aplicación.

`miPuerto` Puerto que se utilizará para los mensajes de la red P2P.

`MaxF` Número máximo de saltos que se incluirá en los mensajes SIP.

`N_Id` Identificador del *peer* que se ejecuta en la red P2P.

Métodos

`OnTimerExpired()` Función que se ejecuta ante la expiración de un temporizador. Como argumento recibe el identificador del temporizador.

`Temporizadores200()` Este método controla la cancelación de temporizadores ante la recepción de una respuesta, dependiendo de quién provenga la misma.

`TimeoutE()` Si en la función `OnTimerExpired()` se determina que el temporizador que expiró fue `timerE`, esta función determina la actuación del nodo.

`CrearDiscovery()` Crea un mensaje que tendrá como misión el descubrimiento de una red.

`CrearSolicitud()` Crea una solicitud, acorde con los parámetros que recibe.

`anadeCabeceras()` Función auxiliar en la que `CrearSolicitud()` se apoya para incorporar las cabeceras al mensaje que se está creando.

`Recepcion()` Función que extrae la información que contiene algún mensaje que se ha recibido, devolviéndola para su interpretación.

`CrearResponse()` Este método crea un mensaje de respuesta a una solicitud recibida.

`Proxy()` Función que se encarga de modificar un mensaje recibido para reenviarlo al nodo correspondiente. El mensaje reenviado contiene la información que tenía, más la que es necesaria que se le incorpore para que la respuesta pueda ser entregada correctamente.

`respuestaProxy()` Si se recibe una respuesta a un mensaje que se reenvió, esta función acondiciona la repuesta para ser entregada a quien corresponda.

`buscaString()` Método auxiliar para la búsqueda de un `String` dentro de otro más grande.

`Vecino`

Un objeto de esta clase se corresponde con los datos que un nodo tiene de otros nodos de la red. En estos datos se incluyen los posibles estados en los que el *peer* se puede encontrar con respecto a este otro nodo.

Atributos

`E_proceding` Estado en el que se especifica que se está enviando un mensaje al vecino en cuestión.

`E_trying` Estado que indica que se ha perdido algún paquete de los enviados a este nodo.

`E_meHaSolicitado` Este estado implica que se ha actuado como *proxy* con algún mensaje de este vecino.

`haSolicitado` En caso de que `E_meHaSolicitado` esté activo, esta variable registra a quién se reenvió el mensaje que se recibió de este nodo.

`alvie` Indica que no se ha perdido el contacto con este vecino.

`quienPregunto` En el caso de que algún mensaje se haya reenviado a este nodo, en esta variable se registra quién envió el mensaje que se le ha reenviado.

`estabaContact` Especifica si en el último mensaje recibido por este nodo se encontraba la cabecera *Contact*.

`remoteTag` Almacena la variable del nodo remoto con la que se está intercambiando alguna solicitud.

`localTag` Variable local respecto al vecino en cuestión.

`CallId` Identificación Call-ID que se mantiene con este vecino en el actual intercambio de mensajes o que se utilizó en el último.

`localSeq` Número de secuencia local en el envío de mensajes con respecto a este nodo.

`remoteSeq` Último número de secuencia que utilizó este vecino.

Métodos

`TerminarDialogo()` Se llama a esta función cuando finaliza un intercambio que estaba teniendo lugar entre el *peer* y este vecino, con motivo de reiniciar las variables necesarias con vista a un nuevo intercambio.

`CambiaCallId()` Genera un Call-ID nuevo y lo almacena en la variable correspondiente.

`CambiaLocalTag()` Genera un *Tag* nuevo y lo almacena en la variable correspondiente.

`Finger`

Un objeto de esta clase representa una entrada en la tabla de *fingers* del nodo. Por lo tanto, almacena los datos que cada entrada de la tabla debe tener.

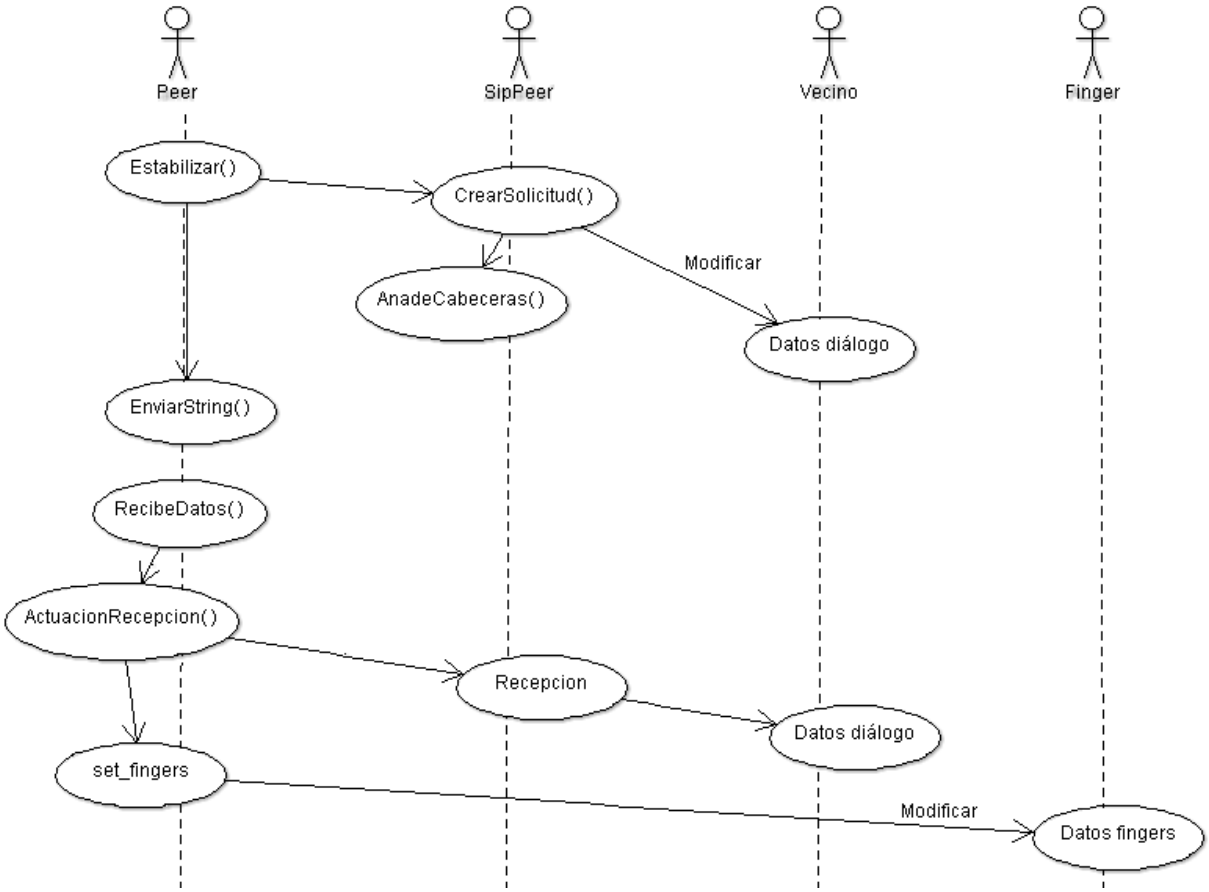


Figura 5.14: Interacción entre clases. Red P2P. Estabilización.

Atributos

`start` Haciendo uso de la nomenclatura especificada en la sección 2.3.2 , esta variable se corresponde con *finger[ ].start* .

`end` Se corresponde con el final del intervalo que abarca esta entrada de la tabla de *fingers.*

`node` De igual modo que el primero, se corresponde con *finger[ ].node* .

Interacción entre clases red P2P

A continuación se describe la interacción entre clases que tiene lugar en dos situaciones concretas, con el fin de aclarar cómo se relacionan las clases entre sí.

En primer lugar, en la Figura 5.14 se analiza un escenario de estabilización en el que el nodo comienza este proceso, recibe una respuesta y actualiza sus tablas.

En este escenario el objeto de la clase `Peer` ejecuta el método `Estabilizar()`. Este método requiere del envío de un mensaje REGISTER que debe ser creado por la

clase que comprende la sintaxis SIP, por lo que se llama al método `CrearSolicitud()` de la clase `SipPeer`. El mensaje creado por este método se apoya en el método auxiliar `AnadeCabeceras()` para añadir las cabeceras del mensaje. Para la creación de estas cabeceras se crean las variables necesarias para identificar el diálogo y que son almacenadas en el objeto de la clase `Vecino` a quien irá dirigido este mensaje. Una vez construido el mensaje, se envía mediante el método `EnviarString()`.

Cuando el nodo recibe la respuesta a este mensaje, se lo entrega al método `ActuacionRecepcion()` que, a su vez, se lo entrega al método `Recepcion()` de la clase `SipPeer` para extraer la información del mismo. Este método debe comprobar que la respuesta contiene los identificadores almacenados en el objeto de la clase `Vecino`, con el fin de confirmar que es la respuesta que se esperaba. Una vez que la función devuelve los datos que contenía el mensaje a `ActuacionRecepcion()`, éste determina que tiene que actualizar sus tablas por lo que llama al método `set_fingers()`, que modificará los datos de los objetos de la clase `Finger`.

En segundo lugar, en la Figura 5.15 se analiza la recepción de un mensaje que desea localizar a un cliente que no le corresponde almacenar, por lo que tiene que actuar como *proxy* y reenviar el mensaje al nodo conocido más cercano.

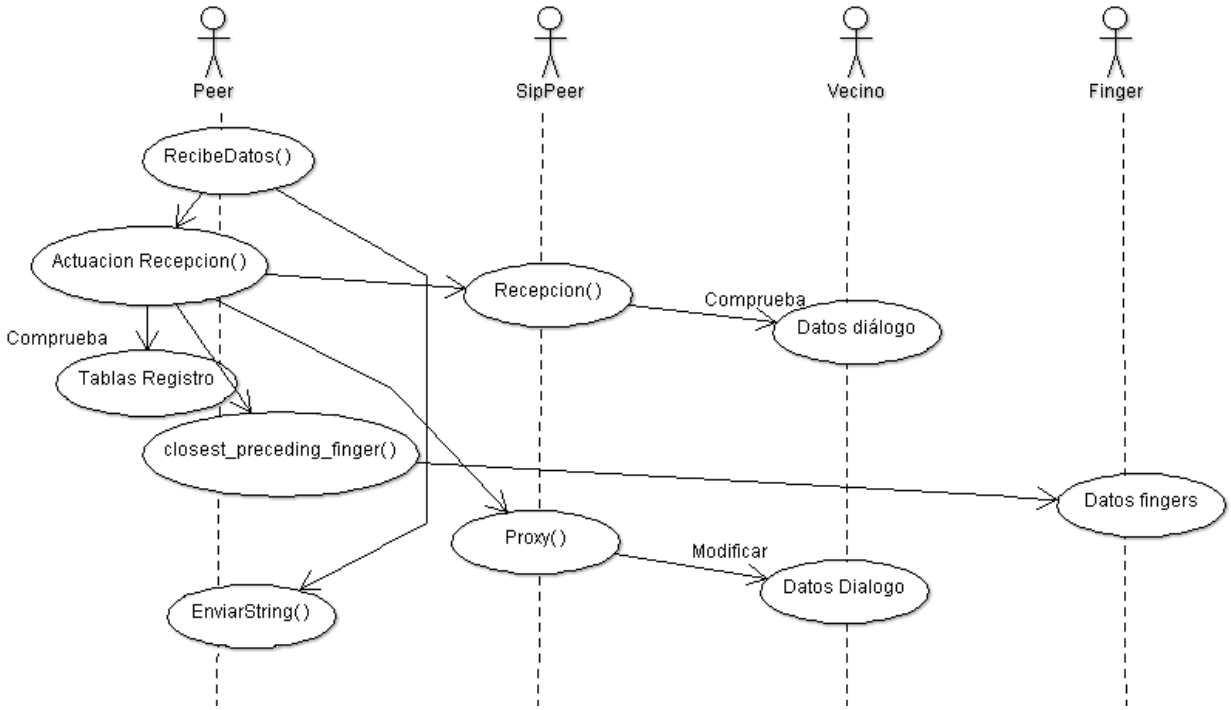


Figura 5.15: Interacción entre clases. Red P2P. Recepción de búsqueda y *proxy*.

El nodo, al recibir el mensaje, lo entrega a la función `ActuacionRecepcion()`, que necesita que la función `Recepcion()` de la clase `SipPeer` lo analice. Este método modifica los datos del objeto `Vecino` que corresponda, con el fin de poder proporcionar una respuesta correcta. Cuando el método devuelve los datos que se encuentran en el mensaje al método `ActuacionRecepcion()`, éste actúa en función de estos datos, comprobando si tiene almacenado al cliente. Puesto que no es el encargado de realizar esa función, busca en su tabla de *fingers* al nodo más cercano conocido al que le corresponde hacerlo. La búsqueda en la tabla de *fingers* se realiza accediendo a los objetos de la clase `Finger`.

