ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

UNIVERSIDAD DE MÁLAGA

PROYECTO FIN DE CARRERA

DISEÑO E IMPLEMENTACIÓN DE UN TELÉFONO SOFTWARE DE VOZ SOBRE IP BASADO EN EL PROTOCOLO DE SESIÓN SIP

INGENIERÍA DE TELECOMUNICACIÓN

MÁLAGA, 2010 JUAN SALVADOR MOYA MARTÍNEZ

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

UNIVERSIDAD DE MÁLAGA

Titulación: Ingeniería de Telecomunicación

Reunido el tribunal examinador en el día de la fecha, constituido por:

D./Dª._____

D./Dª._____

D./Dª._____

para juzgar el Proyecto Fin de Carrera titulado:

DISEÑO E IMPLEMENTACIÓN DE UN TELÉFONO SOFTWARE DE VOZ SOBRE IP BASADO EN EL PROTOCOLO DE SESIÓN SIP

del alumno/a D./Dª. Juan Salvador Moya Martínez

dirigido por D./Dª. Francisco Javier González Cañete

ACORDÓ POR _____ OTORGAR LA

CALIFICACIÓN DE _____

Y, para que conste, se extiende firmada por los componentes del tribunal, la presente diligencia

Málaga, a _____ de _____ de _____

El/La Presidente/a El/La Vocal El/La Secretario/a

Fdo.: _____ Fdo.: _____ Fdo.: _____

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

UNIVERSIDAD DE MÁLAGA

TÍTULO DEL PFC

DISEÑO E IMPLEMENTACIÓN DE UN TELÉFONO SOFTWARE DE VOZ SOBRE IP BASADO EN EL PROTOCOLO DE SESIÓN SIP

REALIZADO POR:

Juan Salvador Moya Martínez

DIRIGIDO POR:

Francisco Javier González Cañete

DEPARTAMENTO DE: Tecnología Electrónica

TITULACIÓN: Ingeniería de Telecomunicación

PALABRAS CLAVE: SIP, RFC 3261, audio conferencia, telefonía

RESUMEN:

En el presente proyecto se ha diseñado una aplicación multimedia capaz de realizar una audio conferencia entre dos terminales telefónicos basados en el protocolo de señalización SIP (Session Initiation Protocol). Este protocolo está definido en el estandar RFC 3261 por el IETF (The Internet Engineering Task Force) . Se ha dotado a la aplicación de un algoritmo de codificación de audio definido en la recomendación G.729 de la ITU (International Telecommunication Union) que ofrece un ancho de banda bajo sin penalizar en exceso la calidad de la voz.

Málaga, Mayo de 2010

A mi abuela

ÍNDICE DE CONTENIDOS

ÍNDICE DECONTENIDOS i
LISTA DE FIGURAS iv
LISTA DE TABLAS v

CAPÍTULO 1: INTRODUCCIÓN 1
1.1. OBJETIVOS 1
1.2. MOTIVACIÓN 1
1.3. POSIBLES APLICACIONES 2
1.4. OBJETIVOS DE DISEÑO 3
1.5. ENTORNO DE TRABAJO 4
1.6. RESUMEN DE CAPÍTULOS 5

CAPÍTULO 2: PROTOCOLO SIP 7
2.1. INTRODUCCIÓN 7
2.2. ARQUITECTURA 7
2.2.1 PILA DE PROTOCOLOS 7
2.2.2 COMPONENTES DEL PROTOCOLO SIP 8
2.3 MENSAJES DEL PROTOCOLO SIP 10
2.3.1 FORMATO GENERAL DE UN MENSAJE 11
2.3.2 MÉTODOS 12
2.3.3 RESPUESTAS 12
2.3.3.1 CÓDIGOS DE RESPUESTA PROVISIONALES 13
2.3.3.2 CÓDIGOS DE RESPUESTA EN CASO DE ÉXITO 14
2.3.3.3 CÓDIGOS DE RESPUESTA POR REDIRECCIONES 14
2.3.3.4 CÓDIGOS DE RESPUESTA POR ERRORES DEL CLIENTE 15
2.3.3.5 CÓDIGOS DE RESPUESTA POR ERRORES DEL SERVIDOR . 16
2.3.3.6 CÓDIGOS DE RESPUESTA POR FALLOS GLOBALES 16
2.4 INTERCAMBIO DE MENSAJES SIP 16
2.4.1 REGISTRO DE UN TERMINAL EN UN SERVIDOR SIP 16
2.4.2 MENSAJES DE ESTABLECIMIENTO DE SESIÓN 18
2.4.3 FINALIZACIÓN DE UNA SESIÓN 20

CAPÍTULO 3: CODIFICACIÓN DE LA VOZ Y PROTOCOLOS DE TRANSMISIÓN …..... 21
3.1 INTRODUCCIÓN …..... 21
3.2 ALGORITMOS DE CODIFICACIÓN …..... 22
3.2.1 OBJETIVOS …..... 22
3.2.2 CODIFICACIÓN DE FORMA DE ONDA …..... 23
3.2.3 CODIFICACIÓN DE FUENTE …..... 24
3.2.3.1 G.729A …..... 24
3.2.3.2 G.723.1 …..... 25
3.2.3.3 GSM …..... 25
3.2.3.4 ILBC …..... 25
3.3 PROTOCOLO RTP …..... 25
3.3.1 FORMATO GENERAL DE UN PAQUETE RTP …..... 26
3.3.2 FORMATO GENERAL DE UN PAQUETE RTCP …..... 27
3.4 CONTROL DE CONGESTIÓN EN RTP …..... 28
3.4.1 LA NECESIDAD DE UN CONTROL DE CONGESTIÓN …..... 28
3.4.2 CONTROL DE CONGESTIÓN PARA DATOS MULTIMEDIA …..... 29
3.4.2.1 ALGORITMOS DE CONTROL DE FLUJO SIMILARES A TCP . 29
3.4.2.2 EMULACIÓN DE LA TRANSMISIÓN DE DATOS TCP …..... 30
3.4.3 SEGURIDAD …..... 34
3.4.3.1 PRIVACIDAD …..... 34
3.4.3.2 CONFIDENCIALIDAD …..... 35
3.4.3.3 AUTENTICACIÓN …..... 38
3.4.3.4 DENEGACIÓN DE SERVICIOS …..... 38
3.4.3.5 PROTECCIÓN A REPETICIONES …..... 39
3.4.3.6 CONTENIDO ACTIVO …..... 39

CAPÍTULO 4: IMPLEMENTACIÓN …..... 41
4.1 INTRODUCCIÓN …..... 41
4.2 LIBRERÍAS SIPXTAPI …..... 41
4.2.1 RESUMEN DE LAS PRINCIPALES FUNCIONES …..... 42
4.2.2 CODIFICACIÓN Y DECODIFICACIÓN …..... 45
4.3 RECOMENDACIÓN G.729 …..... 47
4.3.1 INTRODUCCIÓN …..... 47
4.3.2 IMPLEMENTACIÓN DEL ALGORITMO Y PRUEBAS …..... 48

4.3.3 GENERACIÓN DE LIBRERÍA …. 50
4.4 CLASES GENERADAS Y MODIFICACIÓN DE LA LIBRERÍA SIPXTAPI 50
4.5 PRUEBAS REALIZADAS …. 53
4.6 GENERACIÓN DE LIBRERÍA DE ENLACE DINÁMICO …. 55
4.7 ADAPTACIÓN DE SIPXTAPI A JAVA …. 56
4.7.1 EVOLUCIÓN HISTÓRICA DE JNI …. 57
4.7.2 NECESIDAD DE JNI …. 58
4.7.3 MÉTODOS DESEADOS Y CLASES GENERADAS …. 58
4.8 DISEÑO DE UN INTERFAZ GRÁFICO …. 62
4.8.1 CARACTERÍSTICAS DESEADAS …. 62
4.8.2 PARÁMETROS DE CONFIGURACIÓN …. 64
4.8.3 ESTRUCTURA DE CLASES …. 66
4.8.4 PRUEBAS REALIZADAS …. 69

CAPÍTULO 5: MANUAL DEL USUARIO …. 71
5.1 INTRODUCCIÓN …. 71
5.2 INSTALACIÓN DE LA APLICACIÓN …. 71
5.3 PANTALLAS Y CONFIGURACIÓN …. 72
5.3.1 PANTALLA PRINCIPAL …. 72
5.3.2 CONFIGURACIÓN …. 73
5.4 REQUERIMIENTOS BÁSICOS …. 75
5.5 POSIBLES CASOS DE USO …. 76

CAPÍTULO 6: CONCLUSIONES Y LÍNEAS FUTURAS …. 79
6.1 CONCLUSIONES …. 79
6.2 LÍNEAS FUTURAS …. 80

BIBLIOGRAFÍA Y REFERENCIAS …. 83

LISTA DE FIGURAS

Figura 1.1 Comunicación por voz sobre IP entre dos terminales …..... 1
Figura 1.2 Comparación de distintos codecs en función de su calidad …..... 4
Figura 2.1 Pila de protocolos de voz sobre IP …..... 8
Figura 2.2 Componentes del protocolo SIP …..... 10
Figura 2.3 Formato general de un mensaje SIP …..... 11
Figura 2.4 Registro de terminales SIP en un servidor SIP …..... 17
Figura 2.5 Secuencia exacta de mensajes de registro en un servidor SIP …..... 17
Figura 2.6 Establecimiento correcto de una conexión SIP …..... 18
Figura 2.7 Secuencia exacta de mensajes en el establecimiento de una sesión SIP ... 19
Figura 2.8 Secuencia exacta de mensajes para la finalización de una sesión SIP ….. 20
Figura 3.1 Comparación entre distintos algoritmos …..... 22
Figura 3.2 Codificación de una señal analógica en G.711 …..... 23
Figura 3.3 Formato general de un paquete RTP …..... 26
Figura 3.4 Formato general de un paquete RTCP …..... 27
Figura 3.5 Colapso por congestión …..... 29
Figura 3.6 Algoritmo slow-start …..... 33
Figura 3.7 Encriptación de un paquete RTP …..... 36
Figura 3.8 Modo transporte y modo túnel en IPSec …..... 37
Figura 4.1 Arquitectura global de la librería SipXTapi …..... 44
Figura 4.2 Arquitectura de librería SipXMedia …..... 46
Figura 4.3 Procesos de codificación y decodificación …..... 48
Figura 4.4 Esquema desarrollado para probar la decodificación G.729A …..... 49
Figura 4.5 Diagrama de clases del codificador G.729 …..... 51
Figura 4.6 Diagrama de clases del decodificador G.729 …..... 52
Figura 4.7 Soft-phone básico para realizar pruebas …..... 53
Figura 4.8 Interfaz bidireccional entre Java y sus librerías nativas …..... 56
Figura 4.9 Relación entre librerías C y Java …..... 58
Figura 4.10 Clase InterfazSIP de JNI …..... 60
Figura 4.11 Clase Procesos_InterfazSIP de JNI …..... 61
Figura 4.12 Interfaz gráfico …..... 63
Figura 4.13 Parámetros de configuración básicos …..... 65
Figura 4.14 Parámetros de configuración SIP …..... 66
Figura 4.15 Diagrama de clases …..... 68

Figura 5.1 Ventana principal …..... 72
Figura 5.2 Pantallas de configuración de la aplicación …..... 73
Figura 5.3 Comunicación entre dos terminales …..... 76
Figura 5.4 Entorno de centralita VoIP …..... 77

LISTA DE TABLAS

Tabla 2.1 Códigos de respuesta generales para el protocolo SIP …..... 13
Tabla 2.2 Códigos de respuesta provisionales …..... 13
Tabla 2.3 Códigos de respuesta en caso de éxito …..... 14
Tabla 2.4 Códigos de respuesta debidos a redirecciones …..... 14
Tabla 2.5 Códigos de respuesta por errores de cliente …..... 15
Tabla 2.6 Códigos de respuesta por errores en el servidor …..... 16
Tabla 2.7 Códigos de respuesta por errores de fallo global …..... 16
Tabla 4.1 Anexos de la recomendación G.729 …..... 47

CAPÍTULO 1: INTRODUCCIÓN

1.1 OBJETIVOS

Este proyecto tiene como objetivo el diseño de un teléfono software (*soft-phone*) capaz de transmitir audio usando el protocolo RTP (*Real Time Transport Protocol*) mediante una codificación de bajo ancho de banda y al mismo tiempo ofrecer una calidad en la transmisión de voz suficiente. También se pretende elaborar una biblioteca de funciones básicas telefónicas (llamar, colgar, responder, …) bajo la forma de una librería de enlace dinámico para que todo el desarrollo pueda aprovecharse en otras aplicaciones. Se demostrará el uso de esta librería diseñando dos interfaces gráficos distintos en los lenguajes de programación JAVA y C++. Se pretende montar un sistema como el de la figura 1.1.

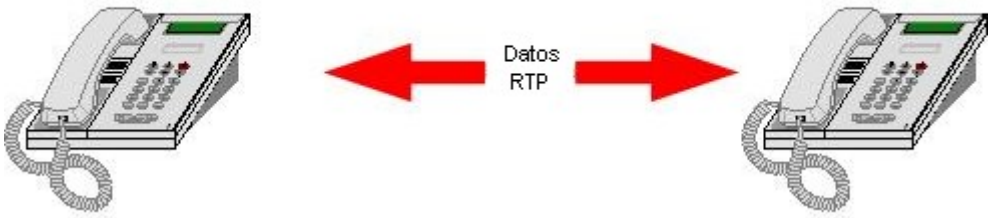

Figura 1.1 Comunicación por voz sobre IP entre dos terminales

Además, se hará un estudio comparativo de los algoritmos de codificación en cuanto al consumo de ancho de banda, coste computacional y calidad en la comunicación.

1.2 MOTIVACIÓN

En las últimas décadas la evolución de las comunicaciones telefónicas ha sido asombrosa, pasando desde enlaces puramente analógicos hasta llegar a enlaces totalmente digitales. Los enlaces de datos también han experimentado una notable evolución, llegando actualmente a velocidades y latencias hace unos años sólo alcanzables en redes muy costosas. Este aumento de calidad en las redes ha propiciado la aparición de nuevos servicios tales como la transmisión de audio y vídeo en tiempo real.

Las comunicaciones de voz a través de redes de datos han llegado a convertirse en un punto fundamental en el mundo empresarial, puesto que la reducción de costes y unificación de tecnologías que ello implica permiten un aprovechamiento más eficiente de los enlaces actualmente desplegados.

En el mercado han aparecido nuevos terminales telefónicos (*hard-phone*) que únicamente se conectan a redes de datos, de manera que ya no es preciso desplegar dos redes de telecomunicaciones distintas, tan sólo una red de datos dimensionada de manera que ofrezca unas comunicaciones de voz adecuadas. Gracias también a los avances en microelectrónica, se dispone de suficiente potencia en un ordenador de gama baja para simular un terminal telefónico mediante un programa informático, lo que se llama *soft-phone*. La unión de redes de datos de calidad basadas en el protocolo IP (*Internet Protocol*) junto con *hard-phones* y *soft-phones* da lugar a la telefonía IP.

Sin embargo, la conexión masiva de este tipo de terminales y *soft-phones* puede llegar a hacer que la calidad de las comunicaciones disminuya debido a la pérdida de paquetes, efecto colateral de una saturación de la red.. Por otra parte, tendremos entornos en los que el ancho de banda no sea un problema y sí lo sean los recursos computacionales de los equipos informáticos. Se hace necesario por tanto la investigación en el uso de algoritmos de codificación adecuados para que el ancho de banda utilizado por una llamada de voz sobre IP sea lo más pequeño posible sin menospreciar su calidad, así como que su consumo en recursos sea moderado o al menos esté bajo ciertos límites.

1.3 POSIBLES APLICACIONES

Las aplicaciones de este tipo de tecnología son muchas y muy variadas:

- Entornos de centro de atención y generación de llamadas, telemarketing.
- Llamadas de larga distancia a costes mínimos.
- Posible integración de todas las comunicaciones por voz en el software.
- Aplicaciones *Click2Call* en páginas Web en las cuales pueda realizarse una comunicación mediante un simple botón en una página Web.
- En entornos empresariales, posible unión de enlaces digitales externos a tantos teléfonos como se requiera mediante centralitas de voz sobre IP y digitales.

- En entornos educativos, comunicación directa entre alumnos y profesores desde casa.
- En entornos Web, comunicación directa entre clientes finales y empresas.

1.4 OBJETIVOS DE DISEÑO

Las aplicaciones multimedia utilizan un esquema de funcionamiento basado en protocolos. El protocolo de señalización es uno de los definidos por IETF (*The Internet Engineering Task Force*), el protocolo SIP (*Session Initiation Protocol*) definido en el RFC (*Request for Comments*) número 3261.

Podría haberse utilizado algún otro protocolo de señalización como el conocido H.323 definido por la ITU (*Internacional Telecommunication Union*). Es un protocolo maduro, robusto y completo, pero su ineficiencia en el uso de ciertos protocolos internos unida a la facilidad y masiva implantación del protocolo de señalización SIP han hecho que se tomara esta decisión de diseño. Se realizará una completa descripción del protocolo SIP en capitulos posteriores, así como un ejemplo de comunicación entre dos dispositivos examinando los paquetes intercambiados y el formato de los mismos.

Existen otros procolos de señalización alternativos tales como H.248 también conocido como MEGACO (*Media Gateway Control*), el protocolo IAX (*Inter-Asterisk eXchange protocol*) o el protocolo LTP (*Lightweight Telephony Protocol*) pero debido a su reducida implantación en los terminales IP y en las centralitas comerciales no se han considerado.

Dado que la implementación de codecs que usen un ancho de banda bajo es una prioridad, se ha elegido la implementación del codec G.729 debido a su habitual utilización en entornos de centralitas comerciales y teléfonos VoIP (Voz sobre IP) en el mercado. Se realizará una comparación con otros codecs que tienen otras características de ancho de banda utilizado y calidad con el fin de establecer los escenarios en los que deben elegirse unos y otros. Se utilizarán parámetros de calidad cualitativos, como el MOS (*Mean Opinión Score*) y cuantitativos como valores de ancho de banda de ocupación, número de tramas por paquete RTP o número de paquetes perdidos en caso de saturación de red. Un ejemplo ilustrativo podría verse en la figura 1.2.

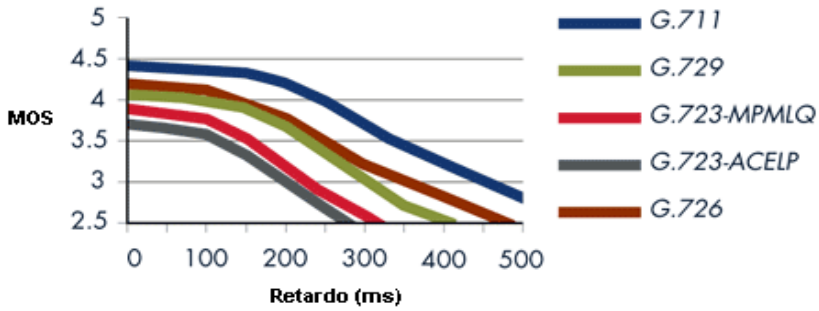

Figura 1.2 Comparación de distintos codecs en función de su calidad

1.5 ENTORNO DE TRABAJO

Durante la ejecución del proyecto se siguieron las siguientes fases para el desarrollo de cada una de las funcionalidades presentadas:

- Recolección de información y estudio de las alternativas disponibles. En esta primera fase se pretende llegar a un conocimiento del estado actual de la tecnología para evaluar las distintas posibilidades existentes. Se encontró una serie de proyectos englobados en el proyecto *SipX* de voz sobre IP en forma de librerías sobre las cuales se podían implementar la funcionalidad que se requiere. Estas librerías eran directamente compilables en C++.
- Diseño. Se deciden las estructuras de datos y algoritmos necesarios para la aplicación. Se diseñaron las clases que tenían que añadirse al proyecto *SipX* para poder integrar el codec G.729, al mismo tiempo que se localizaron las clases que serían necesarias modificar para que este codec fuera permitido por el sistema.
- Implementación. Se codifican las ideas esbozadas en el apartado anterior.
- Prueba y depuración de errores. Se verifica el correcto funcionamiento del software y se realizan los las pruebas necesarias para evaluar los distintos algoritmos de codificación de audio.

Los lenguajes de programación elegidos para la implementación han sido C++ y Java. El núcleo de la aplicación se desarrolló en C++, generando una librería de enlace dinámico con la que se pudo trabajar desde un entorno Java a través de la tecnología JNI (*Java*

Native Interface). En todo momento se ha hecho uso de técnicas tan extendidas como la programación orientada a objetos.

La plataforma de desarrollo ha sido un PC con el sistema operativo Windows XP con los entornos MS Visual Studio 2005, MS Visual C++ 6.0 y JBuilder 6.0. En primera instancia, se desarrolló sobre MS Visual Studio 2005 todo el proyecto hasta la creación de una librería de enlace dinámico, para a través de JNI usando MS Visual C++ 6.0 pasar a un entorno de programación Java con JBuilder 6.0.

1.6 RESUMEN DE CAPÍTULOS

En este primer capítulo se han detallado los objetivos del proyecto y situado en su contexto la aplicación a desarrollar, como un programa multimedia con el propósito de establecer, gestionar y finalizar correctamente una comunicación por voz sobre ip realizando la negociación de la comunicación utilizando el protocolo SIP. Se ha destacado la importancia del uso de algoritmos de codificación de voz adecuados al entorno en el que se vaya a operar, así como posibles problemas inherentes al protocolo de señalización que pueden surgir en determinadas circunstancias.

En el capítulo 2 se describe con detalle el protocolo de señalización SIP, sus principales características y los elementos que definen el protocolo. Se expondrán ejemplos de funcionamiento y una completa y detallada explicación de cada uno de los paquetes y sus principales campos. También se especificará que nivel ocupa el protocolo SIP dentro de la pila de protocolos y se comentarán sus desventajas.

En el capítulo 3 se explican los algoritmos de codificación de voz utilizados y otros protocolos que se utilizan para la transmisión de voz.

En el capítulo 4 se detalla cada una de las partes realizadas en el diseño: parte de integración de codecs en librerías y obtención de una librería de enlace dinámico, creación de otra librería dinámica mediante JNI para su con Java, y diseño de un interfaz gráfico con control de eventos desde Java usando la librería creada en el paso anterior.

En el capítulo 5 se describe la solución proporcionada a modo de manual.

Ya por último, el capítulo 6 expone las conclusiones obtenidas tras el desarrollo del proyecto y las posibles líneas futuras.

CAPÍTULO 2: PROTOCOLO SIP

2.1 INTRODUCCIÓN

El protocolo SIP fue desarrollado por el grupo MMUSIC (*Multiparty Multimedia Session Control*) del IETF, definiendo una arquitectura de señalización y control para voz sobre IP. Inicialmente fue publicado en febrero del 1996 en la RFC 2543, ahora obsoleta con la publicación de la nueva versión RFC 3261 [RFC3261'2002] que se publicó en junio del 2002.

