UNIVERSIDAD AUTONOMA DE MADRID

ESCUELA POLITECNICA SUPERIOR

PROYECTO FIN DE CARRERA

Introducción del codec MELP en la plataforma IP PBX ***Asterisk®***

Cristina Romero Macías

Junio 2008

Introducción del codec MELP en la plataforma IP PBX ***Asterisk®***

AUTOR: Cristina Romero Macias

TUTOR: Jose Manuel Ruiz de Marcos

PONENTE: Doroteo Torre Toledano

ATVS - Biometric Research Lab.

Ingeniería Telecomunicación

Escuela Politécnica Superior

Universidad Autónoma de Madrid

Junio de 2008

PROYECTO FIN DE CARRERA

Título: *Introducción del codec MELP en la plataforma IP PBX* ***Asterisk***®

Autor: Dª Cristina Romero Macías

Tutor: D. José Manuel Ruiz de Marcos

Tribunal:

Presidente: Javier Ortega García

Vocal: José María Martínez Sánchez**.**

Secretario: Doroteo Torre Toledano

Fecha de lectura:

Calificación:

Palabras clave: Codec MELP, protocolo SIP, plataforma Asterisk, tecnología VoIP, softphone, codificación y decodificación.

Resumen: El objetivo principal de este Proyecto Fin de Carrera es la inserción en la plataforma IP PBX Asterisk, del codec de voz MELP de forma nativa, es decir, introducirlo en el núcleo de la centralita para poder realizar de forma automática la codificación y decodificación de la voz humana a la hora de realizar comunicaciones de voz sobre IP. Este proyecto también incluirá una pequeña implementación de una pasarela sobre Skype para posibilitar el desarrollo de un softphone con MELP, que permita la realización de las comunicaciones a través de Asterisk, funcionando como servidor, sobre la red IP.

Abstract: The main objective of this PFC is the insertion of the vocoder MELP natively into the platform IP PBX Asterisk, that is, entering it in the heart of the switchboard to carry out automatically the coding and the decoding of the human speech when making voice communications over IP. This project will also include a small implementation of a Skype Gateway to enable the development of a softphone with MELP, allowing the realization of communications across Asterisk, working as a server, over an IP network.

Agradecimientos

Quisiera agradecer la realización de este proyecto a todas esas personas que han aportado su granito de arena para que yo llegue hasta el final, empezando por mis padres, que tanto se esforzaron para que me aplicase y pudiera estudiar Telecomunicaciones. Siguiendo por los profesores que he tenido a lo largo de la carrera, que en muchas ocasiones han sido fuente de inspiración y sobre todo a mi tutor José Manuel Ruiz de Marcos y su empresa InTecDom que tanto me han enseñado y ayudado, y a mi ponente Doroteo Torre Toledano, por su cooperación e inestimable ayuda. Y por supuesto, tampoco me olvido de mis amigos y compañeros a los que quiero agradecer su ayuda y sostén durante toda la carrera.

Cristina Romero Macías
Junio 2008

INDICE DE CONTENIDOS

INDICE DE FIGURAS

INDICE DE TABLAS

1. Introducción

1.1 Introducción

En la actualidad se está gestando una increíble revolución en el campo de las tecnologías. Particularmente, en la industria de las telecomunicaciones, la revolución empezó con la centralita PBX open source llamada Asterisk y los sistemas basados sobre VoIP (Voz sobre IP) debido a su enorme potencial para cambiar la telefonía tal como la hemos conocido durante los últimos cien años.

Las Telecomunicaciones han sido, posiblemente, el último campo de la industria electrónica donde la revolución del GPL y el código abierto se han mantenido al margen. Los principales fabricantes y vendedores en este sector todavía continúan construyendo sistemas muy caros e incompatibles, programando con lenguajes antiguos y complicados en hardware casi obsoleto, donde la mayoría de las veces, si se quiere realizar algún proyecto realmente interesante es necesario pagar tasas adicionales por la concesión de licencias privadas, funcionalidades y aplicaciones limitadas. Junto al hecho de que los aparatos y software comprados no están personalizados.

Todo esto cambió con el desarrollo de la plataforma open source Asterisk. Esta plataforma creada por Mark Spencer y Digium a principios del siglo XXI fue una revolución debido primeramente, a que se distribuía bajo licencia GPL y segundo porque con Asterisk, nadie te limita como desarrollador e implementador de tus propias ideas para tu empresa, te garantiza una increíble flexibilidad y libertad a la hora de crear aplicaciones o mejorar las existentes.

La tecnología de VoIP siempre ha sido percibida como poco más que un método para la obtención y realización de llamadas a larga distancia más baratas, pero la realidad es que permite a la voz llegar a ser una aplicación más dentro de la red de datos, permitiendo a la gente la posibilidad de comunicarse de una manera más flexible y sin limitaciones.

Hoy en día, la voz se transmite, se almacena y manipula digitalmente, dándola un formato robusto y compacto que hace más simple su procesamiento y transmisión. Los codecs son los traductores que se encargan de convertir la voz analógica en una señal digital y transmitirla a través de la red de Internet. Hoy en día el desarrollo y la utilización de codecs de baja tasa binaria ha adquirido gran importancia debido a la falta de ancho de banda disponible en los

sistemas digitales de telecomunicación, usado principalmente para otras aplicaciones. En este punto, es interesante el desarrollo de codificadores de voz que permitan una mayor compresión de los datos y un ahorro de ancho de banda sin por ello perder calidad. El codec MELP, es un codec relativamente nuevo, que no ha sido utilizado todavía en grandes empresas ni ha tenido gran expansión, pero que consigue los dos objetivos antes mencionados.

1.2 Motivación

La principal motivación de mi proyecto es aunar dos grandes revoluciones en el sector de las telecomunicaciones y poder alcanzar uno de los principales objetivos: la transmisión de voz de calidad a baja tasa binaria a disposición de todos los usuarios gracias a Asterisk.

1.3 Objetivos

El objetivo de este Proyecto Fin de Carrera es la introducción de un codificador/decodificador de voz denominado MELP (Mixed Excitation Linear Prediction) en la plataforma IP PBX Asterisk para permitir el uso de este codec para la transmisión de voz a través de la tecnología VoIP (Voice over Internet Protocol) mediante el protocolo SIP.

Para lograr el objetivo propuesto, se ha instalado la plataforma Asterisk en el sistema operativo Debian, para a continuación realizar la inserción del código fuente del MELP mediante la creación de una interfaz basada en las funciones de traducción de Asterisk y por otra parte el reconocimiento de sus librerías. Para las pruebas, es necesaria a su vez la implementación de un softphone que realice llamadas con MELP.

Este sistema permitirá la realización y recepción de llamadas a través de Asterisk, que funciona como servidor, mediante canales SIP y encapsulado RTP, para la transmisión de voz sobre la red IP. La introducción en el núcleo de Asterisk del codec MELP, permite a su vez su uso en todas las aplicaciones desarrolladas en la plataforma que impliquen comunicaciones de voz y traducción.

1.4 Organización del PFC

El PFC consta de los siguientes capítulos:

- **Capítulo 1:** Introducción, motivación y objetivos del proyecto
- **Capítulo 2:** Análisis breve de la tecnología VoIP, protocolo SIP, Asterisk y del codec MELP.
- **Capítulo 3:** Descripción del diseño e implementación del proyecto
- **Capítulo 4:** Descripción de las pruebas realizadas
- **Capítulo 5:** Conclusiones obtenidas durante el desarrollo del proyecto. Relación con posibles mejoras e implementaciones para el futuro
- **ANEXO A:** Descripción del protocolo SIP
- **ANEXO B:** Funcionalidades y Elementos de una PBX.
- **ANEXO C:** Descripción de los diferentes codificadores de voz
- **ANEXO D:** Descripción de las tecnologías existentes para la síntesis de la voz
- **ANEXO E:** Descripción detallada del codec de voz MELP
- **ANEXO F:** Presupuesto de la realización del proyecto como producto para una empresa.
- **ANEXO G:** Código fuente del proyecto

2. Estado del Arte

2.1 Introducción

Las centralitas telefónicas actuales se basan todas en la tecnología VoIP e incorporan la mayor parte de los codecs con licencia libre que existen en el mercado. Además, poseen inteligencia de red, lo que permite la traducción entre los diferentes estándares de codificación. Este proyecto no trata específicamente sobre el codec de voz MELP, sino de su introducción en una centralita PBX.

El codec MELP es un codec militar propietario y su implementación a velocidades de 1.2 y 2.4 kbps es propiedad de Microsoft. Básicamente por estas razones, ninguna centralita en el mercado lo tiene incorporado. Por otra parte, MELP posee una complejidad similar con respecto a codecs como el G.729 o el CELP y consume muchos recursos en la CPU, por lo que ha sido imposible su implementación.

El interés básico ahora mismo sólo se encuentra en el campo militar, ya que filtra muy bien los ruidos porque reconoce la portadora y envolvente pero rechaza el resto, excelente para transmisiones tierra-aire. La transmisión hasta ahora se realizaba punto a punto.

Su interés para la integración en centralitas telefónicas no ha estado muy promovido debido, sobre todo, a la falta de recursos tecnológicos. Hoy en día, la tecnología de microprocesadores está en un punto que la inserción en centralitas es totalmente viable para conexiones en banda estrecha, gracias al dual-core e inteligencia de red IP donde los extremos no están necesariamente conectados directamente a la red. La inteligencia de red detecta MELP, lo codifica o decodifica o directamente lo enruta.

No hay conocimiento de momento que exista un producto similar en el mercado. Por lo tanto no hay estado del arte porque no ha habido hasta ahora procesadores lo suficientemente avanzados como para soportar la carga que conlleva la inserción del MELP sin que la calidad de la información se deteriore. De momento solo se podrían gestionar como mucho tres o cuatro conversaciones simultáneas en un sistema operativo Linux sin reducción de la calidad. Otra posibilidad para poder implantar el codec es su desarrollo en forma de hardware mediante FPGAs o DSPs liberando al procesador de la tarea de hacer la traducción.

2.2 Tecnología VoIP

2.2.1 Introducción

Hoy en día existen muchos problemas a la hora de gestionar el inmenso número de redes de telecomunicaciones existentes. Por esta razón es imprescindible el estudio de mecanismos que favorezcan la integración en una única red la transmisión simultánea de voz y datos. Esto es posible gracias a la aparición de nuevos y mejorados estándares y al abaratamiento de las tecnologías de compresión de voz. Los avances producidos en el campo de las telecomunicaciones y la gran expansión de Internet, está favoreciendo la implantación del transporte de voz sobre redes IP.

Los primeros intentos se centraron en la utilización de multiplexores que permitían utilizar las redes WAN de datos para la transmisión del tráfico de voz. Pero la falta de estándares, así como el largo plazo para su amortización imposibilitó la implantación de forma masiva de esta solución.

La necesidad de estándares y soluciones aceptadas a nivel mundial era evidente y junto con el éxito masivo de Internet, el protocolo IP parecía ser el medio de mayor importancia para esta integración. Esta solución recibe el nombre de VoIP (Voice over Internet Protocol).

La tecnología conocida como VoIP permite utilizar redes de datos IP para realizar llamadas de voz. Para este proceso se utiliza un software o hardware especializado que convierte la voz humana en señales digitales y las envía a través de las redes de datos, siendo la más común Internet.

Esta tecnología permite encapsular la voz en paquetes para ser transportados sobre redes IP sin necesidad de disponer de circuitos conmutados como es el caso de la telefonía tradicional (PSTN). La red convencional de telefonía se basa en la conmutación de circuitos, estableciendo circuitos físicos durante todo el tiempo que se mantenga la conversación. Esto implica la reserva de recursos hasta que la comunicación finalice no pudiendo ser utilizados por otras comunicaciones. Por otro lado, la telefonía IP no utiliza circuitos físicos, sino que envía múltiples conversaciones a través del mismo canal (circuito virtual) mediante codificación en paquetes y flujos independientes.

Desde que las primeras comunicaciones de voz aparecieron en 1995 gracias a las tecnologías desarrolladas por la empresa VocalTec, han aparecido distintos niveles de desarrollo hacia la convergencia de redes. Las más importantes son:

- Voz en Internet: ofrece servicios de telefonía sobre la red pública global mediante la interconexión de redes de conmutación de paquetes basadas en IP.
- Voz sobre IP (VoIP): ofrece servicios de telefonía sobre redes IP "privadas" sin interconexión a la red PSTN o a la Red Digital de Servicios Integrados (RDSI).
- Telefonía IP: ofrece servicios de telefonía sobre redes IP privadas en interconexión con la PSTN y/o RDSI
- Multimedia sobre IP (MoIP): ofrece servicios multimedia (vídeo, audio, imagen, etc.) sobre redes IP.
- Fax sobre IP (FoIP): ofrece servicios de transmisión de fax sobre redes IP.
- XoIP: ofrece la integración global de todos los servicios actuales y futuros que se puedan ofrecer sobre una red IP. El término X puede referirse a: F (fax), M (multimedia), V (voz) o D (datos).

2.2.2 Características Principales

Las redes IP parecen a priori la solución más rápida y factible para alcanzar la convergencia de redes debido sobre todo a la gran cobertura actual y a su aceptación por parte de los usuarios. La integración de la voz en redes IP mediante la tecnología VoIP aporta múltiples ventajas:

- Se administra una única red y permite el control del tráfico de la red (reducción de fallos y caídas en el rendimiento).
- Estándares abiertos e internacionales: Interoperabilidad, bajada de precios en proveedores y fabricantes de hardware VoIP.
- Calidad: Es posible ofrecer calidades parecidas a la red telefónica conmutada.
- Fiabilidad: Tanto en LAN como en Internet se puede garantizar una gran fiabilidad, aunque en Internet hay que tener en cuenta muchos más factores. Es independiente del tipo de red física que lo soporta.
- Gran expansión actual de las redes de datos (LAN, Internet, WIFI,..) y posibilidad de desarrollar nuevos servicios rápidamente. Ofrece servicios de valor añadido como el correo de voz (voicemail), centro de llamadas (call center) vía web, etc.
- Menor inversión inicial y menos costes para los clientes: Sociedad de consumo.