Se debe modificar el mensaje una vez determinado a quién se va a reenviar, mediante el método `Proxy()` de la clase `SipPeer`, que debe almacenar en el Vecino al que se va a enviar la información de los estados pertinentes. Tras modificar el mensaje, éste se envía mediante el método `EnviarString()`.

Diagrama de estados de la red P2P

El diagrama de estados del *peer* se distribuye en cada `Vecino`. Esto quiere decir que un nodo mantiene una máquina de estados con cada uno de los nodos con los que intercambia mensajes.

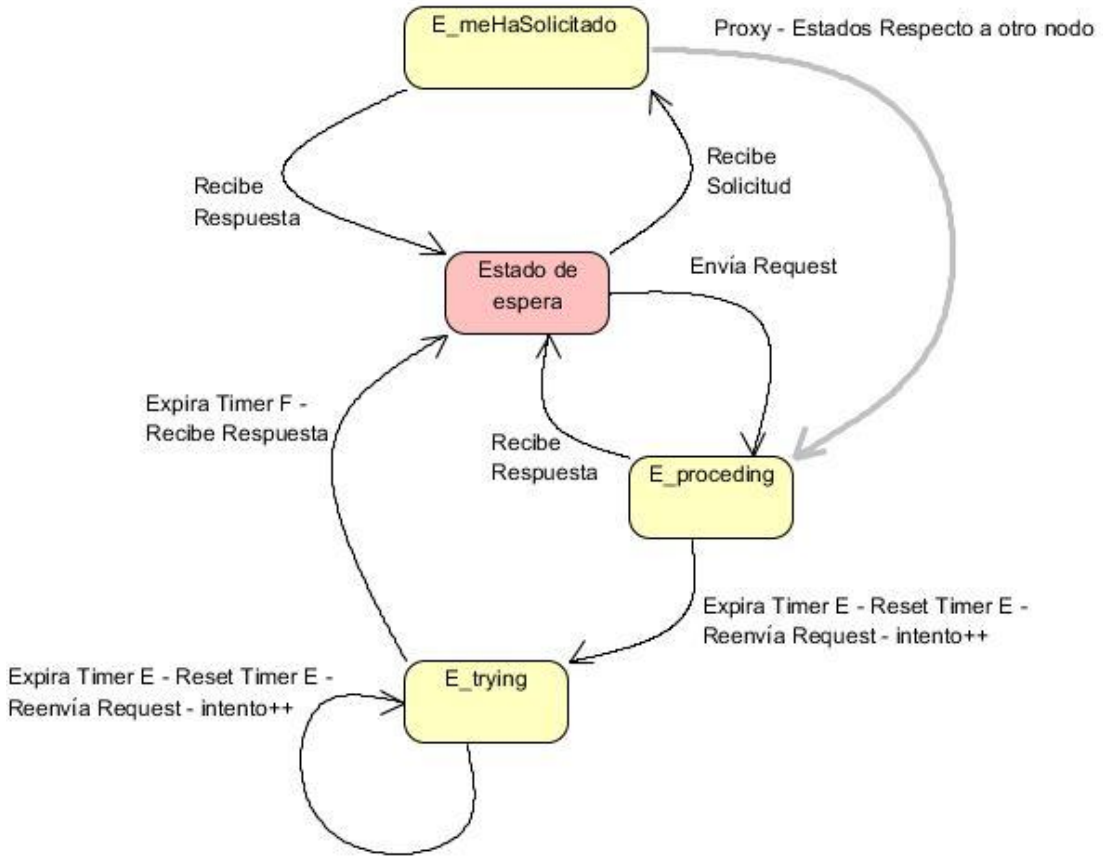


Figura 5.16: Diagrama de estados *peer*.

En la Figura 5.16 se puede ver el diagrama de estados implementado. Cabe destacar dos detalles para la compresión del mismo:

- El procedimiento usado para la recuperación de tramas perdidas es similar al empleado en la sección 5.3.1 *Diagrama de estados del cliente*. Este procedimiento[7] era el empleado para la pérdida de solicitudes que no eran INVITE. Esto se debe a que en este caso son solicitudes REGISTER, por lo que se sigue este mecanismo.
- En la figura se observa que el enlace que une `E_meHaSolicitado` con `E_proceding` es diferente a los demás. El motivo de esta diferencia consiste en que el nodo no abandona el estado `E_meHaSolicitado` respecto a aquel que realizó la solicitud, sino que sigue en ese estado con ese nodo. Por el contrario, pasa a estar en el estado `E_proceding` con el nodo al que reenvía la solicitud por la que tuvo que actuar como *proxy*. Por tanto, entran en juego máquinas de estados de diferentes nodos, estando relacionadas unas con otras.

5.4 Otras consideraciones de diseño

En esta sección se especifican otros detalles del diseño de la aplicación: la función hash utilizada, cómo actuar ante una colisión de la función hash y el uso de una *caché* de nodos conocidos.

5.4.1 Función Hash utilizada

La función hash que se utiliza para obtener un identificador a partir de un nombre de usuario es la función SHA1 [22].

Dado que esta función genera una salida de 160 bits, si se utilizase directamente la salida de la función hash como identificador, tendríamos `m` = 160. Esto implicaría que la administración de la red fuese mucho más costosa dado que, por ejemplo, cada par tiene `m` entradas en su tabla de *fingers*. Por este motivo, se reduce el número de bits que se usan para los identificadores de la red. Por defecto, el número de bits es 8 para una interpretación más sencilla de las tablas.

La reducción del número de bits se realiza cogiendo el número de bits deseado de la parte menos significativa del resultado obtenido de la función SHA1.

5.4.2 Colisión del algoritmo Hash

Si bien el uso de los 160 bits del resultado del algoritmo SHA1 proporcionaría una probabilidad casi nula de colisión, la reducción del número de bits utilizados puede hacer que esta probabilidad sea considerable. Si se reduce hasta 8 bits, existen 256 identificadores posibles, por lo que es relativamente probable que dos nombres de usuario distintos generen el mismo resultado.

[7] Procedimiento que usaba los temporizadores E y F.

El hecho de que dos nodos utilicen la misma identificación se traduciría en un funcionamiento incorrecto del sistema. Para solucionar este problema, ante una solicitud de registro, el nodo que la recibe debe comprobar si ya existe ese identificador asociado a otra dirección IP diferente. De ser así, se comunica al usuario que intenta registrarse que ese nombre de usuario no es válido y que deberá elegir otro.

5.4.3 Agilizar la búsqueda de conocidos

Cuando un cliente desea establecer una llamada, debe realizar una búsqueda en la red para obtener la dirección IP del cliente al que desea llamar. Esta búsqueda se puede traducir en el envío de un mensaje que, dependiendo del número de nodos que haya en el sistema, se deberá reenviar más o menos veces hasta localizarlo.

Si en un terminal sólo se ejecuta un cliente, es decir, no se ejecuta un *peer*, el cliente debe hacer la petición de búsqueda al *peer* con el que se haya registrado. Una vez encontrado, este *peer* realiza la búsqueda por él. Si por lo contrario se ejecutan ambos, la búsqueda la realiza directamente el *peer* que se está ejecutando en el dispositivo. Ambos modos de ejecución requieren que se realicen búsquedas en la red.

Dado que la ejecución se realiza de tal forma que el identificador del cliente coincide con el identificador del *peer*, como ya se comentó, un nodo que conoce la dirección IP de un identificador tiene la dirección perteneciente tanto al cliente como al par. Es por ello que se propone llevar a cabo una búsqueda previa entre los nodos que se conocen antes de comenzar a buscar en la red. De esta forma, la búsqueda de usuarios que estén ejecutando un *peer* puede llevarse a cabo de forma mucho más rápida, localizando al cliente a partir de la información que se tiene del *peer*. Si tras la búsqueda en la lista de vecinos conocidos no se localiza el identificador, se lleva a cabo la búsqueda habitual en la red.

Capítulo 6. Implementación

La implementación del diseño realizado se hace con el lenguaje de programación C++, con las particularidades del uso de la interfaz para programación de aplicaciones (API – *Application Programming Interface*) del sistema operativo Bada.

En este capítulo se detallan, en primer lugar, estas particularidades. También se expone el pseudocódigo de los algoritmos encargados de la gestión de la red P2P, así como el código usado en el codificador y el decodificador del códec ley-A.

6.1 Uso de la API de Bada

Para la implementación de la aplicación deseada se han hecho uso de *listeners* y de una forma particular de comunicar las diferentes clases, detallados a continuación.

6.1.1 Uso de *listeners*

Los *listeners* proporcionan información sobre eventos que han tenido lugar. Haciendo uso de ellos, se convierte en prescindible el uso de hebras para la implementación de la aplicación. Esto se debe a que no hace falta que una hebra se quede esperando a que algún evento tenga lugar, puesto que los *listeners* se encargan de notificarlos.

A continuación se detallan los *listeners* que han sido utilizados para la programación de la aplicación, así como las clases que implementan estas interfaces y sus métodos más relevantes.

`INetConnectionEventListener`

Proporciona información sobre los eventos que tienen lugar en la conexión de red. Los métodos más relevantes de esta interfaz son:

- `OnNetConnectionStarted`: Indica que la conexión de red ha sido establecida.
- `OnNetConnectionStopped`: Indica que se ha cerrado la conexión.
- `OnNetConnectionSuspended`: Indica que se ha suspendido la conexión.

- `OnNetConnectionResumed`: Indica que la conexión se ha recuperado de un estado de suspensión.

Esta interfaz la implementan las clases `Cliente` y `Peer`, puesto que son las dos clases que tienen acceso a la conexión de red.

`ISocketEventListener`

Proporciona información sobre los eventos que tienen lugar en un `socket`. Su método más relevante de que dispone es el siguiente:

- `OnSocketReadyToReceive`: Informa de que se han recibido datos y que están listos para ser recuperados.

Dado que las clases que tienen acceso a los `sockets` son `Cliente`, `Peer` y `Llamada`, éstas son las que implementan esta interfaz.

`ITimerEventListener`

Es la interfaz que notifica de los eventos ocurridos en los temporizadores. Su principal método es el siguiente:

- `OnTimerExpired`: Se llama cuando expira un temporizador.

Todas las clases que utilizan temporizadores implementan esta interfaz. Estas clases son `Peer`, `SipCliente`, y `SipPeer`.

`IAudioInEventListener`

Notifica de los eventos ocurridos durante la captura de audio. El método principal de esta interfaz es el siguiente:

- `OnAudioInBufferIsFilled`: Se ejecuta cuando el buffer en el que se estaban guardando las muestras PCM (*Pulse Code Modulation*) obtenidas por el micrófono se llena.

La clase encargada de gestionar el micrófono, `LLamada`, implementa esta interfaz.

`IAudioOutEventListener`

Esta interfaz notifica de los eventos ocurridos referentes al altavoz del dispositivo. Su principal método es:

- `OnAudioOutBufferEndReached`: Notifica que el dispositivo ha escrito un buffer completo.

`IAdHocServiceEventListener`

Proporciona información sobre una conexión Ad-Hoc. Sus principales métodos son los siguientes:

- `OnAdhocServiceStarted`: Notifica el comienzo de la conexión Ad-Hoc.
- `OnAdhocServiceStopped`: Comunica que la conexión Ad-Hoc se ha cerrado.

Listeners para la Interfaz Gráfica

Para la interfaz gráfica se usan diversos *listeners* que se enumeran a continuación:

- `IActionEventListener`: Este *listener* es el encargado de notificar la pulsación de botones en la interfaz gráfica.
- `ITextEventListener`: Es el *listener* que notifica de los eventos que tienen lugar en el teclado virtual que ofrece el dispositivo para escribir.
- `IScrollPanelEventListener`: Comunica los eventos que tienen lugar respecto a los paneles deslizables de la interfaz gráfica.

Estos *listeners* son implementados por la clase encargada de la interfaz gráfica, `Forma1`.

6.1.2 Comunicación entre clases

Es necesario que las clases intercambien información entre ellas. Para ello se utilizan dos métodos que proporciona la API de Bada, `SendUserEvent()` y `OnUserEventReceivedN()`.

El funcionamiento de estos métodos consiste en que un objeto de una clase puede enviar un evento identificado con un entero mediante el método `SendUserEvenet` a otro objeto de otra clase, que lo recibirá mediante el método `OnUserEventReceivedN`.

Así pues, la implementación de la aplicación se ha realizado usando como centro de la comunicación entre clases el objeto de la clase `Forma1`. Esto se debe a que la mayoría de los eventos que deben ser comunicados están relacionados con la interfaz gráfica. Por ejemplo, si se recibe una solicitud de llamada, se debe comunicar este evento a la interfaz gráfica.

A continuación se detallan los mensajes que son enviados desde las distintas clases.

Cliente

- `RID_LLAMANDO`: El cliente está realizando una llamada.
- `RID_CLIENTE_UNIDO`: El cliente se ha unido satisfactoriamente a una red. Sólo es necesario en caso de que no se ejecute *peer* en el terminal.

- `RID_CLIENTE_ENCUENTRA`: La búsqueda de un cliente ha sido fructífera. Al igual que el anterior, sólo es necesario en caso de que no se ejecute un *peer* en el terminal.