El propósito de SIP es la comunicación entre dispositivos multimedia. SIP hace posible esta comunicación gracias a tres protocolos que son RTP, RTCP (*Real Time Control Protocol*) y SDP (Session *Description Protocol*).

Los protocolos RTP y RTCP se usan para transportar los datos de voz en tiempo real, mientras que el protocolo SDP se usa para la negociación de las capacidades de los participantes, tipo de codificación, puertos de transmisión, etc.

SIP fue diseñado de acuerdo al modelo de Internet. Es un protocolo de señalización extremo a extremo que implica que toda la lógica es almacenada en los dispositivos finales (salvo el rutado de los mensajes SIP). El estado de la conexión es también almacenado en los dispositivos finales. El precio a pagar por esta capacidad de distribución y su gran escalabilidad es una sobrecarga en la cabecera de los mensajes producto de tener que mandar toda la información entre los dispositivos finales.

SIP es un protocolo de señalización a nivel de aplicación para establecimiento y gestión de sesiones con múltiples participantes. Se basa en mensajes de petición y respuesta y reutiliza muchos conceptos de estándares anteriores como HTTP y SMTP.

2.2 ARQUITECTURA

2.2.1 PILA DE PROTOCOLOS

El protocolo SIP se encuentra integrado dentro del modelo OSI (*Open System Interconnection*) a nivel de aplicación y numerosos protocolos pueden coexistir con el tales como otros protocolos de señalización o protocolos para asegurar una determinada calidad

de servicio. En la figura 2.1 se puede comprobar la cantidad de tecnologías y protocolos que pueden usarse en cada nivel del modelo OSI.

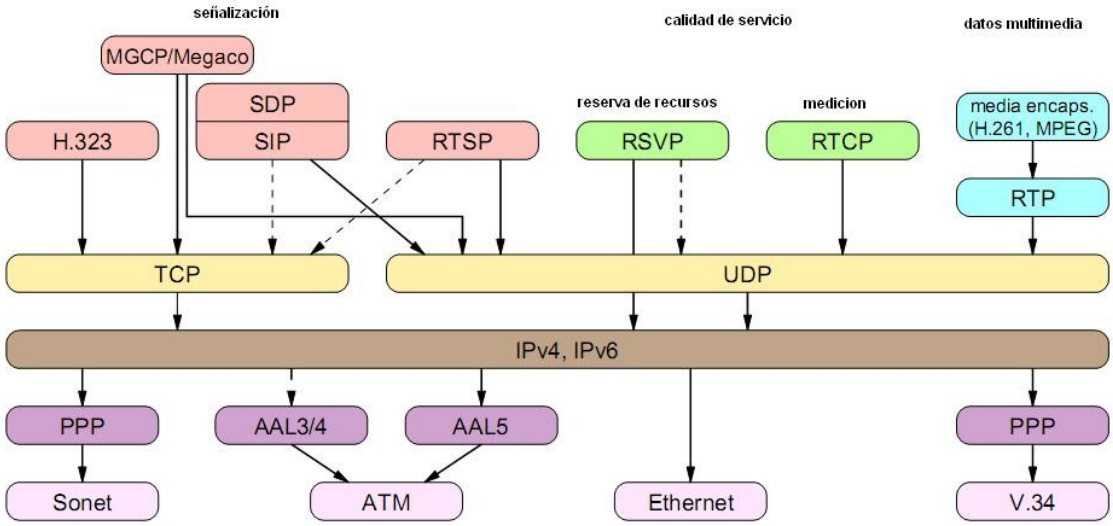

Figura 2.1 Pila de protocolos de voz sobre IP

En la figura 2.1 se puede observar que el protocolo SIP puede trabajar indistintamente con el protocolo de transporte TCP (*Transmission Control Protocol*) o UDP (*User Datagram Protocol*), así como se pueden utilizar distintas tecnologías de nivel físico tales como ATM (*Asynchronous Transfer Mode*), redes Ethernet (estándar estándar IEEE 802.3) o redes SONET (*Synchronous Optical Network*).

2.2.2 COMPONENTES DEL PROTOCOLO SIP

El protocolo SIP define una serie de entidades lógicas para llevar a cabo su función, denominan componentes del protocolo SIP. En ese sentido, se debe distinguir primero la diferencia entre cliente y servidor:

- Un cliente será cualquier elemento que envíe peticiones SIP y reciba respuestas SIP. Más adelante se verán con detalle los mensajes de petición y respuesta. Los clientes pueden o no actuar directamente con un usuario humano.
- Un servidor es un elemento de red que recibe peticiones para darles servicio y enviar de vuelta respuestas a esas peticiones. Ejemplos de servidores son los

proxy, los servidores de agentes de usuario, servidores de redirección y proxy de registro.

Los elementos SIP pueden actuar como cliente, como servidor o como ambos. Éstos elementos se definen como:

- Un agente de usuario o UA (*User Agent*) es una entidad lógica que puede actuar como cliente de agente de usuario y servidor de agente de usuario.
- El cliente de agente de usuario o UAC (*User Agent Client*) es una entidad lógica que crea nuevas peticiones y las envía empleando el mecanismo de transacciones que se describirá en el punto 2.4. El papel de UAC dura el tiempo que se está realizando dicha transacción. Esto significa que si este agente de usuario es el que ha enviado la petición como cliente, más tarde recibe una petición. En ese momento adopta el papel de servidor de agente de usuario durante el procesamiento de esa petición.
- El servidor de agente de usuario o UAS (*User Agent Server*) es una entidad lógica que genera una respuesta a una petición SIP. Esta respuesta puede aceptar, rechazar o redireccionar la petición.
- Proxy o servidor proxy es una entidad intermedia que actúa como servidor, y que puede actuar como cliente para el propósito de realizar una petición en nombre de otros clientes, actuando como representante. Un servidor proxy principalmente realiza el papel de enrutamiento, lo que significa que su labor es asegurarse de que una petición recibida sea enviada a otra entidad más cercana al usuario destino.
- Un servidor de redirección es un servidor de agente de usuario que genera respuestas de redirección (respuestas 3xx, como se verá más adelante) a peticiones recibidas, dirigiendo al cliente a contactar con direcciones alternativas.
- Un proxy de registro (*registrar*) es un servidor que acepta peticiones de registro, y almacena la información que recibe en esas peticiones en el servicio de localización para el dominio que dirige.

Estos elementos, y algunos más, son entidades lógicas que define el protocolo SIP, por lo que en una implementación, muchas de ellas pueden encontrarse en una misma aplicación. En la figura 2.2 se representa un esquema de los elementos anteriormente citados y de las relaciones existentes entre los mismos.

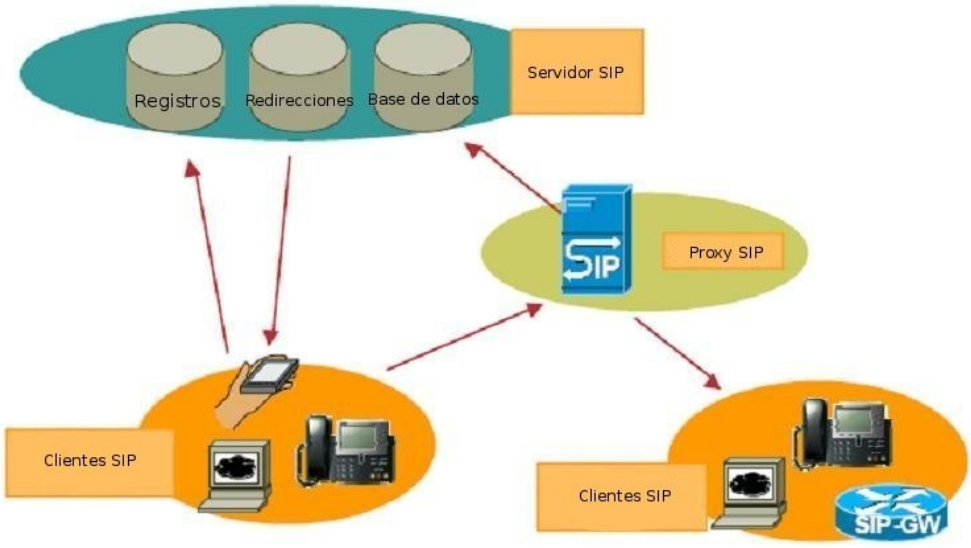

Figura 2.2 Componentes del protocolo SIP

2.3 MENSAJES DEL PROTOCOLO SIP

El protocolo SIP es un protocolo en el que se produce un intercambio de mensajes en texto plano con una estructura similar a HTTP (*HyperText Transfer Protocol*) a través del puerto 5060 sobre TCP o UDP. Los mensajes del protocolo permiten que se pueda transmitir cualquier tipo de dato, y son fácilmente ampliables para satisfacer cualquier necesidad o servicio.

Existen dos tipos de mensajes en el protocolo SIP:

- Mensajes de petición. Son mensajes que el llamante envía al llamado.
- Mensajes de respuesta. Son los mensajes que el llamado envía tras la petición del llamante.

A continuación se mostrarán uno por uno los mensajes que forman parte del establecimiento de la sesión.

2.3.1 FORMATO GENERAL DE UN MENSAJE

El formato general de un mensaje SIP puede verse en la figura 2.3.

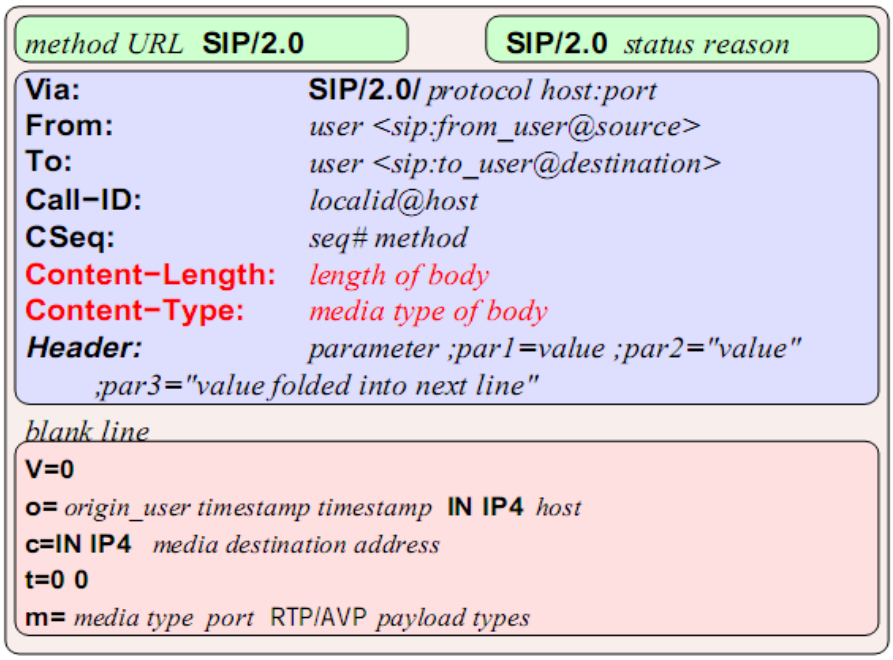

Figura 2.3 Formato general de un mensaje SIP

En la figura 2.3 pueden verse tres filas bien diferenciadas. La primera de ellas indica si el mensaje es una petición que se realiza o una respuesta a una petición ya realizada anteriormente, donde *method* es uno de los métodos SIP que se describirán en el punto 2.3.2 y *status* es el código de respuesta en formato XXX ante una petición.

La segunda fila corresponde al protocolo SIP en sí mismo. Es en ese lugar donde se indican hacia qué usuario va dirigida la llamada (campo *To*), quién la origina (campo *From*), la dirección IP y el puerto a la que se ha de transmitir el paquete de vuelta (campo *Via*) y demás parámetros como un identificador único de la sesión y contenidos del mensaje. El campo *Via* es de capital importancia ante uno de los más graves problemas del protocolo SIP: el problema del NAT (*Network Address Translation*).

La tercera fila corresponde a la cabecera SDP (*Session Description Protocol*). El protocolo SDP proporciona un formato adecuado para el intercambio de parámetros en la

comunicación de los datos en sí mismos. Particularizado al protocolo SIP, transporta sobre qué puertos se establecerá el protocolo RTP, los codecs utilizados en la comunicación, las direcciones IP que intervienen en la misma y el tipo de información que lleve (audio o vídeo).

2.3.2 MÉTODOS

Los distintos métodos que componen el protocolo SIP son los siguientes:

- Método INVITE. Es el método que sirve para el inicio del establecimiento de una sesión SIP. Lleva información del origen de la comunicación, del destino y de todas las capacidades que tiene el origen para poder establecer la comunicación
- Método REGISTER. Sirve para notificar al servidor SIP la posibilidad de realizar y recibir establecimientos de sesión por parte del terminal que envía este método.
- Método ACK. Realiza una función de reconocimiento (*acknowledge*) de mensajes para que el origen sepa que se ha recibido su petición.
- Método BYE. Es usado cuando se desea finalizar una sesión.
- Método OPTIONS. Utilizado cuando se requiere algún tipo de información, se suele usar una vez que ya se ha establecido correctamente la sesión.
- Método CANCEL. Realiza una cancelación del establecimiento de la sesión.
- Método INFO. Se usa cuando se quieren transmitir pulsaciones de teclado en forma de tonos DTMF (*dual tone multifrequency*) al destino.

2.3.3 RESPUESTAS

Los códigos de respuesta son números de tres cifras, cuyo dígito más importante es el primero. Dependiendo del primer dígito la respuesta será de un tipo o de otro según las opciones de la tabla 2.1. Más adelante, en las tablas 2.2 hasta la 2.7 se especificará más claramente todas las respuestas posibles para el protocolo SIP.

Esta nomenclatura es heredada del protocolo de Internet HTTP (*Hypertext Transfer Protocol*) para visualización de paginas web. De forma conceptual los códigos de respuesta con similares en cuanto al primer dígito se refiere. Esta manera de comunicar los sucesos

que ocurren en la parte servidora es ampliamente utilizada en otros protocolos, tales como el correo electrónico POP3 (*Post Office Protocol version 3*) y SMTP (*Simple Mail Transfer Protocol*).

Código	Razón	Explicación
1xx	Provisional (*provisional*)	Petición recibida, se continua con el proceso de la petición
2xx	Éxito (*sucess*)	La acción fue satisfactoriamente recibida, entendida y aceptada
3xx	Redirección (*redirection*)	Se necesitan realizar otras acciones para poder completar la petición
4xx	Error de cliente (*client error*)	La petición contiene una sintaxis incorrecta o no se puede desempeñar en este servidor
5xx	Error de servidor (*server error*)	El servidor falló al intentar cumplir una petición aparentemente válida
6xx	Fallo global (*global failure*)	La petición no se puede llevar a cabo en ningún servidor

Tabla 2.1 Códigos de respuesta generales para el protocolo SIP

2.3.3.1 Códigos de respuesta provisionales

Código	Texto asociado
100	Trying
180	Ringing
181	Call is being forwarded
182	Queued
183	Session progress

Tabla 2.2 Códigos de respuesta provisionales

El código "100 *Trying*" se devuelve cuando tras realizar una petición el servidor busca en sus registros al usuario con el que se quiere establecer la comunicación. Una vez que se ha localizado y se ha comprobado que el destino está disponible se transmite el código "180 *Ringing*" para indicar al llamante que el destino está siendo avisado. Si el destino tuviera activa una redirección de llamada automática a otro destino se devolvería el código

"191 *Call is being forwarded*", mientras que si tiene activada una cola de llamadas se transmitiría el código "182 *Queued*". El código "183 *Session progress*" se utiliza en el caso en que se quiera transmitir al origen de la comunicación que comience a transmitir los datos antes de que la comunicación se haya establecido en su totalidad. Esta modalidad del protocolo SIP se utiliza en en entornos donde la velocidad sea importante.

2.3.3.2 Códigos de respuesta en caso de éxito

Código	Texto asociado
200	OK

Tabla 2.3 Códigos de respuesta en caso de éxito

2.3.3.3 Códigos de respuesta por redirecciones

Código	Texto asociado
300	Multiple Choices
301	Moved Permanently
302	Moved Temporarily
305	Use Proxy
380	Alternative Service

Tabla 2.4 Códigos de respuesta debidos a redirecciones

Los códigos de respuesta debidos a redirecciones se utilizan en el caso en el que el inicio de sesión no sea suficiente para completar la petición, sino que se han de realizar otras acciones por parte del llamante. Dichas acciones consisten en realizar otro inicio de sesión SIP a otro destino que se indica en uno de los campos del paquete de datos de respuesta. El código de respuesta "300 *Multiple choices*" indica al llamante que tiene múltiples destinos que se indican en el paquete de respuesta. El código "301 *Moved Permanently*" proporciona al cliente datos para realizar un nuevo inicio de sesión en otro destino, puesto que el usuario al que llama ya no se encuentra en esa localización. De manera similar, el código "302 *Moved temporarily*" indica al llamante que el usuario temporalmente no está en esa localización, pero sí puede localizarlo en la que se

proporciona. El código "305 *Use proxy*" tiene como objetivo proporcionarle al llamante la dirección de un proxy SIP en el que podrá localizar al destino inicialmente requerido. Por último, el código "380 *Alternative service*" trata de indicar al usuario que el servicio que requiere no está actualmente disponible, pero le proporciona servicios similares que pueden interesarle.

Es responsabilidad del llamante tomar las decisiones que considere oportunas tras la recepción de estos códigos de respuesta.

2.3.3.4 Códigos de respuesta por errores de cliente

Código	Texto asociado
400	Bad Request
401	Unauthorized
402	Payment Required
403	Forbidden
404	Not Found
405	Method Not Allowed
406	Not Acceptable
407	Proxy Authentication Required
408	Request Timeout
410	Gone
413	Request Entity Too Large
414	Request-URI Too Large
415	Unsupported Media Type
416	Unsupported URI Schema
420	Bad Extension
421	Extension Required
423	Interval Too Brief
480	Temporarily not available
481	Call Leg/Transaction Does Not Exist
482	Loop Detected
483	Too Many Hops
484	Address Incomplete
485	Ambiguous
486	Busy Here
487	Request Terminated
488	Not Acceptable Here
491	Request Pending
493	Undecipherable

Tabla 2.5 Códigos de respuesta por errores de cliente

2.3.3.5 Códigos de respuesta por errores en el servidor

Código	Texto asociado
500	Internal Server Error
501	Not Implemented
502	Bad Gateway
503	Service Unavailable
504	Server Time-out
505	SIP Version not supported
513	Message Too Large

Tabla 2.6 Códigos de respuesta por errores en el servidor

2.3.3.6 Códigos de respuesta por fallos globales

Código	Texto asociado
600	Busy Everywhere
603	Decline
604	Does not exist anywhere
606	Not Acceptable

Tabla 2.7 Códigos de respuesta por errores de fallo global

2.4 INTERCAMBIO DE MENSAJES SIP

2.4.1 REGISTRO DE UN TERMINAL EN UN SERVIDOR SIP

El registro de los terminales en un servidor SIP es un paso previo e indispensable para dos terminales conectados al mismo servidor y no directamente. Es conveniente precisar que este paso no sería necesario si no existiera la presencia de un servidor SIP en la topología de la red.

El registro de terminales se realiza a través del mensaje REGISTER tal y como se indica en la figura 2.4.

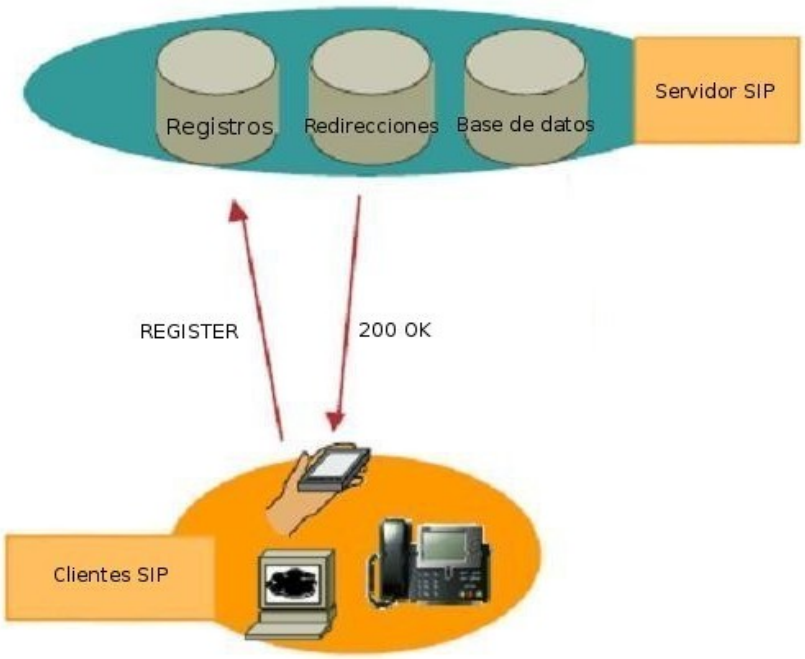

Figura 2.4 Registro de terminales SIP en un servidor SIP

Como se puede ver en la figura 2.4, el registro consiste únicamente en el envío y recepción de un par mensaje-respuesta. La secuencia exacta de mensajes se puede observar en la figura 2.5.

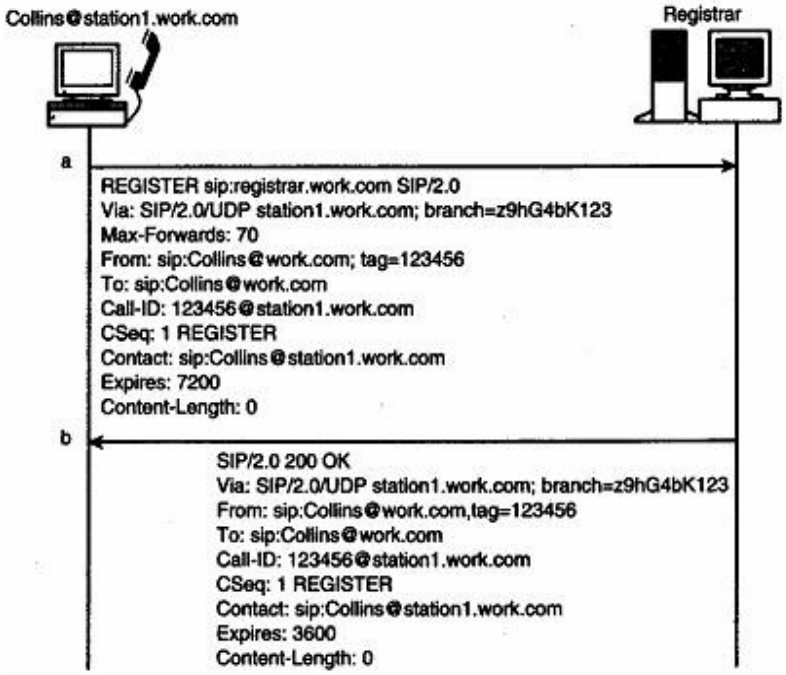

Figura 2.5 Secuencia exacta de mensajes de registro en un servidor SIP

Se pueden ver claramente en los mensajes todos los campos definidos en el apartado 2.3.1, remarcando aquellos más importantes como el campo *From*, *To* y *Via*.