Sin embargo, existe un gran inconveniente que ha ralentizado la expansión de VoIP: la dificultad para ofrecer QoS (Quality of Service). Para realizar una transmisión de voz, es

necesario que todos los paquetes lleguen ordenados, que no haya pérdidas y que se garantice una mínima tasa de transmisión. Al ser un servicio en tiempo real es necesario diferenciar entre los paquetes de voz y los paquetes de datos, priorizar la transmisión y evitar que la transmisión no supere los 150 milisegundos (según recomendaciones de la ITU-T G.114).

La calidad de servicio se está logrando mediante la aplicación de los siguientes criterios:

- Supresión de silencios y VAD (Voice Activity Detection), otorgando mayor eficiencia a la hora de realizar una transmisión de voz ya que se aprovecha mejor el ancho de banda al transmitir menos información.
- Compresión de cabeceras aplicando los estándares RTP/RTCP
- Cancelación de eco
- Priorización de paquetes con mayor latencia
- Implantación de IPv6 para un mayor espacio de direccionamiento.

2.2.3 Arquitectura

La arquitectura para la transmisión de voz sobre una red IP, define los siguientes elementos fundamentales en su estructura:

- Terminales: Puntos finales de la comunicación. Pueden ser hardware o software.
- Servidor: provee el manejo y funciones administrativas para soportar el enrutamiento de llamadas a través de la red. El servicio se nombra en función del protocolo que soporta
- Gateways: enlace de la red VoIP con la red telefónica analógica o RDSI. Se encarga de adaptar de forma transparente las señales a redes VoIP y viceversa. El gateway tiene puertos LAN, interfaces FXO, FXS, E&M, BRI, PRI, G703/G.704. Soportan generalmente los protocolos SIP y H.323 y numerosos codecs. Las principales funciones que realiza son:
    - Discriminar la salida: enlutar por IP o conmutar por PTSN
    - Utilizar la línea como backup: En caso de fallo de Internet o del proveedor de servicios, las llamadas pueden ser enrutadas por la línea tradicional PTSN.
    - Recibir llamadas por PTSN y encaminarlas por IP.
- Red IP: provee conectividad entre todos los terminales. La red IP puede ser una red IP privada, una Intranet o Internet.

Los distintos elementos pueden residir en plataformas físicas separadas o en la misma, pudiendo encontrar juntos servidor y gateway.

2.2.3.1 Protocolos

La señalización en VoIP es muy importante a la hora de establecer, mantener, administrar y finalizar una conversación entre dos puntos. Además de ofrecer funciones de supervisión, marcado, llamada y retorno de tonos de progreso; también se encarga de proveer QoS en cada canal de transmisión.

Los protocolos son los "lenguajes" que se utilizan a la hora de negociar y establecer las comunicaciones de voz sobre IP. Los más importantes son:

- H.323: Protocolo desarrollado por la ITU-T para la interconexión de dispositivos, sistemas y servicios multimedia a través de redes IP que no pueden garantizar la provisión de una calidad de servicio concreta. Para el streaming se basa en RTP/RTCP. Existen cinco tipos de flujos de tráfico: audio, datos, video, control de comunicaciones y control de llamadas. Existe control y señalización para negociar las posibilidades de la comunicación:
    - Negociación de codecs
    - Verificación para establecer canales multimedia
    - Control de secuencia.
- IAX (Inter Asterisk eXchange): Protocolo estandarizado por la empresa Digium para la centralita Asterisk. El objetivo es minimizar el ancho de banda utilizado en la red IP y proveer con un soporte nativo para ser transparente a los NATs. Hoy en día se utiliza IAX2 porque es más robusto y soporta con cantidad de codecs y datos:
    - NAT: Media y señalización por el mismo flujo de datos sobre un puerto UDP. Se evitan los problemas derivados del NAT y no es necesario abrir rangos de puertos para RTP.
    - Trunking (truncado): posibilidad de enviar varias conversaciones por el mismo flujo de datos con reducción del ancho de banda.
    - Cifrado.
- SIP (Session Initial Protocol): Protocolo creado por el IETF en 1999 para VoIP, control de llamadas multimedia e implementación de servicios telefónicos avanzados. Es básicamente un protocolo para la iniciación, modificación y terminación de sesiones de comunicación multimedia entre usuarios.
- MGCP (Media Gateway Control Protocol): Protocolo desarrollado por el IETF que está basado en un modelo maestro/esclavo donde el servidor es el encargado de controlar al gateway. De esta forma se consigue separar la señalización de la transmisión de la información, simplificando la integración con el protocolo SS7.
    - Control de señalización de llamada escalable.

- Control de calidad de servicio QoS integrado en el gateway o en el controlador de llamadas MGC.
- Sesión punto a punto o multipunto.

2.2.3.2 Codecs

Los codec o codificadores de audio se utilizan para digitalizar, comprimir y codificar la señal de audio analógica para poder ser transmitida por la red IP. Los codecs son algoritmos matemáticos implementados en software. Existen diversos modelos utilizados en VoIP dependiendo del algoritmo escogido en la transmisión, la calidad de la voz, el ancho de banda necesario y la carga computacional. El principal objetivo es aunar la eficiencia y la calidad de la voz. El sistema auditivo del ser humano es capaz de captar las frecuencias comprendidas entre 20 Hz y 20 kHz y la mayoría de codecs procesan la información dentro de la banda de 400 Hz-3,5 kHz para que a la hora de reconstruir la señal, ésta siga siendo inteligible.

Entre los codecs más comunes se encuentran:

- G.711: Estándar de la ITU-T para la compresión de audio para telefonía. Representa las señales de audio mediante muestras comprimidas en una señal digital con tasa de muestreo de 8000 muestras por segundo con un flujo de datos de 64 kbps. El algoritmo es logarítmico y existen dos leyes:
  - μ-law: Usado sobre todo en Norte América y Japón. Codifica cada 14 muestras en palabras de 8 bits
  - a-law: Usado en Europa y en el resto del mundo. Codifica cada 13 muestras en palabras de 8 bits
- G.723: Algoritmo estandarizado por la ITU-T en 1995 puede operar a 6.3 kbps o 5.3 kbps.
- G.726: Estándar de la ITU-T, conocido también como ADPCM (Adaptative Differential Pulse Code Modulation), sustituyó a G.721 en 1990. Permite trabajar con velocidades de 16, 24, 32 y 40 kbps. Este codec proporciona una disminución considerable del ancho de banda sin aumentar en gran medida la carga computacional.
- G.729: Se usa sobre todo en aplicaciones de Voz sobre IP por los bajos requerimientos en ancho de banda. Opera con tasas de 8 kbps pero existen extensiones para tasas de 6.4 y 11.8 kbps para peor o mejor calidad de voz respectivamente.
- GSM (Global System Mobile): Estándar que opera a 13 kbps con una carga de CPU aceptable. Inicialmente fue desarrollado para la telefonía móvil.

- iLBC (Internet Low Bit rate Codec): Algoritmo complejo desarrollado por Global IP Sound (GIPS) que ofrece una buena relación ancho de banda/calidad de voz a cambio de una mayor carga computacional. Opera a 13.3 kbps y 15.2 kbps.
- Speex: Implementa un algoritmo capaz de variar la velocidad de transmisión dependiendo de las condiciones actuales de la red (VBR: Variable Bit Rate). El ancho de banda puede variar desde 2.15 a 22.4 kbps.
- MP3 (Moving Picture Experts Group Audio Layer 3 Enconding Standard): Codec de audio optimizado para música y no para telefonía creado por la ISO. Es utilizado sobre todo en los teléfonos IP para la música en espera.

2.2.4 Elementos Implicados

Hay varias maneras de implementar la tecnología VoIP para usar Internet como teléfono:

- Adaptadores IP: Son dispositivos hardware que permiten la conexión de teléfonos analógicos a la red IP mediante la tecnología VoIP. Entre los tipos más comunes de adaptadores encontramos:
    - ATA (Analog Telephone Adapter): Es la manera más fácil de usar la tecnología VoIP, ya que permite aprovechar los teléfonos analógicos actuales transformando su señal en paquetes digitales de acuerdo con protocolos sobre IP. El hardware es simplemente un adaptador con dos conectores: uno FXS que se conecta al teléfono analógico y el otro se conecta al enlace a Internet mediante LAN. Soporta normalmente los protocolos SIP o IAX2 así como varios codecs.
    - FXS to USB: Es un dispositivo hardware que permite la conexión de un teléfono tradicional al PC para enviar y recibir el audio. Es necesaria la instalación de un softphone que trabaje sobre VoIP.
    - FXO to USB: Este dispositivo está casi siempre relacionado con el softphone propietario Skype. Permite reenviar las llamadas recibidas por el softphone por una línea analógica.

Figura 2.1: Adaptador Linksys PAP2

- Teléfonos IP: Son teléfonos especializados que físicamente se parecen a cualquier teléfono normal sin embargo, en vez de tener conectores estándar RJ-11 tienen un conector RJ-45 Ethernet para conectarlos directamente a una red IP. Algunos incluso no tienen conector que ya funcionan mediante una red inalámbrica.
    - Normalmente sólo soportan un único protocolo de VoIP (IAX, SIP, H233…)
    - Soportan diferentes codecs de audio: GSM, G.729, etc.
    - Se configuran desde los menús del propio teléfono o desde la interfaz web.
    - Algunos teléfonos soportan funcionalidades avanzadas: llamada en espera, trabajar como switch, auto configuración desde un servidor remoto etc.

Figura 2.2: a) Thomson ST 2030 b) Philips 3211 VoIP

- Softphones: Son programas (software) que permiten llamar desde un ordenador a otro ordenador o a un teléfono utilizando tecnologías de voz sobre IP. Se ejecuta en estaciones o en servidores de trabajo:
    - El audio puede ser capturado desde: Micrófono, Entrada de línea, Dispositivos de entrada de audio USB o Bluetooth.
    - Softphones Propietarios: Protocolos estándar, Protocolos propios abiertos y Protocolos propios cerrados. El más conocido es Skype.
    - Softphones Libres: Protocolos estándar y abiertos.
    - Integración con el entorno de trabajo, con plataformas de acceso y validación de usuarios (LDAP)
    - Permiten la importación/exportación de datos, varias conversaciones simultáneas e incluso varias líneas.
- Centralitas IP: Son centralitas de telefonía que permiten usar la tecnología VoIP (mixtas) o exclusivamente IP (puras). La más extendida actualmente es la plataforma Asterisk.

Figura 2.3: Icono de la plataforma Asterisk

Los principales agentes implicados a la hora de realizar una comunicación sobre VoIP son:

- Usuarios VoIP: realizan llamadas mediante la tecnología VoIP
- Proveedores de VoIP: permiten realizar llamadas desde una red VoIP a una red telefónica tradicional. Cobran por el servicio.
- Carriers de VoIP: venden las rutas de VoIP a los proveedores.
- Terminadores de VoIP: venden las líneas para hacer llamadas de telefonía tradicional a los proveedores de VoIP.
- Integradores de soluciones VoIP: Conectan centralitas a VoIP, servidores dedicados a servicios adicionales, conexiones CRM-> VoIP, softphones, etc.

La principal tarea de los proveedores de voz sobre IP es el de hacer de pasarela hacia la red telefónica pública (POTS) con unos costes muy reducidos:

- Soportan protocolos estándar (SIP, IAX2, H323) y propietarios (SKYPE)
- Soportan determinados codecs: GSM, G.729, etc.
- Permiten realizar más de una llamada a la vez.
- Las llamadas entre usuarios del mismo proveedor suelen ser gratuitas.

2.3 Centralitas Telefónicas IP PBX y Asterisk

Una Centralita Telefónica IPBX (Intranet Private Branch/Business eXchange) es un sistema avanzado de telefonía que permite realizar llamadas a través de la red IP. Las conversaciones de voz son codificadas para poder realizar la transmisión.

2.3.1 Introducción

Una central PBX es una central telefónica conectada a la red pública de teléfono (RTC) por medio de líneas troncales que permite al usuario o a una empresa conectar sus aparatos telefónicos independientemente del proveedor de servicios, es decir, no tienen asociada ninguna central de teléfono pública ya que el PBX actúa como tal, lo que permite a su vez realizar

llamadas internas mediante conmutación directa sin necesidad de salir al exterior. La principal ventaja es que se obtiene una reducción de los costes en telefonía. También es posible conectar dos centrales distintas mediante la red de Internet.

Las primeras centralitas PBX requerían de una persona que se encargara de la conexión de los distintos cables para poder realizar la comunicación entre los terminales. Estas centrales se denominaban PBMX (Manual PBX). Los avances tecnológicos permitieron crear las PABX (Automatic PBX) que mediante un sistema electromecánico de conmutación automático permitían prescindir de los operadores humanos.

Las extensiones de una PBX son los dispositivos conectados a ella y pueden ser tanto teléfonos, como faxes o módems. Además, es posible conectar cierto número de líneas troncales o varías PBX entre sí para realizar llamadas entre sedes.

La tendencia actual en las PBX es incorporar a las centralitas la posibilidad de transmitir la voz sobre redes de datos. Este nuevo modelo permite la reducción de costes gracias a la gestión de una única infraestructura, la simplificación de la integración y la posibilidad de ampliar el número de servicios de valor añadido.

El término IP-PBX (Internet Protocol PBX) hace referencia a las centralitas capaces de transmitir la voz sobre redes IP basándose en la tecnología VoIP. Es un sistema de telefonía diseñado para entregar voz sobre una red de datos a la vez que interopera con la red PSTN.

En esencia, los gateways de VoIP se combinan con las funcionalidades tradicionales de las PBXs para permitir a las empresas gestionar su intranet para reducir costes a la hora de realizar llamadas a larga distancia, gozar de los beneficios de una sola red para voz y datos y de las características avanzadas CTI.