SipCliente

- `RID_LLAMADA_ENTRANTE`: El cliente ha recibido una petición de llamada.
- `RID_SONANDO`: Ha recibido una respuesta 180 RINGING a una solicitud de llamada que ha enviado.
- `RID_HABLANDO`: Ha recibido una respuesta 200 OK a una solicitud de llamada que ha enviado.
- `RID_ADIOS`: Ha recibido una solicitud BYE del cliente con el que estaba manteniendo una llamada.
- `RID_RECHAZADA`: Ha recibido una respuesta 603 DECLINED de un cliente a quien envío una solicitud INVITE.
- `RID_REINVITE`: Ha expirado el `Timer A` de una petición INVITE.
- `RID_TIMEOUT_INVITE`: Ha expirado el `Timer B` o el `Timer D` de una petición INVITE.
- `RID_RE200`: Ha expirado el `Timer E` perteneciente al envío de una respuesta 200 OK.
- `RID_TIMEOUT_200`: Ha expirado el `Timer F` perteneciente al envío de una respuesta 200 OK.
- `RID_RENINV`: Ha expirado el `Timer E` perteneciente al envío de una solicitud que no era INVITE.
- `RID_TIMEOUT_NINV`: Ha expirado el `Timer F` perteneciente al envío de una solicitud que no era INVITE.

Peer

- `RID_UNIDO`: Se ha completado el proceso de unión a una red.
- `RID_PEER_ENCUENTRA`: La búsqueda de un cliente ha sido fructífera.
- `RID_PERDIDO`: Un nodo ha desaparecido de la red.
- `RID_PEER_NOENCUENTRA`: Una búsqueda de un cliente no ha sido satisfactoria.
- `RID_NOMBRE_NODISPONIBLE`: Ha recibido la notificación de que el nombre de usuario que desea utilizar no está disponible.

SipPeer

- `RID_TIMEOUT_E`: Ha expirado el `Timer E` perteneciente al envío de una solicitud.
- `RID_TIMEOUT_F`: Ha expirado el `Timer F` perteneciente al envío de una solicitud.

En la Figura 6.1 se puede ver cómo se relacionarían el cliente y el *peer* que se ejecutan en un mismo terminal. Se puede ver el ejemplo del establecimiento de una llamada, que transcurre de la siguiente forma:

- El usuario indica mediante la interfaz gráfica que quiere establecer una llamada con un usuario.
- La clase `Forma1` comunica al *peer* que debe realizar la búsqueda. Para ello envía el mensaje oportuno.
- Cuando en el `socket` de la clase `Peer` se recibe la respuesta con la IP que se buscaba, envía un mensaje `RID_PEER_ENCUENTRA` indicando que ya la ha recibido.
- Puesto que en el mensaje anterior sólo se envía un entero, la clase `Forma1`, recoge la dirección IP que el objeto de la clase `Peer` le ha comunicado que ya ha encontrado.
- Una vez que ya dispone de la IP, le pide al cliente que envíe el mensaje INVITE oportuno a esa dirección.

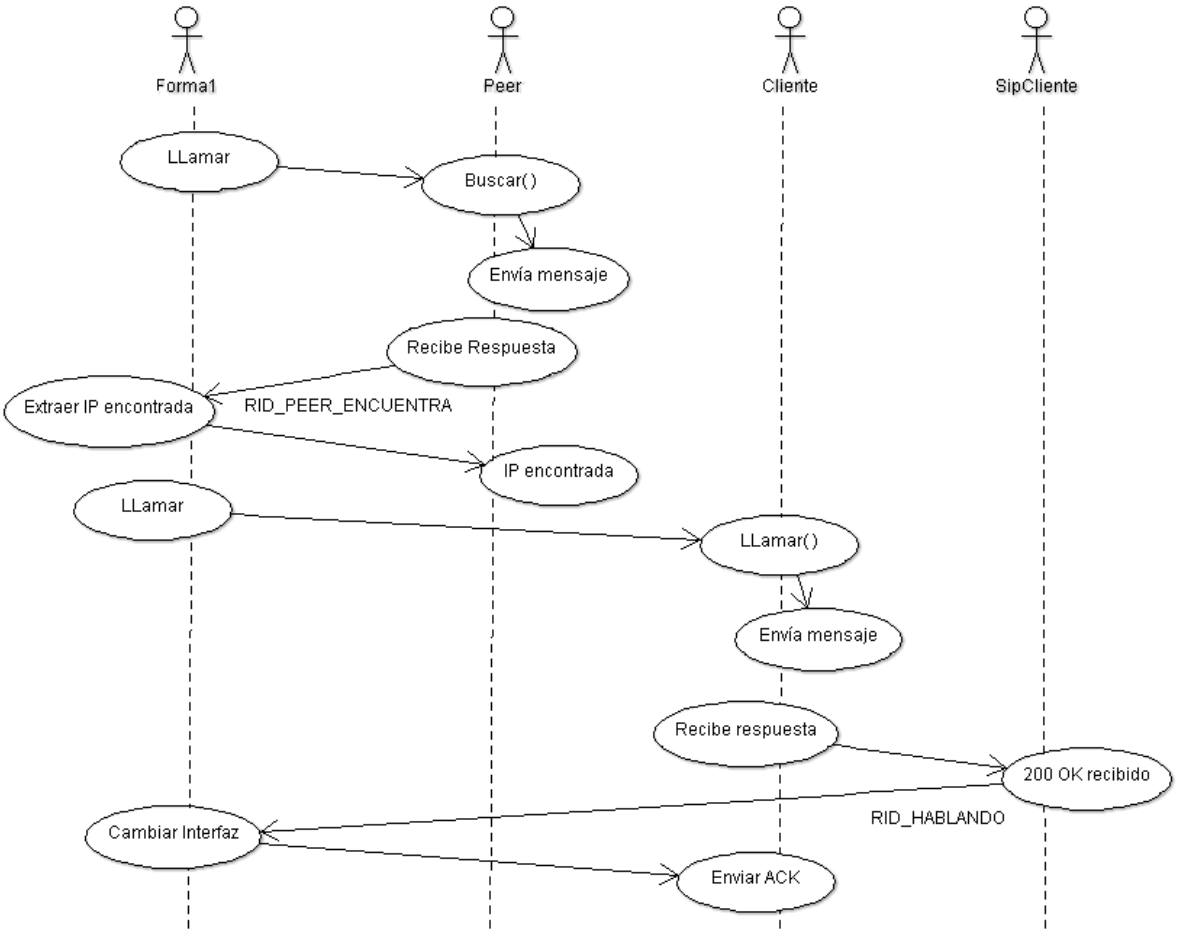


Figura 6.1: Comunicación entre *peer* y cliente

- Cuando el cliente recibe la respuesta se la pasa a `SipPeer` que, tras analizar que se trata de una respuesta afirmativa, lo comunica a `Forma1` mediante un mensaje `RID_HABLANDO` para que modifique la interfaz y para que haga que se envía la solicitud ACK.

6.2 Red P2P

La implementación de los algoritmos para la gestión de la red P2P es uno de los puntos más importantes del presente proyecto. Por ello, en este apartado se detallan los principales algoritmos implementados.

Para facilitar la comprensión estos algoritmos se expondrán como pseudocódigo en lugar del código C++ programado para la aplicación. Es necesario tener en cuenta la siguiente nomenclatura utilizada para comprender el pseudocódigo:

- N_{Id} Identificador del nodo.
- N_{addr} Dirección de transporte del nodo (IP y puerto).
- N_{pred} Identificador del predecesor.
- N_{succ} Identificador del sucesor.
- F^i Entrada en la tabla de *fingers* para un índice *i*.
- F^i_{start} Campo comienzo, de una entrada en la tabla de *fingers*.
- F^i_{end} Campo final, de una entrada en la tabla de *fingers*.
- F^i_{node} Campo siguiente salto, de una entrada en la tabla de *fingers*.
- N^{list}_{succ} Lista de sucesores.
- N^i_{succ} Sucesor número *i*.

La gestión de la red se centra en la actuación ante los mensajes recibidos y en el algoritmo de estabilización. Así pues, esta sección de divide en: recepción de solicitud, recepción de respuesta y estabilización. Pero antes se describen dos funciones básicas, que serán utilizadas por el resto de algoritmos.

En la Tabla 6.1 se detalla la función encargada de modificar la tabla de *fingers*, comenzando en el índice *i*. Esta función devuelve el índice de la última posición de la tabla que se actualizó.

También se hace uso de una función que localiza el predecesor más cercano a una clave (*key*) dada. El pseudocódigo de esta función se muestra en la Tabla 6.2.

6.2.1 Recepción de solicitud

Las solicitudes que puede recibir un nodo son de dos tipos y se diferencian por la presencia o no de la cabecera *Contact* en el mensaje. Si contiene esta cabecera se trata de un nodo que ya se ha unido a la red. Por el contrario, si no contiene esta cabecera se trata de un nodo que se quiere unir a la red. Puesto que son dos tipos de mensajes distintos, se siguen dos procedimientos distintos ante la recepción de cada uno de estos mensajes.

set_fingers(*i:* start index , *node:* localización del nodo **)**

$F_{node}^{i} \leftarrow node$
while $i \leq m - 1$ **do**
  if $F_{start}^{i} \in [N_{Id}, F_{node}^{i}]$ **then**
    $F^{i+1} \leftarrow F_{node}^{i}$
  else
    return *i*
  $i \leftarrow i + 1$
return *i*

Tabla 6.1: set_fingers

El procedimiento que se sigue cuando se recibe una solicitud sin la cabecera *Contact* se puede ver en la Tabla 6.3. En éste se comprueba, en primer lugar, si el propio nodo que recibe el mensaje ya está unido a la red. De no ser así, ignora el paquete recibido. Tras comprobar que la cabecera *Contact* no está en el mensaje, comprueba si el identificador del nodo que envió la solicitud se encuentra entre él y su sucesor. De ser así, envía una respuesta indicando que él es su predecesor y que su sucesor es el actual sucesor del nodo. De no ser así, busca el predecesor más cercano que conoce mediante el procedimiento de la Tabla 6.2 y actúa como *proxy* reenviando el mensaje al nodo pertinente.

closest_preceding_finger(*key***)**

$node \leftarrow N_{Id}$
for $i \leftarrow m$ hasta 1 **do**
  if F_{node}^{i} está activo **and** $F_{node}^{i} \in (N_{Id}, key)$
    $node \leftarrow F_{node}^{i}$
    break
return $node$

Tabla 6.2: closest_preceding_finger

Recibe solicitud, siendo *M*, la solicitud recibida

if unión no completada **then**
  ignora *M*
else if *M*.Contact no está **then**
  $to \leftarrow M.\text{To}$
  if $to \neq N_{Id}$ **and** $to \notin (N_{Id}, N_{succ}]$ **then**
    $node \leftarrow$ closest_preceding_finger(to) /*Tabla 6.2*/
    proxy *M* **to** *node*
  else
    enviar respuesta 200 OK
      Contact: N_{succ} **;** predecessor = N_{Id}

Tabla 6.3: Recepción de solicitud sin *Contact*

Cuando la cabecera *Contact* está presente en la solicitud, indica que es un mensaje de estabilización de un nodo que ya está unido a la red. En este caso, se actúa siguiendo el procedimiento que se muestra en la Tabla 6.4. En éste, tras comprobar que la cabecera *Contact* está presente, se comprueba si el nodo que ha enviado la solicitud se encuentra entre el propio nodo y su predecesor. De ser así, el predecesor del nodo que recibe la solicitud pasa a ser el nodo que la envió. Después, se envía una respuesta con la lista de sucesores para que el nodo pueda actualizar su propia lista.

Recibe solicitud, siendo *M*, la solicitud recibida

if unión no completada **then**
  ignora *M*
else if *M*.Contact está **then**
  if N_{pred} está vacío **or** *M*.From ∈ (N_{pred}, N_{Id}) **then**
    $N_{pred} \leftarrow M.\text{From}$
  enviar respuesta 200 OK
    Contact: N_{Id}; predecessor = N_{pred}
    Contact: N_{succ}^{0}
    Contact: N_{succ}^{1}
    …

Tabla 6.4: Recepción de solicitud con *Contact*

6.2.2 Recepción de respuesta

Cuando el nodo recibe una respuesta a una solicitud que envió, su actuación depende de si era una solicitud con la cabecera *Contact* o no.

Si la solicitud que envió tenía esta cabecera, el nodo actúa como especifica en la Tabla 6.5.