2.4.2 MENSAJES DE ESTABLECIMIENTO DE SESIÓN

El establecimiento de una sesión SIP conlleva una sucesión determinada de los mensajes y respuesta anteriormente expuestos. En este apartado se desarrollará un ejemplo gráfico de cómo dichos mensajes y respuestas van evolucionando a lo largo del tiempo hasta llegar a un completo establecimiento de la conexión.

En la figura 2.6 puede observarse el establecimiento de una conexión.

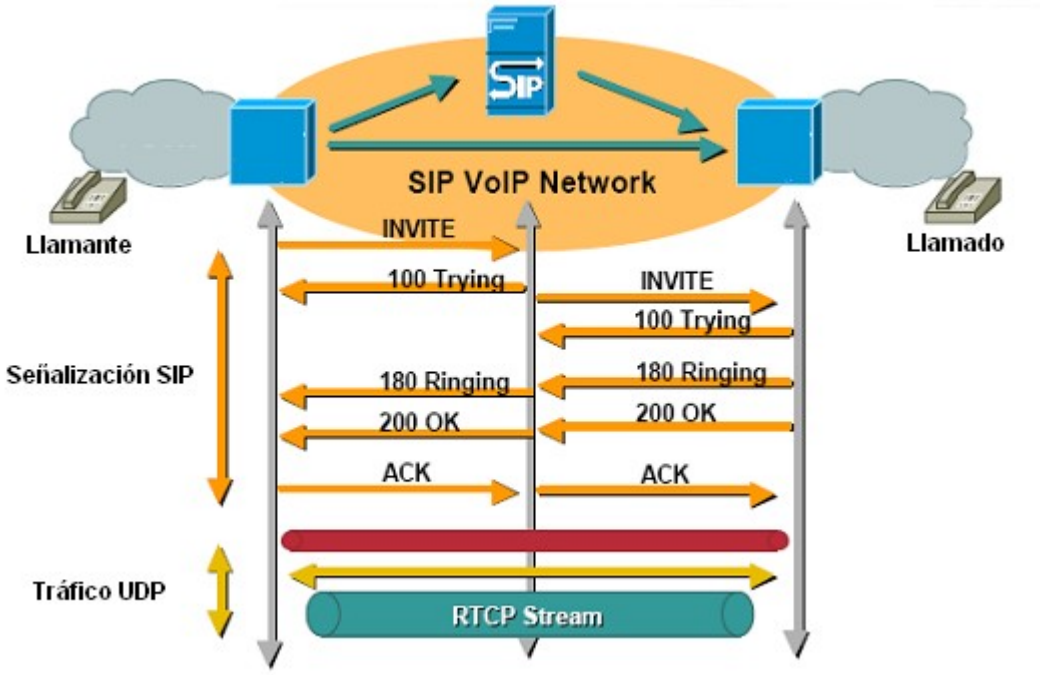

Figura 2.6 Establecimiento correcto de una conexión SIP

Tal y como se observa en la figura 2.6, se inicia el establecimiento con un mensaje INVITE que es respondido inmediatamente con una respuesta "180 *Trying*" por parte del servidor SIP. Se suponen ambos terminales registrados previamente con un mensaje de tipo REGISTER, ante lo cual el servidor SIP sabe que ambos terminales están operativos y listos para el establecimiento de sesiones. Tras la llegada del mensaje INVITE al servidor SIP, este consulta si el terminal llamado esta operativo, en cuyo caso redirecciona el mensaje INVITE del terminal llamante. Tras obtener una respuesta positiva, redirecciona al origen de la sesión la respuesta "180 *Ringing*" para informar de que el destino ha sido

altertado de una comunicación entrante. Tras aceptar la comunicación, el destino envía la respuesta "200 *OK*" al origen para indicarle que ya puede enviar tráfico RTP/RTCP con los datos que quiera comunicar.

El tráfico exacto de mensajes puede verse en la figura 2.7.

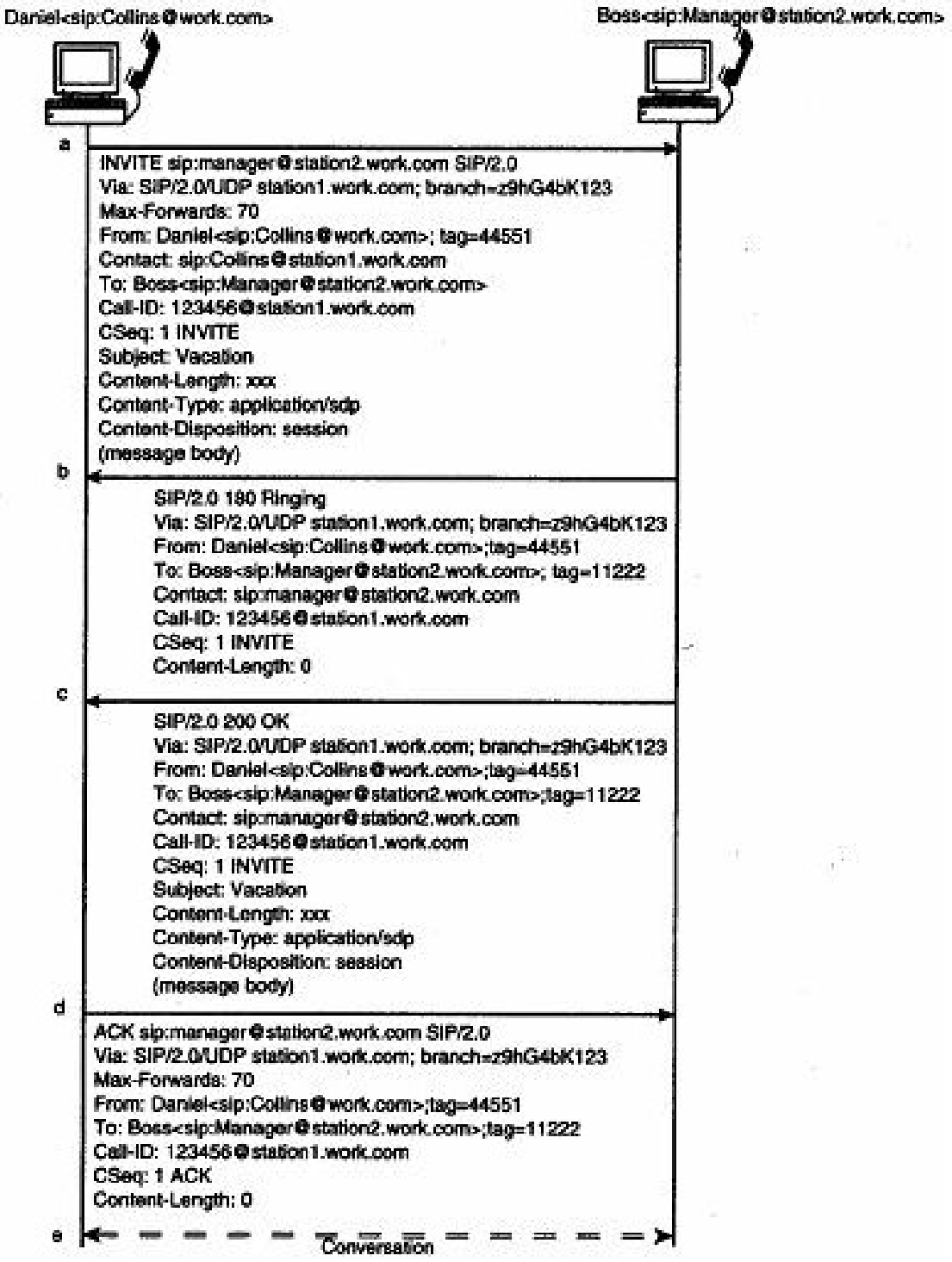

Figura 2.7 Secuencia exacta de mensajes en el establecimiento de una sesión SIP

2.4.3 FINALIZACIÓN DE UNA SESIÓN

El fin de una sesión se produce mediante el envío por cualquiera de los participantes de una sesión de un mensaje BYE. El diagrama de mensajes puede observarse en la figura 2.8.

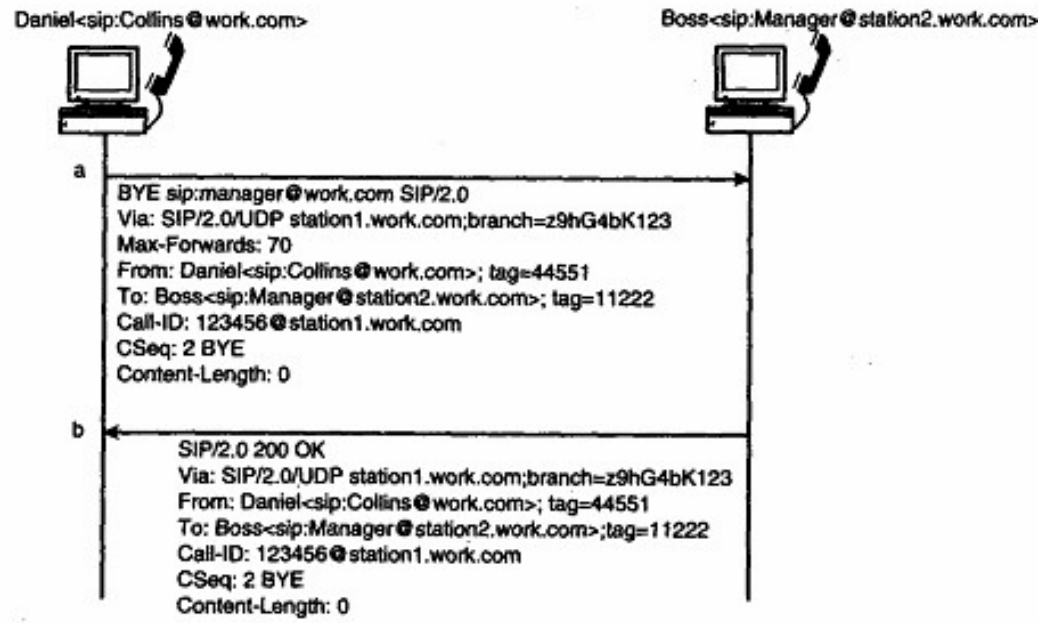

Figura 2.8 Secuencia exacta de mensajes para la finalización de una sesión SIP

CAPÍTULO 3: CODIFICACIÓN DE LA VOZ Y PROTOCOLOS DE TRANSMISIÓN

3.1 INTRODUCCIÓN

El proceso de codificación de voz permite transmitir y almacenar la señal de voz en forma digital eficientemente. Desde el punto de vista de la transmisión de la señal de voz, la codificación permite optimizar la utilización del canal de comunicación transmitiendo el máximo de información, por ejemplo para transmitir varias comunicaciones por un solo canal, con la mínima pérdida de calidad optimizando la relación entre velocidad de transmisión (bits por segundo) e inteligibilidad del mensaje. Desde el punto de vista de almacenar señal de voz en formato digital, la codificación de voz permite minimizar el número de bits necesarios para el almacenamiento manteniendo un nivel de calidad adecuado. En este contexto se definen diversos algoritmos de codificación de voz tales como GSM (utilizado en telefonía móvil), G.729, G.723, G.711, etc.

La proliferación de equipos, sumada a la disponibilidad de hardware de audio/video económicos y la posibilidad de contar con velocidades de conexión cada vez más rápidas, ha aumentado el interés en el uso de Internet para enviar audio y video, tipos de datos que tradicionalmente se reservaban para redes especializadas. Durante los últimos años, las audioconferencias y las videoconferencias se han convertido en una práctica común. Sin embargo, la misma naturaleza de Internet indica que esta red no está preparada para la transmisión de datos en tiempo real y, por consiguiente, la calidad del audio transmitido por Internet generalmente tiene una calidad mediocre. Los protocolos de transmisión de datos multimedia específicamente tratan el análisis y la solución de estos problemas para permitirle a una llamada o aplicación telefónica por Internet cambiar su funcionamiento para mantener una calidad auditiva aceptable, incluso en los casos en los que la red esté algo congestionada. Se describirá el protocolo RTP (*Real-Time Transport Protocol*) al ser el protocolo más utilizado en la actualidad para voz sobre IP.

3.2 ALGORITMOS DE CODIFICACIÓN

3.2.1 OBJETIVOS

El proceso de conversión de ondas analógicas a información digital se realiza con un codificador-decodificador (el CODEC). Los algoritmos de codificación persiguen el objetivo de digitalizar la voz recogida de un dispositivo de audio de manera eficiente en cuanto a ancho de banda utilizado y coste computacional. Existe por tanto un compromiso entre estas dos variables que cada uno de los algoritmos matemáticos que se pueden implementar resuelve de manera distinta.

Si se realiza una comparación con el ancho de banda que ocupa una conversación efectuada de manera analógica, la conversión digital ofrece un menor ancho de banda. Una transmisión analógica convencional ocupa una canal de 64 Kbps, mientras que una transmisión digital de la misma mediante un algoritmo de compresión puede llegar a ocupar perfectamente 8 Kbps. Del mismo modo, la calidad de la señal de salida que se obtiene al realizar esta compresión de audio disminuye conforme la compresión aumenta. Para medir la calidad se utiliza un parámetro subjetivo llamado MOS (*Mean Opinion Score*) que evalúa la calidad del algoritmo de codificación desde el valor uno (peor calidad) hasta el valor 5 (mejor calidad). Se basa en recoger las opiniones de un grupo de pruebas cambiando el algoritmo de la conversación variando a su vez distintos parámetros como el número de paquetes perdidos o el tanto por ciento de saturación en la red. En la figura 3.1 se puede ver un ejemplo donde se comparan tres algoritmos distintos que más tarde se explicarán.

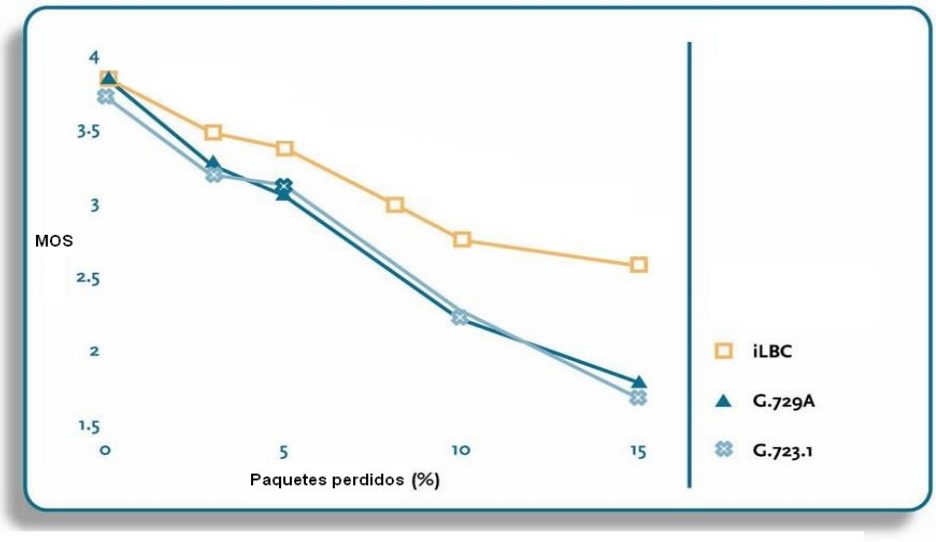

Figura 3.1 Comparación entre distintos algoritmos

Los algoritmos pueden dividirse en tres grandes grupos: algoritmos de codificación de forma de onda, algoritmos de codificación de fuente y algoritmos híbridos. El primer grupo de ellos reune los algoritmos con mayor calidad y bajo coste computacional pero con un índice de compresión nulo. El tercer grupo obtiene mejores índices de compresión a costa de disminuir la calidad y unos mayores recursos computacionales. El segundo grupo se ubica como una solución de compromiso en la que el ancho de banda ocupado y los recursos utilizados son moderados y la calidad de la señal es buena. Dependiendo del entorno deberían usarse unos tipos de algoritmos u otros.

3.2.2 CODIFICACIÓN DE FORMA DE ONDA

El algoritmo que identifica a este grupo es el recomendado por la ITU-T como G.711, también llamado PCM (*Pulse Code Modulation*). Este estándar es usado principalmente en telefonía, y fue liberado para su uso en el año 1972. G.711 es un estándar para representar señales de audio con frecuencias de la voz humana, mediante muestras comprimidas de una señal de audio digital con una tasa de muestreo de 8000 muestras por segundo. El codificador G.711 proporciona un flujo de datos de 64 Kbps. Para este estándar existen dos algoritmos principales, el μ-law (usado en Norte América y Japón) y el A-law (usado en Europa y el resto del mundo). Estos algoritmos utilizan un sistema de mapeo logarítmico que enfatiza las partes de la señal a las que el oído humano es más sensible. A efectos prácticos, el mapeo logarítmico se traduce en que valores pequeños de señal se codifican con un mayor número de bits mientras que valores altos de señal se codifican con un menos número de bits tal y como se puede observar en la figura 3.2.

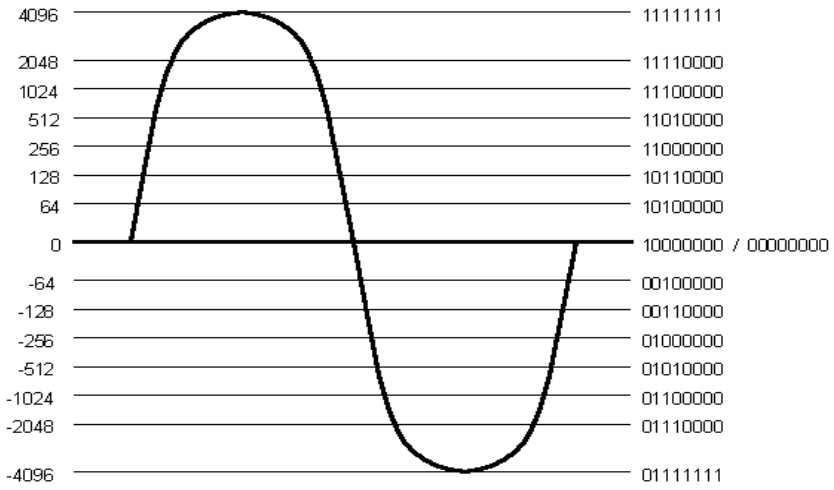

Figura 3.2 Codificación de una señal analógica en G.711

Utilizado para flujos de voz sobre IP, es el algoritmo que mejor calidad ofrece y más ancho de banda ocupa. En la actualidad, la mayoría de los proveedores de voz sobre IP ofrecen la posibilidad de operar con este algoritmo.

Existen más algoritmos que pertenecen a este grupo, tales como

- DPCM (*Differential Pulse Code Modulation*)

  Utiliza predictores de manera interna para sólo codificar la diferencia entre la muestra actual y la anterior.
- *Adaptive* DPCM

  Codifica la diferencia entre la muestra actual y la anterior, pero tanto los predictores como los cuantificadores internos son capaces de adaptarse a la señal de entrada que haya que codificar.
- Técnicas de codificación en el dominio de la frecuencia tales como SBC (*Sub-Band Coding*), que dividen la señal a codificar en sub-bandas de frecuencia y las codifica con algún algoritmo de los mencionados anteriormente. Tiene como ventaja que codifica con más bits la parte mas representativa de la señal en cuanto a calidad se refiere, pero a costa de aumentar la complejidad del proceso de codificación.

3.2.3 CODIFICACIÓN DE FUENTE

Los algoritmos de codificación de fuente se basan en examinar la señal de voz. La idea principal se basa en comprender los mecanismos que producen la señal de voz, modelando los distintos sonidos como los coeficientes de un filtro variante en el tiempo que, alimentado con una fuente de ruido blanco, produce la señal de voz deseada. Estos coeficientes del filtro, entre otros parámetros, es lo que realmente se envía al decodificador.

3.2.3.1 G.729A

Proporciona una respetable calidad de voz a una tasa de transferencia baja (8 Kbps). La codificación G.729A fue diseñada para aplicaciones que precisan un bajo retardo (el retardo algorítmico es de 15 ms). Incorpora mecanismos de recuperación de tramas perdidas, lo que le hace útil cuando la red empleada no ofrece una calidad de servicio garantizada como ocurre por ejemplo en Internet. La calidad de salida del audio ha resultado muy satisfactoria, sobre todo cuando se emplea en entornos ruidosos.

3.2.3.2 G.723.1

Este algoritmo se incluye dentro de los que requieren un ancho de banda muy pequeño, operando a 5.3 Kbps. El codificador G.723.1 está concebido para aplicaciones de video telefonía a bajas tasas binarias. Para este tipo de aplicaciones los requisitos de retardo no son excesivamente críticos, porque de todos modos el retardo del codificador de vídeo es superior al de voz. Así, el retardo algorítmico del codificador es de 37.5 ms. Al igual que el algoritmo G.729A, implementa técnicas de recuperación de tramas perdidas y ofrece una calidad de audio aceptable.

3.2.3.3 GSM (*Global System for Mobile telecomunications*)

Es el codec por excelencia utilizado en telefonía móvil. Ofrece una implementación muy sencilla con una tasa de transferencia adecuada (13.3 Kbps en modo full rate, la mitad en modo half rate). Su principal ventaja reside en su bajo coste computacional (es capaz de codificar en tiempo real en un procesador Intel 486).

3.2.3.4 ILBC (*Internet Low Bit rate Codec*)

Ofrece una alta calidad de voz para el ancho de banda ocupado de 13.3 Kbps, llegando incluso a compararse con la calidad ofrecida por G.711. Se caracteriza por ser un algoritmo de codificación libre de derechos de uso definido en RFC 3951. Actualmente se usa con frecuencia en sistemas de mensajería instantánea y cada vez con más frecuencia se implementa en teléfonos VoIP.

3.3 PROTOCOLO RTP

RTP es el protocolo que se encarga de transportar la voz propiamente dicha. Existe cierta confusión que lleva a pensar que ese es el trabajo de SIP, pero no es así. Una vez que SIP establece una llamada, es RTP quien toma el control para transportar la voz a su destino.