Una centralita IP PBX puede implementarse como un dispositivo hardware, o virtualmente como un sistema software, como Asterisk o FreeSWITCH. Dado que la mayor parte de las funcionalidades de una IP PBX se presentan en software, es relativamente sencillo y barato añadir nuevas funcionalidades, tales como conferencias, control XML-RPC para llamadas en vivo, IVR (Interactive Voice Response), TTS/ASR (text to speech/automatic speech recognition), habilidad para la interconexión de circuitos tanto analógicos como digitales a través de la red PTSN, protocolos de VoIP como SIP, IAX, H.323, Jabber, GoogleTalk, etc.

2.3.2 IP-PBX Asterisk

No hace mucho tiempo, las telecomunicaciones tanto de voz como de datos, así como el software, eran todos propietarios, controlados por unas pocas empresas que creaban las tecnologías y otras que utilizaban los productos y servicios. A finales de la década de los 90, los datos se expandieron gracias a la propagación de Internet, los precios bajaron y surgieron nuevas e innovadoras tecnologías, servicios y empresas.

También surgió el concepto de software libre GNU con licencia GPL (General Public Licence) y una plataforma de software llamada Linux. Sin embargo, las comunicaciones de voz, omnipresentes en las telecomunicaciones eran todavía un campo propietario. El surgimiento de las centralitas telefónicas IP PBX junto con el trabajo desarrollado por Mark Spencer (Digium), dieron lugar a la centralita software IP PBX de código abierto y gratuito Asterisk.

Asterisk es líder mundial como motor de telefonía no propietaria y herramientas para su uso. Ofrece una flexibilidad sin precedentes en el mundo de las comunicaciones y facilita tanto a desarrolladores como integradores la tarea de crear soluciones avanzadas en comunicaciones y de forma gratuita. Al ser GPL está disponible para su descarga, teniendo cada cierto tiempo la posibilidad de versiones nuevas y mejoradas.

2.4 Procesamiento Digital de Señales

2.4.1 Introducción

El Procesamiento Digital de Señales es una tecnología innovadora que ha introducido cambios revolucionarios en un amplio abanico de ámbitos: comunicaciones, imágenes médicas, localización mediante radar y sonar, la música de alta calidad, etc. En cada una de estas áreas se ha desarrollado una tecnología propia DSP con sus propios algoritmos matemáticos y técnicas especializadas.

El Procesamiento Digital de Señales tiene que ver con los aspectos teóricos y prácticos que conlleva la representación de la información a la hora de transmitir señales en forma digital o bien a la hora de utilizar hardware digital para extraer información o manipular las señales para transmitirlas.

Actualmente, el procesamiento digital de señales abarca muchos campos:

- Telecomunicaciones: Transmisión inalámbrica, Telefonía digital, modems, etc.
- Audio Digital: CD, MPEG, DCC, etc.
- Vídeo Digital/Imagen: HDTV, DVD, JPEG, etc.
- Sistemas basados en voz: Reconocimiento, procesado y síntesis del habla
- Multimedia: PC Fax, Gráficos, teleconferencia, software por radio, etc.
- Sistemas Biomédicos: BCG-electrocardiograma, EEG-electroencefalograma, etc.
- Industrial: Control de motores, controladores, cancelación del ruido.
- Defensa: Sistemas de sonar, radares, comunicaciones seguras.
- Automóviles: Navegación por GPS, motores.

Las señales que encontramos en la realidad física en nuestro alrededor casi invariablemente tienen un dominio de definición continuo. Nosotros modelamos la señal procedente del habla como una función continua de amplitud, donde el domino de definición es un intervalo finito de números reales. En los sistemas electrónicos modernos se trata, en esencia, con bastantes señales continuas en un modo digital

Los métodos de Procesamiento Digital de Señales fundamentalmente requieren que las señales estén cuantificadas en el dominio discreto del tiempo y representadas como una secuencia de palabras que consisten en ceros y unos. En la naturaleza, las señales no están cuantificadas y varían en el tiempo. Las señales tales como las ondas del aire procedentes del habla son convertidas por un traductor en la correspondiente señal eléctrica analógica. Consecuentemente, es necesario realizar una conversión de la señal analógica a una representación digital o viceversa si se necesita una salida analógica.

Para realizar la conversión de las señales continuas en un modo digital, no es necesario tratar directamente con estas señales, sino que basta con utilizar versiones muestreadas de las mismas. Mediante el muestreo se obtiene un conjunto discreto de puntos y gracias a la gran precisión aritmética de los sistemas digitales, se puede grabar un valor aproximado de cada punto del conjunto muestreado. Si definimos el muestreo como un proceso que restringe la señal a un conjunto discreto, sin la cuantificación de los valores, se puede describir el método para relacionar señales continuas con sus versiones muestreadas. El número de niveles de cuantificación usados para representar la señal analógica y la tasa a la cual es muestreada está en función de la precisión con la que se desea recibir la misma, el ancho de banda disponible y el coste del sistema.

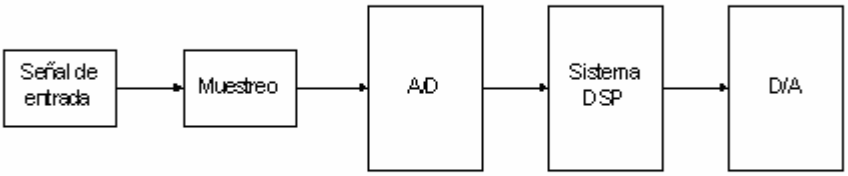

Figura 2.4: Modelo de procesamiento de las señales

El conversor analógico/digital (A/D), codifica una señal analógica en una palabra digital. Los convertidores convencionales primero muestrean la señal analógica variante en el tiempo mediante el teorema de muestreo de Nyquist, donde la mínima tasa de muestreo es el doble de la frecuencia de la mayor frecuencia que contenga la fuente. La tasa de muestreo es decisiva a la hora de escoger una arquitectura u otra. Una vez muestreada la señal, ésta se pasa a través del convertidor que cuantifica la señal en palabras digitales. Una vez obtenida la señal digital, se aplican los métodos de procesamiento digital con sus respectivos algoritmos. Dependiendo del sistema la salida puede ser usada directamente en forma digital o convertida de nuevo a analógica. Un filtrado digital puede producir una señal analógica a su salida, mientras que el procesamiento de voz puede pasar la señal producida por el método DSP a un sistema software para una segunda etapa de procesamiento.

Los pasos a seguir a la hora de realizar el procesamiento digital de señales son:

1. Filtrado lineal: Eliminación del ruido y de componentes no deseados que interfieren en la señal como puede ser la discriminación de fase o frecuencia.
2. Transformación de la señal: Transformación del dominio de la señal a frecuencia.
3. Filtrado / Mejora no lineal de la señal: Eliminación de los pulsos de ruido mediante filtrado con orden n.
4. Análisis de la señal/Interpretación/Clasificación: Clasificación automática de las características de la señal.
5. Compresión/Codificación: Reducción del ancho de banda o almacenamiento de los requisitos para una señal.
6. Grabación/Reproducción: En dispositivos tales como CDs.

2.4.2 Procesamiento de la voz

El Procesamiento de la voz es la ciencia que estudia las señales de voz y los métodos de procesamiento de estas señales. Las señales son procesadas normalmente mediante una representación digital en donde el procesamiento de la voz puede verse como una intersección del procesamiento digital de señales y el procesamiento del lenguaje natural del ser humano.

El procesamiento de la voz puede ser dividido en las siguientes categorías:

- Generación y reconocimiento del habla: Trata con el análisis del contenido lingüístico de una señal de voz.
- Reconocimiento del hablante: La finalidad es reconocer la identidad del hablante
- Mejora de las señales de voz: Mediante la reducción de ruido, cancelación del eco, etc.
- Codificación del habla: Método de compresión de datos muy importante en el campo de las telecomunicaciones
- Síntesis del habla: Síntesis artificial del habla generando una voz computerizada.
- Mejora del habla: Mejora la calidad que se percibe de la señal de voz quitando los efectos negativos del ruido, del equipo de grabación, del procesador, etc.

La generación de la voz y su reconocimiento son usados para comunicar entre las personas y las máquinas. Antes que usar las manos o los ojos, siempre se tiene a usar la boca y los oídos. A la hora de generar la síntesis del habla se usan dos aproximaciones: la grabación digital y la simulación del tracto vocal. En la grabación digital, la voz del hablante es digitalizada, almacenada y normalmente comprimida. Durante la reproducción, los datos almacenados son descomprimidos y convertidos en una señal analógica. Los simuladores del tracto vocal son más complejos ya que el tracto es una cavidad acústica con frecuencias resonantes determinadas por el tamaño y la forma de las cámaras. Los simuladores operan mediante la generación de señales digitales que se asemejen a la excitación de los sonidos producidos. Las características de la cámara resonante son simuladas pasando la señal excitada a través de un filtro digital con resonancias similares.

El reconocimiento de la voz es inmensamente más difícil que la generación. Para ello se utiliza el procesamiento digital de señales que resuelve el problema mediante dos aproximaciones: Extracción de las características y su conexión. Cada palabra de una conversación es aislada y analizada para identificar el tipo de excitación y las frecuencias resonantes. Estos parámetros son comparados con ejemplos previos de palabras habladas para identificar la comparación más exacta. A menudo, estos sistemas están limitados a unos cientos de palabras y pueden aceptar pausas no sonoras entre palabras.

Las características de la señal procedente del habla surgen de las propiedades y limitaciones del aparato vocal del ser humano. Es por lo tanto necesario en el diseño de estas aplicaciones tener en cuenta la generación del habla.

El habla se produce por la excitación variante con el tiempo acústica de la cavidad vocal (tracto vocal), la cual pertenece a la región de la boca limitada por las cuerdas vocales y los labios. Los diferentes sonidos son producidos por el ajuste de ambos tipos de excitación y la forma del tracto vocal. Hay varios métodos para clasificar los sonidos del habla. El principal a la hora de hablar de codecs de voz es el basado en el tipo de excitación que producen:

- Sonidos sonoros: Son producidos por la excitación del tracto en soplos casi periódicos de aire producidos por la vibración de las cuerdas vocales de la laringe. Las cuerdas vocales modulan el chorro de aire procedente de los pulmones a una tasa que como mínimo tiene un valor de 60 veces por segundo para adultos y hasta 400 o 500 veces por segundo para niños. Todas las vocales son producidas de esta manera.
- Sonidos Nasales: Son también señales sonoras pero parte del flujo de aire es diferido hacia el tracto nasal gracias a la apertura del velum
- Sonidos Plosivos: Son producidos por la excitación del tracto por una repentina salida de presión. Hay sonidos sonoros e insonoros .Las cuerdas vocales comienzan a vibrar antes de la salida para los sonidos sonoros.
- Sonidos Fricativos: Son producidos por la excitación del tracto por un flujo turbulento creado por el aire que pasa por un agujero estrecho.
- Sonidos Fricativos sonoros: Son producidos por la excitación del tracto gracias a una turbulencia y una vibración de la cuerda vocal
- Sonidos Africativos: Son sonidos que comienzan con una parada y son soltados como fricativos. Hay sonoros e insonoros.

A parte de controlar el tipo de excitación, la forma del tracto vocal es ajustada mediante la manipulación de la lengua, los labios y la mandíbula inferior. La forma determina la respuesta en frecuencia del tracto vocal. La respuesta en frecuencia en cualquier frecuencia es definida como la amplitud y la fase de los labios en respuesta a la excitación sinusoidal de amplitud la unidad y fase cero en la fuente. La respuesta en frecuencia, generalmente, muestra la concentración de energía en una vecindad de frecuencias llamadas frecuencias de pico.

Para las vocales, tres o cuatro resonancias usualmente pueden ser distinguidas claramente en el rango de frecuencias desde 0 hasta 4 kHz. De media más del 99% de la energía en una señal

procedente del habla está en este rango. La configuración de estas resonancias es lo que distingue las diferentes vocales entre sí.

Para los sonidos fricativos y plosivos, las resonancias no se distinguen tal fácilmente. Sin embargo existen ciertas características en amplias regiones de frecuencias donde la energía se concentra.

Para los sonidos nasales, aparte de las frecuencias de pico hay anti-resonancias o ceros en la respuesta en frecuencia. Estos ceros son el resultado de acoplar el movimiento de la onda en los tractos nasales y vocal.

Las diferentes partes del aparato que genera el habla humana pueden ser juntadas para producir un discurso fluido. Para considerar la implementación digital de este proceso se tiene en cuenta la teoría estándar del muestreo en tiempo y en frecuencia para convertir señales continuas en discretas para ser representadas digitalmente según el número de bits deseado.

Los codificadores de la voz se dividen en dos categorías: Codificadores basados en modelos y codificadores basados en la forma de la onda:

- Codificadores basados en modelos: Están basados en modelos de producción del habla.
- Codificadores de voz según la forma de onda en el dominio del tiempo: Tienen la propiedad de que no hay error en la cuantificación y la señal original puede ser reproducida con total exactitud. Entre los más comunes se encuentran:
- Codificadores de voz según la forma de onda en el dominio de la frecuencia:

La síntesis de la voz es la producción artificial del habla humana. Para su realización se usan sistemas llamados sintetizadores de voz y pueden ser implementados mediante software o hardware. Un ejemplo es el sistema TtS (Text-to-Speech) que convierte el lenguaje normal modelado como texto a voz. Este modelado es generalmente realizado mediante un conjunto de reglas derivadas de teorías fonéticas o análisis acústicos. Esta tecnología se suelen denominar síntesis de la voz basada en reglas.

En contraste con la aproximación basada en reglas, también se ha desarrollado una aproximación basada en corpus. En esta aproximación, los datos bien definidos del habla han sido anotados en diferentes niveles con información, tales como etiquetas fonéticas-acústicas y triangulación sintáctica, para producir un modelado estadístico. Parámetros funcionales espectrales y prosódicos de los datos del habla son analizados en relación con la información etiquetada y sus características de control son descritas cuantitativamente. Basándose en los

resultados de estos análisis se crea un modelo computacional es creado y entrenado usando el corpus. Una vez creado el modelo, se aplican los resultados a datos no chequeados para comprobar su validez.