Recibe respuesta a una solicitud sin Contact, siendo *M*, la respuesta recibida

if *M*.To = N_{Id} **and** N_{succ} está vacío **then**
  /* Me estoy uniendo a la red*/
  $k \leftarrow \text{set\_fingers}(1, M.\text{Contact}) + 1$ /*Tabla 6.1*/
  if $k \leq m$ **then**
    /*Quedan entradas vacías en la tabla de figners*/
    /*Se envía una solicitud para la siguiente entrada vacía*/
    $id \leftarrow N + 2^{k-1}$
    enviar REGISTER *M*.Contact
      To: sip:*id*@desconocido
  else
    /*Estabilizar con el predecesor*/
    enviar REGISTER *M*.Contact
      To: N_{Id}
      Contact: N_{Id}; predecessor = N_{pred}
    unión completada
else
  if $\exists i$ t. q. $F_{start}^{i} = M.\text{To}$ **then**
    $i \leftarrow \text{set\_fingers}(i, M.\text{Contact})$
    if $i < m$ **then** /*Aún quedan entradas vacías en la tabla*/
      $id \leftarrow F_{start}^{i+1}$
      enviar REGISTER *M*.Contact
        To: sip:*id*@desconocido
    else if *i* = *m*
      enviar REGISTER *M*.Contact
        To: N_{Id}
        Contact: N_{Id}; predecessor = N_{pred}
      unión completada

Tabla 6.5: Recepción de respuesta, a solicitud sin *Contact*

Cuando el nodo recibe esta respuesta comprueba si el campo N_{succ} está vacío, lo que quiere decir que envió un mensaje para unirse a la red. Comprueba también si la cabecera *To* coincide con su identificador, lo que significa que es la respuesta al primer mensaje que envió para tratar de unirse a la red. De ser así, actualiza la tabla de *fingers* comenzando por la primera entrada. Si toda la tabla ha sido actualizada, envía un mensaje de estabilización y da por completada la unión a la red. Si faltan entradas de la tabla por completar, enviará una solicitud con el fin de conocer al nodo que corresponda para seguir completando la tabla.

En el caso contrario al anterior, es decir, que la cabecera *To* no coincida con su identificador, cuando recibe la solicitud entiende que envió un mensaje con la finalidad de terminar de completar la tabla. Al recibirlo, actualiza la tabla a partir de la entrada correspondiente y vuelve a actuar como en el caso anterior tras actualizarla.

Si, en cambio, se recibe una respuesta a una solicitud que contenía la cabecera *Contact*, el nodo actúa como se especifica en la Tabla 6.6.

Recibe respuesta a una solicitud con Contact, siendo *M*, la respuesta recibida

if $N_{succ} = M.\text{To}$ **then**
  $pred \leftarrow M.\text{Contact.predecessor}$
  if $pred \neq N_{Id}$ **and** $pred \in (N_{Id}, N_{succ})$
    set_fingers(1, *pred*)
  if $pred = N_{Id}$ **then**
    $N_{succ}^{list} \leftarrow M.\text{Contacts}$
  if unión completada
    /*Estabilizar la próxima entrda de la tabla de fingers*/
    $i \leftarrow \lceil \log_2(N_{succ} - N_{Id}) \rceil$
    if $i < m$ **then**
      $id \leftarrow F_{start}^{i+1}$
      if $id \in (N_{Id}, N_{succ}]$ **then**
        $node \leftarrow N_{Id}$
      else
        $node \leftarrow$ closest_preceding_finger(id)
      if $node = N_{Id}$ **then**
        $node \leftarrow N_{succ}$
      enviar REGISTER *node*
          To: sip:*id*@desconocido

Tabla 6.6: Recepción de respuesta, a solicitud con *Contact*

En esta ocasión comprueba si debe actualizar la tabla de *fingers* con motivo del mensaje recibido y si debe actualizar la lista de sucesores. De ser así, las actualiza, cabiendo la posibilidad de que tenga que enviar alguna solicitud para rellenar alguna entrada de la tabla de *fingers*.

6.2.3 Estabilización

La estabilización se lleva a cabo mediante el simple envío de un mensaje, combinado con los algoritmos de actuación ante la recepción de un mensaje vistos anteriormente. En la Tabla 6.7 se puede ver el algoritmo de estabilización que se ejecuta periódicamente. Se envía un mensaje de estabilización al sucesor y al predecesor. En el caso de que sólo hubiera dos nodos, el sucesor y el predecesor serían los mismos, por lo que se comprueba que no sea así para no enviar un mensaje repetido.

Estabilización

if unión completada **then**
  if $N_{succ} \neq N_{Id}$ **then**
  enviar REGISTER N_{succ}
  To: N_{succ}
  Contact: N_{Id}; predecessor = N_{pred}
  if $N_{succ} \neq N_{pred}$ **then**
  enviar REGISTER N_{pred}
  To: N_{pred}
  Contact: N_{Id}; predecessor = N_{pred}

Tabla 6.7: Estabilización

6.3 Códecs

Para la aplicación se ha implementado un códec estándar de telefonía IP, como es el códec G.711. De sus dos variantes, ley-μ y ley-A, se ha implementado el segundo de estos por ser el utilizado normalmente en Europa.

A continuación se pueden ver el código usado para codificar una muestra lineal a ley-A y viceversa.

6.3.1 Codificador

La función del codificador recibe como argumento la muestra de voz lineal, de 16 bits y en complemento a dos. El código de esta función se muestra en la Tabla 6.8.

Codificador de muestra lineal a ley-A

```
byte
ALaw::Lin2Alaw(short pcm_val)    /* 2's complement (16-bit range) */
{
        short  mask;
        short  seg;
        short  out_range;
        unsigned charaval;

        if (pcm_val >= 0) {
                mask = 0x55; //00   /* signo septimo bit = 1 */
        } else {
                mask = 0xD5; //80   /* sign bit = 0 */
                pcm_val = -1*pcm_val;
        }

        out_range = pcm_val & 0x7000;
        if(out_range != 0){
                return (unsigned char) (0x7F ^ mask);
        }
        pcm_val = pcm_val & 0xFFF;

        /* Obtiene el segmento de la escala en el que se encuentra*/
        seg = search(pcm_val, seg_aend, 8);

        /* Combina el signo, el segment, y los bits de cuantización
        if (seg >= 8)        /* Fuera de rango, devuelve el máximo valor */
                return (unsigned char) (0x7F ^ mask);
        else {
                aval = (unsigned char) seg << SEG_SHIFT;
                if (seg < 2)
                        aval |= (pcm_val >> 1)& QUANT_MASK;
                else
                        aval |= (pcm_val >> seg)& QUANT_MASK;
                return (aval ^ mask);
        }

}
```

Tabla 6.8: Codificador de muestra lineal a ley-A

6.3.2 Decodificador

La función del decodificador tiene como misión devolver una muestra de voz lineal, de 16 bits y en complemento a dos, a partir de una muestra codificada previamente.

El código de esta función, se puede ver en la Tabla 6.9.

Decodificador de ley-A, a lineal.

```
Short
ALaw::Alaw2Lin(byte a_val)
{
        short          t;
        short          seg;
        short          signo;
        short         ret;
        bool           sign;
        a_val ^= 0x55;
        if((a_val & (0x80)) !=0 ){
                signo = 0x8000;
                sign = true;
        }else{
```

```
                signo = 0x0;
                sign = false;
        }
        t = (a_val & QUANT_MASK);
        seg = ((unsigned)a_val & SEG_MASK) >> SEG_SHIFT;
        switch (seg) {
        case 0:
        {
                t <<= 1;
                ret = t^0x1;

        }break;
        case 1:
        {
                t<<=seg;
                ret = t^0x21;
        }break;
        case 2:
        {
                t<<=seg;
                ret = t^0x42;
        }break;
        case 3:
        {
                t<<=seg;
                ret = t^0x84;
        }break;
        case 4:
        {
                t<<=seg;
                ret = (0x108) ^ t;
        }break;
        case 5:
        {
                t<<=seg;
                ret = t^0x210;
        }break;
        case 6:
        {
                t<<=seg;
                ret = t^0x420;
        }break;
        case 7:
        {
                t<<=seg;
                ret = t^0x840;
        }break;
        }
        if(signo){
                return (-ret);
        }else{
                return (ret);
        }
}
```

Tabla 6.9: Decodificador de ley-A, a lienal

Capítulo 7. Evaluación

En este capítulo se hará una evaluación de los resultados obtenidos. En primer lugar, se hará una evaluación haciendo uso de la herramienta Wireshark analizando un escenario típico de actuación del sistema. Después, se analizará el correcto funcionamiento de la red P2P con un escenario en el que cinco nodos se conectan a la misma. Por último, se analizarán algunos aspectos de la robustez del sistema.

7.1 Análisis con Wireshark

Mediante el analizador de protocolos Wireshark se puede analizar el correcto funcionamiento del sistema ante distintas situaciones. En primer lugar, se analizará el establecimiento de una llamada entre dos nodos. En segundo lugar, se revisará la actuación del sistema ante los diversos casos posibles de pérdida de tramas.

7.1.1 Llamada

El escenario que se usa para el análisis de la llamada, es el que se puede ver en la figura Figura 7.1.

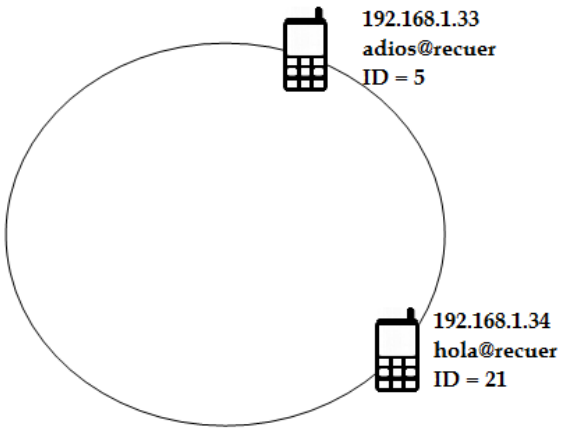


Figura 7.1: Escenario de la llamada

En este escenario, el establecimiento de una llamada del usuario *adios@recuer*, al usuario *hola@recuer* transcurre como se puede ver en la Figura 7.2.

En esta captura, las tramas número 271, 707, y 729 no están relacionadas con la llamada, ya que son parte de la estabilización de la red.

Para el establecimiento de la llamada, en primer lugar, el usuario debe obtener la dirección IP de aquel a quien quiere llamar. Para ello, *adios@recuer* envía un mensaje REGISTER (trama número 273 en la figura) solicitando la dirección IP del usuario *hola@recuer*.[8] Una vez que el usuario recibe la dirección del nodo al que desea llamar, la llamada transcurre de la siguiente forma:

- *adios@recuer* envía un mensaje INVITE con destino *hola@recuer*. Trama 275 en la Figura 7.2.
- Tras recibir la solicitud de llamada, *hola@recuer* envía una respuesta provisional 180 RINGING, indicando que ha recibido la solicitud. Trama 276.
- Cuando el usuario *hola@recuer* acepta la llamada, se envía una respuesta 200 OK. Trama 286.
- El cliente *adios@recuer*, ante la recepción de la respuesta anterior, envía automáticamente el mensaje ACK. Trama 287.
- Comienza el intercambio de mensajes RTP. No se muestra en la Figura 7.2 porque sería imposible abarcar el intercambio en una figura.
- Cuando el usuario *adios@recuer* desea finalizar la llamada, envía una solicitud BYE. Trama 913.
- Tras recibir esta solicitud, *hola@recuer* envía una respuesta 200 OK y finaliza la llamada. Trama 919.

Este intercambio descrito anteriormente, se puede ver también en el análisis de telefonía que proporciona Wireshark a través del gráfico de flujo con los principales mensajes intercambiados pertenecientes a una llamada (véase la Figura 7.3).

No.	Time	Source	Destination	Protocol	Length	Info
271	127.702721	192.168.1.33	192.168.1.34	SIP	263	Request: REGISTER sip:192.168.1.34:5061 (fetch bi
273	128.623000	192.168.1.33	192.168.1.34	SIP	253	Request: REGISTER sip:192.168.1.34:5061 (fetch bi
274	128.688258	192.168.1.34	192.168.1.33	SIP	276	Status: 200 OK (1 bindings)
275	128.725610	192.168.1.33	192.168.1.34	SIP/SDP	449	Request: INVITE sip:hola@recuer, with session descri
276	128.785219	192.168.1.34	192.168.1.33	SIP	268	Status: 180 Ringing
286	131.126103	192.168.1.34	192.168.1.33	SIP/SDP	416	Status: 200 OK, with session description
287	131.145313	192.168.1.33	192.168.1.34	SIP	298	Request: ACK sip:hola@192.168.1.34:0
707	137.475877	192.168.1.33	192.168.1.34	SIP	263	Request: REGISTER sip:192.168.1.34:5061 (fetch bi
729	137.674971	192.168.1.33	192.168.1.34	SIP	263	Request: REGISTER sip:192.168.1.34:5061 (fetch bi
913	140.348970	192.168.1.33	192.168.1.34	SIP	316	Request: BYE sip:hola@192.168.1.34:0
919	140.549637	192.168.1.34	192.168.1.33	SIP	260	Status: 200 OK

Figura 7.2: Captura de Wireshark de la llamada

[8] Para este escenario, no se hace uso de la mejora propuesta en el apartado 5.4.3 para agilizar la búsqueda de nodos conocidos, con el fin de que evaluar la búsqueda del cliente. En el caso de que se utilizara, no enviaría ningún mensaje para buscar la dirección, dado que es su sucesor y ya la conoce.