RTP trabaja sobre UDP y por lo tanto no hay mucho control de transmisión. El equipo emisor envía la voz hacia el otro extremo con la esperanza de que llegue, pero no espera recibir confirmación de esto y a decir verdad tampoco hay tiempo para hacerlo pues la voz necesita ser transmitida en tiempo real. Si un paquete de voz se pierde en el camino simplemente se rellenará ese espacio con un silencio, lo que técnicamente se llama ruido

confortable (*comfort noise*). Es por esta necesidad de transmitir la información en tiempo real que resulta obvio que RTP sea un acrónimo de *Real Time Protocol*. A pesar de encargarse de casi toda la labor de transportar la voz, RTP no está sólo y tiene un protocolo de apoyo llamado RTCP. RTCP no es del todo indispensable pero proporciona valiosa ayuda al momento de transportar la voz de manera óptima pues proporciona estadísticas e información de control que le permiten a un servidor SIP tomar decisiones para mejorar la transmisión en caso de ser posible. Los paquetes RTCP se transmiten periódicamente para comunicar dicha información a los equipos de voz involucrados.

3.3.1 FORMATO GENERAL DE UN PAQUETE RTP

El formato general de un mensaje RTP puede verse en la figura 3.3

<table>
<tr><td colspan="6">Byte 1</td><td colspan="2">Byte 2</td><td colspan="2">Byte 3</td><td>Byte 4</td></tr>
<tr><td>V=2</td><td colspan="2">P</td><td>X</td><td>CC</td><td>M</td><td colspan="2">PT</td><td colspan="3">Número de secuencia</td></tr>
<tr><td colspan="11">Marca de tiempo</td></tr>
<tr><td colspan="11">Identificación de la fuente de sincronización (SSRC)</td></tr>
<tr><td colspan="11">Identificación de la fuente de contribución (CSRC)</td></tr>
<tr><td colspan="11">Extensión (opcional, dependiendo del bit X)</td></tr>
<tr><td colspan="11">Datos</td></tr>
</table>

Figura 3.3 Formato general de un paquete RTP

A continuación se indican los significados de los diferentes campos de encabezados:

- Campo de versión V: 2 bits de longitud. Indica la versión del protocolo (V=2).
- Campo de relleno P: 1 bit. Si P es igual a 1, el paquete contiene bytes adicionales para rellenar y finalizar el último paquete.
- Campo de extensión X: 1 bit. Si X = 1, el encabezado está seguido de un paquete de extensión.
- Campo de cuenta de campos CRSC CC: 4 bits. Contiene el número de CRSC que le sigue al encabezado.
- Campo de marcador M: 1 bit. Un perfil de aplicación define su interpretación.
- Campo de tipo de carga útil PT: 7 bits. Este campo identifica el tipo de carga útil (audio, video, imagen, texto, html, etc.).

- Campo Número de secuencia: 16 bits. Su valor inicial es aleatorio y aumenta de a 1 por cada paquete enviado. Puede utilizarse para detectar paquetes perdidos.
- Campo Marca de tiempo: 32 bits. Refleja el instante de muestreo del primer byte del paquete RTP. Este instante debe obtenerse a partir de un reloj que aumenta de manera monótona y lineal para permitir la sincronización y el cálculo de la variación de retardo en el destino.
- Campo SSRC (*Synchronization source*): 32 bits. Identifica de manera única la fuente. La aplicación elige su valor de manera aleatoria. SSRC identifica la fuente de sincronización (simplemente llamada "la fuente"). Este identificador se elige de manera aleatoria con la intención de que sea único entre todas las fuentes de la misma sesión. La lista de CSRC identifica las fuentes (SSRC) que han ayudado a obtener los datos contenidos en el paquete que contiene estos identificadores. La cantidad de identificadores se proporciona en el campo CC.
- Campo CSRC (*Contributing source*): 32 bits. Identifica las fuentes contribuyentes.

3.3.2 FORMATO GENERAL DE UN PAQUETE RTCP

Un paquete RTCP está formado por los campos que se reflejan en la figura 3.4.

<table>
<tr><td colspan="3">Byte 1</td><td>Byte 2</td><td colspan="2">Byte 3</td><td>Byte 4</td></tr>
<tr><td>V</td><td>P</td><td>C</td><td>Tipo</td><td colspan="3">Longitud</td></tr>
<tr><td colspan="7">Datos</td></tr>
</table>

Figura 3.4 Formato general de un paquete RTCP

- Versión (V, 2 bits). La versión en RTCP es igual a la versión de los paquetes RTP.
- Relleno (P, 1 bit). Si P es igual a 1, el paquete contiene bytes adicionales para rellenar y finalizar el último paquete.
- Contador (C, 5 bits) de bloques de datos contenidos en el paquete.
- Tipo de paquete (8 bits). Los tipos de paquetes que existen se muestran a continuación:
  - FIR (*full INTRA-frame request)*

- NACK (*negative acknowledgement*)
- SR (*sender report)*
- RR (*receiver report*)
- SDES (*source description*)
- BYE (*goodbye*)
- APP (*application defined*)
- RTPFB (*Generic RTP Feedback*)
- PSFB (*Payload-specific*)
- XR (*RTCP extension*)

- Longitud (16 bits). Se refiere a la longitud del paquete RTCP en palabras de 32 bits menos una, incluyendo la cabecera y cualquier relleno.

3.4 CONTROL DE CONGESTIÓN EN RTP

3.4.1 LA NECESIDAD DE UN CONTROL DE CONGESTIÓN

Hasta ahora se ha asumido que se envía un flujo de datos multimedia a su velocidad nominal, y que si el ancho de banda disponible en el canal de comunicación es menor se producen pérdidas de paquetes. En el mundo real el flujo de datos compartirá el canal de comunicación con otros tipos de tráfico. Deben considerarse los efectos debidos a ese tráfico en el flujo de datos RTP.

Como es sabido, una red IP proporciona un tipo de servicio *best-effort*. Una de las características que definen una red IP es, por tanto, que no existe un control de admisión de paquetes. El nivel de red acepta a todos los paquetes y hace todo lo posible por entregarlos. De todos modos, no hay garantía de entrega y serán descartados paquetes si el enlace llega a congestionarse. Esto no supone ningún problema siempre y cuando las capas superiores disminuyan las velocidad de transmisión cuando haya congestión. Si no lo hicieran existiría un riesgo potencial de colapso en la red debido a la congestión.

Un colapso por congestión se define como la situación en la cual un incremento en la carga de la red resulta en un decremento en el número de paquetes entregados de manera exitosa hacia la red. Gráficamente este efecto puede observarse en la figura 3.5.

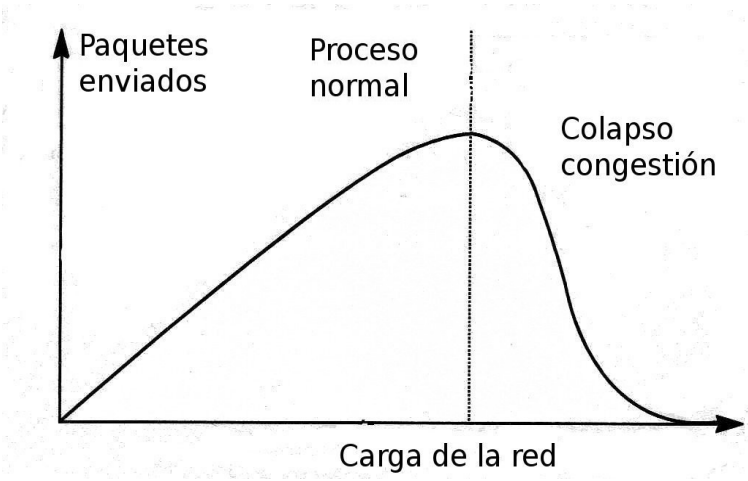

Figura 3.5 Colapso por congestión

El protocolo de transporte TCP implementa mecanismos de control de congestión, pero no ocurre lo mismo con el protocolo de transporte UDP. Con el incremento de tráfico multimedia sobre UDP existe un riesgo cada vez mayor de que se produzcan colapsos en la red por congestión.

3.4.2 CONTROL DE CONGESTIÓN PARA DATOS MULTIMEDIA

Actualmente no existen estándares para el control de congestión de flujo de datos de audio y vídeo a través de Internet. Es posible de todos modos usar el protocolo de transporte TCP para transmitir los datos y no usar por tanto el protocolo UDP. Sin embargo se pueden usar algoritmos parecidos a los que usa el protocolo TCP

3.4.2.1 Algoritmos de control de flujo similares a TCP

TCP tiene varias propiedades que lo hacen inapropiado para aplicaciones de tiempo real, en particular el énfasis que pone en la fiabilidad. Prácticamente ninguna aplicación multimedia usa TCP.

En vez de usar TCP directamente es posible emular el algoritmo de control de congestión de TCP sin usar los mecanismos de fiabilidad. Aunque no existen estándares que implementen esta solución, existen intentos de diseñar un protocolo que lo implemente, donde el protocolo RAP (*Rate Adaptation Protocol*) es quizás el más completo. Al igual que TCP, una fuente RAP envía paquetes de datos que contienen números de secuencia, los cuales son confirmados por un receptor RAP. Gracias a esta

confirmación en los paquetes un transmisor puede detectar pérdidas de paquetes y mantener una media estable del tiempo de ida y vuelta del paquete.

Un transmisor RAP ajusta su velocidad de transmisión usando algoritmos de incrementos aditivos y decrementos multiplicativos de la misma manera en que lo hace un transmisor TCP. La única diferencia con TCP es que RAP no utiliza mecanismos que aseguren la entrega de un paquete. Cuando se detecta una pérdida en algún paquete, el protocolo RAP reduce su velocidad de transmisión pero no tiene la obligación de reenviar el paquete perdido. La acción más probable del protocolo RAP sería renegociar con el otro extremo de la comunicación el algoritmo de codificación utilizado para adaptarse al nuevo ancho de banda disponible y continuar sin recuperar los paquetes perdidos.

El problema de usar TCP o protocolos que usen ciertas características de TCP con RAP es que la aplicación tiene que adaptar su velocidad de transmisión muy rápidamente para adaptarse a las nuevas condiciones del canal. Esto en algunos casos puede degradar la comunicación lo suficiente para que ambos extremos noten pérdidas en la calidad.

3.4.2.2 Emulación de la transmisión de datos TCP

Tal y como se dijo anteriormente, el principal problema de los algoritmos de control de la congestión similares a TCP son los rápidos cambios que pueden ocurrir en cortos periodos de tiempo en el canal de comunicación. Muchos codecs de audio son no adaptativos y trabajan a una velocidad de transmisión fija (por ejemplo GSM o G.711). Otros codecs de audio pueden adaptarse únicamente entre un conjunto fijo y determinado de velocidades de transmisión (por ejemplo ILBC). Los codecs de vídeo generalmente pueden adaptarse mucho mejor porque tanto el número de tramas por segundo como el índice de compresión pueden ser ajustados fácilmente, pero la velocidad a la que lo hacen es frecuentemente baja. Aunque el algoritmo de codificación pueda adaptarse rápidamente no está claro que hacerlo sea lo mas adecuado puesto que se han realizado estudios en los que los usuarios prefieren una calidad estable a una calidad que varía con el tiempo.

La emulación de la transmisión de datos TCP consiste en examinar la función respuesta de la máquina de estados TCP. Se puede desarrollar un modelo matemático para la velocidad de paquetes salientes en una conexión TCP (*throughput* medio) dados el índice de pérdidas y el tiempo de ida y vuelta de una red. Se puede demostrar que el *throughput* medio T de una conexión TCP puede modelarse de la siguiente manera:

$$T = \frac{s}{R * \sqrt{\frac{2p}{3}} + 3p\left(1 + 32p^2\right) * T_{rto} \sqrt{\frac{3p}{8}}} \qquad (1)$$

En la anterior expresión se distinguen los siguientes parámetros:

- Tamaño del paquete en bytes *s*
- Tiempo de ida y vuelta entre transmisor y receptor *R*
- Probabilidad de pérdida de paquetes *b*
- *Timeout* de retransmisión TCP en segundos T_{rto}

Esta ecuación parece compleja pero los parámetros que la definen son relativamente fáciles de medir. Una aplicación basada en RTP conoce el tamaño de los paquetes que envía, el tiempo de ida y vuelta puede ser obtenido de la información proporcionada por RTCP, y una aproximación de la probabilidad de paquetes perdidos también puede ser obtenida mediante el protocolo RTCP. El único parámetro que puede ser algo más complicado de calcular es el *timeout* de retransmisión TCP, pero un valor satisfactorio en la mayoría de los casos es cuatro veces el tiempo de ida y vuelta.

Teniendo todos estos parámetros un transmisor puede calcular la velocidad media que una conexión TCP podría albergar asumiendo que la probabilidad de pérdidas de paquetes es constante. Este dato puede ser usado como parte de un mecanismo de control de congestión. Si la aplicación está enviando datos a una velocidad mayor que la calculada debería reducir la velocidad de transmisión hasta hacerla coincidir con el valor calculado o se arriesga a congestionar la red. Si está enviando datos a una velocidad menor podría incrementarla hasta el valor calculado. La aplicación funciona mediante un bucle de realimentación: cambiar velocidad de transmisión, medir probabilidad de pérdidas de paquetes, cambiar velocidad de transmisión, ... Este bucle de realimentación podría ser ejecutado cuando lleguen los paquetes de informes RTCP.

Por ejemplo, si el tiempo de ida y vuelta es 100 milisegundos la aplicación que envíe datos con el codec PCM con paquetes de 20 milisegundos y una probabilidad de pérdidas de paquetes del 1% la velocidad ideal seria de 21.9 Kbps. Dado que la velocidad del flujo de datos PCM es 64 Kbps el transmisor sabe que está causando congestión y que debe

reducir su velocidad de transmisión. Podría hacerlo conmutando a un codec de menor ancho de banda, por ejemplo GSM.

Esto parece muy simple pero en la práctica hay problemas que deben ser resueltos. El problema más crítico es cómo se mide y se promedia la probabilidad de pérdidas de paquetes. Otros problemas secundarios están relacionados con el tamaño de los paquetes, el algoritmo comienzo lento (*slow-start*) y la transmisión discontinua.

- **Cálculo de la probabilidad de pérdida de paquetes.**

  Las aplicaciones RTP no miden directamente la probabilidad de pérdidas de paquetes, pero a cambio sí pueden contar el número de paquetes perdidos entre cada intervalo de paquetes RTCP y a partir de ahí aproximar la probabilidad de paquetes perdidos. Es necesario destacar que esta aproximación a veces puede no ser lo suficientemente válida. El porcentaje de paquetes perdidos en un intervalo particular no tiene que ser un fiel reflejo del porcentaje total de paquetes perdidos porque puede haber ráfagas de pérdidas. Sin embargo, se podrían amortiguar estas ráfagas si se midiera el porcentaje de paquetes perdidos sobre un periodo de tiempo prefijado, pero no existe un consenso sobre este valor de tiempo.

- Tamaño del paquete.

  La expresión matemática expuesta anteriormente asume que el tamaño del paquete es fijo mientras la velocidad de transmisión varía. Esto es cierto para algunos codecs pero es difícil de conseguir con otros.

- **Algoritmo slow-start.**

  El protocolo TCP implementa el algoritmo *slow-star*t cuando empieza a transmitir. Con él se consigue que un transmisor compruebe la capacidad de la red de manera gradual antes de comenzar a transmitir una ráfaga de datos que puedan causar congestión. Este algoritmo es fácil de implementar para TCP porque el protocolo recibe una notificación inmediata cuando ocurre una pérdida de paquete, pero es mas difícil de implementar en protocolos que envían notificaciones a intervalos mucho más grandes. Una descripción gráfica puede observarse en la figura 3.6.

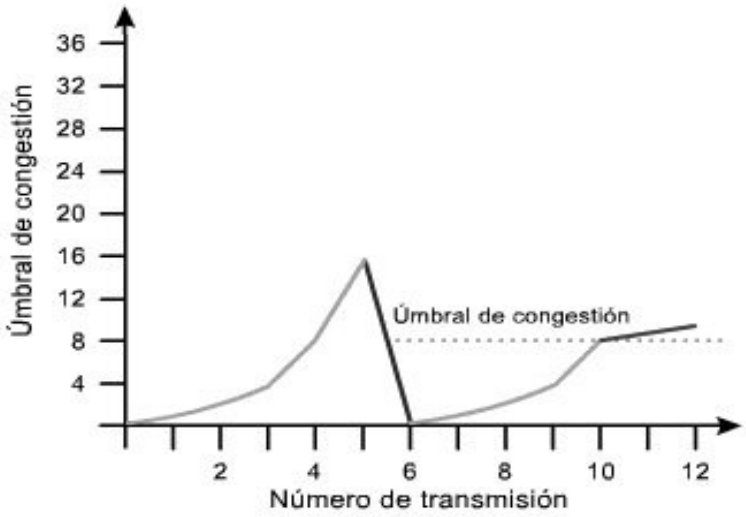

Figura 3.6 Algoritmo *slow-start*

La solución adoptada para este tipo de protocolos consiste en que el receptor envíe paquetes una vez cada tiempo de ida y vuelta durante la ejecución del algoritmo *slow-start*. El transmisor comienza a una baja velocidad de transmisión y la dobla cada vez que llega un paquete hasta que el primer paquete perdido sea detectado. Una vez que esto ocurra, el transmisor empieza su operación normal usando la velocidad de transmisión que se detectó justamente antes de la pérdida.

Esto es fácil de implementar para su uso con el protocolo RTP adaptando el protocolo RTCP. Algunas aplicaciones pueden no ser capaces de implementar este algoritmo debido a las limitaciones en el conjunto de codecs que soportan, pero siempre podrán primero enviar datos aleatorios inicialmente y cuando hayan calculado la velocidad idónea, elegir el codec más adecuado para empezar a transmitir los datos reales.

- Transmisión discontinua.

  Si un transmisor deja de enviar datos durante algún tiempo puede afectar a los datos que necesita el algoritmo. Ante este problema poco se puede hacer aparte de comenzar de nuevo el algoritmo cuando la comunicación se restaure.

Si todos estos problemas pueden ser resueltos, el algoritmo propuesto tiene el potencial para llegar a ser un estándar en el control de congestión de audio y vídeo.

3.4.3 SEGURIDAD

Hasta ahora se ha considerado que la aplicación que hace uso del protocolo RTP está aislada de cualquier otra aplicación. Sin embargo en el mundo real existen aquellos que buscan ilegitimar usuarios, desconectar sesiones y otros propósitos maliciosos. Un sistema RTP correctamente implementado es seguro contra tales ataques y puede proveer de privacidad, confidencialidad y servicios de autenticación a sus usuarios.

3.4.3.1 Privacidad

El problema de la privacidad trata de resolver cómo prevenir que terceras personas puedan adquirir datos de los participantes de una sesión RTP establecida.

La información referente a una fuente, posiblemente detalles personales, es comunicada en ciertos paquetes RTCP. Esta información tiene varios usos, la mayoría de los cuales son legítimos, pero algunos de ellos pueden ser inapropiados para algunos usuarios. Un ejemplo de legitimidad podría ser una aplicación de teleconferencia que usa los paquetes RTCP para nombrar a todos los participantes por su nombre, puesto de trabajo e identificación del número llamante. Otros usos inapropiados podrían ser incorporar información personal de un usuario que está escuchando una emisora de radio a través de Internet, permitiendo a la misma emisora y a las empresas que publicitan sus servicios adquirir tales datos. Es por tanto importante que las aplicaciones no envíen ningún tipo de información personal en los paquetes RTCP sin previamente informar al usuario que esta información está disponible.

La única excepción que es obligatoria cumplir es el envío de la dirección IP del participante. Esto no representa un riesgo en la privacidad porque la misma información está disponible en la cabecera del paquete IP. La dirección IP es un único identificador que puede ser usado para reconocer al usuario. Si el participante se conectara desde una máquina con una dirección IP temporal (direcciones IP asignadas dinámicamente por el proveedor de servicios) el problema queda resuelto, pero si el sistema tiene una dirección IP estática no puede hacerse nada para proteger tal dirección de ser reconocida.

Algunos participantes pueden no querer ser visibles al resto de participantes. Podrían no enviar paquetes RTCP, pero perderían la posibilidad de proveer información de calidad en la transmisión a otros participantes para que ellos pudieran adaptarse. En algunos casos incluso un transmisor que no reciba paquetes RTCP puede asumir que el receptor no está disponible y por tanto finalizar la comunicación.

3.4.3.2 Confidencialidad

Uno de los principales requerimientos para RTP es la confidencialidad, asegurando que sólo los participantes deseados puedan decodificar los paquetes RTP. El contenido del paquete RTP se mantiene confidencial encriptándolo ya sea a nivel de aplicación (encriptando el paquete entero o sólo los datos que lleva) o a nivel IP.

La encriptación puede realizarse a nivel IP, por ejemplo usando protocolos de seguridad IPSec (*Internet Protocol Security*). La ventaja principal es que es transparente al protocolo RTP y provee de un conjunto de algoritmos de encriptación bien definidos. La desventaja principal es que ya no se permite compresión de la cabecera del paquete y requiere cambios extensos en los sistemas operativos.

La encriptación a nivel de aplicación tiene ventajas pero también alguna desventaja. La principal ventaja es que permite la compresión de la cabecera del paquete RTP. Otra ventaja es que es simple de implementar y de desplegar, ya que no requiere cambios en los sistemas operativos o dispositivos encaminadores. Desafortunadamente esto también es una desventaja potencial porque da libertad a que las aplicaciones puedan desarrollar su propia manera de realizar la encriptación, lo que puede dar lugar a incompatibilidades. Otra desventaja es que algunos campos de la cabecera del paquete no pueden ser encriptados, lo que en algunos casos puede revelar información sensible. Por ejemplo, el conocimiento del tipo de carga que lleva el paquete puede permitir a un atacante averiguar algunos valores de la carga encriptada, quizás porque cada trama comienza con una cabecera con un formato estándar.

- **Características de confidencialidad en la especificación del protocolo RTP**

  La propia especificación RTP provee de soporte para encriptación de los paquetes RTP (incluyendo la cabecera). Pueden elegirse varios esquemas de encriptación. Dependiendo del algoritmo usado pueden añadirse bytes de relleno (*padding*) a la carga del paquete antes de que pueda realizarse la encriptación. Por ejemplo, el algoritmo de encriptación DES (*Data Encryption Standard*) opera en bloques de 64 bytes, así que se necesitaran añadir bytes de relleno si la longitud no es múltiplo de 8 bytes. Un ejemplo puede verse en la figura 3.7.