Las cualidades más importantes en un sistema sintetizador de voz son la naturalidad y que sea inteligible. La naturalidad describe cómo se parecen los sonidos sintetizados al habla humana, mientras que sea inteligible significa cuan fácil la salida se entiende. El sintetizador ideal tiene las dos características y se tiende a maximizarlas.

Las dos tecnologías clave para la generación del habla sintética son la síntesis por concatenación y la síntesis por picos de frecuencias, entre otras:

- Síntesis por concatenación.
- Síntesis por resonancia
- Síntesis por articulación de la voz.
- Síntesis basada en HMM.
- Síntesis por onda sinusoidal.

2.5 Codec de voz MELP

2.5.1 Introducción

Un codec de audio es dispositivo hardware o un programa software definido mediante librerías que incluye un conjunto de algoritmos e instrucciones que sirven para codificar/decodificar datos de audio digital en función de un formato de archivo o un flujo de datos. El objetivo principal de estos algoritmos es representar con la mayor exactitud posible la señal de audio de alta calidad con el mínimo número posible de bits. La finalidad es reducir el ancho de banda necesario para la transmisión y el espacio necesario para el almacenamiento.

Normalmente la compresión conlleva una perdida de calidad, por lo que interesará utilizar aquellos codecs con una buena relación compresión/calidad según las necesidades de la transmisión.

Los principales atributos de un codec de voz son:

- Tasa binaria: Sirve para ver el grado de compresión que el codec logra. El ancho de banda para la voz usado en telefonía es muestreado a 8 kHz y digitalizado con un cuantificador logarítmico de 8 bits, resultando una tasa binaria de 64 kbps. Para los codecs usados en

telefonía o para la transmisión de voz, se mide el grado de compresión comparando cuánto se ha reducido la tasa de 64 kbps. Los codificadores de voz no tienen porqué tener una tasa constante ya que se puede ahorrar ancho de banda al no transmitir durante los intervalos de silencio en una conversación. Tampoco es necesario mantener fija la tasa durante los intervalos sonoros de la conversación.

- Retraso: El retraso en la comunicación de un codificador es más importante durante la transmisión que en el almacenamiento. En conversaciones en tiempo real, un retraso importante puede poner trabas a la hora de mantener un discurso fluido. No se deben permitir retrasos superiores a 300 ms incluso si no hay eco. La mayoría de codificadores de voz de baja tasa son codificadores de bloque. Codifican un bloque del habla, conocido como trama.

- Complejidad: El grado de complejidad es un factor determinante a la hora de asumir el coste y la potencia de un codificador. El coste es casi siempre un factor importante a la hora de elegir un codec u otro para una determinada aplicación. Con el advenimiento de la tecnología inalámbrica y portátil en las comunicaciones, el consumo de potencia se ha convertido también en un importante factor a tener en cuenta. Son necesarios cuantificadores simples escalares, tales como PCM lineales o logarítmicos en cualquier sistema de codificación así como la menor complejidad posible. En codificadores más complejos, primero son simulados en procesadores y luego implementados en DSP chips y quizás al final son implementados en dispositivos VLSI para aplicaciones especiales. La velocidad y el acceso aleatorio a memoria (RAM) son los dos factores más importantes a la hora de contribuir a aumentar la complejidad.

- Calidad: El atributo de la calidad tiene muchas dimensiones. La última instancia de la calidad es determinada por cómo son escuchados los sonidos por un oyente. Algunos de los factores que afectan a la ejecución de un codificador son si la señal de entrada está limpia o está afectada por el ruido, si el flujo de bits ha sido corrompido por errores, y si han tenido lugar múltiples codificaciones. La clasificación de las calidades es determinada mediante test auditivos subjetivos. Otra dimensión de la calidad es la sensibilidad ante errores en bits. Para aplicaciones con baja tasa binaria, la distribución de los errores suele ser aleatoria por lo que se necesitan codecs robustos.

2.5.2 Codificación y Decodificación MELP

El Codec MELP (Mixed Excitation Linear Prediction) es un codec de voz basado en el modelo paramétrico LPC (Linear Prediction Coding) que presenta la posibilidad de trabajar a 2400 o 1200 bps. Consta de técnicas de codificación avanzadas como la cuantificación vectorial ponderada y los libros de códigos.

MELP pertenece a la familia de codificadores paramétricos, aunque también podría ser clasificado como híbrido o multimodo controlado por la fuente, aunque posee parámetros adicionales que son capaces de representar con exactitud los procesos dinámicos que tienen lugar durante la producción de la voz humana, pudiendo representar la mayoría de las características de la voz humana.

- Excitación Mixta: Uno de los problemas más comunes en la codificación de voz LPC es el rumor introducido en el habla sintetizada debido a la excitación de los pulsos puros.

Muchas de las características del ruido y la información de la fase en el primer residuo son ignoradas una vez que la información del pitch es extraída. Un único tren de impulsos es inadecuado para sintetizar los diferentes tipos de características de la voz humana. Cuando usamos la excitación mixta en la síntesis del habla, se usa una mezcla de pulso y ruido para realizar una aproximación más exacta a las características del primer residuo y de esta manera poder eliminar el rumor introducido.

Esta técnica es implementada mediante un modelo de mezclas multibanda, pudiendo simular la intensidad de voz en frecuencia mediante filtrado adaptativo usando un banco de filtros fijos.

- Pulsos Aperiódicos: Durante el habla hay periodos sonoros e insonoros. El codec MELP es capaz de sintetizar estos periodos sonoros mediante pulsos periódicos o aperiódicos, aunque los aperiódicos se utilizan sobre todo en las regiones de transición entre segmentos sonoros e insonoros de la señal de voz. Este mecanismo permite al decodificador reproducir tonos glotales erráticos sin necesidad de añadir sonidos tonales.

El jitter usado en tramas débilmente sonoras proporciona un modelo más ajustado de los pulsos glotales erráticos en el habla humana. Este tercer estado sonoro evita el error causado al sintetizar tramas débilmente sonoras mediante ruido puro o al sintetizar tramas no sonoras usando la excitación de los pulsos puros.

- Mejoramiento Espectral Adaptativo: Este filtro está basado en los polos que se obtienen a través de un filtro sintetizador de predicción lineal. De esta manera se ensalza la estructura del habla sintetizada y se mejora la armonía entre las ondas en paso banda sintetizada y natural.

- Magnitudes de Fourier: Las magnitudes utilizadas corresponden a las diez primeras magnitudes armónicas del pitch en la predicción del espectro de la magnitud residual. Las magnitudes de estos armónicos son cuantizadas y pueden ser usadas en el sintetizador para generar la excitación de los pulsos.

La información en estos coeficientes mejora la exactitud del modelo de producción del habla sobre todo a bajas frecuencias. Esto incrementa la calidad del habla sintetizada, particularmente en voces masculinas y cuando hay ruido de fondo.

Este algoritmo de codificación de voz MELP es una excelente opción para realizar su inserción en Asterisk por las siguientes razones:

- Tiene una tasa binaria muy baja por lo que se pueden realizar conexiones en banda estrecha.
- Relativamente poca complejidad con respecto a sus predecesor CELP, lo que facilita su implementación y eficiencia en el procesamiento.
- Las diferente test realizados por empresas dan como resultado una mayor robustez frente al ruido y mayor inteligibilidad en condiciones poco favorables como transmisiones aéreas.
- La voz sintetizada posee una calidad superior con una valoración en el test MOS de 3.3.
- El codec MELP supone toda una innovación tecnológica y aunque no exista en el mercado civil, su proceso está en marcha mediante el desarrollo de aplicaciones software y hardware.
- El retardo inducido por las tramas MELP en los microprocesadores actuales es inferior a la codificación CELP a 4.8 kpbs.

El codificador MELP trabaja sobre fragmentos de voz de 22.5ms, es decir sobre bloques de 180 muestras con datos de audio de 16 bits provenientes del muestreo a 8 kHz de la señal analógica. Según el modo de funcionamiento la información se procesa de diversas maneras:

- Codificación a 2400 bps: El codificador procesa individualmente los fragmentos de voz con bloques de 180 muestras y las codifica en palabras de 7 bytes.

- Codificación a 1200 bps: El codificador procesa tres ventanas, cada una con fragmentos de voz de 22.5ms que forman bloques de 180 muestras. En total trabaja con un bloque de 540 muestras. El resultado es una palabra codificada de 11 bytes.

El codificador se divide en las siguientes partes:

- Filtro paso alto: Chebychev tipo II de cuarto orden
- Análisis de la banda de voz con filtros Butterworth de sexto orden en cinco sub-bandas de frecuencia separadas
- Filtro de predicción lineal de décimo orden
- Protección frente a errores

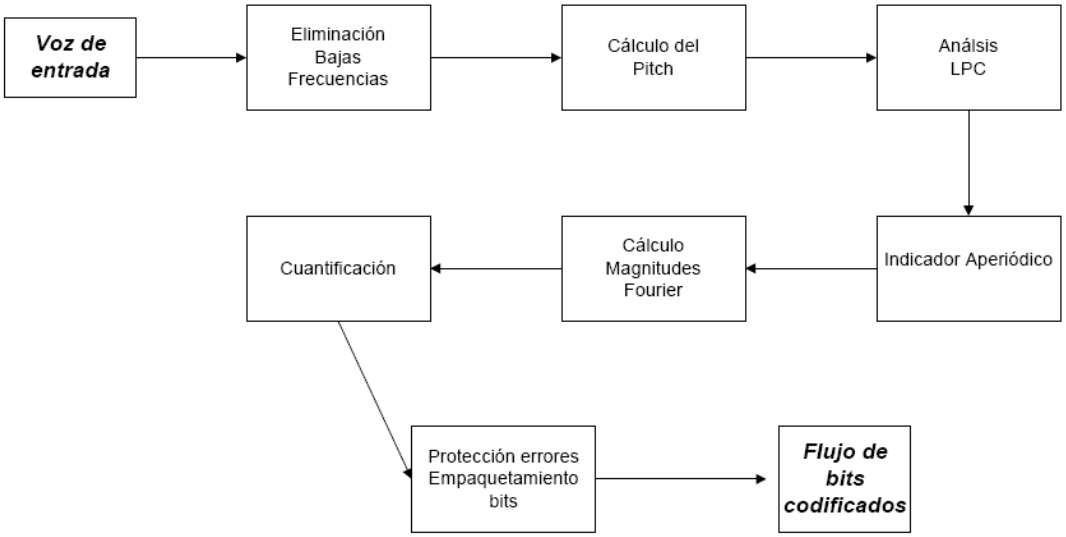

Figura 2.5: Diagrama de bloques del codificador MELP

El decodificador se divide en las siguientes partes:

- Generación de la voz con excitación mixta mediante la suma de impulsos y ruido filtrados:
  - Pulsos: Se aplica la transformada inversa discreta de Fourier con duración igual al período del pitch
  - Ruido: Se obtienen mediante un generador uniforme de números aleatorios.
  - Filtro de pulsos: Suma de los coeficientes del filtro en paso banda para las sub-bandas de frecuencia de voz
  - Filtro de ruido: Suma de los coeficientes del filtro donde las bandas de frecuencia están calificadas como no sonoras
- Filtro adaptativo de mejora del espectro
- Síntesis de predicción lineal
- Dispersión de pulso

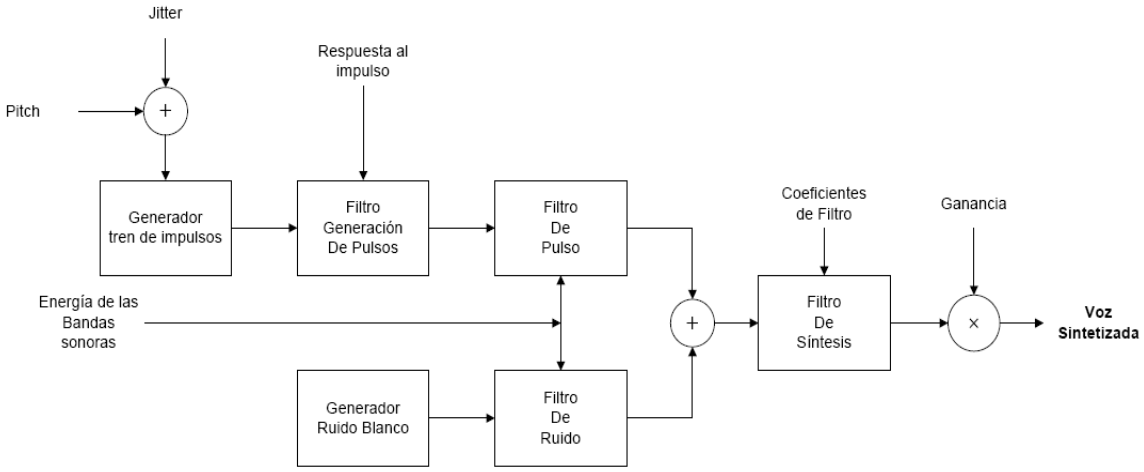

Figura 2.6: Diagrama de bloques del Decodificador MELP

3. Diseño y Desarrollo

3.1 Objetivo

Hoy en día, tanto grandes como pequeñas empresas, buscan optimizar sus recursos y abaratar costes. Todo esto junto con el auge de las comunicaciones de voz sobre IP, han hecho posible la rápida expansión de la plataforma open source Asterisk.

Las ventajas del uso de Asterisk como centralita de telefonía son que puedes realizar tus propias adaptaciones para tu uso personal y no requiere de una amplia infraestructura de software o hardware para que funcione. Todo esto unido al hecho de que puedes desarrollar nuevas aplicaciones, hace de Asterisk el software ideal a la hora de decidir qué producto escoger. La principal desventaja que puede tener es la dificultad inicial a la hora de programar para realizar tus modificaciones, ya que Asterisk es un software escrito en el lenguaje C, que en un principio puede parecer sencillo, pero posee un entramado complejo y no resulta fácil a la hora de hacer un debug y ver los posibles errores.