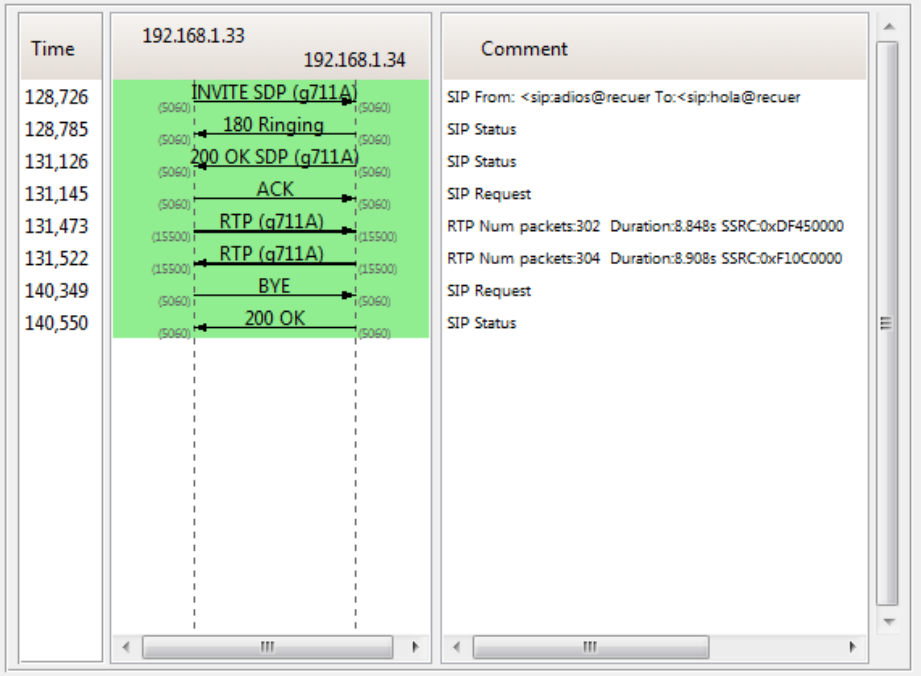


Figura 7.3: Flujo mensajes para la llamada

Es interesante, con el fin de comprobar la correcta formación de los mensajes, observar el análisis pormenorizado que Wireshark hace del mensaje INVITE, usado en el establecimiento de llamada anterior.

```
⊟ Session Initiation Protocol
  ⊟ Request-Line: INVITE sip:hola@recuer SIP/2.0
      Method: INVITE
    ⊞ Request-URI: sip:hola@recuer
      [Resent Packet: False]
  ⊟ Message Header
    ⊟ Via: SIP/2.0/UDP 192.168.1.33:5060;branch=z9hG4bK3
        Transport: UDP
        Sent-by Address: 192.168.1.33
        Sent-by port: 5060
        Branch: z9hG4bK3
    ⊟ From: <sip:adios@recuer>;tag=0GaXSMYh
      ⊟ SIP from address: sip:adios@recuer
          SIP from address User Part: adios
          SIP from address Host Part: recuer
        SIP tag: 0GaXSMYh
    ⊟ To: <sip:hola@recuer>
      ⊟ SIP to address: sip:hola@recuer
          SIP to address User Part: hola
          SIP to address Host Part: recuer
    ⊟ CSeq: 3 INVITE
        Sequence Number: 3
        Method: INVITE
      Call-ID: r2CpO5NUDqrcttSGFOPJe3QuCbmbM5PN
      Max-Forwards: 70
    ⊟ Contact: <adios@192.168.1.33:5060>
      ⊞ Contact-URI: adios@192.168.1.33:5060
      Content-Type: application/sdp
```

Figura 7.4: Cabeceras SIP

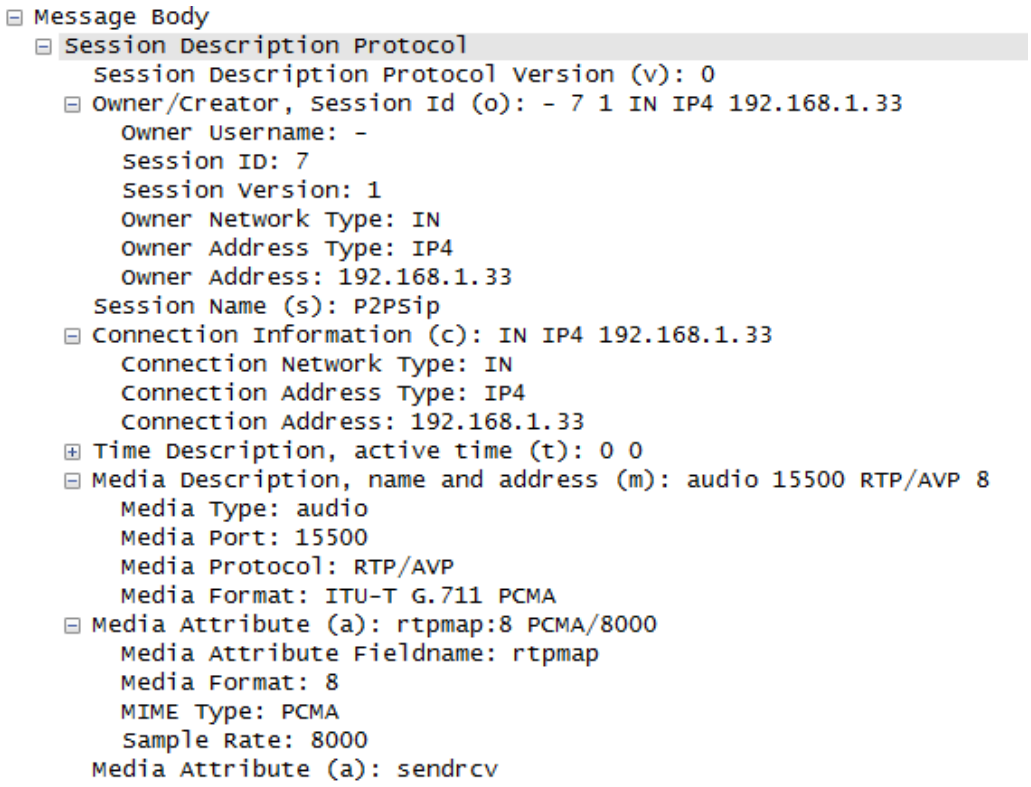


Figura 7.5: Mensaje SDP

Este mensaje se compone de dos partes, las cabeceras del mensaje SIP y el cuerpo del mensaje que contiene el mensaje SDP. En la Figura 7.4 puede verse el análisis de Wireshark de las cabeceras del mensaje SIP, mientras que en la Figura 7.5 puede verse el análisis que hace del cuerpo formado por SDP.

Del análisis pormenorizado que hace Wireshark del mensaje SIP, cabe destacar los siguientes puntos:

- El mensaje se divide en tres secciones, *Request-Line, Message Headers, y Message Body*.
- La *Request-Line* se descompone en el método de la solicitud, “INVITE”, y la URI del usuario al que se dirige, “Hola@recuer”.
- Las cabeceras que contiene el mensaje son: “Via”, “From”, “To”, “CSeq”, “Call-ID”, “Max-Forwards”, y “Contact”.
- En el cuerpo del mensaje se puede ver la descripción de la sesión multimedia, en la que se detalla el códec que se va a utilizar, así como el protocolo y el puerto que se usará durante la sesión.

Puesto que Wireshark es un analizador de protocolos contrastado, el hecho de que éste desglose cada uno de los parámetros del mensaje anterior de forma correcta, indica que el diseño que se ha hecho de los diferentes protocolos está basado fielmente al RFC de cada uno de ellos.

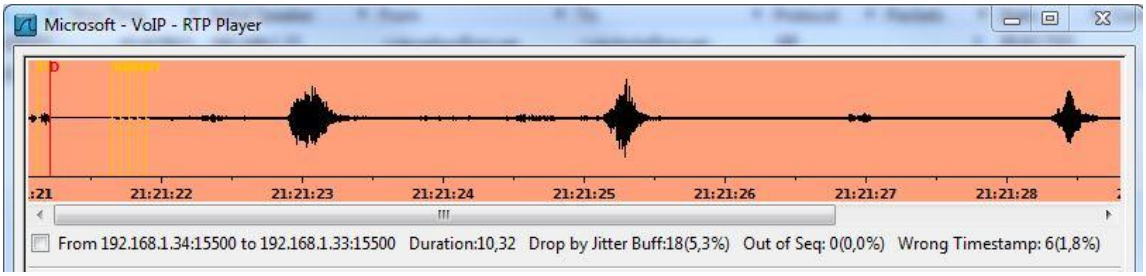


Figura 7.6: Decodificación de una llamada

Para evaluar la correcta implementación del códec de voz, y del apartado multimedia en general, se hace uso del decodificador de llamadas VoIP que proporciona Wireshark. En la Figura 7.6 se puede ver la correcta decodificación de una llamada.

7.1.2 Recuperación de tramas perdidas

A continuación se evalúa la eficacia de los mecanismos propuestos para proporcionar robustez al sistema en cuanto a pérdida de tramas se refiere.

Los tres escenarios que requieren estos mecanismos para el establecimiento de una llamada son los siguientes: pérdida del mensaje INVITE, pérdida de la respuesta 200 OK y pérdida de alguna solicitud que no sea INVITE (BYE, REGISTER).

El análisis de la pérdida de un mensaje, para las diferentes situaciones, se puede ver en la Figura 7.7.

Source	Destination	Protocol	Length	Info
192.168.1.37	192.168.1.35	SIP	301	Request: INVITE sip:luis@recuerda
192.168.1.37	192.168.1.35	SIP	301	Request: INVITE sip:luis@recuerda
192.168.1.37	192.168.1.35	SIP	301	Request: INVITE sip:luis@recuerda
192.168.1.37	192.168.1.35	SIP	301	Request: INVITE sip:luis@recuerda
192.168.1.35	192.168.1.37	SIP	274	Status: 180 Ringing
192.168.1.35	192.168.1.37	SIP	269	Status: 200 OK

a) INVITE

Source	Destination	Protocol	Length	Info
192.168.1.58	192.168.1.57	SIP	295	Request: INVITE sip:luis@recuerda
192.168.1.57	192.168.1.58	SIP	270	Status: 180 Ringing
192.168.1.57	192.168.1.58	SIP	265	Status: 200 OK
192.168.1.57	192.168.1.58	SIP	265	Status: 200 OK
192.168.1.57	192.168.1.58	SIP	265	Status: 200 OK
192.168.1.57	192.168.1.58	SIP	265	Status: 200 OK
192.168.1.57	192.168.1.58	SIP	265	Status: 200 OK
192.168.1.57	192.168.1.58	SIP	265	Status: 200 OK
192.168.1.58	192.168.1.57	SIP	298	Request: ACK sip:luis@192.168.1.57:5060

b) 200 OK

Source	Destination	Protocol	Length	Info
192.168.1.33	192.168.1.58	SIP	316	Request: BYE sip:luis@192.168.1.58:5060
192.168.1.33	192.168.1.58	SIP	316	Request: BYE sip:luis@192.168.1.58:5060
192.168.1.33	192.168.1.58	SIP	316	Request: BYE sip:luis@192.168.1.58:5060
192.168.1.33	192.168.1.58	SIP	316	Request: BYE sip:luis@192.168.1.58:5060
192.168.1.58	192.168.1.33	SIP	267	Status: 200 OK

c) NON-INVITE

Figura 7.7: Recuperación ante pérdida de tramas

En el caso a), pérdida de un mensaje INVITE, se puede ver como un usuario envía una solicitud INVITE a otro. El temporizador A (véase la sección 5.3.1) del usuario expira y vuelve a enviar la solicitud dado que el paquete puede haberse perdido. Este proceso se repite varias veces hasta que recibe una respuesta.

En el caso b), pérdida de un mensaje 200 OK, la situación que se observa es la siguiente. Un usuario envía una solicitud INVITE a otro, que la recibe y envía una notificación 180 RINGING y una respuesta 200 OK, ésta última cuando el usuario descuelga. Cuando envía esta respuesta, pasa a esperar el mensaje ACK. Al no recibirlo, su temporizador E (véase la sección 5.3.1) expira y el mensaje se reenvía. Se sigue este procedimiento hasta que se recibe el mensaje ACK.

El caso de pérdida de un mensaje no INVITE, c), es similar al anterior. En este caso el usuario envía una solicitud BYE y lanza el temporizador E (véase la sección 5.3.1), si no recibe ninguna respuesta final y el temporizador expira, el mensaje se reenvía. Sigue este procedimiento hasta que recibe una respuesta final 200 OK aceptando el colgado.

7.2 Red P2PP con 5 nodos

En esta sección se evalúa la red P2P mediante un escenario con 5 nodos, analizando las tablas de cada uno de ellos. Las tablas que se mostrarán por parte de cada nodo siguen la sintaxis que se muestra en la Tabla 7.1.