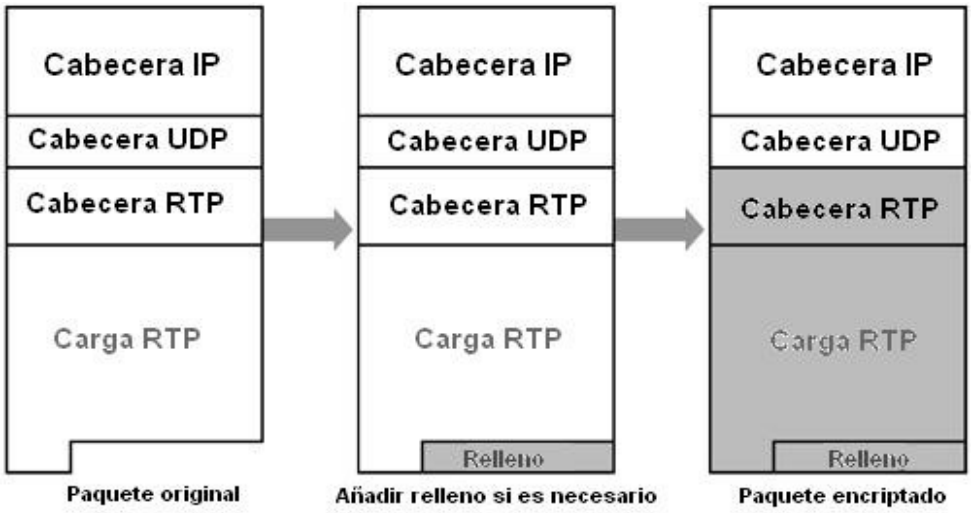

Figura 3.7 Encriptación de un paquete RTP

Cuando se usan bytes de relleno se activa el bit P en la cabecera del paquete, y el último byte de relleno indica el número de bytes que han sido añadidos.

El algoritmo de encriptación usado por defecto es el algoritmo DES. Cuando el protocolo RTP fue diseñado el algoritmo DES ofrecía un nivel de seguridad suficiente, pero los avances en la capacidad de proceso lo han convertido en un algoritmo débil. Es recomendable utilizar otros algoritmos más robustos cuando sea posible, tales como pueden ser triple DES o AES (*Advanced Encryption Standard*). Para maximizar la compatibilidad, todas las implementaciones que soporten encriptación deberían soportar DES aún a sabiendas de su debilidad.

La especificación RTP no define ningún mecanismo para el intercambio de claves de encriptación. Éste es un punto de fundamental importancia en cualquier sistema y debería ser llevado a cabo al inicio de la sesión. El protocolo SIP provee de un mecanismo de intercambio de claves.

- Confidencialidad usando el perfil SRTP (*Secure* RTP)

  SRTP fue diseñado teniendo en mente las necesidades de la telefonía inalámbrica. Provee confidencialidad y autenticación apropiada para su uso con enlaces que puedan tener una alta probabilidad de pérdida de paquetes y que requieran compresión de paquetes para resultar eficientes. SRTP no es un protocolo aún completo pero sigue evolucionando lentamente.

SRTP proporciona confidencialidad de los datos del paquete RTP encriptando únicamente la sección del paquete correspondiente a la carga del mismo. La cabecera del paquete es enviada sin encriptar. El algoritmo de encriptación por defecto es AES. El protocolo también proporciona confidencialidad a los paquetes RTCP. Todo el paquete RTCP es encriptado excluyendo la cabecera inicial (los primeros 64 bytes) y algunos campos adicionales que son añadidos al final de cada paquete RTCP. Los campos adicionales son un índice SRTCP (*Secure* RTCP), un bit para indicar si la carga del paquete está encriptada (bit E), un identificador de clave maestra opcional y un campo de autenticación. El algoritmo de encriptación por defecto también es AES.

- Confidencialidad usando seguridad IP

  Como agregado a la seguridad que RTP y SRTP aportan a nivel de aplicación, es posible usar seguridad a nivel IP. IPSec es implementado como parte de la pila de protocolos de red del sistema operativo, no por las aplicaciones. Proporciona seguridad para todas las comunicaciones desde un equipo y no es específico para RTP.

  IPSec tiene dos modos de trabajo: modo transporte y modo túnel. El modo transporte inserta una cabecera adicional entre la cabecera IP y la cabecera del nivel de transporte, proporcionando confidencialidad de la cabecera TCP o UDP y su carga, pero dejando la cabecera IP tal como está. En el modo túnel se encapsula todo el datagrama IP dentro de una cabecera de seguridad. Ambos modos de trabajo se representan en la figura 3.8.

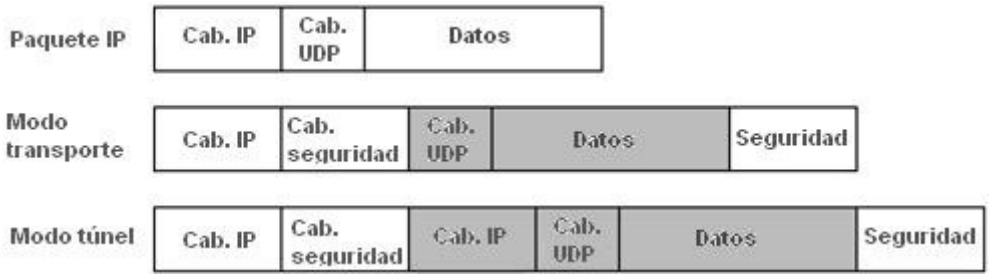

Figura 3.8 Modo transporte y modo túnel en IPSec

El modo túnel se usa frecuentemente para construir redes privadas virtuales entre dos dispositivos encaminadores para extender una red privada a través de una red pública. El modo de transporte se usa cuando quiere establecerse un enlace seguro entre dos equipos individuales.

Ambos modos de trabajo proporcionan confidencialidad y autenticación de los paquetes. La confidencialidad se proporciona gracias al protocolo ESP (*Encapsulating Security Payload*). ESP añade una cabecera adicional y un campo al final del paquete. La cabecera incluye un parámetro de seguridad para determinar el contexto criptográfico y un número de secuencia, mientras que el campo añadido al final es un campo de relleno e indica el tipo de datos encapsulados (TPC o UDP si se usa el modo transporte, IP si se usa el modo túnel). Entre dicha cabecera y el campo añadido al final se sitúan los datos protegidos.

Si se usa ESP con RTP todo el paquete RTP (incluyendo cabecera y carga) será encriptado, así como las cabeceras UDP e IP si se usa el modo túnel.

3.4.3.3 Autenticación

Pueden distinguirse dos tipos de autenticación: aquella en la que los paquetes no han sido modificados, conocida como integridad, y aquella en la que los paquetes vienen del origen correcto, conocida como autenticación por origen.

La integridad de los paquetes se consigue gracias al uso de códigos de autenticación. Si tanto transmisor como receptor poseen una clave que sólo ellos conocen pueden generar una única clave que puede usarse para proteger al paquete de datos. A un posible atacante que no conozca la clave le será imposible cambiar el contenido del paquete. La autenticación por origen es un problema mucho más complejo para aplicaciones que usen el protocolo RTP. El problema reside en que es necesario identificar al origen con su clave, significando esto que la clave tiene que ser mucho más grande y por tanto más compleja de procesar. Esto hace que frecuentemente sea inviable autentificar la fuente de cada paquete en un flujo RTP.

Al igual que en los métodos explicados que aseguran la confidencialidad, la autenticación puede ser aplicada a nivel de aplicación o a nivel IP.

3.4.3.4 Denegación de servicios

Existe un riesgo potencial de denegación de servicio cuando determinados formatos de datos que usan compresión suponen una alta carga computacional. Si la carga del paquete

tiene tales propiedades, un atacante podría ser capaz de inyectar determinados paquetes en un flujo de datos que sean muy complejos de decodificar y hacer que el receptor se sobrecargue. El receptor puede hacer muy poco para evitar este problema, lo único que podría hacer sería dejar de procesar paquetes en situaciones de sobrecarga.

Otra posible causa de denegación de servicio puede ser debida a los fallos de implementación en la temporización de los paquetes RTCP. Si varios participantes de una comunicación envían paquetes RTCP a una velocidad constante, al aumentar el número de participantes el tráfico proveniente de los paquetes RTCP crecerá linealmente. Para grupos de participantes muy grandes este crecimiento puede derivar en una congestión de la red.

3.4.3.5 Protección a repeticiones

El objetivo es que un posible atacante no pueda grabar paquetes de datos de una sesión RTP para luego reinyectarlos en la red con propósitos maliciosos. En principio el paquete RTP contiene un número de secuencias y un identificador temporal que podrían solucionar este problema, pero un atacante podría grabar el flujo de paquetes y modificarlos convenientemente para que el receptor reciba el identificador temporal y el número de secuencia esperados. La única solución es autentificar los mensajes y verificar la integridad de los mismos.

3.4.3.6 Contenido activo

La gran mayoría del contenido audiovisual es pasivo, es decir, datos codificados se hacen pasar por un decodificador para obtener el contenido original. Sin embargo hay otra clase de contenido: aquel que contiene código ejecutable que puede ser incluido en determinados flujos de datos. Tal contenido activo tiene el potencial de ejecutar operaciones arbitrarias en el sistema receptor.

CAPÍTULO 4: IMPLEMENTACIÓN

4.1 INTRODUCCIÓN

Para el desarrollo de un soft-phone con las características especificadas en capítulos anteriores se ha usado un conjunto de librerías básicas llamadas SipXTapi. Tras realizar un estudio detallado de su funcionamiento, se han encontrado puntos clave en su funcionamiento necesarios para incluir las características deseadas, tales como la integración de codecs adicionales o la generación de librerías de enlace dinámico. Tras la realización de dichas modificaciones, es posible implementar un interfaz gráfico en el lenguaje de programación Java que haga uso de las librerías generadas con anterioridad. En este capítulo se describe con exactitud los pasos dados hasta conseguir una implementación real de un soft-phone con un interfaz gráfico Java con toda la funcionalidad requerida.

4.2 LIBRERÍAS SIPXTAPI

El conjunto de librerías denominadas SipXTapi se compone de un conjunto de funciones y herramientas para proporcionar soluciones en la construcción y diseño de clientes SIP. Estas librerías son distribuidas bajo la licencia LGPL (*Lesser General Public License*) [LGPL], lo que permite un uso comercial de la misma. SipXTapi es desarrollado bajo sistemas Windows de 32 bits, Windows CE y Linux (32 y 64 bits). También dispone de soporte parcial para arquitecturas MacOS, Solaris, VxWorks, Symbian y, próximamente, Android. El desarrollo de estas librerías fue iniciado por la empresa Pingtel Corporation [SIPFoundry'10], y más tarde fue donado en su totalidad.

Las características principales se pueden resumir en las siguientes:

- Múltiples llamadas simultáneas
- Retención y silenciado de canales
- Volumen y control de ganancia
- Dispositivos de entrada, micrófono y altavoces seleccionables
- Generación de tonos DTMF, ocupado, etc
- Reproducción de audio desde un fichero
- Selección de interfaces de red

- Pila SIP compatible con RFC 3261
- Señalización fuera de banda compatible con RFC 2833
- Supresión de tráfico RTP en modo silencio
- Puertos configurables para UDP, TCP y RTP
- Múltiples cuentas SIP
- Algoritmo de codificación G.711

El algoritmo de codificación G.729 es un algoritmo propietario, por lo que se exige un pago para su uso. Su código fuente optimizado no está disponible, pero existe una versión distribuida libremente por la ITU que, aunque poco eficiente en su diseño, sirve para la consecución del objetivo de integrar este algoritmo en un cliente SIP. Será labor de este capítulo describir los pasos dados en la compilación de este código fuente y en la posterior generación de una librería que pueda integrarse al proyecto global SipXTapi.

En este punto se examina a fondo la librería SipXTapi haciendo especial hincapié en aquellas funcionalidades principales a la hora del diseño de un cliente SIP. Al mismo tiempo se estudia la manera de introducir nuevos algoritmos de codificación de audio en la librería, las clases que se generarán y las llamadas a las mismas para que la integración sea perfecta. Como objetivo final se genera una librería de enlace dinámico con toda la funcionalidad deseada de la librería SipXTapi para que otras aplicaciones puedan hacer uso de ella.

4.2.1 RESUMEN DE LAS PRINCIPALES FUNCIONES

La librería SipXTapi dispone de un muy amplio conjunto de funciones que se pueden englobar dentro de los siguientes bloques:

- Funciones de inicialización

  Son funciones que se encargan de inicializar los protocolos que se van a usar para la comunicación (SIP, RTP y RTCP) y de crear las entidades o cuentas de usuario bajo las cuales el usuario podrá identificarse. Como ejemplo de este tipo de funciones se tienen los siguientes métodos:

  sipXInitialize()

  sipXUnInitialize()

  sipXLineAdd()

sipXLineRegister()

sipXConfigSetRegisterExpiration()

sipXSetUserAgentName()

- Funciones de adquisición de dispositivos

  Dispone de funciones para averiguar qué dispositivos de entrada y salida están conectados al sistema, así como para que se puedan escoger aquellos que se quiera en cada momento. Algunas funciones específicas son las que siguen a continuación:

  sipXAudioGetNumInputDevices()

  sipXAudioGetInputDevice()

  sipXAudioGetNumOutputDevices()

  sipXAudioGetOutputDevice()

  sipXAudioSeCallInputDevice()

  sipXAudioMute()

  sipXAudioEnableSpeaker()

  sipXAudioSetVolume()

- Funciones de control

  Aquí es donde se encuentran las funciones telefónicas básicas tales como llamar, colgar, rechazar una llamada, poner tono de marcado u ocupado, adquirir el identificador de la llamada, etc. Como ejemplos se pueden citar:

  sipXCallAccept()

  sipXCallReject()

  sipXCallAnswer()

  sipXCallCreate()

  sipXCallConnect()

  sipXCallDestroy()

  sipXCallGetRemoteId()

- Funciones avanzadas del protocolo SIP

  En este bloque residen funcionalidades avanzadas tales como uso de notificaciones, mensajería instantánea SIP, etc. Ejemplos de tales funciones son:

  sipXPublisherCreate()

  sipXPublisherDestroy()

  sipXConferenceCreate()

Todas estas funciones de alto nivel hacen uso de unas librerías de bajo nivel que en forma de capas se acercan paulatinamente al nivel más bajo del modelo OSI. Así se pueden encontrar librerías que hacen uso de los eventos recibidos, aquellas que manejan todo el protocolo SDP y las que controlan todo el tráfico multimedia sobre RTP. Será en esta última la que se estudiará y modificará posteriormente para añadir nuevos algoritmos de codificación.

Se explicará de manera general como añadir cualquier algoritmo de codificación, y a continuación se añadirá el algoritmo G.729. La estructura del proyecto completo SipXTapi aparece en la figura 4.1.

En la estructura se puede observar que existe una librería denominada SipXStack que implementa todo el protocolo SIP según el estándar RFC 3261. En ella se especifican el formato de los paquetes de datos y la información que tienen que llevar. La siguiente librería, SipXCall, implementa funciones básicas de llamadas con SIP proporcionándole los datos necesarios al protocolo desarrollado por SipXStack. Con estas dos librerías se tendría toda la arquitectura SIP completa, pero es necesario transmitir el flujo real de datos que se realiza con la librería SipXMedia. En ella se diseñan los algoritmos de codificación y decodificación y se ponen a disposición del resto de librerías que lo quieran usar. La implementación del protocolo SDP se encuentra dentro de esta librería. Si se necesitara conectar toda la arquitectura con un servidor SIP entraría en juego la librería SipXCommServer. Ya por último, para transmitir todos los datos a nivel físico se utiliza la librería SipXPort.

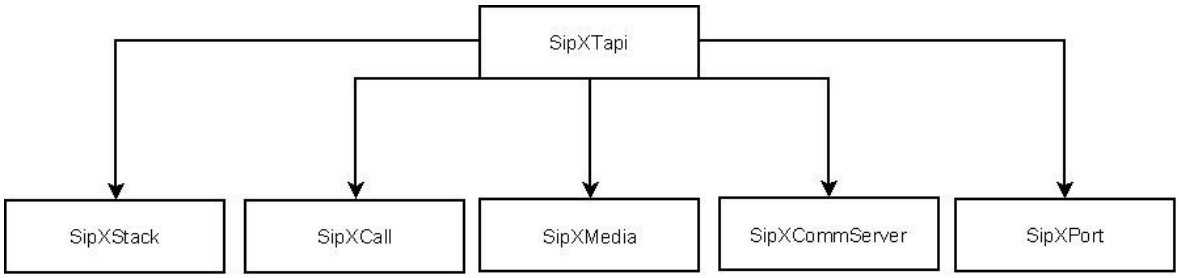

Figura 4.1 Arquitectura global de la librería SipXTapi

Las funciones de alto nivel descritas anteriormente hacen uso de todas estas funcionalidades que ofrecen las capas inferiores mediante los puntos de acceso al servicio.

Cada librería puede compilarse de manera independiente y generar un fichero de librería de extensión “lib” del que podrá hacer uso la librería de alto nivel SipXTapi. Esto proporciona una gran versatilidad en el desarrollo de aplicaciones, puesto que al generar cada librería un subproducto único e indivisible en forma de fichero, cualquier aplicación externa podría hacer uso de los servicios que ofrezca.

A modo de ejemplo, una aplicación de enseñanza del protocolo de señalización SIP podría usar la librería SipXCall para mostrar por pantalla la formación de cada uno de los paquetes que implementa el protocolo, mientras que otra aplicación distinta que muestre el protocolo RTP haría uso únicamente de la librería SipXMedia. Este enfoque modular es una de las ventajas derivadas del uso de este conjunto de librerías.

4.2.2 CODIFICACIÓN Y DECODIFICACIÓN

Todo el proceso algorítmico de codificación y decodificación de audio se produce en la librería SipXMedia. Por defecto se implementa el algoritmo G.711, pero ya se discutió en capítulos anteriores la necesidad de escoger algoritmos de codificación que usen un bajo ancho de banda si es posible. Se ha escogido el algoritmo G.729 por su amplia difusión comercial y disponibilidad de código gratuito - en entornos académicos y no comerciales - para su implementación.

Tras estudiar la estructura del código se distinguen tres tipos de clases principales que son las que implementan todo el control de codificación y decodificación de audio:

- Clases codificadoras

  Se nombran MpeSipx seguido del nombre del codec. En esta clase se definen tres métodos básicos: initEncode(), freeEncode() y encode(). Rellenando estos tres métodos junto con los constructores y destructores de la clase es posible implementar cualquier algoritmo de codificación

- Clases decodificadoras

  Se nombran MpdSipx seguido del nombre del codec. Con una estructura muy similar a las clases codificadoras, desarrollando los métodos initDecode(), freeDecode() y decode() se puede implementar cualquier algoritmo de decodificación.

- Clase que inicia los algoritmos de codificación – decodificación
  La clase MpCodecFactory ha de modificarse para añadir nuevas instancias de las clases codificadoras y decodificadoras.

Se puede observar la estructura de clases resultante tras añadir el algoritmo G.729 en la figura 4.2, donde se distinguen con claridad las clases de codificación y las clases de decodificación.

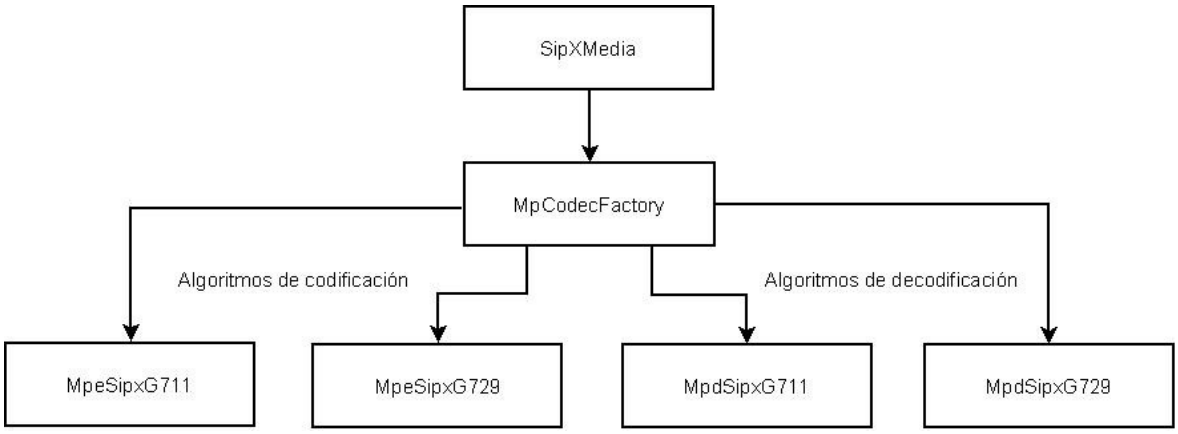

Figura 4.2 Arquitectura de librería SipXMedia

Siguiendo estas directrices es posible añadir los algoritmos de codificación que se requieran según la aplicación, abriéndose todo un abanico de posibilidades. Si se requiriera una aplicación en la que el coste de ancho de banda es elevado podrían incluirse algoritmos que usen anchos de banda adaptativos tales como ARM (*Adaptative Multi Rate*). Si por el contrario las especificaciones son bajos requerimientos computacionales puede optarse por G.711. También se puede encontrar la situación en la que uno de los interlocutores use un algoritmo específico no estándar, ante lo cual habría que redefinir las clases anteriormente mencionadas para adaptarlas a la situación. Todos los cambios realizados en la librería SipXMedia estarán automáticamente disponibles para las funciones de alto nivel de la capa superior, por lo que la flexibilidad es muy alta.

4.3 RECOMENDACIÓN G.729

4.3.1 INTRODUCCIÓN

El algoritmo de codificación de señales de audio G.729 consiste en un codificador de 8 Kbps usando operaciones con aritmética de punto fijo. Existen anexos a la recomendación, identificados por las letras del abecedario, que implementan funcionalidades nuevas o mejoran ciertos aspectos del algoritmo base, pero todos ellos forman parte de la familia CS-ACELP (*Conjugate-Structure Algebraic Code Excited Linear Prediction*). Esta familia de codificadores se caracteriza por la buena calidad ofrecida a la hora de codificar la señal, pero al mismo tiempo por su alto contenido de proceso computacional. Las características propias de cada uno de los anexos de la recomendación se representan de modo esquemático en la tabla 4.1.

	Anexos											
Funcionalidad	**-**	**A**	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**C+**	**J**
Baja complejidad		X	X									
Punto fijo	X	X	X		X	X	X	X	X	X		X
Punto flotante				X							X	
8 kbit/s	X	X	X	X	X	X	X	X	X	X	X	X
6.4 kbit/s					X		X		X	X	X	
11.8 kbit/s						X		X	X	X	X	
Transmisión discontinua			X				X	X		X	X	
Ancho de banda variable												X

Tabla 4.1 Anexos de la recomendación G.729

En la tabla 4.1 puede observarse que existen anexos a la recomendación que implementan el algoritmo en punto fijo y en punto flotante, así como aquellos que implementan módulos de transmisión discontinua DTX (*Discontinuous Transmission*) mediante detectores de voz VAD (*Voice Activity Detector*). Los anexos cuya complejidad computacional es baja en comparación con el algoritmo original constituyen los anexos A y B, y es por ello que son los más utilizados en los sistemas telefónicos comerciales tales como teléfonos VoIP, teléfonos software y centralitas. También puede observarse que

existen recomendaciones que son flexibles en cuanto al ancho de banda utilizado, ofreciendo por tanto la posibilidad de tener una mayor o menor calidad de señal de audio en función de las condiciones del canal de comunicación en cada momento.