El objetivo principal de mi proyecto es insertar el codec MELP de forma nativa dentro de la plataforma Asterisk para que haga las traducciones directamente sin necesidad de recurrir a recursos externos. La relevancia de usar Debian y Asterisk, se ve patente a la hora de la ejecución de MELP, puesto que al realizar los procesos de codificación y decodificación, el codec consume muchos recursos de la CPU y sería prácticamente imposible su realización sobre el sistema operativo Windows de una forma fluida.

Insertar un codec de forma nativa dentro de Asterisk significa insertarlo dentro del núcleo del sistema para que lo reconozca como propio. Para ello, es necesaria la creación de una interfaz cargable dentro de la API *Codec Translator*, que contenga todas las librerías y funciones necesarias que implementen la traducción en ambos sentidos: señal LPC a trama MELP y viceversa.

Cuando el proceso de inserción esté realizado y Asterisk reconozca como propio el codec MELP, es necesario la configuración de un dialplan (o plan de llamadas) y del protocolo SIP para poder realizar comunicaciones de voz con el codec MELP y de un softphone que contenga MELP para que las comunicaciones sean bidireccionales.

3.2 Configuración de Asterisk

3.2.1 Instalación de Asterisk

El primer paso para la instalación de Asterisk es tener en cuenta los requisitos mínimos necesarios para su correcto funcionamiento:

- Hay que tener en cuenta las llamadas concurrentes, las conferencias y aplicaciones complejas necesarias y las transcodificaciones que se vayan a realizar.
- Es necesario un microprocesador con velocidad mínima de 1.8 GHz y una memoria RAM de 1 Gb para poder soportar unas 60 llamadas concurrentes con el codec G.729

Para mi proyecto, he contado con un PC portátil con un microprocesador AMD Turion 64 Mobile Technology MK-36 a 2 GHz y una memoria RAM de 1Gb.

Una vez comprobados los requisitos técnicos es necesario contar con plataforma software basada en GNU/Linux ya que la elección de la plataforma es clave. Debido al hecho de que las plataformas open source están más probadas y son configurables, para mi proyecto he contado con la plataforma Debian 2.6.18-4-k7. Una vez instalada, mediante el terminal se procede a la instalación de las librerías necesarias para que Asterisk pueda funcionar:

- openssh-server: para poder realizar accesos remotos seguros
- gcc, make: compilador de lenguaje C y sus herramientas
- libc-dev, libssl-dev, zlib1g-dev, libncurses5-dev, libmysqlclient14-dev

La instrucción necesaria para la instalación es apt-get install y ésta busca en el repositorio de Debian las últimas actualizaciones disponibles y las instala.

Una vez instaladas todas las librerías, se procede a la descarga de los códigos fuente. Asterisk suele estar disponible en dos modos distintos: estable y cabeceras. En el modo estable es la rama establecida para el uso en sistemas de producción. Las cabeceras es el modo que usan los desarrolladores para probar nuevas características y corregir errores.

La corrección de errores es fusionada con el modo estable después de un período de pruebas. Es posible también que el modo de desarrollo se pueda romper en ciertos puntos durante el período de pruebas. Las publicaciones estables se pueden obtener bien vía FTP o mediante CVS. Mediante CVS se puede descargar la publicación que se desee, mientras que por FTP siempre se descarga la última versión. Cada versión estable se puede descargar en un

fichero .tar cuyo nombre contiene la versión de Asterisk. Para mi proyecto he contado con la versión de Asterisk 1.4.18.

El código de Asterisk se divide en las siguientes partes:

- Asterisk: Es el núcleo del sistema
- Asterisk-sounds: Voces de calidad pregrabadas
- Asterisk-addons: Software adicional
- Libpri: Librería para gestionar enlaces RDSI primarios
- Libiax: Librería para utilizar el protocolo IAX
- Zaptel: Interfaz del Kernel para acceder a tarjetas de comunicaciones para líneas analógicas o digitales.

Para el proyecto sólo serán necesarias las dos primeras, ya que las comunicaciones se van a realizar sobre VoIP mediante el protocolo SIP para la iniciación y el protocolo RTP para las comunicaciones.

Una vez descargados los .tar necesarios se procede a su extracción y compilación:

1. Nos metemos en el directorio donde vamos a instalar asterisk y sus dependencias:
   # cd/usr/src
2. Una vez en el directorio se procede a la extracción por terminal de los ficheros:
   # tar zxvf asteriskl-*.tar.gz
   # tar zxvf asterisk-sounds-*.tar.gz
3. En el mismo directorio donde se han extraído los ficheros , nos metemos en la carpeta de Asterisk descomprimida y se procede a su instalación:
   # ./configure
   # make
   # make install
   # make samples -> Instala ficheros de ejemplo con una configuración inicial.
4. Para la instalacion de los sonidos se sigue el mismo procedimiento:
   # cd/usr/var/asterisk-sounds
   # make install
5. En este punto Asterisk ya se encuentra instalado, ahora se realiza la comprobación de que todos sus directorios y ficheros se han instalado correctamente:
   - /etc/asterisk/: En este directorio se encuentran todos los ficheros de configuración. Al realizar el make samples, los ficheros de ejemplo se guardan aquí.

- /usr/lib/asterisk/modules/: En este directorio se encuentran todos los módulos que se cargan al iniciar la plataforma. Se encuentran las diversas aplicaciones, codecs, formatos y canales usados por Asterisk. Por defecto, se cargan al inicio, pero se pueden deshabilitar en el fichero modules.conf. Hay que tener cuidado de no deshabilitar módulos imprescindibles a la hora de que funcione correctamente.
- /var/lib/asterisk/: Este directorio contiene los ficheros *astdb* y un número de subdirectorios. Los ficheros *astdb* contienen las bases de datos locales de Asterisk.
- /var/lib/sounds/: Directorio donde se encuentran las voces pregrabadas.

6. Una vez completados todos los pasos anteriores, Asterisk se encuentra instalado y listo para ejecutarse. La instrucción necesaria para su ejecución es:

   #usr/var/asterisk/ asterisk

   - Asterisk ofrece la posibilidad de inicialización con el intérprete de comandos CLI, basta con ejecutar el comando:

     # usr/var/ asterisk asterisk –r

   - Para matar el proceso que ejecuta Asterisk basta con teclear alguno de los siguientes comandos: stop now (detención instantánea), stop when convenient (detención cuando no hay carga) y stop gracefully (detención cuando no hay carga y se terminen las llamadas).
   - Otra aplicación interesante es el Verbose: Indica la cantidad de mensajes que se recibirán por el terminal sobre los eventos generales del sistema. Cuanto más alto, más información. Se ejecuta con el comando:

     # usr/var/ asterisk asterisk –vvvvvvvvvvgc (con tantas letras v como se quiera).

3.2.2 Creación de la interfaz para el codec MELP

Con Asterisk ya funcionando correctamente lo primero que hay que hacer es ver que hay que hacer para insertar un codec en la plataforma. En la página de Asterisk hay un link para ver las conexiones entre ficheros y el flujo de información (ver referencia)

1. En la carpeta #usr/src/asterisk/codecs, se encuentran las interfaces en forma de ficheros en C, creadas por Asterisk para el reconocimiento de codecs y la creación de módulos dinámicos. En ese directorio es donde se coloca la carpeta a la que he denominado melp con todos los códigos fuente en C del codec. El primer paso para crear la interfaz para el codec MELP, es la creación de un fichero para programar en C. Para que este fichero sea reconocido por Asterisk el nombre tiene que tener la forma codec_*.c. En este caso se ha llamado codec_melp.c (ver referencia).

2. Una vez creado el fichero, Asterisk posee una serie de funciones definidas que son necesarias implementar para poder realizar la traducción correctamente:

   - Estructuras de Datos:

     struct melp_translator_pvt: Estructura donde se crea un buffer con el número de muestras que va a procesar el codec MELP.

   - Definiciones:

     - # *define BUFFER_SAMPLES 8000*: El número de muestras que el buffer de Asterisk va a recopilar de la señal LPC.

     - # *define MELP_FRAME_LEN 11*: El número en bytes de la trama codificada por el codec MELP.

     - # *define MELP_SAMPLES*: El número de muestras necesarias que puede procesar a la vez el codec MELP

   - Funciones:

     - *static void melp_destroy_stuff (struct ast_trans_pvt *pvt):* Destruye las posibles variables temporales creadas que tienen reservada memoria al final del proceso.

     - *static int melptolin_framein (struct ast_trans_pvt *pvt, struct ast_frame *f*): Función que implementa la decodificación de la trama en MELP y lo guarda en un buffer de salida

     - *static int lintomelp_framein (struct ast_trans_pvt *pvt, struct ast_frame *f):* Función que guarda las muestras en un buffer temporal para luego realizar la codificación

     - *static struct ast_frame* melptolin_sample (void):* Función que devuelve los valores de la estructura de Asterisk ast_frame para la codificación que se ponen manualmente según los parámetros de tu codec al crear este fichero.

     - static struct *ast_frame* lintomelp_frameout (struct ast_trans_pvt *pvt):* Función que realiza la codificación de la muestra en una trama melp.

     - *static struct ast_frame* lintomelp_sample (void):* Función que devuelve los valores de la estructura de Asterisk ast_frame para la decodificación que se ponen manualmente según los parámetros de tu codec al crear este fichero.

     - *static int load_module (void):* Función que usa Asterisk para cargar el fichero en tiempo de ejecución

     - *static void parse_config (void):* Función que comprueba que el módulo se ha cargado correctamente.

     - *static int reload (void):* Vuelve a cargar el módulo y comprueba su corrección.

     - *static int unload_module(void):* Función que detiene el registro del módulo durante esa ejecución para deshabilitarlo.

-*AST_MODULE_INFO(ASTERISK_GPL_KEY, AST_MODFLAG_DEFAULT, "MELP Coder/Decoder" , . load = load_module, . unload = unload_module, . reload = reload)*: Función que realiza el registro del módulo en la API de Asterisk.

- Variables:

  - *static struct ast_translator melptolin*

  - *static struct ast_translator lintomelp*

3. Creación de ficheros en el lenguaje C que contengan la estructura de los datos y su longitud:
   - *slin_melp_ex.c*: En este fichero se define una función llamada static signed short slin_melp_ex[] y contiene los datos de audio de 16 bits con signo. Contiene tantos shorts como muestras procesa el codec. En este caso MELP procesa 540 shorts(ver referencia).
   - *melp_slin_ex.c:* En este fichero se define una función llamada unsigned char gsm_slin_ex[] [static] y contiene los datos en bruto de 8 bits. Contiene tantos char como muestras procesa el codec. MELP procesa 88 bytes(ver referencia).

Las estructuras propias definidas por Asterisk siempre empiezan por *ast* y son necesarias para el correcto funcionamiento del módulo:

- *ast_frame*: Contiene la información de una trama. Se utiliza en las funciones lintomelp_sample y melptolin_sample

  - void* data= slin_melp_ex;

  - int dalalen= sizeof(slin_melp_ex);

  - timeval delivery

  - unsigned int flags

  - enum ast_frame_type frametype= AST_FRAME_VOICE;

  - long len

  - int mallocd= 0;

  - size_t mallocd_hdr_len

  - int offset= 0;

  - int samples= sizeof(slin_melp_ex)/2;

  - int seqno

  - const char * src = _PRETTY_FUNCTION_;

  - int subclass = AST_FORMAT_SLINEAR;

  - long ts

- *ast_trans_pvt*: La estructura por defecto de los traductores. Contiene los campos básicos y buffers, todos en la misma zona de memoria. Se utiliza en las funciones lintomelp_framein, melptolin_framein y lintomelp_frameout:
  - int datalen =2*MELP_SAMPLES; Espacio usado en el búfer de salida
  - ast_frame f
  - ast_trans_pvt * next : Define la siguiente cadena para traducir
  - timeval nextin
  - timeval nextout
  - char * outbuf
  - plc_state_t * plc
  - void * pvt
  - int samples =MELP_SAMPLES; Número de muestras en el búfer de salida.
  - ast_translator * t
- *ast_translator*: Descriptor del traductor. Nombres, operaciones en tiempo de ejecución, etc. El codec se registra pasando los datos de esta estructura a través de las funciones ast_register_translator() y ast_unregister_translator(). Se usa en las funciones load_module, parse_config, reload y unload_module.

Las librerías necesarias para que funcione la interfaz y que a su vez definen las relaciones con otras interfaces son:

#include "**asterisk.h**"
#include <fcntl.h>
#include <stdlib.h>
#include <unistd.h>
#include <netinet/in.h>
#include <string.h>
#include <stdio.h>
#include "**asterisk/lock.h**"
#include "**asterisk/translate.h**"
#include "**asterisk/config.h**"
#include "**asterisk/options.h**"
#include "**asterisk/module.h**"
#include "**asterisk/logger.h**"
#include "**asterisk/channel.h**"
#include "**asterisk/utils.h**"
#include "**melp_slin_ex.h**"
#include "**slin_melp_ex.h**"

Es necesario también incluir la librería creada específicamente para poder usar el código fuente del melp: #include "melp/sc1200.h"

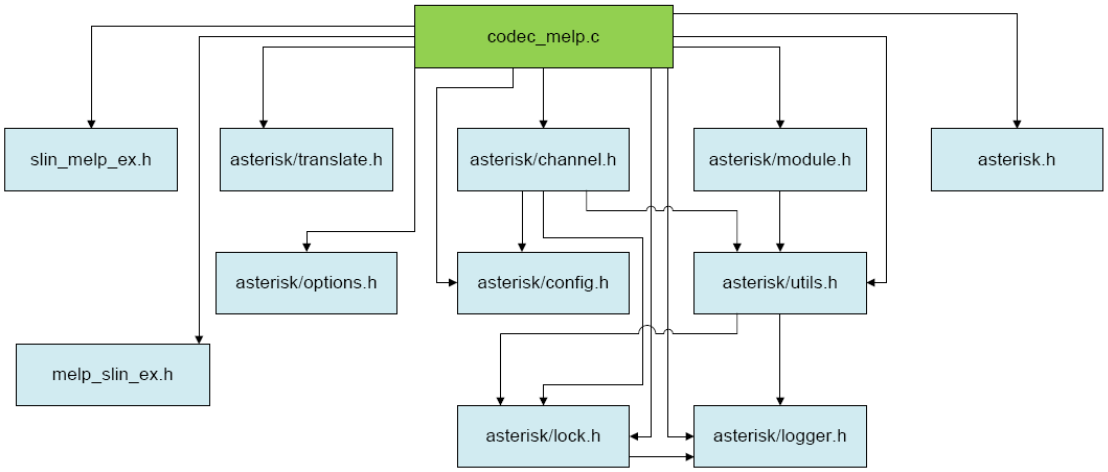

Figura 3.1: Diagrama de funcionamiento de las librerías de Asterisk

Una vez creados los ficheros, definidas todas las variables y estructuras es necesario completar las funciones para incluir los datos procedentes del código fuente de MELP

La función que implementa el codificador MELP es lintomelp_frameout (); Aquí, llamamos a la función melp_encode() y le pasamos un puntero a un búfer temporal con tamaño 540 muestras y un puntero a un búfer temporal de tamaño 11 bytes donde se recogen los datos codificados. La función melp_encode se repite para todas las tramas hasta completar el búfer general con 8000 muestras.