En estas tablas se muestra, en primer lugar, el identificador del nodo que se está ejecutando en ese terminal (*N_Id*). Posteriormente se muestran las *m* (número de bits utilizado para la identificación en la red) entradas de la tabla de *fingers*. Cada entrada de la tabla de *fingers* se representa por el índice de la entrada correspondiente, seguido por un intervalo cerrado-abierto en el que se encuentran el comienzo de esa entrada (*finger(i).start*) y el final de la misma (*finger(i).end*). Por último, se muestra el próximo salto correspondiente para esa entrada (*finger(i).node*).

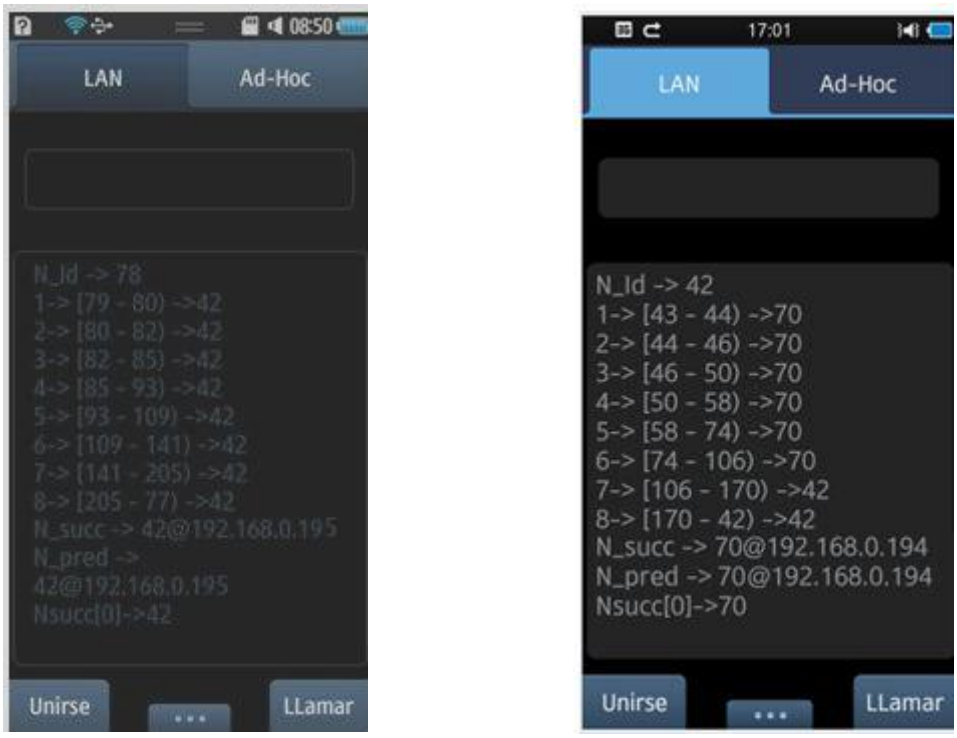
Tras las entradas de la tabla, se muestran el sucesor y el predecesor del nodo como *N_pred* y *N_succ* respectivamente. Y por último, se muestran las entradas en la tabla de sucesores en la que se muestran los sucesores del nodo, hasta un máximo de *m* sucesores.

El escenario que se muestra sigue la siguiente serie de acontecimientos:

- La red está formada en primera instancia por los nodos 78 y 42 (Figura 7.8).
- Se une el nodo 70 (Figura 7.9).
- Se une el nodo 151 (Figura 7.10).
- Se une el nodo 7 (Figura 7.11).

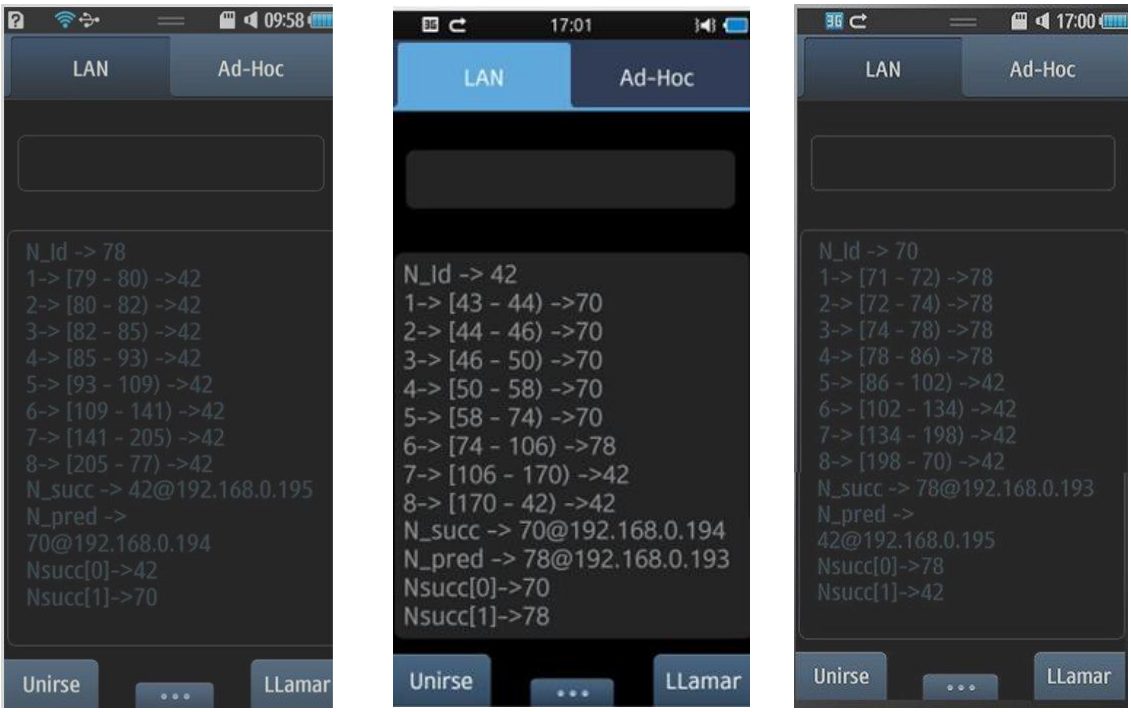
$N\_Id \rightarrow N_{Id}$	
$i \rightarrow [N\_Id+2^{i-1} \bmod m - N\_Id+2^{i} \bmod m) \rightarrow$	Próximo nodo conocido mayor que $N\_Id+2^{i-1} \bmod m$
$N\_succ \rightarrow N_{succ}$	
$N\_pred \rightarrow N_{pred}$	
$Nsucc[j] \rightarrow N_{succ}^{j}$	

Tabla 7.1: Sintaxis de las tablas



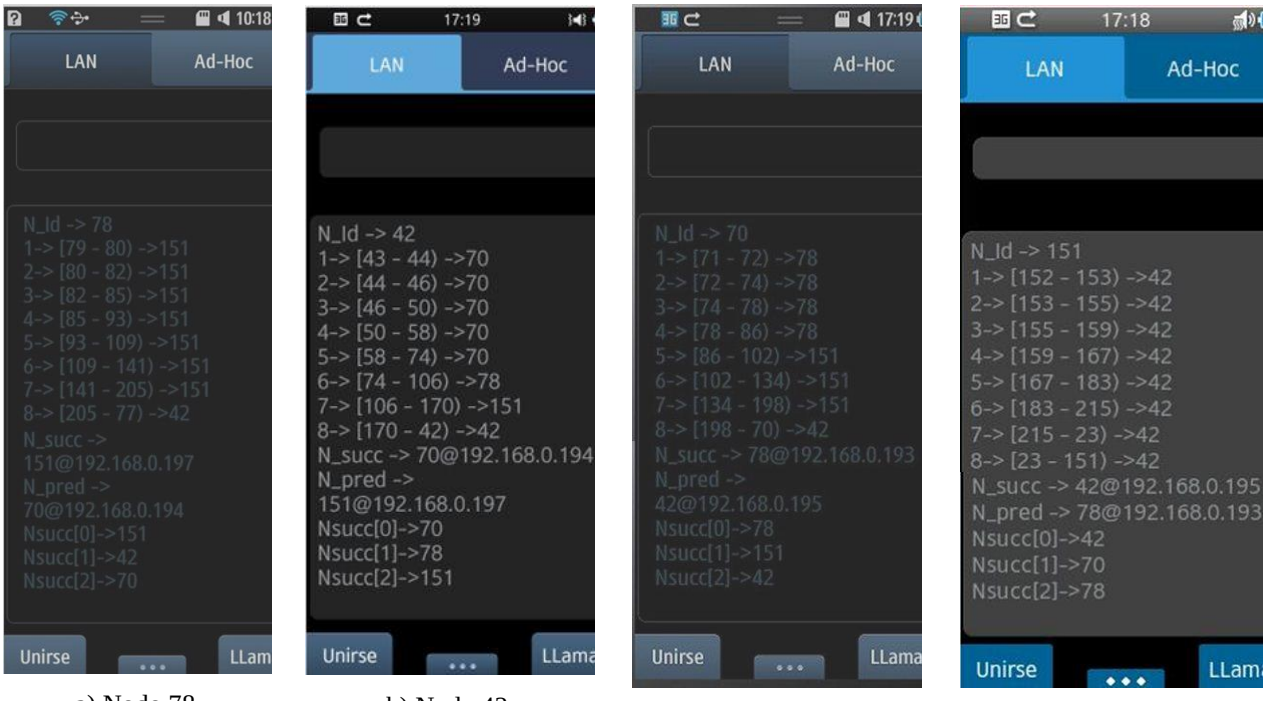
a) Nodo 78 b) Nodo 42

Figura 7.8: Red con 2 nodos.



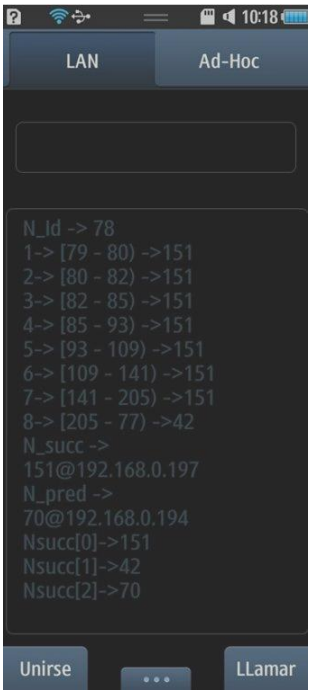
a) Nodo 78 b) Nodo 42 c) Nodo 70

Figura 7.9: Red con 3 nodos

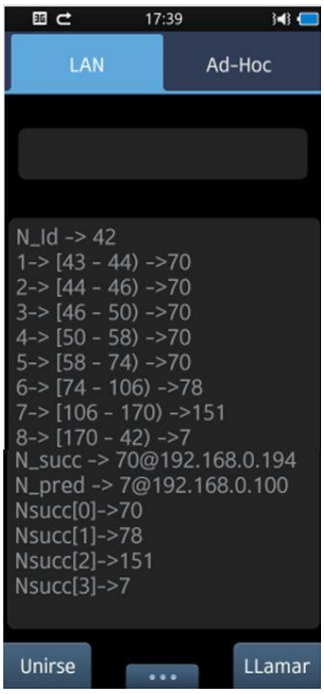


a) Nodo 78 b) Nodo 42 c) Nodo 70 d) Nodo 151

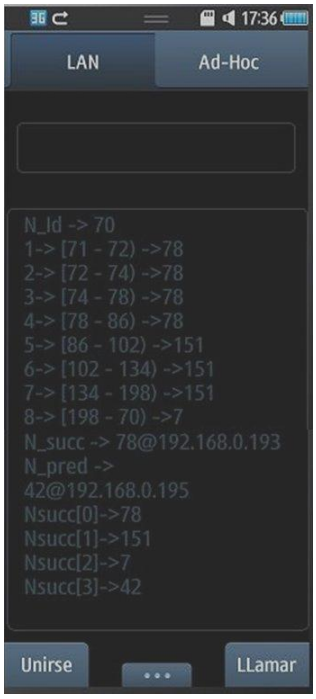
Figura 7.10: Red con 4 nodos



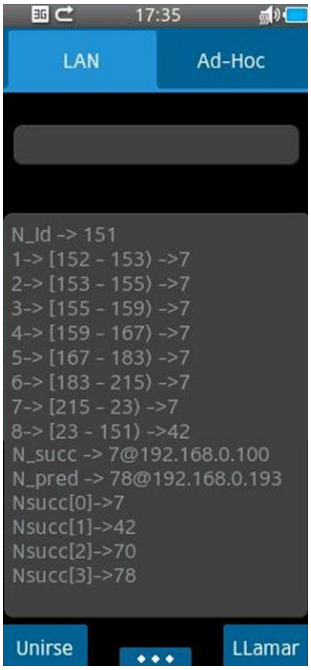
a) Nodo 78



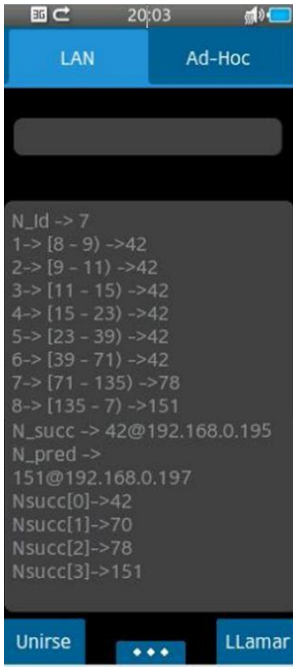
b) Nodo 42



c) Nodo 70



d) Nodo 151



e) Nodo 7

Figura 7.11: Red con 5 nodos

En la Figura 7.11 se pueden ver las tablas de cada uno de los nodos, cuando todos están unidos a la red P2P. La topología que tiene la red en ese momento es la que se puede ver en la Figura 7.12.