4.3.2 IMPLEMENTACIÓN DEL ALGORITMO Y PRUEBAS

El algoritmo que se ha implementado es el perteneciente al anexo A G.729A. Para ello se dispone del código fuente proporcionado por la ITU con fines académicos y de varios vectores de prueba que sirven para probar el algoritmo. Los pasos a seguir son los siguientes:

- Descarga del código fuente desde la página web de la ITU.
- Compilación del código C para generar dos ejecutables que se encarguen de codificar y decodificar un señal de audio G.729A.
- Utilizar vectores de prueba proporcionados por la ITU para verificar que la compilación ha sido correcta y el algoritmo funciona perfectamente.

Una vez compilado el código y generados dos ejecutables para probar el algoritmo, se procede a realizar una codificación de un fichero de audio PCM codificada a 8 bits por muestra con una frecuencia de muestreo de 8 Khz de duración 10 segundos. En la figura 4.3 se representa un esquema de los ejecutables generados junto con sus entradas y salidas. Ambos procesos admiten un fichero con un formato determinado y devuelven otro fichero con otro formato.

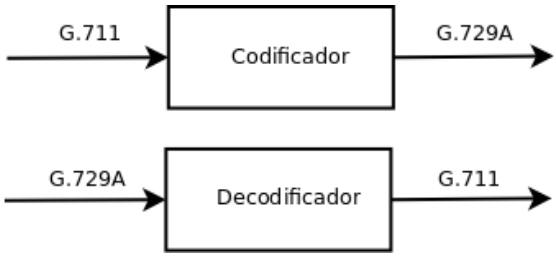

Figura 4.3 Procesos de codificación y decodificación

Para comprobar el estado del fichero codificado se usa un software reproductor de audio [COREPLAYER] que admite la posibilidad de reproducir múltiples formatos, entre ellos el deseado. El objetivo es verificar que el reproductor es capaz de decodificar el fichero de audio y que la calidad del audio generado es aceptable de manera cualitativa. Una vez

cargado el fichero y tras comprobar que no da ningún fallo, se comprueba que la calidad audible es aceptable, aunque inferior a la del fichero en formato PCM como cabía esperar.

El correcto funcionamiento del proceso de decodificación se verifica con la ayuda de un software de filtrado de paquetes llamado WireShark [WIRESHARK]. Utilizando dos teléfonos VoIP hardware conectados a un conmutador y realizando una llamada entre ellos es posible establecer un flujo de paquetes RTP entre ellos que se puede capturar. El software de captura permite guardar dicho flujo de paquetes en un fichero para más tarde poder realizar un postprocesado del mismo con herramientas externas. El esquema de las pruebas se representa en la figura 4.4.

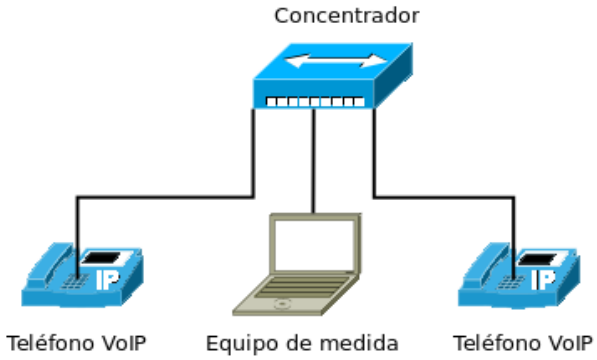

Figura 4.4 Esquema desarrollado para probar la decodificación G.729A

Una vez capturado un fichero con las muestras G.729A es posible utilizar el decodificador generado para obtener un fichero en formato G.711. Se comprueba que el proceso no genera ningún error y que la calidad de la voz generada en formato G.711 es adecuada.

En este punto puede asegurarse que el algoritmo está bien definido y es consistente, pues es capaz de codificar una muestra de audio en G.729A y decodificarla posteriormente para obtener el original. Ya se dispone, por tanto, de una herramienta software para generar ficheros de audio codificados. Sin embargo, el objetivo final es realizar esta codificación – decodificación en tiempo real con un flujo de datos proveniente de una fuente externa tal como un micrófono o un enlace de comunicaciones. El objetivo es integrar este desarrollo dentro de la librería SipXTapi como un algoritmo de codificación más, y para ello ha de generarse una librería que SipXTapi pueda usar.

4.3.3 GENERACIÓN DE LIBRERÍA

Se pretende encapsular todas las funciones de codificación y decodificación que proporciona la ITU en su código fuente en una librería para que otras aplicaciones puedan hacer uso de ella. De esta manera la librería elegida para la implementación de un cliente SIP podrá modificarse para hacer uso de los servicios que G.729A puede ofrecerle.

Una vez implementado y probado correctamente el algoritmo, el procedimiento para generar una librería es sencillo. Modificando las opciones del compilador es posible generar un fichero de extensión “lib” directamente importable desde cualquier aplicación a desarrollar. Esta filosofía de trabajo aporta grandes ventajas, a destacar entre ellas la gran modularidad en los proyectos que ofrece y la enorme reutilización de código que posibilita, ambos pilares básicos de cualquier metodología de programación moderna.

4.4 CLASES GENERADAS Y MODIFICACIÓN DE LA LIBRERÍA SIPXTAPI

En el apartado 4.2.2, tras un exhaustivo análisis del código fuente de la librería, se consiguieron aislar las partes necesarias para la consecución del objetivo de integrar la recomendación G.729 como un algoritmo más dentro de la librería.

Tras importar la librería generada para el algoritmo G.729, ya se disponen de herramientas para generar las clases necesarias. Las nuevas clases que se generaron fueron las siguientes:

- Clase codificadora MpeSipxG729.

  Realiza la codificación de un flujo de muestras de audio sin formato proporcionado por distintos dispositivos de adquisición a un formato G.729. Para ello se utilizan las herramientas que la librería G.729 generada ofrece. Posee métodos de inicialización, liberación y codificación de audio. Hereda de la clase MpEncoderBase tal y como representa en la figura 4.5.

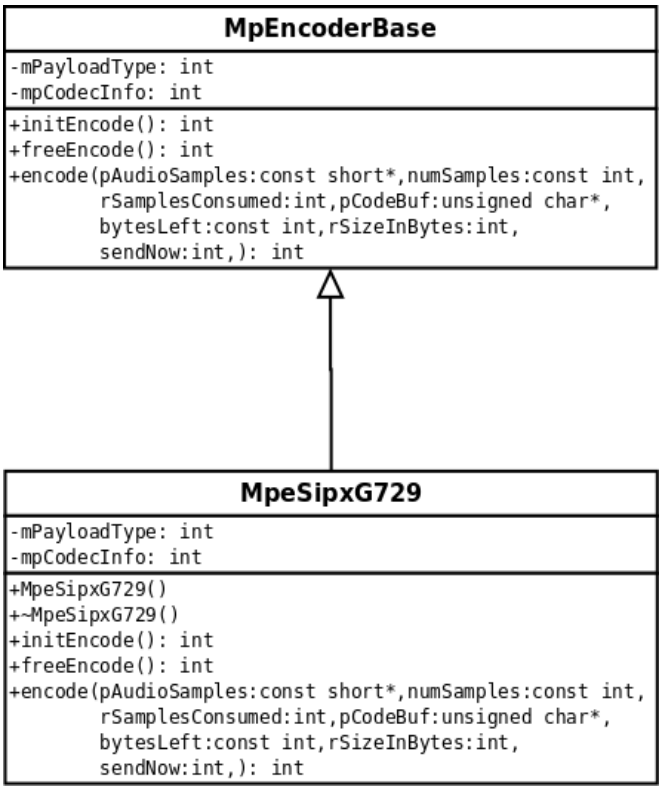

Figura 4.5 Diagrama de clases del codificador G.729

- Modificaciones en las clases CpPhoneMediaInterface y sipXMediaFactoryImpl.

  Añade en la parte del protocolo SDP necesaria el nuevo algoritmo de la librería. Con esto se añaden unas constantes que posibilitan que todo el conjunto de librerías quede informado de la incorporación de nuevo algoritmo de codificación.

- Modificación de la clase MpCodecFactory.

  Es fundamental incluir las diferentes instancias de las clases codificadora y decodificadora recién creadas, y es aquí donde se debe hacer por coherencia con el resto de los algoritmos ya implementados de serie. Desde esta clase se llama a los métodos de codificación y decodificación de los algoritmos, por lo que siguiendo una lógica similar al resto de algoritmos es posible añadir G.729.

- Modificación de la clase MpJitterBuffer.

  Es fundamental que los paquetes que tengan que ser decodificados con las clases implementadas estén sujetas a la seguridad proporcionada por un buffer que ayude

a soportar las posibles variaciones de las condiciones de la red. De esta manera se conseguirá amortiguar la variación del retardo entre paquetes. Dado que el resto de los algoritmos implementan este tipo de mecanismo de calidad de servicio, es relativamente sencillo incluir el nuevo algoritmo.

- Clase decodificadora MpdSipxG729.

  Implementa el proceso inverso a la clase MpeSipxG729, realiza una decodificación de un conjunto de muestras codificadas en G.729 y las convierte a un formato entendible por clases de nivel superior. Hereda de la clase MpDecoderBase según la figura 4.6.

Una vez que todas las clases están modificadas y compiladas correctamente se puede dar por concluida la integración del algoritmo de codificación de audio G.729 en la librería SipXTapi. Debido al alto nivel de organización que el código de la librería SipXTapi proporciona, los cambios para añadir nuevos algoritmos de codificación no son extensos.

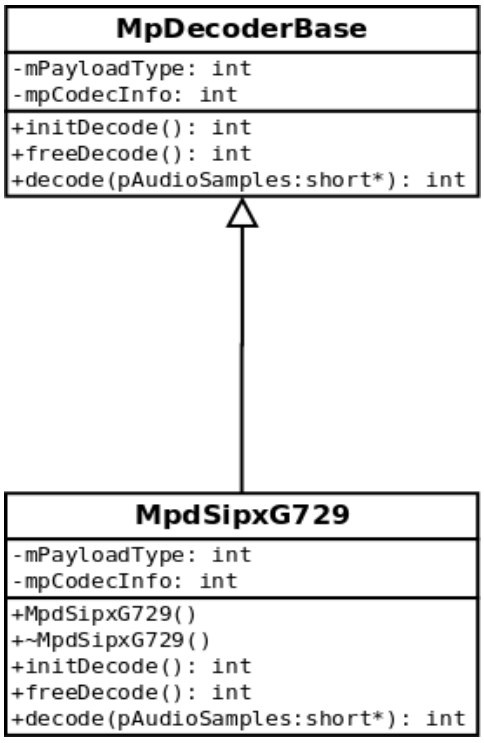

Figura 4.6 Diagrama de clases del decodificador G.729

4.5 PRUEBAS REALIZADAS

Llegados a este punto el objetivo general de este apartado está cumplido, pero aún es necesario comprobar que las modificaciones realizadas funcionen correctamente. Para ello se hizo uso de una clase de alto nivel llamada sipXmgr que es la que implementa la funcionalidad básica de cualquier soft-phone. Sólo se necesita diseñar un básico interfaz gráfico en el lenguaje de programación C++ que ejecute los métodos que sipXmgr ofrece tal y como aparece en la figura 4.7. La misma librería ofrece un soft-phone de pruebas muy simple que se modificó para eliminar ciertas funcionalidades como son la adquisición de los dispositivos de video y otros factores que no son objeto de las pruebas deseadas. Al mismo tiempo se introdujo un pequeño cuadro en el que poder ver representadas algunas trazas de la aplicación para monitorizar su comportamiento.

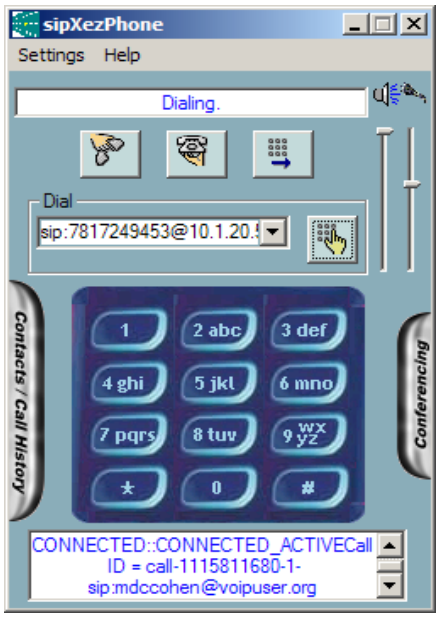

Figura 4.7 Soft-phone básico para realizar pruebas

Una vez diseñado un soft-phone en C++ totalmente funcional, el entorno de pruebas fue el que se describe a continuación. Se dispuso del siguiente equipamiento:

- Dos teléfonos VoIP Linksys SPA942 [SPA942]
- Adaptador para terminal analógico Linksys PAP2T [PAP2T]
- Centralita de voz sobre ip ejecutando el software Asterisk versión 1.4 [ASTER]
- Soft-phone X-lite de la empresa Counter Path [XLITE]

Todo el equipamiento se configuró adecuadamente con nombres de usuario numéricos empezando por el nombre de usuario 201. La centralita se configuró para que pudieran acceder los teléfonos anteriormente descritos y para que pudieran realizar llamadas internas entre ellos. Del mismo modo se configuró una extensión en la centralita que implementara un servicio de eco en el cual lo que se le pide al usuario es que grabe su voz para que más tarde se le reproduzca.

Todos los teléfonos, adaptadores telefónicos y soft-phone admiten el algoritmo G.711, pero sólo el equipamiento de Linksys admite el codec G.729. La centralita no es capaz de realizar la traducción desde el codec G.729 a G.711, pero sí es posible configurarla para que trabaje en modo *passthrough* en el que la centralita no trata el tráfico RTP que venga por el canal de transmisión, tan sólo lo deja pasar.

Se realizaron cuatro tipos de pruebas diferentes para evaluar el trabajo realizado:

- Llamada telefónica entre el terminal Linksys al soft-phone diseñado.

  Iniciando la conversación inicialmente desde el terminal Linksys y posteriormente desde el soft-phone diseñado, se comprobó que la comunicación se estableció usando el codec G.729. La calidad de la voz enviada y recibida en ambos extremos de la comunicación fue adecuada, hechos que se corroboran midiendo con un analizador de redes el retardo de los paquetes RTP *delta* y la variación media en la llegada de los paquetes RTP *jitter*. Los valores obtenidos no superan un retardo de 50 msg y una variación del retardo de 1 msg, por lo que puede considerarse que la comunicación puede realizarse de manera fluida.

- Llamada telefónica entre el soft-phone X-lite y el soft-phone diseñado.

  Dado que el soft-phone gratuito X-lite sólo soporta los codec G.711 y GSM la comunicación sólo puede establecerse mediante el algoritmo G.711. Con esta prueba se quiso demostrar que aún habiendo añadido un nuevo codec en la librería SipXTapi, los algoritmos que ya estaban seguían funcionando con total normalidad. El resultado fue el esperado y se estableció la comunicación correctamente. Esto indica que la señalización SIP funcionó como se esperaba.

- Llamada telefónica entre el adaptador analógico Linksys PAP2T y el soft-phone diseñado.

  La peculiaridad de este adaptador analógico es que dispone de una entrada para un terminal analógico convencional que lo convierte en un teléfono VoIP. Los

resultados fueron equivalentes a los realizados en la primera prueba con el terminal VoIP Linksys.

- Llamada entre dos extensiones VoIP de la centralita donde una de ellas es el soft-phone diseñado.

  Se comprobó que la señalización entre la centralita y cada una de las extensiones fue correcta y que el algoritmo escogido para la voz fue G.729. La calidad del audio fue adecuada.

Tras estas cuatro pruebas se puede concluir que el funcionamiento corresponde a lo esperado. Los terminales de voz sobre ip comerciales coexistieron adecuadamente con el soft-phone modificado con el codec G.729. Con el fin de determinar su operatividad en un entorno empresarial real se hizo uso de una plataforma de voz sobre ip Asterisk debidamente configurada. Su configuración con respecto al soft-phone diseñado no supuso mayores problemas que la configuración de cualquier otro terminal VoIP, y su comportamiento fue totalmente equivalente al de cualquier producto comercial probado.

Tras las pruebas realizadas se concluye que algoritmo queda perfectamente bien integrado dentro de la librería SipXTapi y que su interoperabilidad con otras plataformas comerciales es adecuada.

4.6 GENERACIÓN DE LIBRERÍA DE ENLACE DINÁMICO

Una vez que se ha comprobado que toda la integración realizada funciona correctamente, todo el trabajo realizado puede condensarse en un único fichero en forma de librería de enlace dinámico. Mediante la creación de esta librería se posibilita que todo el trabajo realizado pueda aprovecharse para cualquier otra aplicación. Por ejemplo, podría diseñarse una aplicación de gestión en la que hubiera un botón en el que poder llamar a un número de teléfono. También podría implementarse una aplicación web en la que los clientes que requieran atención especializada pudieran reclamarla desde la misma web hablando con un interlocutor. Cualquier aplicación podría hacer uso de los servicios SIP diseñados gracias a la importación de esta librería de enlace dinámico.

Para la creación de esta librería es necesario crear un nuevo proyecto en el entorno de programación elegido (en este caso Visual C++ 6.0) e indicarle que el resultado de la compilación no sea un fichero ejecutable sino un fichero de librería de enlace dinámico. Después de añadir todo el código a ese nuevo proyecto y tras la compilación del mismo, el compilador generará un fichero con extensión "dll" y un fichero con extensión "lib" como resultado que almacenará todas las funciones básicas de comunicación diseñadas.

4.7 ADAPTACIÓN DE SIPXTAPI A JAVA

JNI (*Java Native Interface*) es un mecanismo que nos permite ejecutar código nativo desde Java y viceversa. El código nativo son funciones escritas en un lenguaje de programación como C o C++ para un sistema operativo donde se está ejecutando la máquina virtual. Una máquina virtual se define como un entorno de programación formado por:

- El programa que ejecuta los *bytecodes* Java
- Las librerías (API) de Java

Se define *host environment* como el ordenador en que se ejecuta el sistema operativo y a las librerías nativas de este ordenador. Estas librerías se suelen escribir en C o C++ y se compilan dando lugar a códigos binarios del hardware donde se ejecuta el *host environment*. De esta forma se puede decir que las máquinas virtuales Java se hacen para un determinado *host environment*.

JNI tiene un interfaz bidireccional que permite a las aplicaciones Java llamar a código nativo y viceversa. La Figura 4.8 muestra gráficamente esta bidireccionalidad.

Figura 4.8 Interfaz bidireccional entre Java y sus librerías nativas

Es decir, JNI soporta dos tipos de interfaces:

- Métodos nativos.

  Permiten a Java llamar a funciones implementadas en librerías nativas .

- Interfaz de invocación.

  Permiten incrustar una máquina virtual Java en una aplicación nativa, como por ejemplo una máquina virtual Java en un navegador web escrito en C. Para ello, la aplicación nativa llama a librerías nativas de la máquina virtual y luego usa el llamado interfaz de invocación para ejecutar métodos Java en la máquina virtual.

4.7.1 EVOLUCIÓN HISTÓRICA DE JNI

Desde los orígenes de Java se incluía ya una forma de hacer llamadas nativas desde la máquina virtual a las librerías del *host environment*, así como desde el *host environment* a la máquina virtual (en ambas direcciones) pero con dos problemas:

- Las llamadas a código nativo desde Java accedían a estructuras C, pero no estaba definida la posición exacta que debían ocupar estos campos en memoria, con lo que una llamada nativa en una máquina virtual no coincidía con las llamadas en otra máquina virtual.
- Las llamadas nativas al JSDK (*Java Software Development Kit*) 1.0 se basaban en el uso de un recolector de basura conservativo, ya que no se controlaba cuando ya no quedaba ningún puntero nativo apuntaba a un objeto Java, con lo que una vez apuntado un objeto Java desde un método nativo éste no se liberaba nunca más.

Esta forma de llamadas nativas es lo que se usó en el JSDK 1.0 para implementar clases que accedían a partes cruciales del *host* como son los paquetes io, net y awt.

Debido a los problemas mencionados, se decidió implementar una nueva forma de acceso a métodos nativos a la que se llamó JNI, y que es una evolución del JRI (*Java Runtime Interface*), la técnica de acceso nativo de Netscape. Con el JSDK 1.1 surgió la primera versión de JNI, pero todavía las clases Java que accedían al *host* seguían usando la forma de llamada antigua. En JSDK 1.2 se reescribieron todas las clases que accedían directamente al *host* usando JNI.

4.7.2 NECESIDAD DE JNI

El objetivo es siempre escribir las aplicaciones completamente en Java para evitar incompatibilidades con las plataformas, pero existen algunas situaciones en las que no será posible. Algunas de las situaciones en las que se tienen que escribir métodos nativos son:

- Cuando se quiera utilizar una característica dependiente de la plataforma en donde se vaya a ejecutar la aplicación, como puede ser por ejemplo una tarjeta de recogida de datos.
- Cuando se tenga que trabajar con librerías escritas en otros lenguajes y sean imprescindibles para la aplicación.
- Métodos críticos en ejecución. Sin embargo, con los sucesivos avances en tecnología Java cada vez es menos necesario implementar métodos nativos por esta razón.

Para llevar a cabo la funcionalidad requerida en este capítulo, se utiliza JNI por la segunda razón, puesto que el objetivo es desarrollar un interfaz gráfico en Java utilizando unas librerías escritas en C, tal y como se observa en la figura 4.9.

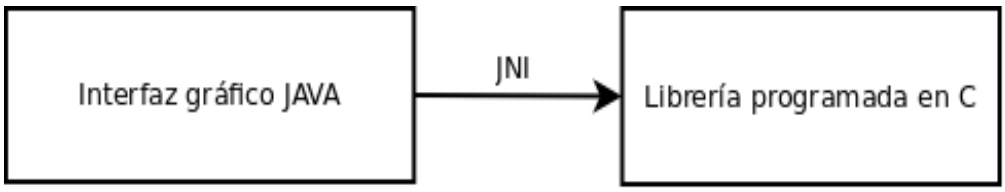

Figura 4.9 Relación entre librerías C y Java

4.7.3 MÉTODOS DESEADOS Y CLASES GENERADAS

Para implementar un soft-phone Java sería idóneo disponer de los siguientes métodos:

- Métodos de inicialización y finalización.