En cambio, la función que implementa el decodificador MELP es melptolin_framein(); En donde se le pasan tramas codificadas de tamaño 11 bytes en un puntero unsigned char y devuelve los datos decodificados en un puntero a un búfer temporal.

A la hora de compilar este fichero, es necesario declarar ciertas variables en otros ficheros para que Asterisk reconozca el codec:

1. build_tools/embed_modules.xml: Este archivo realiza la carga de todos los módulos cuando Asterisk se inicializa. Se incluyen todos los archivos objetos y las librerías estáticas de los codecs:

<member name="codecs" displayname="Coders/Decoders" remove_on_change="codecs/*.o docecs/gsm/src/*.o codecs/ilbc/*.o codecs/lpc10/*.o *codecs/melp/*.o* codecs/gsm/lib/libgsm.a codecs/lpc10/liblpc10.a codecs/ilbc/libilbc.a *codecs/melp/libmelp.a*">

2. include/asterisk/frame.h: Se define la variable global usada en la interfaz codec_melp.c, AST_FORMAT_MELP que define la máscara de bits para reconocer al MELP a la hora de trabajar internamente.
3. main/frame.c: Se definen las rutinas para la manipulación de tramas. En este archivo se realizan cuatro modificaciones:
    a. En la estructura ast_format_list:

       int visible = 1;
       int bits =AST_FORMAT_MELP;
       char * name = "melp";
       char * desc = "MELP";
       int fr_len =11;
       int min_ms = 68;
       int max_ms =300;
       int inc_ms =68;
       int def_ms =68;
       unsigned int flags
       int cur_ms= 68;

    b. En la function show_codecs_deprecatec() es necesario aumentar en una unidad el bucle for para que recorra todos los codecs.
    c. En la función show_codecs() es necesario aumentar también una unidad en el bucle for para que recorra todas los codecs.
    d. En la función ast_codec_get_samples() hay que poner en la estructura del switch el caso para AST_FORMAT_MELP:

       case AST_FORMAT_MELP:
          len= (simples / 540)* 11;
          break;
4. main/channel.c: Entre el conjunto posibles de codecs que Asterisk puede usar durante una comunicación de voz, siempre va a escoger el mejor para cada contexto específico. En este fichero se implementa la función ast-best_codec, donde están definidas todas las máscaras de los codecs. Por lo tanto es necesario introducir el AST_FORMAT_MELP, para que el codec MELP esté entre los candidatos.
5. main/translate.c: En la función show_translation, es necesario cambiar el *#define SHOW_TRANS* e incrementarlo una unidad para poder ver desde el terminal los procesos y habilitar el manejador de comandos.
6. main/rtp.c: En este fichero se realizan dos modificaciones:
    a. Se rellena la estructura del mimeType del codec MELP para el encapsulamiento de tramas: *{{1, AST_FORMAT_MELP}, "audio", "MELP"}*.

b. Se rellena la estructura que define el Payload de las tramas en el encapsulamiento RTP: *[96]= {1, AST_FORMAT_MELP}*.

Para la realización de estos pasos, es ventajosa la creación de un parche, llamado melp-patch.txt (ver ANEXO G) que se ejecuta en la carpeta donde está situado Asterisk y que contenga los datos a introducir y en qué líneas del código se deben de meter. Los parches siempre tienen extensión .txt y su ejecución se realiza por el terminal mediante las instrucciones:

patch –dry-run –p0 < /usr/src/asterisk/melp-asterisk-patch.txt

patch –p0 < /usr/src/asterisk/melp-asterisk-patch.txt

3.2.3 Configuración de Asterisk para la realización de llamadas

Cuando por fin Asterisk reconoce el módulo del codec melp es hora de configurar el resto de módulos para poder realizar una transmisión por voz sobre IP.

El plan de llamadas es la base del sistema Asterisk, ya que define cómo la plataforma maneja las llamadas entrantes y salientes. Es decir, consiste en una lista de instrucciones o pasos a realizar que va a seguir Asterisk.

El fichero donde se describen los planes de llamadas se denomina *extensions.conf* y se encuentra en la carpeta # /etc/asterisk, junto con el resto de ficheros .conf. El plan de llamadas se divide en cuatro partes principales:

- Contexts: Los contextos son grupos de extensiones y mantienen las diferentes partes del plan de llamadas separadas. Las extensiones definidas en un plan están totalmente aisladas de otras extensiones en otro plan, al menos que se permite explícitamente la interacción. Cada contexto se denota con un nombre entre corchetes. Todas las instrucciones que se escriban a continuación de una definición de contexto, pertenecen a ese contexto. La seguridad es muy importante en las comunicaciones y es necesario realizar el plan de llamadas en función de la seguridad que quieras obtener.
- Extensions: Dentro de cada contexto, se definen una o más extensiones. Una extensión es una instrucción dentro del plan de llamadas, cuyo detonante puede ser una llamada entrante o una marcación. Las extensiones guían a las llamadas durante todo el proceso de la conversación. La sintaxis básica de una extensión es: exten => name, priority,aplication() y se compone de tres partes:
    - El nombre de la extensión

    - La prioridad
    - La aplicación que realiza durante la llamada
- Priorities: Cada extensión tiene varios pasos llamados prioridades. Las prioridades se numeran secuencialmente, empezando por el 1.
- Applications: Cada aplicación realiza una acción dentro del canal de comunicación. Asterisk tiene una amplia gama de aplicaciones. Dentro de las más comunes se encuentran: Answer(), Playback() y Hangup().

El objetivo primordial de la tecnología VoIP es la paquetización del flujo de audio para su transporte sobre redes basadas en el protocolo IP. Los protocolos de transporte originariamente no fueron diseñados para un flujo en tiempo real y un retraso superior a 300 milisegundos puede suponer deficiencias a la hora de la comunicación. El protocolo SIP se basa en que cada extremo de la conexión es un peer y se negocian las capacidades entre los dos extremos.

En Asterisk el módulo que gestiona el protocolo SIP se denomina chan_sip.so. El fichero de configuración del protocolo SIP se llama sip.conf y contiene toda la información correspondiente a la configuración.

Al principio del fichero sip.conf en la sección [general] se describen los parámetros generales de las conexiones SIP. Estos parámetros afectan a todos los canales de comunicación que vayan por este protocolo. Entre las más importantes se encuentran:

- allowguest: permite o no las conexiones SIP sin la autentificación
- bindport: define el puerto por donde van a pasar las conexiones
- callevents: define la generación de eventos externos
- useragent: Especifica en la cabecera SIP un string con el nombre del agente. Por defecto está useragent=asterisk.

Cada canal SIP se define con un nombre entre corchetes y debajo se describen sus características. Éstas pueden referirse al usuario, al peer o a ambos:

- allow y disallow (ambos): Permiten el uso o no de codecs específicos.
- callerid (ambos): si se define un caller id para un usuario o un peer, todas las llamadas entrantes por ese canal tendrán esa identificación.
- canreinvite (ambos): El protocolo SIP por defecto tiende a conectar los extremos directamente, pero Asterisk debe mantener el camino entre los terminales si se requiere DTMF.

- context (ambos): A cada canal se le asigna un contexto proveniente del fichero extensión.conf para gestionar las llamadas entrantes. Todos los canales tienen que tener siempre un contexto asignado.
- nat (ambos): permite el uso de este protocolo. Se puede poner a yes, no o never.
- port (peer): define el puerto de comunicaciones para ese canal en concreto. En los softphones el puerto por defecto es 5060.
- qualify (peer): permite o no que el tiempo de latencia no supere los 2 milisegundos
- secret (ambos): pone la contraseña para la autentificación.
- username (peer): permite intentar contactar con un peer antes de que se registre contigo.
- type: define el tipo de extensión: user para autenticar llamadas entrantes, peer para llamadas salientes y friend para ambas.

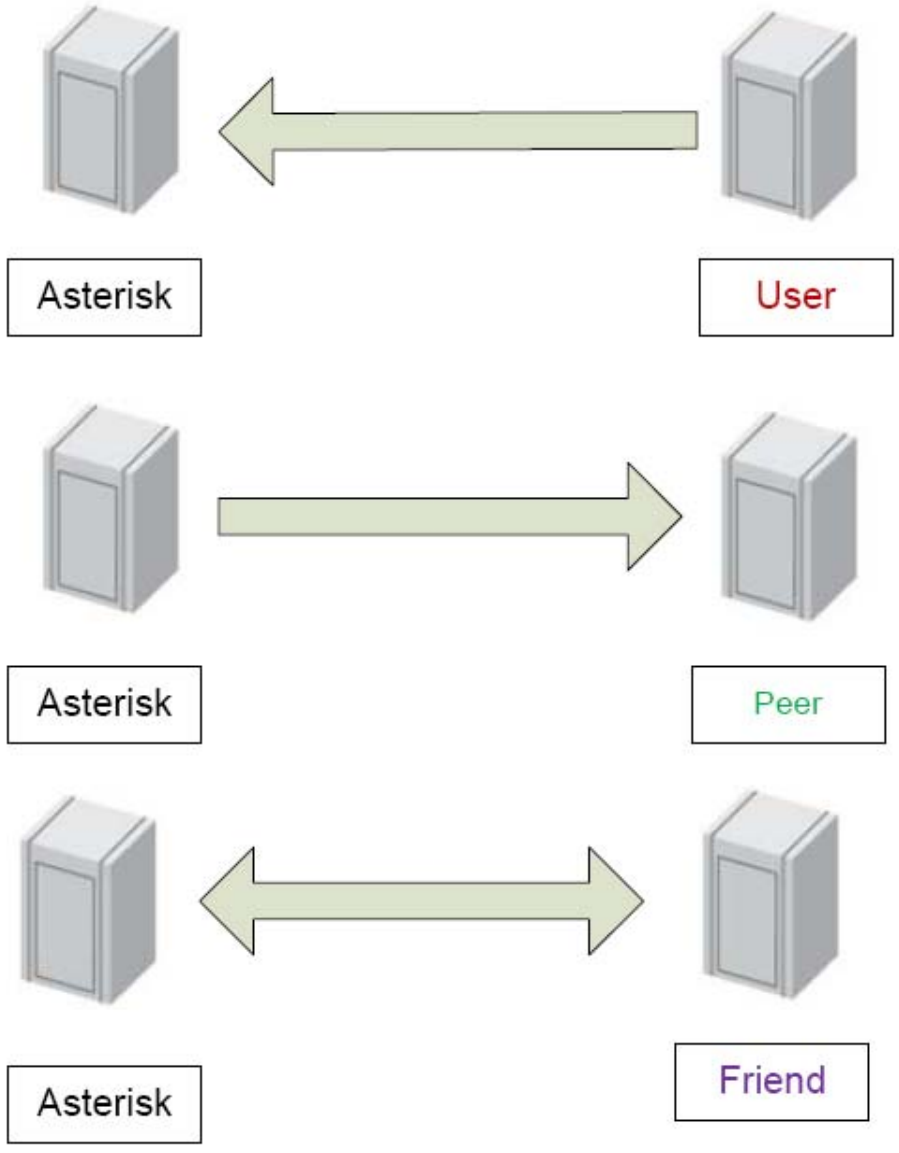

Figura 3.2: Modelo de los usuarios SIP

En el fichero sip.conf de este proyecto se han definido tres canales de comunicación:

1. Canal para un usuario con nombre Cristina para un teléfono IP:

```
[Cristina]; teléfono IP para realizar y recibir llamadas
type=friend; Permite ser user y peer a la vez
host=dynamic
username=Cristina
secret=user2_password
nat=yes; Permite la utilización de nat
canreinvite=no
context=office; Se refiere  al dialplan que va a seguir a la hora de gestionar las llamadas.
callerid="User2"
allow=gsm; Permite el uso de estos codecs a la hora de recibir llamadas.
allow=ulaw
allow=alaw
regexten=201; Extensión asignada al teléfono.
mailbox=201@default; buzón de voz.
```

2. Canal para un usuario con nombre 201 para realizar llamadas por un softphone:

```
[201] ; sofphone IP para realizar y recibir llamadas
type=friend
host=dynamic
secret=crisxu_password
username=crisxu
canreinvite=no
qualify=yes
nat=no
context=office
port=5061; puerto por donde se va a conectar el softphone
disallow=all
allow=gsm
allow=alaw
;allow=ilbc
allow=ulaw
;allow=speex
```

3. Canal para la pasarela Skype donde está insertado el codec MELP para que pueda realizar llamadas con ese codec.

```
[SPYKE]
type=friend
host=dynamic
;username=gateway
;secret=cristina
nat=no
;callerid="User3"
canreinvite=no
;qualify=yes
context=skype
callerid="SkypeGW <5001>"
disallow=all
;allow=gsm
;allow=ulaw
;allow=alaw
```

```
allow=melp
;regexten=5001
```

3.3 Configuración del Codec MELP

El codec MELP es un codificador de voz estándar usado principalmente en aplicaciones militares y comunicaciones por satélite, transmisiones con seguridad y aparatos de radio. Fue desarrollado por NSA y NATO. El primer codec MELP a 2.4 kbps fue creado por Texas Instruments y estandarizado en 1997. Entre 1998 y 2001, se creó un nuevo codec MELP que realizaba la codificación a 1.2 kbps.