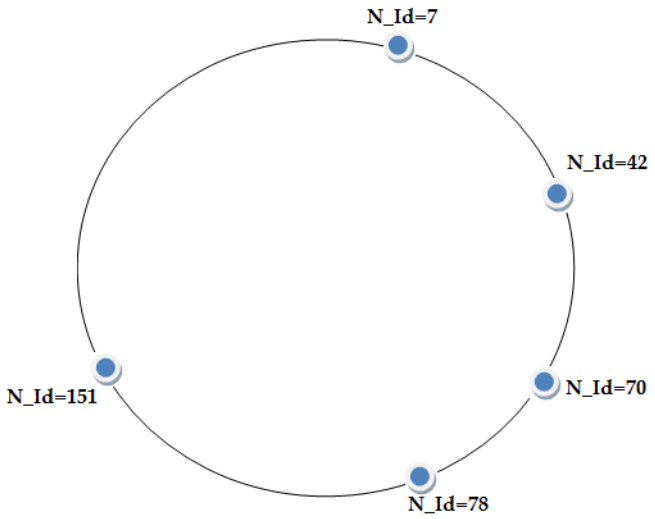


Figura 7.12: Topología de la red con 5 nodos

7.3 Robustez del sistema

Además de la recuperación de pérdida de tramas, la robustez del sistema se puede evaluar desde otros puntos de vista. En primer lugar, se evaluará el tiempo de ejecución que el sistema puede estar activo y, en segundo lugar, se usará una herramienta que proporciona el entorno de desarrollo de Bada para evaluar la robustez.

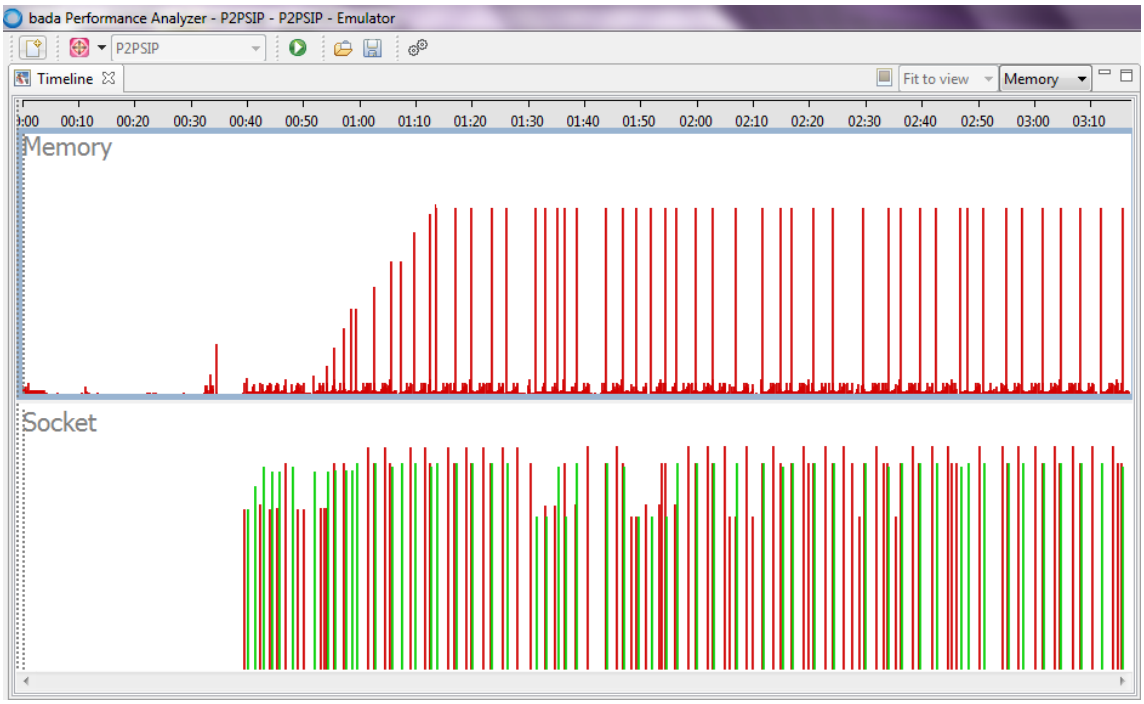


Figura 7.13: Análisis de la memoria y el uso de los sockets

7.2.1 Tiempo de ejecución

En cuanto al tiempo que la aplicación se puede ejecutar, tras realizar diversas pruebas, el sistema no tiene una limitación concreta. Así pues, la limitación de tiempo de ejecución continuo viene dada por la duración de la batería del dispositivo.

Las pruebas que se han realizado consisten en la ejecución de la aplicación en un terminal móvil de forma ininterrumpida en diversas ocasiones, evaluando el tiempo que el dispositivo es capaz de estar ejecutándola de forma continua.

7.2.2 Análisis con Performance Analyzer

Uno de los mayores problemas de la programación de aplicaciones móviles es el uso de la memoria del dispositivo. Para controlar ésta, el entorno de desarrollo de Bada proporciona una herramienta llamada *Performance Analyzer* que evalúa el correcto uso y liberación de la misma. En la Figura 7.13 se puede ver una captura de pantalla donde se observa el uso de la memoria y de los `sockets` durante la ejecución de la aplicación.

Las líneas verticales que se pueden apreciar en la figura, se corresponden con el uso que se hace del recurso evaluado. En el caso de la memoria, las líneas rojas indican el uso que la aplicación hace de ésta en cada momento de la ejecución. En cuanto al `socket`, se pueden apreciar líneas de dos colores, las líneas verdes son el tráfico entrante mientras que las rojas el tráfico saliente. Se puede ver que éstas siguen un patrón periódico de envío y recepción, que se corresponde con el proceso de estabilización de la red que se hace de forma periódica.

Si las líneas rojas que se ven en la parte destinada a la memoria no decrecieran, significaría que hay un problema en el uso que la aplicación hace de la memoria. Por tanto, puesto que las líneas rojas se estabilizan, quiere decir que la memoria se está usando y liberando de forma correcta.

Capítulo 8. Conclusiones

En este capítulo se van a detallar los resultados obtenidos en el proyecto, analizando las aportaciones de éste. También se describen las líneas de trabajo futuras con las que se podría continuar trabajando sobre los resultados obtenidos en este trabajo.

8.1 Resultados

En el proyecto se ha abordado el problema de la telefonía VoIP en redes P2P y su uso en dispositivos móviles. Para ello, se diseña e implementa una aplicación que ofrece las siguientes aportaciones:

- La aplicación implementada permite su uso sin infraestructura de red, haciendo uso de una red inalámbrica Ad-Hoc.
- Se ha diseñado una aplicación que proporciona flexibilidad en cuanto al tipo de red utilizada, permitiendo su uso mediante una red WiFi tanto Ad-Hoc como de infraestructura (con punto de acceso).
- Se ha estudiado e implementado el protocolo SIP con las funcionalidades necesarias para su correcto funcionamiento. También se han realizado las modificaciones necesarias para cambiar la arquitectura cliente-servidor, para la que está diseñado, por una arquitectura *peer-to-peer*.
- También se ha implementado el protocolo SDP, necesario para describir la sesión multimedia que los usuarios tienen que establecer.
- Se ha diseñado e implementado una red P2P que permite a un usuario localizar a cualquier otro con un número de saltos relativamente bajo, requiriendo únicamente el nombre de usuario de éste. Esta red P2P se basa en tablas hash distribuidas.
- Permite flexibilidad al usuario en cuanto a su implicación con la red P2P, permitiendo elegir a éste si quiere formar parte de la misma o simplemente registrarse con algún nodo de la red.

- Se ha implementado el codificador de voz G.711 (en su variante que hace uso de ley-A), muy extendido en el uso de aplicaciones VoIP.
- Se ha implementado el protocolo RTP, para las sesiones multimedia.
- Se ha realizado una implementación que permite la codificación y decodificación de muestras de voz, así como la captura y reproducción de éstas en tiempo real. Todo esto simultáneamente con el encapsulado y envío, recepción y desencapsulado, de paquetes RTP con las muestras de voz. Así pues, permite mantener una conversación que no se vea frustrada por los retardos del sistema.

Si bien existen aplicaciones de este tipo para otros sistemas operativos, no es el caso para el sistema operativo Bada, como ya se comentó en la sección 1.2 . Esto se debe a la prohibición de la empresa propietaria de usar la tecnología VoIP en este sistema operativo que existió hasta Marzo de 2011. Así pues, esta prohibición se tradujo en una imposibilidad de incorporar librerías ya existentes que facilitasen la implementación del sistema.

En definitiva, la aplicación desarrollada cubre las funcionalidades de un usuario de telefonía IP en redes P2P, implementando cada uno de los bloques y protocolos necesarios, teniendo acceso únicamente a `sockets`, micrófono y altavoz.

8.2 Líneas de trabajo futuras

Tras el proyecto realizado, quedan abiertas las siguientes líneas de trabajo abiertas:

- Con las futuras versiones del sistema operativo Bada, en las que se ha comunicado que se dará soporte para VoIP, se podrían incorporar más códecs al sistema que requieran un menor ancho de banda.
- Se podría incorporar la posibilidad de que un nodo con acceso a Internet, actuara como pasarela para comunicarse con servidores VoIP existentes (e.g. Ekiga [23]), de forma que el sistema incorporaría una importante funcionalidad.
- Se podría añadir esta aplicación a una plataforma que incorporara más funcionalidades, tales como el intercambio de mensajes de texto o el intercambio de ficheros.
- La aplicación desarrollada opera sobre la interfaz de red WiFi del dispositivo. Se podría modificar el trabajo realizado para permitir su uso mediante HSDPA, conectándose a la red móvil. Las principales modificaciones requeridas serían las siguientes:
    - En el diseño realizado, el descubrimiento de los nodos de la red se realiza mediante mensajes multicast. Esto no sería posible, por lo que se debería disponer de otro mecanismo.

  - Habría que implementar de nuevo el acceso a la red, modificando las funciones que acceden a ésta a través de la interfaz WiFi.

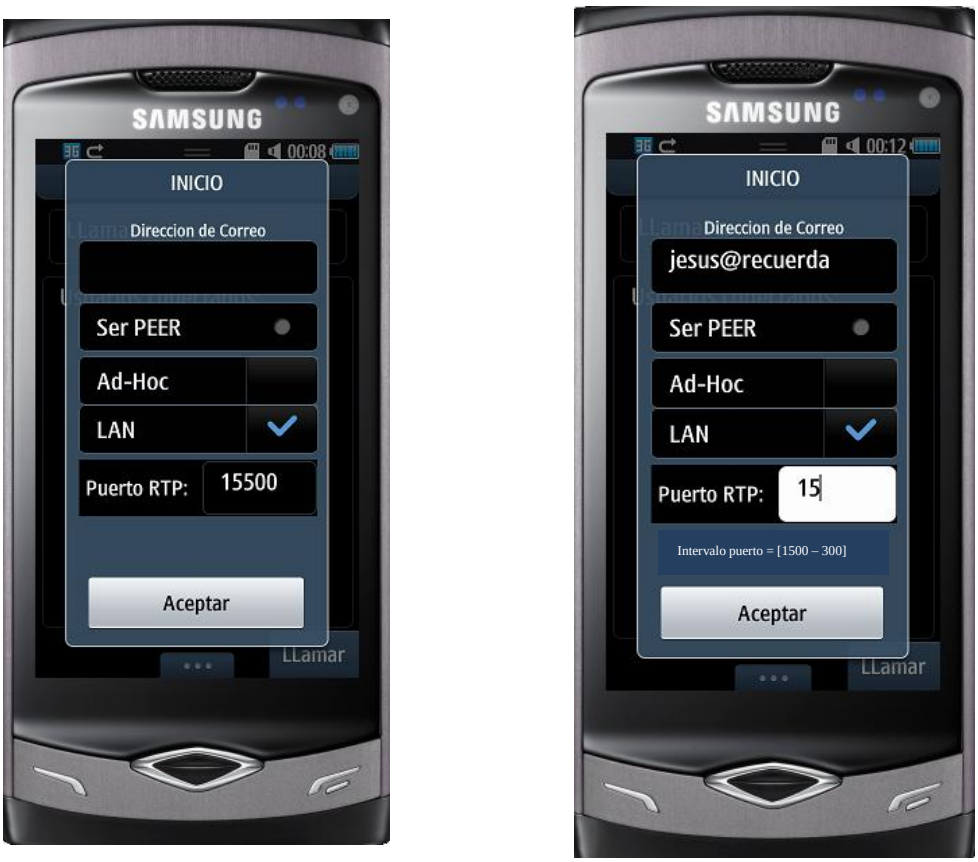
- En cualquiera de los casos en los que se permita al sistema ser usado en Internet, sería necesario usar algún mecanismo para solucionar los problemas que presentaría el direccionamiento privado (NAT) de los nodos. Con este motivo, se podrían usar mecanismos como *NAT TRAVERSAL* [24].