  Con estos métodos se inicia y finaliza todo el núcleo del soft-phone. Admiten parámetros tales como puertos de comunicación SIP, puertos de comunicación RTP, etc.

- Métodos de configuración.

  Con estos métodos se configuran las opciones generales de un sof-phone, tales como el volumen del altavoz, el volumen del micrófono, los dispositivos de entrada/salida, y otras características más técnicas como el tamaño del buffer de

jitter, el número de tramas por paquete o el rango de puertos UDP que se utiliza en RTP.

- Métodos de operación.

  Constituyen el verdadero centro de operación del soft-phone con funciones como llamar, colgar, rechazar o responder.

Utilizando las herramientas que ofrece JNI es posible realizar un mapeo entre estas funciones deseadas y las funciones disponibles en la librería SipXTapi definidas en el punto 4.2.1. A modo de ejemplo, la función llamar hace uso de las funciones sipxCallCreate() y sipxCallConnect(), mientras que la función colgar utiliza la función sipxCallDestroy().

En la figura 4.10 se representa la clase InterfazSIP cuyo objetivo es abstraer de la complejidad de la librería SipXTapi resumiendo toda su funcionalidad en un pequeño conjunto de métodos tales como inicializar, llamar o colgar entre otros. Dichos métodos serán directamente llamados desde la clase que se representa en la figura 4.11 (Procesos_InterfazSIP), que es directamente la clase generada con JNI. Los métodos de la clase representada en la figura 4.11 son los que podrán invocarse desde una clase Java.

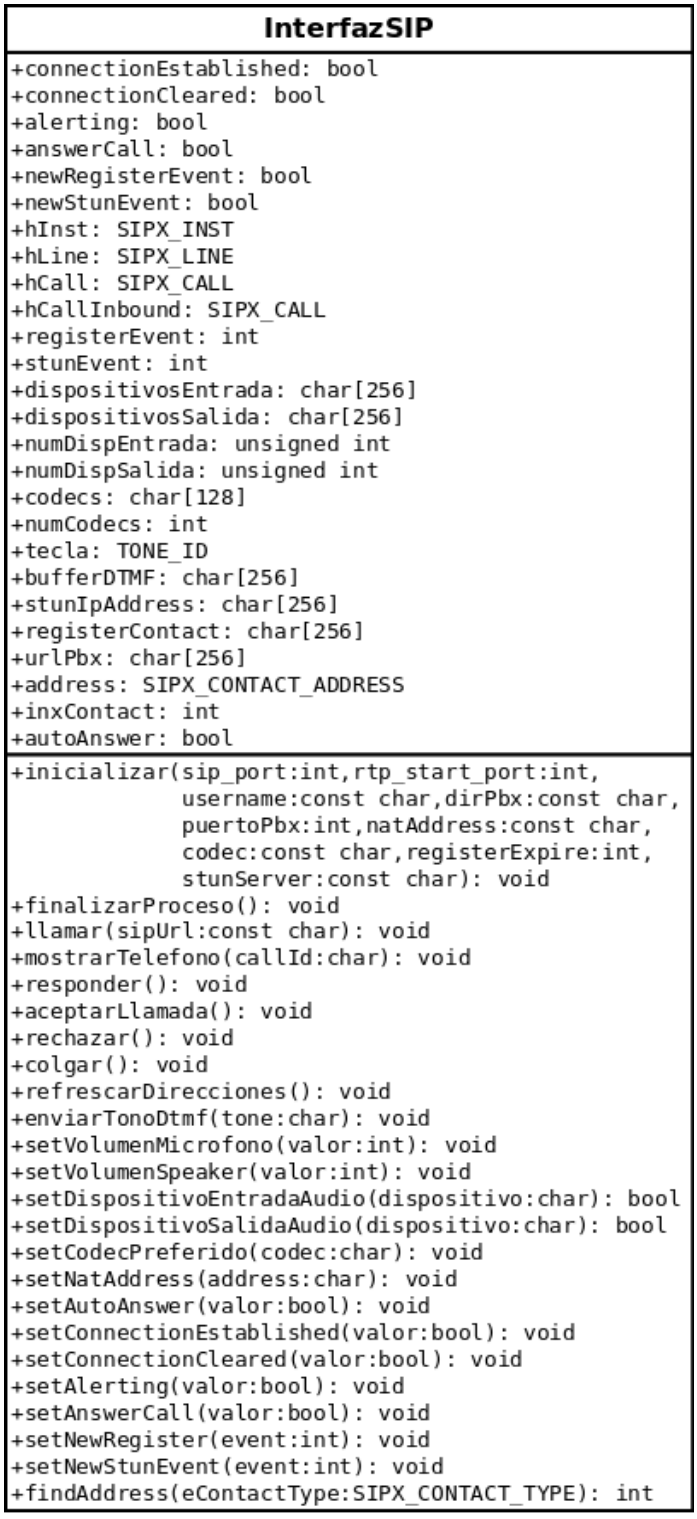

Figura 4.10 Clase InterfazSIP de JNI

Procesos_InterfazSIP

```
+Java_Procesos_InterfazSIP_inicializarSIP(env:JNIEnv*,obj:jobject,
                                          jpuertoSIP:jint,jpuertoUDP:jint,
                                          jusuario:jstring,jdir:jstring,
                                          jpuertoPbx:jint,jstun:jstring,
                                          jcodec:jstring,jregisterexpire:jint): void
+Java_Procesos_InterfazSIP_cicloComprobacion(env:JNIEnv*,obj:jobject): void
+Java_Procesos_InterfazSIP_finalizarSIP(env:JNIEnv*,obj:jobject): void
+Java_Procesos_InterfazSIP_llamar(env:JNIEnv*,obj:jobject,jstr1:jstring,
                                 jstr2:jstring): void
+Java_Procesos_InterfazSIP_responder(env:JNIEnv*,obj:jobject): void
+Java_Procesos_InterfazSIP_rechazar(env:JNIEnv*,obj:jobject): void
+Java_Procesos_InterfazSIP_colgar(env:JNIEnv*,obj:jobject): void
+Java_Procesos_InterfazSIP_getCallerId(env:JNIEnv*,obj:jobject): void
+Java_Procesos_InterfazSIP_setVolumenMicrofono(env:JNIEnv*,obj:jobject,
                                               value:jint): void
+Java_Procesos_InterfazSIP_setVolumenSpeaker(env:JNIEnv*,obj:jobject,
                                             value:jint): void
+Java_Procesos_InterfazSIP_setDispositivoEntradaAudio(env:JNIEnv*,
                                                      obj:jobject,
                                                      str:jstring): void
+Java_Procesos_InterfazSIP_setDispositivoSalidaAudio(env:JNIEnv*,
                                                     obj:jobject,
                                                     str:jstring): void
+Java_Procesos_InterfazSIP_setAutoAnswer(env:JNIEnv*,obj:jobject,
                                         value:jboolean): bool
+Java_Procesos_InterfazSIP_setAudioJitterDelay(env:JNIEnv*,obj:jobject,
                                               int1:jint,int2:jint): void
+Java_Procesos_InterfazSIP_setProfundidadBuffer(env:JNIEnv*,
                                                obj:jobject,value:jint): void
+Java_Procesos_InterfazSIP_setNumeroFramesPorPaquete(env:JNIEnv*,
                                                     obj:jobject,
                                                     value:jint): void
+Java_Procesos_InterfazSIP_setAnchoDeBanda(env:JNIEnv*,obj:jobject,
                                          value:jint): void
+Java_Procesos_InterfazSIP_setTipoCalidadServicio(env:JNIEnv*,
                                                  obj:jobject,
                                                  value:jint): void
+Java_Procesos_InterfazSIP_setRangoPuertosUDP(env:JNIEnv*,obj:jobject,
                                             value1:jint,value2:jint): void
+Java_Procesos_InterfazSIP_setPuertoServicio(env:JNIEnv*,obj:jobject,
                                            value:jint): void
+Java_Procesos_InterfazSIP_setNATRouter(env:JNIEnv*,obj:jobject,
                                       str:jstring): void
+Java_Procesos_InterfazSIP_enviarTonoDtmf(env:JNIEnv*,obj:jobject,
                                         value:char): void
```

Figura 4.11 Clase Procesos_InterfazSIP de JNI

Una vez que se han definido por completo todos los métodos y tras utilizar JNI para su implementación, se obtiene un fichero de librería de enlace dinámico “dll” para su uso con el lenguaje Java. Esta librería es directamente importable por las herramientas que Java proporciona.

De esta forma se ha conseguido trasladar toda la funcionalidad implementada en C al lenguaje Java. Es en este punto donde todas las posibles opciones que Java nos ofrece pueden usarse para desarrollar un interfaz gráfico vistoso y adecuado que muestre un soft-phone capaz de transmitir y recibir audio con la codificación G.729.

4.8 DISEÑO DE UN INTERFAZ GRÁFICO

El objetivo de este punto es dotar a un usuario de un interfaz gráfico agradable y funcional sin perder potencia y versatilidad. Para conseguir dicho objetivo se disponen de las herramientas propias del lenguaje Java, como la tecnología Swing [SWING], para generar interfaces de usuario. Esta tecnología proporciona al programador una sencilla biblioteca de funciones con la que es posible dibujar botones, etiquetas de texto y programar funciones de respuesta ante eventos determinados, como pueden ser el click de ratón o la pulsación de un determinado botón.

4.8.1 CARACTERÍSTICAS DESEADAS

Las características básicas de un buen interfaz gráfico podrían resumirse en:

- Facilidad de comprensión, aprendizaje y uso.
- Representación fija y permanente de un determinado contexto de acción.
- El objeto de interés ha de ser de fácil identificación.
- Diseño ergonómico mediante el establecimiento de menús, barras de acciones e iconos de fácil acceso.
- Las interacciones se basarán en acciones físicas sobre elementos de código visual o auditivo (iconos, botones, imágenes, mensajes, etc).
- Las operaciones serán rápidas, incrementales y reversibles, con efectos inmediatos.
- Existencia de herramientas de ayuda y consulta.

Se persigue diseñar un interfaz gráfico que reúna todas las cualidades anteriormente mencionadas, siendo de mucha importancia la simplicidad con respecto al usuario final. Dada la gran cantidad de parámetros técnicos que podrían configurarse, se ha determinado que el interfaz debe ser lo más sencillo posible para que el usuario final pueda acceder a

una versión funcional del software sin requerir unos conocimientos previos. De este modo, se propone un diseño gráfico como el que representa la figura 4.12.

Figura 4.12 Interfaz gráfico

El diseño realizado esta compuesto por las siguientes partes:

- Una botonera con los dígitos del uno al nueve, asterisco, almohadilla y la tecla R.
- Un botón de color verde con las funciones llamar y responder.
- Un botón de color rojo con las funciones colgar y rechazar llamada.
- Un botón de configuración.
- Una caja de texto donde se visualiza el número llamante y el número marcado, así como otros parámetros de interés como si el soft-phone está listo para llamar o tiene conexión de red.
- Dos barras de desplazamiento verticales en ambos laterales con las que modificar el volumen de entrada y del micrófono.

El diseño propuesto se asemeja mucho al ofrecido por cualquier terminal de telefonía móvil actual, asegurando por tanto su inmediato manejo por la gran mayoría de los usuarios. Del mismo modo se cumple con la facilidad en las funciones del teléfono, identificándose de forma sencilla las funciones de llamada y colgado con distintos iconos

de color verde y rojo, respectivamente, una intuitiva botonera y una pantalla de color muy similar a la ofrecida por cualquier pantalla de un terminal telefónico. Esta claridad en la funcionalidad que se ofrece hace que los errores de uso que un usuario produzca se minimicen.

4.8.2 PARÁMETROS DE CONFIGURACIÓN

Se necesitan unos mínimos parámetros de configuración para que la aplicación funcione correctamente. Dichos parámetros son clasificables en dos pestañas: una configuración básica y una configuración SIP. Se entiende por configuración básica aquella relacionada con parámetros que hacen referencia al propio programa, tales como los dispositivos de entrada y salida, el fichero de sonido a reproducir ante una llamada entrante, la dirección de red donde se encuentra el servidor SIP al que se conecta y el puerto de escucha del mismo. La configuración relativa a las posibilidades que la aplicación ofrece a un servidor SIP se especifica en otra pestaña para diferenciarla claramente de la configuración básica, puesto que se manejan parámetros tales como los algoritmos de codificación disponibles, puertos UDP de señalización y RTP desde donde se esperan datos y frecuencia de envío de paquetes SIP REGISTER.

Los parámetros de la configuración básica son los siguientes:

- Dispositivos de entrada y salida de audio.

  La aplicación escoge los primeros que el sistema operativo le ofrece cuando se le pregunta. En la gran mayoría de los casos suele corresponder con la tarjeta de sonido instalada en el sistema, aunque existen dispositivos comerciales externos conectables a un puerto USB (*Universal Serial Bus*) que implementan toda una tarjeta de sonido con auriculares y micrófono. Este tipo de dispositivos se suelen usar en entornos de distribución masiva de llamadas por su comodidad.

- Dirección IP donde se encuentra el servidor SIP al que ha de registrarse el soft-phone.

  Este dato es obligatorio si se quiere usar el soft-phone junto con un servidor SIP, lo que en la práctica constituye la mayoría de los casos.

- Puerto en el que el servidor SIP recibe los paquetes de señalización SIP.

  Por defecto es el puerto estándar 5060, aunque puede modificarse si fuera necesario.

- Fichero de sonido que se reproduce cuando se recibe el evento de llamada entrante.
- Modo de respuesta automática.

  En diversos entornos es una opción muy interesante que los teléfonos se descuelguen automáticamente al recibir una llamada entrante, como por ejemplo en entornos de emisión o recepción masiva de llamadas. Con este modo de trabajo activado y en el tipo de entorno anteriormente descrito, es común que a esta forma de trabajar se le denomine extensión software.

En la figura 4.13 aparecen representados los parámetros básicos de un modo claro.

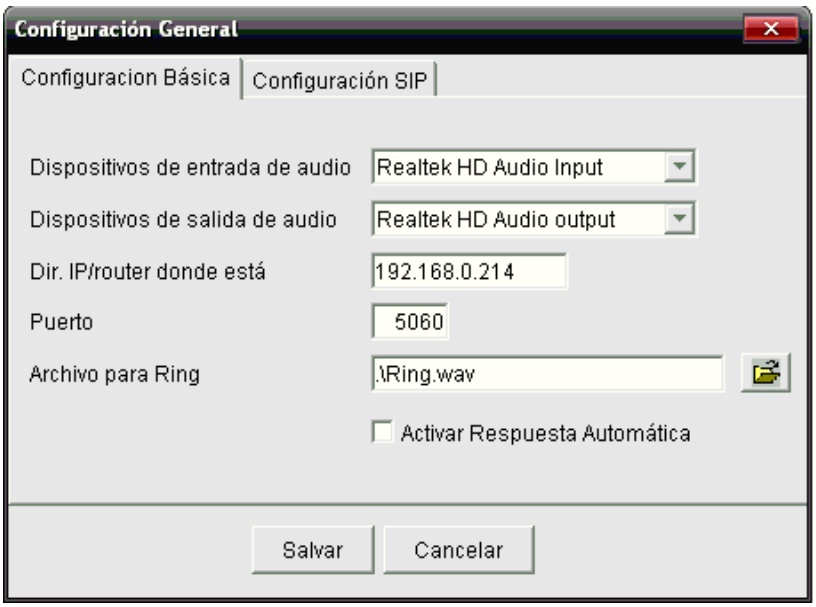

Figura 4.13 Parámetros de configuración básicos

Los parámetros de configuración SIP son los que representan en la figura 4.14.

Figura 4.14 Parámetros de configuración SIP

- Codecs disponibles.

  Representa en una lista desplegable todos los algoritmos de codificación que la librería SipXTapi ofrece: G.711 y el agregado G.729.

- Puerto SIP.

  Es el puerto UDP en el que se reciben los paquetes de señalización SIP.

- Puerto base UDP.

  Es el puerto UDP a partir del cual se recibirá todo el tráfico multimedia mediante el protocolo RTP.

- Frecuencia de registro.

  Establece un valor en segundos para el envío periódico de paquetes de registro SIP.

4.8.3 ESTRUCTURA DE CLASES

El diseño realizado consta de tres clases principales, un interfaz y las bibliotecas generadas en el apartado 4.7:

- Clase Principal.java.

  Es el punto de entrada a la aplicación, desde donde se instancia la clase PanelSoftphone.java y se configuran algunos parámetros como el modo de visualización del entorno gráfico (*look and feel*), el título de la ventana principal y sus dimensiones.

- Interfaz InterfazSoftphone.java.

  Define una serie de métodos que toda clase que implemente este interfaz debe tener. Estos métodos hacen referencia a lo que hay que hacer cuando se establece una conexión, cuando se corta una conexión, cuando el destino empieza a sonar y cuando se recibe una llamada.

- Clase PanelSoftPhone.java.

  Dibuja cada uno de los componentes de la aplicación tales como los botones numéricos (*dialpad*), el visor del número a llamar o número llamante, los indicadores del volumen de altavoz y micrófono, la reproducción del sonido de llamada entrante y el estado de la aplicación en general. Otra de sus funciones principales es relacionar los eventos generados por dichos componentes con las funciones propias de un softphone: llamar, colgar, descolgar, etc. Implementa la interfaz InterfazSoftphone. Es en esta clase donde se usa tecnología Swing para pintar los componentes gráficos.

- Clase InterfazSIP.java.

  Esta clase hace uso de toda la librería que se diseñó en el apartado 4.7 dedicado a JNI. Es el verdadero núcleo del programa puesto que se encapsula toda la comunicación con el exterior haciendo uso de todo el diseño realizado a lo largo de este capítulo.

La estructura de clases se representa como un diagrama de clases en la figura 4.15. Se puede comprobar cuáles son los atributos de las clases y las dependencias de las mismas.

Con esta estructura de clases es posible cumplir con toda la funcionalidad requerida, así como ampliarla muy fácilmente. Como ejemplo, si se quisiera utilizar otro protocolo de señalización distinto a SIP sólo sería necesario crear una clase más que implementara la interfaz InterfazSoftphone junto con los métodos de llamar, colgar y otros que son necesarios. Del mismo modo, si se quisiera cambiar el entorno gráfico por uno más depurado y amigable únicamente se requeriría modificar la clase PanelSoftPhone, sin necesidad de modificar ninguna otra parte del código. Esta gran mantenibilidad y posterior soporte del código es básica para cualquier proyecto de software, y la utilización de lenguajes que permitan usar técnicas de programación orientada a objetos como Java es muy recomendable en un elevado número de aplicaciones.

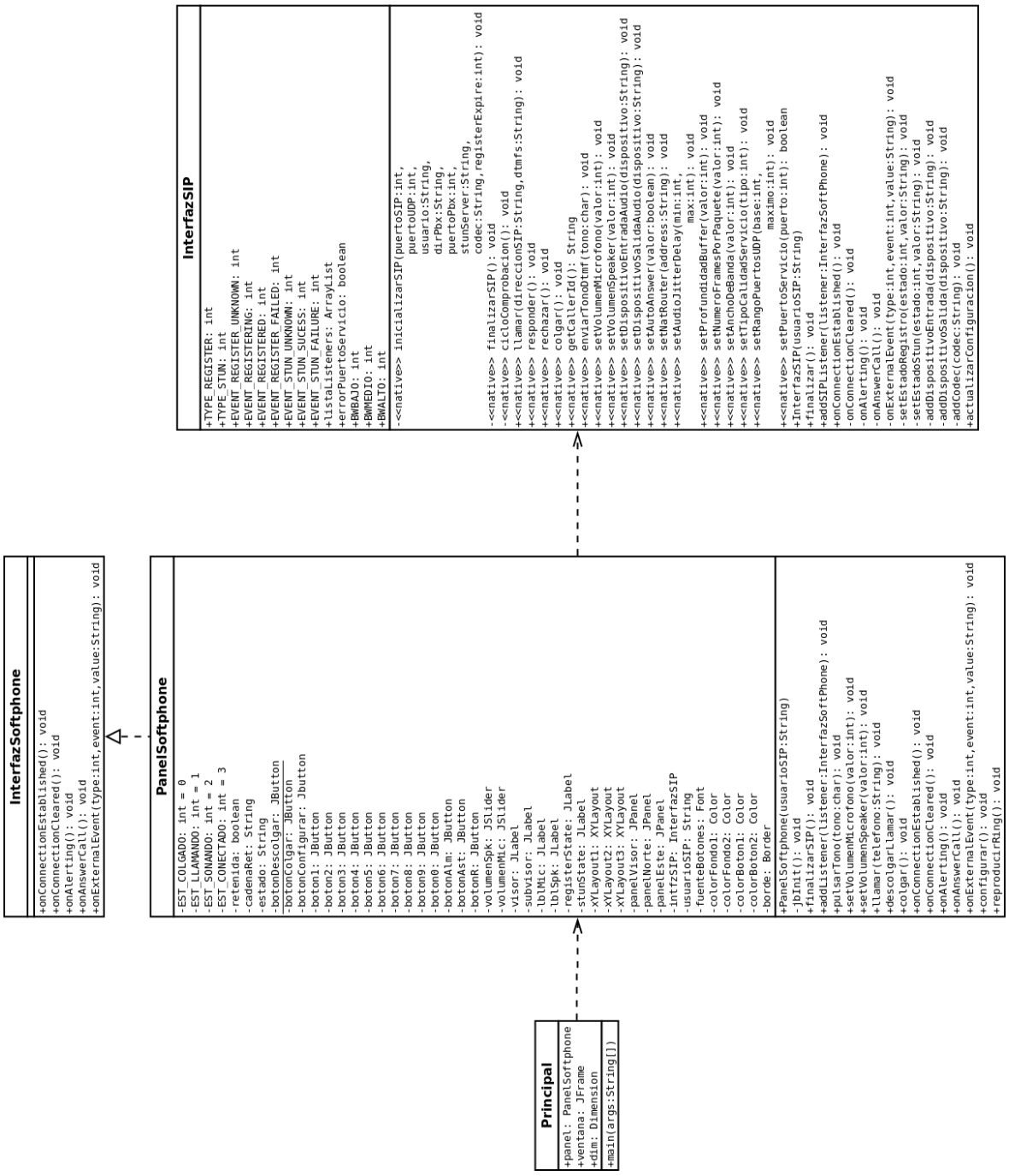

Figura 4.15 Diagrama de clases

4.8.4 PRUEBAS REALIZADAS

Para probar todo el diseño realizado se propone usar la misma batería de pruebas usada en el apartado 4.5 de este capítulo. El material que se utilizó consta de dos terminales VoIP Linksys, un adaptador para terminal analógico Linksys, una centralita VoIP Asterisk y un soft-phone comercial de la empresa Counter Path. Si todo el proceso de diseño e implementación es correcto, el resultado de las pruebas debería ser el mismo. Las pruebas que se propusieron fueron las siguientes:

- Llamada telefónica entre el terminal Linksys al soft-phone diseñado.
  Tras comenzar la comunicación desde el terminal Linksys y aceptando la llamada desde el soft-phone, se pudo comprobar como la señalización SIP entre los terminales se efectuaba correctamente ofreciendo los algoritmos de codificación G.711 y G.729 en la cabecera SDP. Ambos clientes SIP seleccionaron el algoritmo G.729 y la comunicación se estableció sin incidencias. El tráfico que se observó con un software de captura de paquetes tuvo características similares al tráfico medido en el apartado 4.5.
- Llamada telefónica entre el soft-phone X-lite y el soft-phone diseñado.
  El objetivo de esta prueba es comprobar que los algoritmos G.711 implementados de serie en la librería SipXTapi siguen funcionando correctamente con un soft-phone comercial. Se comprobó analizando los paquetes RTP transmitidos que la comunicación se estableció con el algoritmo G.711, tal y como indicó la cabecera SDP.
- Llamada telefónica entre el adaptador analógico Linksys PAP2T y el soft-phone diseñado.
  Los resultados son equivalentes a la primera prueba.
- Llamada entre dos extensiones VoIP de la centralita donde una de ellas es el soft-phone diseñado.
  La comunicación se estableció sin problemas y el flujo de paquetes RTP entre las dos extensiones fue similar al observado en las pruebas anteriores.