El proceso de codificación del MELP procesa tres ventanas de información con 180 muestras de 16 bits cada una, en un total de 540 muestras, provenientes del proceso de muestreado del audio a 8 kHz. El resultado de la codificación son tramas de 11 bytes con una duración de 67.5 milisegundos.

El proceso de decodificación, coge las tramas de 11 bytes codificadas a 1.2 kbps y las decodifica en ventanas con 540 muestras de 16 bits cada una.

Para poder utilizar los códigos fuente en C del MELP es necesario desarrollar dos funciones que implementen el codificador y el decodificador.

3.3.1 Función melp_encode

Está función se encarga de procesar un búfer de entrada procedente del módulo Asterisk con los datos de la conversación y codificarlos a tramas melp de 11 bytes a 1200 bps. La función se encuentra implementada en el fichero melp12_enc.c y definida en la librería sc1200.h (ver ANEXO G).

Los argumentos de entrada son:

- *int16_t *buf_in*: Búfer de entrada que consiste en short int donde se pasan los datos a codificar.
- *char *buf_out*: Búfer de salida que contiene los datos codificados en tramas de 11 bytes

La función devuelve un BOOLEAN con TRUE si todo el proceso se ha ejecutado correctamente. Se divide en dos bloques principales:

1. Inicialización de todas la variables MELP para el modo (mode = ANALYSYS), la tasa binaria de codificación (rate = RATE1200), longitud de palabra codificada (chwordsize = 8), el tamaño de trama (frameSize = 540 shorts) y el tamaño del búfer (bitBufSize = 11 bytes). Después de inicializar todas las variables, se llama a la función melp *melp_ana_init(),* para que configure el algoritmo.
2. Se ejecuta un bucle while, para que recorra todo el búfer de entrada de 540 en 540 shorts, los copie en la memoria local, se limpie el ruido con el noise pre-procesor y se pase a la función de análisis, la cual devuelve la ventana codificada en tramas de 11 bytes. Por último se copian los datos al búfer de salida que pasa a Asterisk.

3.3.2 Función melp_decode

Esta función se encarga de procesar un búfer con los datos codificados en tramas de 11 bytes procedentes de una conversación y decodificarlos. La función se encuentra implementada en el fichero melp12_dec.c y definida en la librería sc1200.h (ver ANEXO G).

Los argumentos de entrada son:

- *unsigned char *src*: Búfer de entrada donde se encuentran almacenados los datos codificados en tramas de 11 bytes procedentes de la conversación.
- *int16_t *dst:* Búfer de salida donde se guardan los datos ya decodificados en short.

La función devuelve un BOOLEAN con TRUE si todo el proceso se ha ejecutado correctamente. Se divide en dos bloques principales:

1. Inicialización de todas la variables MELP para el modo (mode = SYNTHESIS), la tasa binaria de codificación (rate = RATE1200), longitud de palabra codificada (chwordsize = 8), el tamaño de trama (frameSize = 540 shorts) y el tamaño del búfer (bitBufSize = 11 bytes). Después de inicializar todas las variables, se llama a la función melp *melp_syn_init(),* para que configure el algoritmo.
2. Se ejecuta un bucle while, para que recorra todo el búfer de entrada de 11 en 11 bytes, los copie en la memoria loca y se pase a la función de síntesis, la cual devuelve la ventana decodificada en tramas de 540 shorts. Por último se copian los datos al búfer de salida que pasa a Asterisk.

3.3.3 Creación de una librería estática del codec MELP

Una librería es un programa con funcionalidades pre-construidas que pueden ser utilizadas por un ejecutable. En su interior contienen ficheros, variables y funciones.

Normalmente, cuando se habla de librerías, se define un conjunto de módulos objeto *.obj/.o* resultantes de la compilación de los archivos con las funciones y variables. Estos módulos se agrupan en un solo fichero que suele tener las extensiones *.lib*, *.a* o *.dll*. Estos ficheros permiten usar el conjunto de los módulos como una sola unidad y es de gran utilidad cuando se trabaja con aplicaciones grandes.

En el entorno Linux se pueden crear dos tipos de librerías:

- Librerías estáticas: Se vincula a nuestro programa en la compilación. Una vez creado el ejecutable, la librería no sirve para nada.
- Librerías dinámicas: Estas librerías no se vinculan al programa en la compilación, sino que cuando el programa se está ejecutando y necesite una función o variable de la librería, acudirá a esta. En este caso la librería no se puede eliminar nunca.

Para meter las funciones y variables del codec MELP en Asterisk hay que realizar una librería estática, ya que será la propia plataforma la que se encargará de crear una librería dinámica de la interfaz para acceder a sus funciones sólo cuando se use ese codec en concreto. Las librerías estáticas tienen sus inconvenientes y ventajas:

- Los programas resultantes ocupan más espacio en memoria
- Son fácilmente portables entre sistemas
- Los programas son más rápidos en ejecución.
- Si se modifican, el ejecutable no se ve afectado.

Para poder insertar el código fuente del codec MELP junto con las funciones creadas específicamente para Asterisk es necesario el desarrollo de una librería estática *libmelp.a* para que la interfaz reconozca las funciones y pueda usarlas. El desarrollo de la librería debe incluir todos los ficheros y cabeceras del código fuente en formato objeto:

1. El primer paso es realizar un makefile para la compilación de los ficheros y la creación de las extensiones objeto (file.o). El makefile debe incluir también los objetos generados para las funciones melp12_enc.c y melp12_dec.c:

melp12_dec.o, melp12_enc.o, classify.o, coeef.o, dsp_sub.o, fec_code.o, fft_lib.o, fs_lib.o, fsvq_cb.o, global.o, harm.o, lpc_lib.o, mathdp31.o, mathhalf.o, math_lib.o, mat_lib.o, melp_ana.o, melp_chn.o melp_sub.o, melp_syn.o, msvq_cb.o, npp.o, pitch.o, pit_lib.o, postfilt.o, qnt12.o, qnt12_cb.o, transcode.o y vq_lib.o

2. Una vez creados todos los ficheros objeto es necesario meterlos en una librería estática, para ello basta con ejecutar una simple instrucción que contenga todos los objetos y los vincule: ar [OPTION] [LIBNAME] [OBJECTS]. Para el codec MELP en concreto, la instrucción resulta de la siguiente forma:
   ar –rv libmelp.a melp12dec.o melp12_enc.o,... y así hasta especificar todos los objetos creados en el makefile.

3. Una vez realizada la librería, es necesario aparte de crearla en la carpeta # usr/src/asterisk/codecs/melp, copiarla en la carpeta # /usr/lib para que Asterisk pueda buscarla en caso de que no la encuentre en sus directorios.

3.3.4 Inserción de la librería estática de MELP en Asterisk

La realización de un fichero Makefile en Debian, viene descrito por el comando GNU *make.* Esta utilidad determina de forma automática que piezas de un programa necesitan ser compiladas, recompiladas o maneja cuestiones de comandos para recompilar.

Para hacer uso del comando *make*, se debe escribir un archivo llamado makefile que describe las relaciones internas de los ficheros se quieren compilar para realizar un programa y ofrece comandos para actualizar cada fichero. En un programa, normalmente el fichero ejecutable es actualizado con los módulos objeto, que provienen de la compilación del código fuente. Una vez realizado correctamente el makefile, siguiendo las normas esenciales de compilación y escritura, basta con escribir el comando make en el terminal, pero siempre teniendo en cuenta que debemos encontrarnos en la carpeta donde hemos creado el makefile.

Un Makefile se compone de reglas con las siguientes formas:

objetivo . . . : prerequitisitos . . .

comandos

. . .

. . .

- Objetivo: Suele ser el nombre de un archivo que es generado por un programa. Los objetivos son los ejecutables o módulos objeto. También puede ser el nombre de una acción que se lleva a cabo.

- Prerrequisito: Es un archivo que se utiliza como entrada para crear el objetivo. El objetivo normalmente depende de varios archivos.

- Comando: Es la acción que se lleva a cabo. Cada norma puede tener más de un comando, cada uno en su propia línea. Teniendo siempre en cuenta que es necesario un carácter de tabulación al comienzo de cada línea de comandos para que funcione el programa.

- Regla: Explica cómo y cuándo hacer un *make* de ciertos ficheros que son los objetivos de una cierta norma. El *make* lleva a cabo los comandos en los prerrequisitos para crear o actualizar el objetivo. Una regla también define cuándo y cómo se lleva a cabo la acción.

Para poder introducir de manera nativa la librería en el módulo cargable de los codecs, es necesario modificar su makefile donde se crean todas las librerías dinámicas *codec_*.so*. El makefile se encuentra en la carpeta # usr/src/asterisk/codecs:

LIBMELP:=melp/libmelp.a
$(LIBMELP):
@$(MAKE) –C melp all
$(if $ (filter codec_melp, $(EMBEDDED_MODS)),modules.link, codec_melp.so): $(LIBMELP)

Básicamente, estas instrucciones indican que es necesario utilizar la librería libmelp.a para vincularla a la librería dinámica codec_melp.so, creada a raíz del objeto codec_melp.o y libmelp.a, para la cual se han utilizado los ficheros fuente del codec MELP.

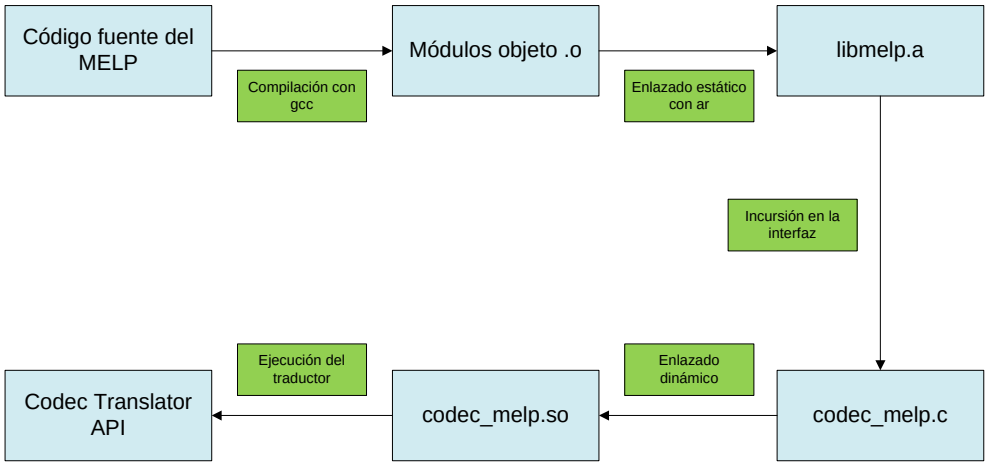

Figura 3.3: Esquema de funcionamiento de la interfaz MELP

3.4 Creación de un softphone con MELP

Un softphone es un software que permite realizar llamadas de teléfono convencionales a otros teléfonos u ordenadores a través de tu propio ordenador. Esto es posible gracias a la tecnología VoIP y los protocolos SIP y H.323.

En la actualidad, existen muchos tipos de softphone, tanto propietarios como libres y dentro de los softphones propietarios el más extendido es el Skype, ya que no es simplemente un teléfono sobre VoIP sino que ofrece un servicio peer-to-peer.

Debido a que el código de Skype es propietario y el código es cerrado, no se puede modificar por desarrolladores que quieran implementar nuevas funcionalidades, pero a su vez sí que permite añadirle nuevas aplicaciones suplementarias para que funcionen simultáneamente.

Para mi proyecto, he utlizado un Gateway desarrollado en la empresa Intecdom que permite, a través de Asterisk, la implementación de un softphone para hablar con usuarios Skype.

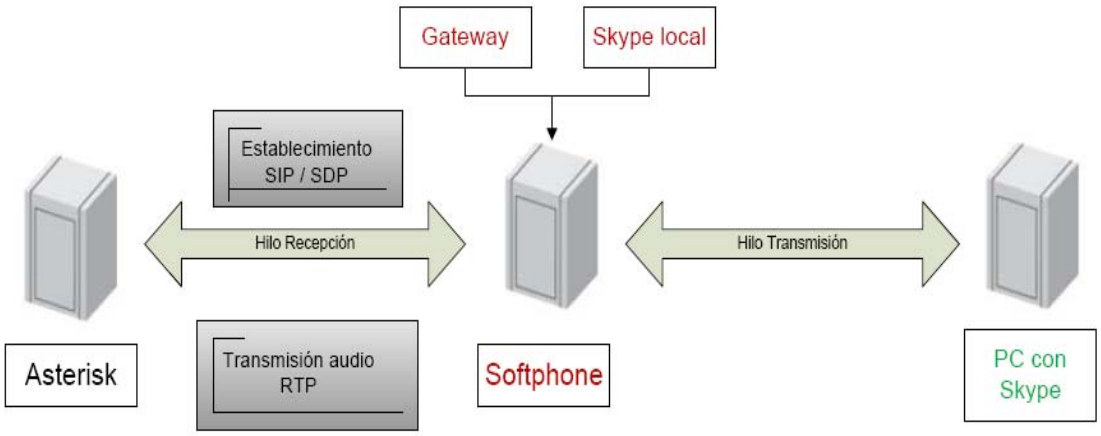

Figura 3.4: Esquema de funcionamiento de la pasalela como softphone

El Gateway funciona a través de dos hilos:

- Hilo de transmisión: coge las tramas codificadas provenientes de un usuario Skype mediante un servidor TCP, las remuestrea por ½, ya que Skype funciona con un muestreo de 16 kHz y lo codifica para finalmente pasarlo a Asterisk, mediante encapsulado RTP
- Hilo de recepción: coge las tramas provenientes de Asterisk a 8 kHz mediante encapsulamiento RTP y las decodifica para remuestrearlas por 2, para pasarlas a 16 kHz. Las codifica y la manda al Skype por TCP.