Apéndice A. Manual de Usuario

Este apéndice describe el manual de usuario de la aplicación. Se divide en tres secciones en las que se detallarán los posibles casos con imágenes.

A.1 Ventana inicial

Al ejecutar la aplicación, aparece una ventana que permite elegir algunas opciones. Ésta se puede ver en la Figura A.1 a).



a) b)

Figura A.1: Ventana inicial

La ventana está formada por los siguientes elementos:

- Un cuadro con el título “Dirección de correo” en el que el usuario debe escribir su dirección de correo, que será el nombre de usuario.
- Un campo seleccionable con el nombre “Ser PEER”. Seleccionándolo se puede elegir si formar parte de la red o ser sólo cliente.
- Dos opciones (“LAN” y “Ad-Hoc”) de las cuales sólo se permitirá seleccionar una. En caso de elegir la primera de ellas, la aplicación se ejecutará usando la conexión a una red WiFi de infraestructura. Si se elige la segunda, la aplicación creará una red Ad-Hoc cuyo nombre será **P2PSip_*dominio***, siendo *dominio* la parte de la dirección de correo introducida que queda tras la arroba. Por ejemplo, si el nombre de usuario fuera *jesus@recuerda*, el nombre de la red creada sería P2PSip_recuerda.
- Por último, se puede elegir el puerto RTP que la aplicación usará durante las sesiones multimedia.

Debajo de estas opciones, la aplicación tiene un campo para informar en caso de que alguna de las opciones se introdujera de forma incorrecta, como puede verse en la Figura A.1 b) en la que el puerto elegido no está en el rango permitido.

A.2 Ventana principal

Tras elegir las opciones de ejecución en la ventana inicial y pulsar el botón aceptar, ésta desaparece dejando lugar a la ventana que se puede ver en la Figura A.2 a). Esta notificación permanecerá bloqueando la pantalla hasta que se una a alguna red, momento en el que se cerrará automáticamente.

La ventana principal consta de los siguientes elementos (véase la Figura A.2 b)):

- En la parte superior de la pantalla, se puede ver el nombre de usuario con el que se ha ejecutado la aplicación.
- Un cuadro que contiene “LLamar a”. En éste se escribe el nombre del usuario al que se quiere enviar una petición de llamada.
- Un campo en el que se mostrará información dependiendo del modo de visualización en el que estemos.
- En la parte inferior se encuentra el botón “LLamar” y un menú desplegable. Pulsando el botón “LLamar” se enviará una petición de llamada al usuario que se encuentre en el campo anteriormente mencionado. El menú desplegable contiene tres opciones (véase la Figura A.2 c)):
    - “Debug/Normal” permite cambiar el modo de visualización de la aplicación. Los diferentes modos se explicarán a continuación.
    - “Abandonar” hace que el nodo abandone la red.
    - “Parar Estabilizacion” da la orden de no enviar más mensajes de estabilización en la red.

a) Notificación b) Modo normal c) Opciones disponibles

Figura A.2: Ventana principal

Los dos modos de ejecución posibles son: “Normal”, y “Debug”. El primero de ellos puede verse en la Figura A.2 b), y muestra los usuarios que hay en ese momento conectados a la red. Escribiendo su nombre en el campo “LLamar a” y pulsando el botón “LLamar” se llamará a ese usuario. El segundo de los modos de visualización se puede ver en la Figura A.3, este modo muestra la información de las tablas de rutas del *peer* del sistema. La sintaxis que sigue la información se detalló en la sección 7.2

A.3 Gestión de las llamadas

Cuando el sistema envía o recibe una llamada, ésta se gestionará mediante ventanas que aparecerán en el centro de la pantalla. Las posibles ventanas que podrían aparecer se pueden ver en la Figura A.4 y son las siguientes:

- Figura A.4 a): Cuando el usuario pulsa el botón “Llamar”, si la dirección que ha introducido es correcta, se verá esta ventana. En el caso de la figura, el usuario está llamando a “Uno@recuerda”. La palabra “Sonando” en la ventana, indica que el terminal del usuario con el que se quiere establecer la llamada ha recibido el mensaje. Hasta que no se tiene esta notificación, no aparece esta palabra. Su aparición equivale al tono de llamada en un teléfono convencional. Si se pulsa el botón “Colgar” se cancela la petición de llamada y la ventana desaparece.

Figura A.3: Modo debug

- Figura A.4 b): Esta ventana indica que la petición que se estaba enviando ha sido cancelada. Sólo cabe la posibilidad de pulsar el botón "Aceptar", que hará la ventana desaparecer.
- Figura A.4 c): El usuario al que se envió la petición de llamada, la ha aceptado. Esta ventana con el mensaje "Conectado" indica que se ha establecido la llamada. Mediante el botón "Colgar" se pone fin a la conversación y desaparece la ventana.
- Figura A.4 d): Un usuario, "Uno@recuerda" en el caso de la figura, ha enviado una petición de llamada. Ésta se puede aceptar o rechazar. Si se rechaza desaparecerá la ventana.
- Figura A.4 e): Si en la ventana comentada en el punto anterior la llamada se acepta, se pasa a esta otra y comienza la conversación. Sólo se puede pulsar el botón "Colgar", lo que finalizaría la llamada y haría desaparecer la ventana.

En caso de que se quiera enviar una petición de llamada a un usuario que no se encuentra en la red, aparece una notificación que lo indica, como se puede ver en la Figura A.5.

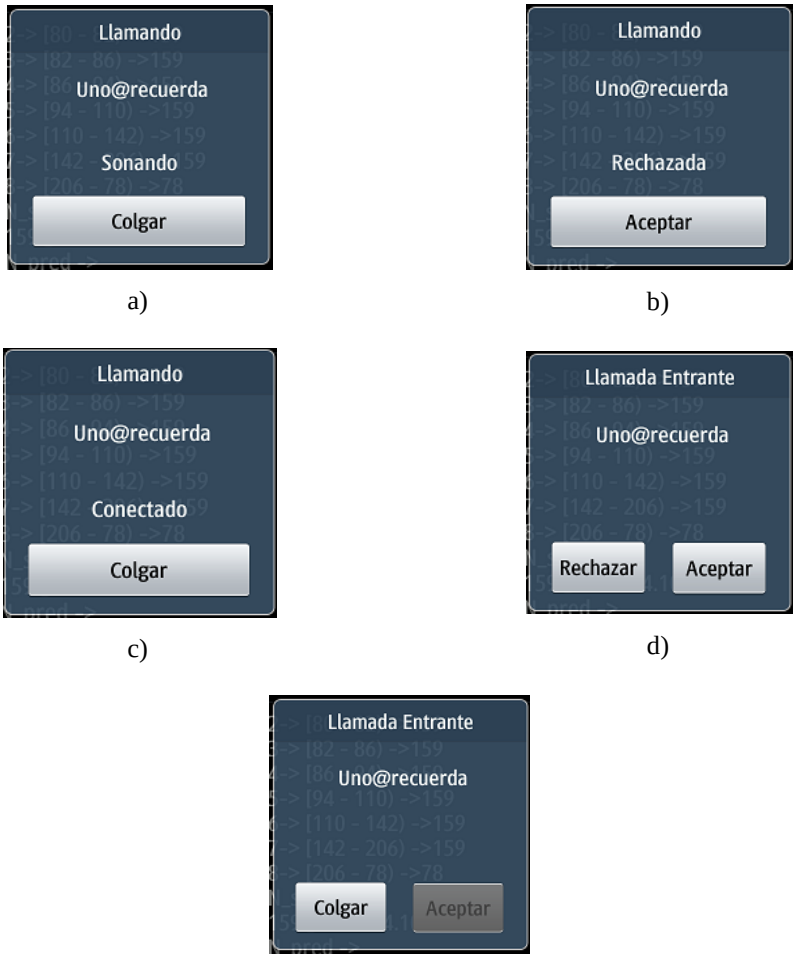


Figura A.4: Posibles ventanas de llamada

Figura A.5: Usuario no encontrado

Bibliografía

[1] comScore, "The comScore 2008 Digital Year in Review," Enero 2009.

[2] IHS. (2011, Agosto) isuppli. [Online]. http://www.isuppli.com/Mobile-and-Wireless-Communications/News/Pages/Smartphones-to-Account-for-Majority-of-Cellphone-Shipments-by-2015.aspx

[3] Hunjoo Jin. (2011) infobae.com. [Online]. http://america.infobae.com/notas/36742-Samsung-vendio-mas-smartphones-que-Apple

[4] Y-N. Ken. (2011, Noviembre) whatjapanthinks. [Online]. http://whatjapanthinks.com/2011/11/10/smartphones-voice-calls-and-voip-part-2-of-2/

[5] Universidad de Granada. (2005, Marzo) [Online]. http://www.ugr.es/informatica/redes/telefonia-ip.htm

[6] (2011, Marzo) joernesdohr. [Online]. http://www.joernesdohr.com/bada/bada-voip-applications-finally-allowed/

[7] ITU-T Recomendation, "H.323v.4: Packet-based multimedia communications systems," Noviembre 2000.

[8] H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, E. Schooler, J. Rosenberg, "SIP: Session Initiation Protocol," *RFC 3261, Internet Engineering Task Force*, Junio 2002.

[9] S. Casner, R. Frederick, V. Jacobson, H. Schulzrinne, "RTP: A Transport Protocol for Real-Time Applications," *RFC 1889, Internet Engineering Task Force*, Enero 1996.

[10] A. Rao, R. Lanphier, H. Schulzrinne, "Real Time Streaming Protocol (RTSP)," *RFC 2326, Internet Engineering Task Force*, Abril 1998.

[11] N. Greene, A. Rayhan, C. Huitema, B. Rosen, J. Segers, F. Cuervo, "Megaco Protocol Version 1.0," *RFC 3015, Internet Engineering Task Force*, Noviembre 2000.

[12] M. Handley, V. Jacobson, Jacobson, "SDP: Session Descripion Protocol," *RFC 2327, Internet Engineering Task Force*, Abril 1998.

[13] F. Yergeau, "UTF-8, a transformation format of ISO 10646," *RFC 2279, Internet Engineering Task Force*, Enero 1998.

[14] J. Gettys, J. Mogul, H. Frystyk, L. Masiner, P. Leach, T. Berners-Lee, R. Fielding, "Hypertext Transfer Protocol -- HTTP/1.1," *RFC 2616, Internet Engineering Task Force*, Junio 1999.

[15] Gonzalo Camarillo, *SIP Demystified*.: McGraw-Hill telecom., 2002.

[16] Shiao-Li Tsao, Jin-Chang, Chien-Ming, Chao-Cheng, "Unstructured Peer-to-Peer Session Initiation Protocol for Mobile Enviroment," in *Personal, Indoor and Mobile Radio Communications, 2007. PIMRC 2007. IEEE 18th International Symposium on* , Athens, Sept. 2007, pp. 1 - 5.

[17] R. Morris, D. Karger, F.KaaShoek ,H. Balakrishnan, I. Stoica, "Chord: A scalable peer-to-peer lookup services for internet applications," Agosto 2001.

[18] S. Casner, R. Frederick, V. Jacobson, H. Schulzrinne, "RTP: A Transport Protocol for Real-Time Applications," *RFC 3550, Internet Engineering Task Force*, Julio 2003.

[19] CISCO, "Voice Over IP − Per Call Bandwidth Consumption," vol. Document ID: 7934, 2006.

[20] K. Vos, S. Jensen, K. Sorensen, Skype Technologies S.A. , "SILK Speech Codec," *Internet-Draft*, Marzo 2010.

[21] J. Valin, A. Heggestad, A. Moizard, G. Herlein, "RTP Payload Format for Speex Codec," *RFC 5574, Internet Engineering Task Force*, Junio 2009.

[22] P. Jones, D. Eastlake, "US Secure Hash Algorithm 1," *RFC 3174, Internet Engineering Task Force*, Septiembre 2001.

[23] Ekiga. Ekiga. [Online]. www.ekiga.net

[24] S. Vaarala, H. Levkowetz, "Mobile IP Traversal of Network Address Translation (NAT) Devices," *RFC 3519, Internet Engineering Task Force*, Abril 2003.

[25] K. Singh, H. Schulzrinne, "SIPpeer: a session initiation protocol (SIP)-based peer-to-peer Internet telephony client adaptor," *White paper. Computer Science Department, Columbia University*, Enero 2005.

[26] D. J. Watts, *Six Degrees: The Science of a Connected Age*, W. W. Norton & Company, Ed., 2003.