Como se esperaba, el resultado de las pruebas fue similar al del apartado 4.5. Se puede concluir por tanto que el comportamiento no ha variado desde un soft-phone realizado en el lenguaje C++ al utilizar herramientas de conversión de librerías a Java. Todas las

pruebas han sido satisfactorias y se puede disponer de un soft-phone realizado en Java que usa librerías que se han diseñado en C++ para implementar toda la pila de protocolos.

CAPÍTULO 5: MANUAL DEL USUARIO

5.1 INTRODUCCIÓN

En este capítulo se describe el proceso de instalación y configuración de la aplicación junto con una explicación de cada uno de los ficheros de los que se compone. También se describirán distintos casos en los que la aplicación puede funcionar con éxito y la configuración que sería necesaria para cada uno de ellos. Al ser un programa realizado en Java, se examina la versión mínima de la máquina virtual que se requiere para su correcto funcionamiento, así como las diferencias que pueden existir entre unas y otras versiones en cuanto a la ejecución de la aplicación se refiere.

A lo largo de este capítulo también se apunta que, dependiendo de la versión del sistema operativo Windows que se use, puede que sean necesarias algunas librerías externas más.

5.2 INSTALACIÓN DE LA APLICACIÓN

Para efectuar la instalación de la aplicación basta con guardar los siguientes ficheros en una carpeta:

- Fichero “softphone.jar”. Contiene todas las clases de la aplicación Java y es el verdadero núcleo de la aplicación.
- Fichero “jbcl.jar”. Es una librería que hace falta para el correcto funcionamiento de la aplicación. Está relacionada con el entorno gráfico Swing.
- Fichero “ring.wav”. Es un fichero de audio que se reproduce cuando se recibe una llamada entrante.
- Fichero “sipXTapi.dll”. Es la librería de enlace dinámico generada en el capítulo 4. Contiene toda la lógica SIP y RTP.
- Fichero “intfzSIP.dll”. El la librería generada en el capítulo 4 una vez que se utilizó el proceso de conversión a Java JNI.
- Fichero “config.prop”. Es un fichero de propiedades que contiene toda la configuración que se puede rellenar en las pantallas.

- Fichero “run.bat”. Es un fichero de procesamiento por lotes que simplifica la llamada a la aplicación, haciéndola mucho más cómoda de ejecutar.

Una aplicación Java se ejecuta llamando al método main que contiene la clase principal. Esta llamada se efectúa mediante el ejecutable de la máquina virtual java.exe cogiendo como argumentos de entrada los ficheros jar de los que depende y el nombre de la clase que contiene el método main. Para ejecutar la aplicación basta con ejecutar el fichero “run.bat”.

5.3 PANTALLAS Y CONFIGURACIÓN

5.3.1 PANTALLA PRINCIPAL

Una vez ejecutada la aplicación el aspecto es el de la figura 5.1:

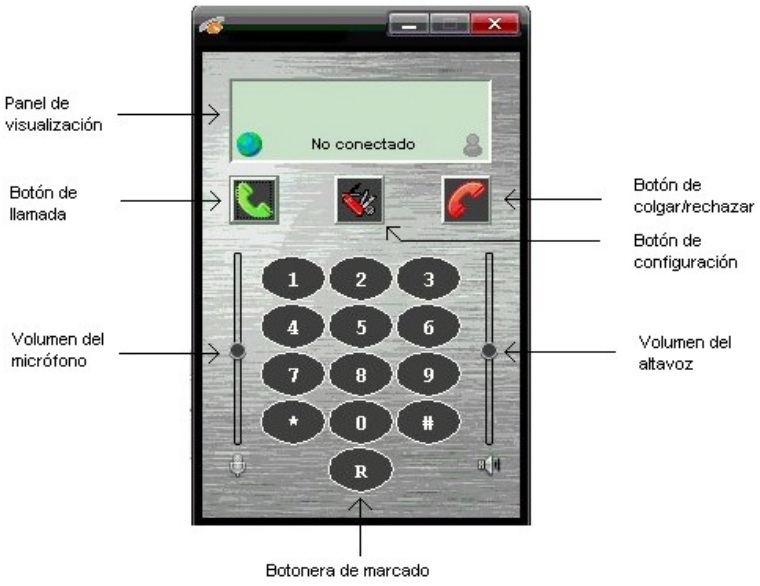

Figura 5.1 Ventana principal

Se observa un panel de visualización principal en el que se representan todos los eventos que se van produciendo a lo largo de la ejecución de la aplicación, como pueden ser los eventos de llamada recibida, número de teléfono marcado y distintos tipos de fallos. A izquierda y derecha se pueden ver unas barras de desplazamiento vertical que están directamente relacionados con el volumen del micrófono y del altavoz respectivamente. Se

observan tres botones superiores que hacen las funciones de llamar (botón izquierdo), colgar o rechazar (botón derecho) y de acceso a la configuración (botón central). Ya por último, se puede ver un marcador común de telefonía con todos los dígitos y los caracteres asterisco, almohadilla y R.

En el panel de visualización se observan dos iconos y una linea de texto. Con el icono izquierdo se representa la conectividad que tiene la aplicación con la red local, mientras que el icono derecho indica si la aplicación esta funcionando correctamente. Un sombreado en los iconos indica que existe un fallo bien en la conectividad a la red local o bien en la configuración de la aplicación. La linea de texto indica si hay una llamada en curso ("Conectado") o no ("No conectado"). Cuando se recibe una llamada entrante se reproduce un sonido identificativo y se representa el origen de la llamada en el panel de visualización.

5.3.2 CONFIGURACIÓN

Las pantallas de configuración son accedidas a través del botón central de configuración, y son las que se representan en la figura 5.2.

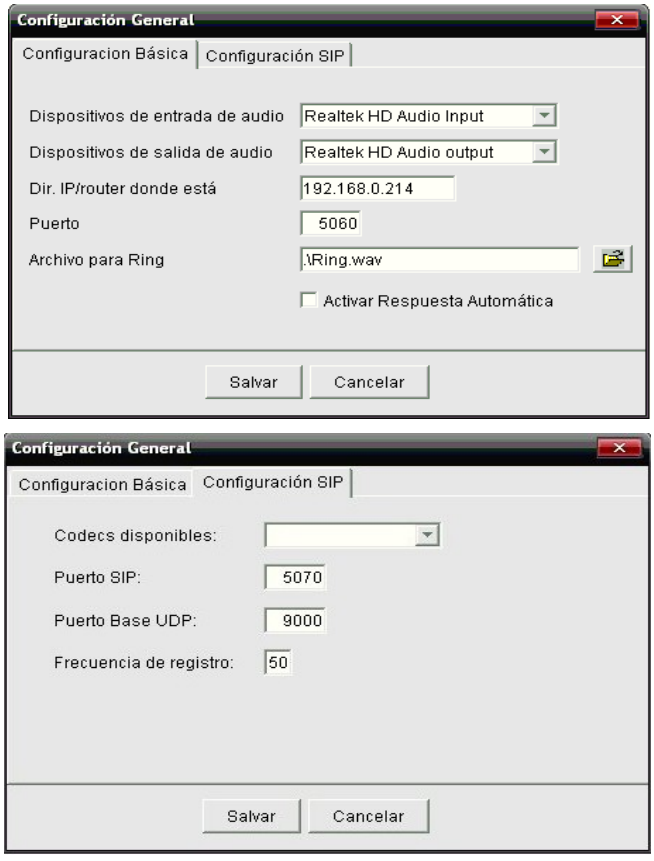

Figura 5.2 Pantallas de configuración de la aplicación.

La información está repartida en dos pestañas, donde una hace referencia a una configuración local de la parte de comunicaciones (pestaña "Configuración SIP") y en la otra se configuran parámetros básicos y de comunicación con servidores externos SIP.

Los datos que se han de rellenar son los que siguen a continuación:

- Dispositivos de entrada y salida de audio.

  Son los dispositivos de audio por los cuales se transmitirá y se capturará el audio recibido. Forma una lista desplegable en la que aparecerán todos y cada uno de los dispositivos que el sistema operativo detecte conectados, tales como tarjetas de sonido, auriculares u otros sistemas de reproducción.

- Dirección IP y puerto de recepción del servidor SIP al que enviar los paquetes de señalización.

  En el caso en el que la aplicación se encuentre en un entorno en el que tenga que conectarse a una centralita, en este campo se pondrá la dirección IP y el puerto de escucha de la misma.

- Fichero de sonido que se reproduce cuando se recibe una llamada.

  Ha de ser un fichero en formato de ondas a 8000 Hz monocanal muestreado a 8 bits.

- Modo de respuesta automática para contestar llamadas de forma inmediata.

  Esta opción es útil en entornos donde la llamada se necesite responder automáticamente y sin demora ninguna (como por ejemplo en entornos de distribución masiva de llamadas entrantes). De esta manera el tiempo de trabajo del teleoperador se maximiza. Por defecto está desactivado.

- Puerto de escucha de la aplicación para señalización SIP.

  Es el puerto que la aplicación abre que usa para recibir toda la información de señalización SIP. Por defecto se utiliza el puerto 5060.

- Puerto base de escucha de la aplicación para tráfico RTP.

  Es el puerto donde la aplicación recibirá los paquetes RTP que generan la voz recibida. Por defecto se utiliza el puerto 9000.

- Frecuencia de envío de paquetes de registro al servidor SIP.

  Cuando se trabaja en un entorno donde existe un servidor SIP, la aplicación necesita enviar paquetes de registro con una frecuencia determinada para indicar

que el terminal sigue disponible para el servidor SIP. Este valor por defecto está puesto a sesenta segundos.

- Codecs disponibles.

  Se representa en una lista desplegable el codec que se usará para la comunicación dentro de todos los codecs disponibles que hay: G.729 y G.711.

Con estos parámetros es posible configurar totalmente la aplicación. Los parámetros como la dirección IP y puerto del servidor SIP deben ser requeridos al administrador del sistema dependiendo del entorno de trabajo de la aplicación.

Cuando la configuración haya sido resuelta y se pulse el botón "Salvar", la aplicación se reiniciará para aplicar los valores seleccionados. A continuación la aplicación estará lista para trabajar.

5.4 REQUERIMIENTOS BÁSICOS

El requerimiento más importante de la aplicación consiste en que en el ordenador que se ejecute debe estar instalada la máquina virtual Java 1.3. Si la versión de la máquina virtual Java es superior la aplicación funcionará correctamente, aunque se observarán cambios en el dibujo de los componentes gráficos. No se observan incrementos en el rendimiento de la aplicación debidos al cambio de máquina virtual.

Cualquier versión de Windows para ordenadores personales que pueda ejecutar la máquina virtual Java 1.3 es válida, si bien existen problemas con algunas librerías de enlace dinámico. Para versiones anteriores a Windows XP es necesario descargar desde la página del fabricante la librería "msvc80.dll" para que funcione correctamente el código reunido en la librería "sipXTapi.dll". Para versiones posteriores o iguales a Windows XP no existen problemas de librerías de enlace dinámico y todo funciona correctamente desde el primer momento.

Para que la aplicación no tenga problemas de recursos computacionales es conveniente que la máquina donde se ejecute disponga de al menos 128 MB de memoria RAM (*Random Access Memory*) y un microprocesador Pentium III 800 Mhz. En otro caso, la codificación y decodificación de la voz con el algoritmo G.729 pueden verse afectadas.

No existen requerimientos desmesurados de espacio en disco duro puesto que la aplicación junto con todos los ficheros necesarios ocupan un espacio de 7 MB, más que asumible para cualquier máquina actual o de varios años atrás.

En cuanto a requerimientos de ancho de banda, depende del algoritmo de codificación configurado y del entorno de red en el que se ejecuta la aplicación, pero en ningún caso la aplicación consumirá mas de 150 Kbps de ancho de banda bidireccional. Es responsabilidad del administrador del sistema el correcto dimensionado de la red de comunicaciones para que la aplicación no sature la red y se produzcan efectos no deseados en el resto de las aplicaciones del sistema.

5.5 POSIBLES CASOS DE USO

La aplicación puede usarse en dos entornos bien diferenciados: llamadas directas desde terminal a terminal o llamadas entre terminales a través de una centralita VoIP.

- Llamadas directas.

  Se pueden establecer llamadas directas desde un terminal a otro simplemente escribiendo con el teclado el identificador SIP del terminal destino como se representa en la figura 5.3. En este caso la aplicación obviará los datos de dirección IP y puerto de recepción del servidor SIP que existen en la configuración.

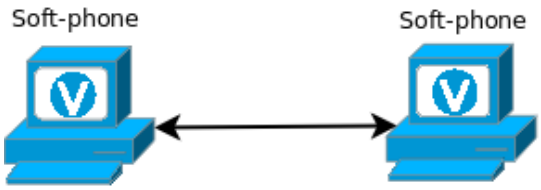

Figura 5.3 Comunicación entre dos terminales

- Llamadas entre terminales a través de una centralita VoIP.

  Este caso es el más habitual en entornos empresariales y domésticos. Todas las llamadas las gestiona una centralita que ejerce funciones de servidor central, puesto que conoce la localización de todos los terminales y redirige la señalización de la manera adecuada tal y como se representa en la figura 5.4. En este caso sólo es necesario marcar en el teclado el nombre del terminal a llamar para que la centralita se encargue de realizar el encaminamiento de la llamada a su destino. Es evidente que en este caso de uso de la aplicación adquieren sentido los campos de dirección IP y puerto de recepción mencionados en el punto 5.3.2.

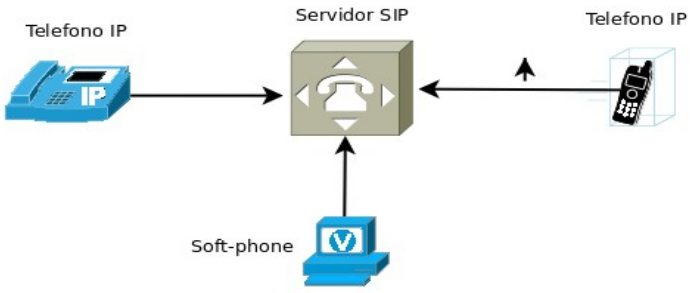

Figura 5.4 Entorno de centralita VoIP

CAPÍTULO 6: CONCLUSIONES Y LÍNEAS FUTURAS

6.1 CONCLUSIONES

La conclusión más inmediata es la correcta consecución de los objetivos planteados al inicio de este proyecto. Se ha conseguido desarrollar un soft-phone dotado con un algoritmo de codificación de bajo ancho de banda del tipo G.729. Dicho soft-phone se ha probado y testeado con productos comerciales de amplia difusión en el mercado para comprobar su interoperabilidad y funcionalidad obteniendo resultados satisfactorios.

Como labor previa al diseño ha sido necesario realizar un pormenorizado estudio del protocolo de señalización SIP y del protocolo RTP. Ha sido fundamental conocer la estructura de todos los paquetes implicados en la comunicación, el seguimiento de los diagramas de flujo existentes en la bibliografía y el profundo conocimiento de los datos transmitidos así como su razón.

Para la consecución de los objetivos se han seguido varias etapas en el desarrollo que han implicado el manejo de varias tecnologías relacionadas. Se ha utilizado una librería con licencia gratuita escrita en el lenguaje C++ que ha requerido de un exhaustivo análisis para comprender su funcionamiento interno. Una vez realizado este estudio previo se modificó para incluir una nueva funcionalidad. El producto de todo este trabajo dio fruto a una librería de enlace dinámico para generalizar y facilitar aún más el uso de toda la funcionalidad. Dicha librería fue utilizada en un lenguaje de programación distinto al de implementación de la librería, Java, utilizando la tecnología JNI que hace de puente entre código nativo y código interpretado Java. Por último, en un entorno de desarrollo Java, se ha dotado de un interfaz gráfico con tecnología Swing para el usuario final de la aplicación. Al usar tecnologías tan distintas se ha conseguido profundizar en el conocimiento de las mismas y en los distintos entornos de desarrollo utilizados.

El soft-phone diseñado fue probado con terminales comerciales de distintos tipos, centralitas de voz sobre IP y otros soft-phone comerciales. Esto implica la creación de un entorno de red de área local y la configuración de todos los equipos implicados: conmutadores, encaminadores, terminales de voz sobre IP, centralitas, etc. Como

conclusión adicional se ha aprendido a trabajar con dispositivos comerciales bastante comunes en el mundo de la voz sobre IP. Del mismo modo, el análisis de los datos recogidos fue realizado con software de monitorización especializado en entornos de redes, por lo que fue necesario estudiar la documentación oportuna para exprimir al máximo las posibilidades del software.

Como conclusión final, se ha conseguido diseñar una aplicación de audio conferencia con la integración de un algoritmo de codificación muy común en el mundo comercial. Es una aplicación capaz de trabajar con dispositivos comerciales sin ningún problema y puede proporcionar un significativo ahorro de costes al evitar la compra de equipos innecesarios cuando ya se dispone de un equipo informático en el que ejecutar la aplicación.

6.2 LÍNEAS FUTURAS

La evolución natural del software diseñado se orienta a los siguientes puntos:

- Sustitución del protocolo de señalización SIP bajo UDP por TCP.

  El protocolo de señalización SIP trabaja bajo el protocolo de transporte UDP, pero sería apropiado cambiar el protocolo por TCP y aprovechar todas las ventajas que este proporciona. Avanzando un paso más, sería adecuada la sustitución de UDP por el protocolo de transporte SCTP (*Stream Control Transmission Protocol*), que ofrece mejores técnicas para evitar la congestión, una estricta entrega de los paquetes ordenador y una baja latencia para transmisiones de datos en tiempo real.

- Sustitución del protocolo RTP por el protocolo SRTP.

  Es conveniente en la medida de lo posible cifrar todas las comunicaciones para asegurar la privacidad de los datos transmitidos.

- Algoritmos de codificación de audio adicionales.

  Pueden implementarse nuevos algoritmos de codificación que tengas otras características, tales como un menor ancho de banda, un ancho de banda adaptativo o transmisión de datos en alta calidad.

- Vídeo.

  Toda la arquitectura de protocolos permite usar el tipo de tráfico que se requiera sin distinguir audio o vídeo, pero sería de interés dotar a la aplicación de un interfaz

gráfico y de los algoritmos necesarios para que se pueda realizar una vídeo conferencia en tiempo real.

- Soluciones contra el problema del NAT.

  Existen protocolos y mecanismos que solucionan el problema del NAT, entre las que es obligatoria citar a ICE (*Interactive Connection Establishment*). Es un protocolo que se encuentra aún en estado de revisión, pero está a la espera de asignación de un RFC. ICE permite que los dispositivos involucrados en la sesión SIP prueben distintos medios o rutas para comunicarse entre sí y acuerden uno común. Gracias a ICE es posible que dos terminales que se encuentran en la misma LAN envíen el tráfico RTP de manera local sin realizar ninguna configuración exótica en el servidor. La inteligencia está en los terminales. Desgraciadamente, apenas existen en la actualidad implementaciones completas del protocolo en soft-phone comerciales, por lo que su funcionamiento dista de ser óptimo a nivel práctico.
- Funcionalidades adicionales.

  Es posible añadir funcionalidades adicionales tales como mensajería instantánea y presencia, transferencia de ficheros, pizarras compartidas, etc.

BIBLIOGRAFÍA Y REFERENCIAS

[1] Colin Perkins, *RTPAudio and video for the Internet*, Addison-Wesley, 2003.

[2] Jim Van Meggelen, Jared Smith, Leif Madsen, *Asterisk The future of telephony*, O'Reilly, 2005.

[3] David Merel, Barrie Dempster, David Gomillion, *Asterisk 1.6*, Packt Publishing, 2009.

[4] Rogelio Martínez Perea, *Internet Multimedia Communicactions using SIP*, Morgan Kaufman, 2008.

[5] J. Zukowski, *Programación en Java 2. J2SE 1.4*, Ediciones Anaya Multimedia, 2003.

[6] Al Stevens, Clayton Walnum, *Programación con C++*, Ediciones Anaya Multimedia, 2000.

[7] Rob Gordon, *Essential JNI*, Prentice Hall, 1998.

[SIPFoundry'10] "Is sipXecs IP PBX right for you?", Documento en formato HTML accesible por internet en la dirección: *http://sipx-wiki.calivia.com/*

[RFC3261'02] Network Working Group, "SIP: Session Initiation Protocol", Internet Engineering Task Force, Request for Comments (RFC) 3261, Junio, 2002.

[SPA942] Documento en formato HTML accesible por internet en la dirección *http://www.cisco.com/en/US/products/ps10039/index.html*

[PAP2T] Documento en formato HTML accesible por internet en la dirección *http://www.cisco.com/en/US/products/ps10029/index.html*

[ASTER] Documento en formato HTML accesible por internet en la dirección *http://www.asterisk.org*

[XLITE] Documento en formato HTML accesible por internet en la dirección *http://www.counterpath.com/x-lite.html*

[SWING] Documento en formato HTML accesible por internet en la dirección *http://java.sun.com/docs/books/tutorial/uiswing/*

[WIRESHARK] Documento en formato HTML accesible por internet en la dirección *http://www.wireshark.org/*

[COREPLAYER] Documento en formato HTML accesible por internet en la dirección *http://coreplayer.com/*

[LGPL] Documento en formato HTML accesible por internet en la dirección *http://www.gnu.org/licenses/lgpl.html*