En mi proyecto, es necesaria la inserción en este sistema del codec MELP. Para su realización se han seguido los siguientes pasos:

- Inserción del código fuente del MELP, válido también para el sistema operativo Windows, en el código fuente del Gateway. Este paso incluye el reconocimiento por parte del sistema de todas las librerías y ficheros incluidos en el MELP
- Creación de un fichero que implemente las funciones de codificación y decodificación del MELP, específicamente para este sistema
- Añadir el codec MELP dentro de los hilos de transmisión y recepción.

A la hora de incluir el MELP dentro del Gateway hay que tener en cuenta ciertos aspectos para que la transmisión vía RTP sea posible:

- Dentro del Asterisk que yo he programado, el MELP tiene definidos un *Payload Type* y un *Timestamp* para las transmisiones. Para que las comunicaciones entre Asterisk y el Gateway funcionen tienen que ponerse iguales estos valores.

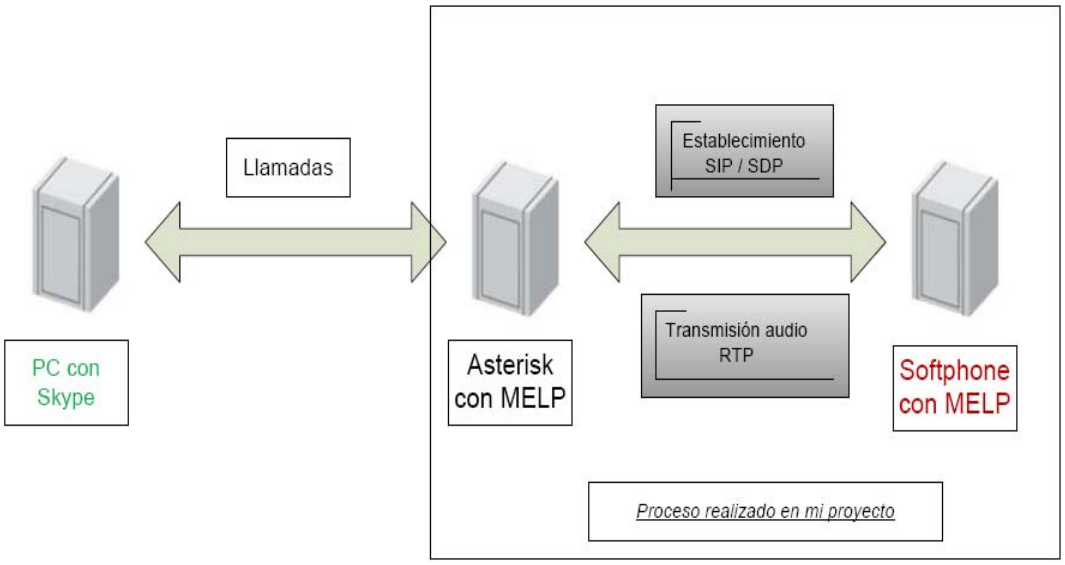

Figura 3.5: Esquema del funcionamiento del softphone con Asterisk

4. Test y Pruebas

4.1 Comprobación de la interfaz MELP

Cuando el proceso de inserción del MELP en Asterisk está realizado a continuación se deben realizar las pruebas pertinentes para comprobar que efectivamente, la plataforma reconoce el nuevo codec y que sus módulos para la traducción están listos.

El primer paso es la eliminación de los ficheros objeto que tiene ya Asterisk donde no se encuentra el módulo MELP. Este paso se realiza en el diretorio # usr/src/asterisk, mediante el comando *make clean*.

Una vez eliminados todos los módulos objeto, es la recompilación de los mismos en el mismo directorio con el comando *make*. En este paso es imprescindible que se creen los módulos *codec_melp.o* y *codec_melp.so*. Si no se han creado durante la ejecución del comando significa que Asterisk no ha reconocido la interfaz y que algún paso intermedio de nuestra configuración ha fallado.

Por último, se ejecuta el proceso *make install*, donde el módulo *codec_melp.so* es comprobado por la plataforma para ver si funciona correctamente o no. Muchas veces durante la ejecución del comando *make*, Asterisk reconoce la nueva interfaz, pero luego durante su iniciación con el comando *asterisk* es donde surgen todos los errores de programación o compilación que se han cometido durante la etapa de desarrollo.

Normalmente, los errores en esta etapa son mostrados al desarrollador mediante mensajes por pantalla. Si los errores son de Asterisk, simplemente aparecerá un mensaje predefinido diciendo el fichero, el módulo o la carpeta donde se ha producido el fallo. Por otro lado, si los errores se producen en las funciones creadas por el desarrollador, Asterisk para la ejecución. Si se ha producido un error, el comando *asterisk -vvvvvvgc* para su ejecución y Asterisk no se puede inicializar puesto que no está completo. Una posibilidad para ver dónde están los errores específicos de programación, es la inserción de las funciones *fprintf* en ciertas partes del código para ver su progreso. Otra opción es la ejecución del *debugger gdb* específico para el compilador *gcc* que te permite poner puntos de ruptura en Asterisk y en MELP.

Si el proceso de inserción se ha realizado correctamente y Asterisk ha podido crear las interfaces para la traducción, en pantalla se mostrará el siguiente mensaje:

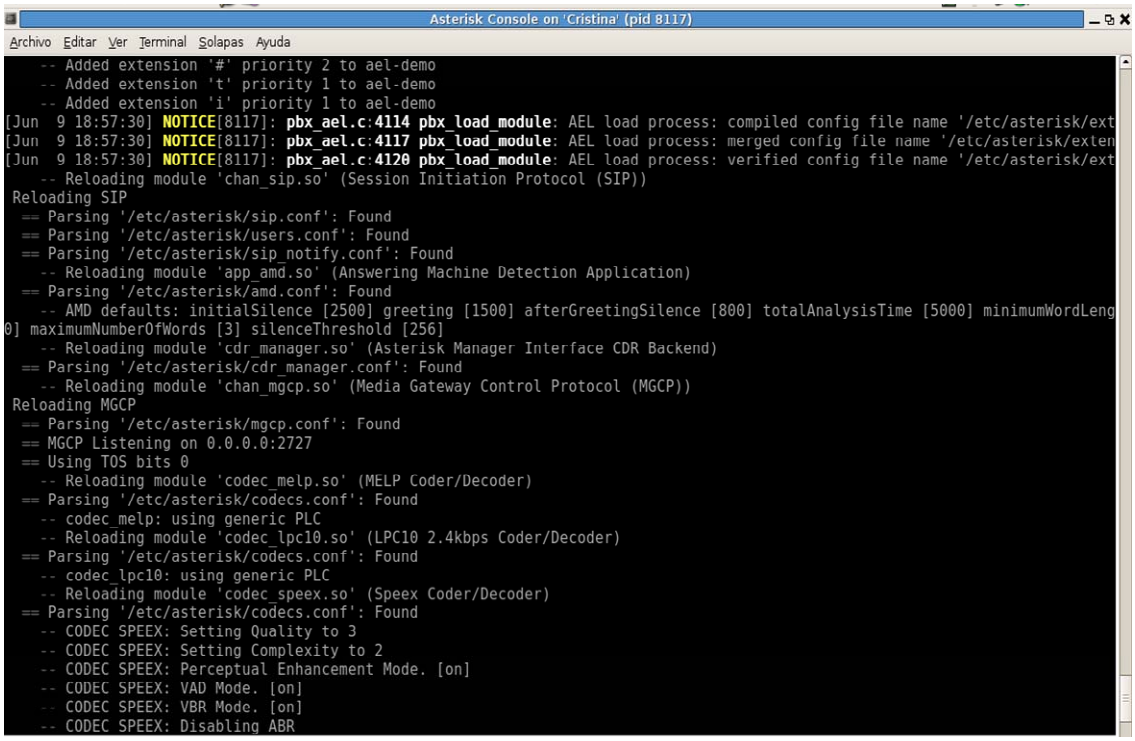

Figura 4.1: Ejecución de Asterisk

4.2 Comprobación de las funcionalidades de Asterisk

Los elementos que hay que comprobar en Asterisk son básicamente el funcionamiento de nuestro plan de llamadas configurado para los clientes SIP y el funcionamiento de los canales SIP que hemos definido.

Lo primero para comprobar el funcionamiento de los archivos *extensions.conf* y *sip.conf* es la instalación y configuración de los dispositivos por donde vamos a realizar las comunicaciones, es decir, los Softphones y un teléfono físico IP.

El teléfono IP es de la casa Teldat, modelo DI-VOZ, versión V.02.11. Antes de realizar su configuración, el teléfono se incluye en la red local LAN mediante el cable de red conectado a un switch Fast Ethernet. Para poder acceder al teléfono por la interfaz web para configurar los parámetros SIP, es necesario mediante el menú manual poner una dirección IP válida con su correspondiente máscara:

- SIP Settings: Se configuran los puertos y direcciones IP de Asterisk por donde van a pasar las comunicaciones.

SIP Phone Setting	
SIP Phone Port Number	5060 [1024 - 65535]
Registrar Server	
Registrar Server Domain Name/IP Address	192.168.3.114
Registrar Server Port Number	5060 [1024 - 65535]
Authentication Expire Time	3600 [60 - 9999]
Outbound Proxy Server	
Outbound Proxy Domain Name/IP Address	192.168.3.114
Outbound Proxy Port Number	5060 [1024 - 65535]
Message Server	
MWI Message Server Domain Name/IP Address	
Voice Message Account	
Others	

Figura 4.2: Configuración de las herramientas SIP

- SIP Account Setting: Se configuran los parámetros para el canal SIP de transmisión. Estos parámetros tienen que ser idénticos a los definidos en el fichero *sip.conf:*

SIP Account Setting	
Default Account	Account 2
Account 1 Setting	
Account Active	Disable Enable
Display Name	Cristina
SIP User Name	Cristina
Authentication User Name	Cristina
Authentication Password	
Register Status	UnRegister

Figura 4.3: Configuración del cliente SIP

Con los parámetros ya configurados, se da al botón de *Submit* para guardar la nueva configuración y reiniciar el teléfono IP.

El softphone que se va utilizar para realizar llamadas, que no posee el codec MELP, es el x-lite en su versión para Linux. Para instalarlo basta con conectarse por Internet a la página web oficial y descargarse la última actualización disponible

A diferencia de otras aplicaciones en Linux, para ejecutar el x-lite sólo es necesario extraer por el terminal el fichero ejecutable comprimido en un archivo *.tar* y teclear el comando *./x-lite* en la carpeta donde hayamos insertado el ejecutable. Para configurar los parámetros SIP, nos vamos al menú a la subcarpeta

Figura 4.4: Configuración del softphone x-lite

Finalmente, para el softphone que hace de pasarela con el codec melp, no hace falta configurar ningún parámetro puesto que durante la fase de inserción ya se han tenido en cuenta las direcciones IP de Asterisk y el puerto por donde realizar las transmisiones.

La comprobación de los ficheros configurables de Asterisk es mucho más dinámica:

- Con todos los aparatos configurados y ejecutándose, se inicia Asterisk. Una vez iniciado, la plataforma automáticamente realiza el registro de los canales SIP predefinidos. En caso de no detectar los dispositivos, la plataforma intenta la conexión un número

predeterminado de veces. Se pueden ver las conexiones mediante el comando *sip show peers*, que detecta si los dispositivos son alcanzables, si el host es dinámico, si se aplica o no el protocolo NAT y el puerto usado para transmitir.

- Si todos los canales SIP están activos, es hora de probar si el plan de llamadas configurado es válido. Lo único que hay que realizar son llamadas entre los dispositivos bien marcando la extensión de cada canal en el teléfono IP, o la *extensión@dirección IP Asterisk* desde los softphones. Durante todo el proceso de llamar y hablar, Asterisk muestra por pantalla todos los parámetros de la ejecución: Inicio, duración, mensajes lanzados, etc.

4.3 Comprobación de las transmisiones entre el softphone y Asterisk

El WireShark es un programa para analizar los protocolos de red. Es capaz de ejecutarse en la mayoría de las plataformas existentes y antiguamente era conocido como Ethereal. El objetivo con este programa es visionar el tráfico de datos entre el Gateway con el codec MELP, el telefóno IP y el softphone.

Durante las primeras pruebas, la transmisión sólo se realizó en un sentido, bien mediante el hilo transmisor o con el hilo receptor del Gateway. Estas pruebas sirvieron para confirmar que realmente se realizaban las funciones de codificación y decodificación tanto en Asterisk como en el Gateway. Una vez probado que las funciones del MELP y las funciones de los protocolos SIP, SDP y RTP se realizaban correctamente se procedió a la segunda etapa.

- Realizar transmisiones bidireccionales entre el teléfono IP y el softphone a través el Gateway.
- Comprobar las transmisiones para ver si efectivamente se transmiten correctamente las tramas MELP.

En la segunda etapa surgieron los siguientes problemas:

- Las transmisiones bidireccionales consumen completamente la CPU en la plataforma Windows por lo que la traducción se ralentiza
- Los paquetes transmitidos sufren un retraso considerable a la hora de transmitir debido a la carga impuesta sobre el microprocesador que durante toda la llamada está al 100%.
- No es posible la realización de llamadas fluidas.

1. La primera parte del programa nos muestra las transmisiones de paquetes realizada durante la llamada:
    - Mediante el protocolo SIP y SDP se establecen las características de la transmisión. Una vez realizadas las funciones SIP de INVITE y ACK, se procede a la realización de la llamada.
    - Por el protocolo RTP se ven los paquetes transmitidos en ambas direcciones una vez que se descuelga el telefono. En la imagen se puede observar que el payload de la transmisión es MELP.

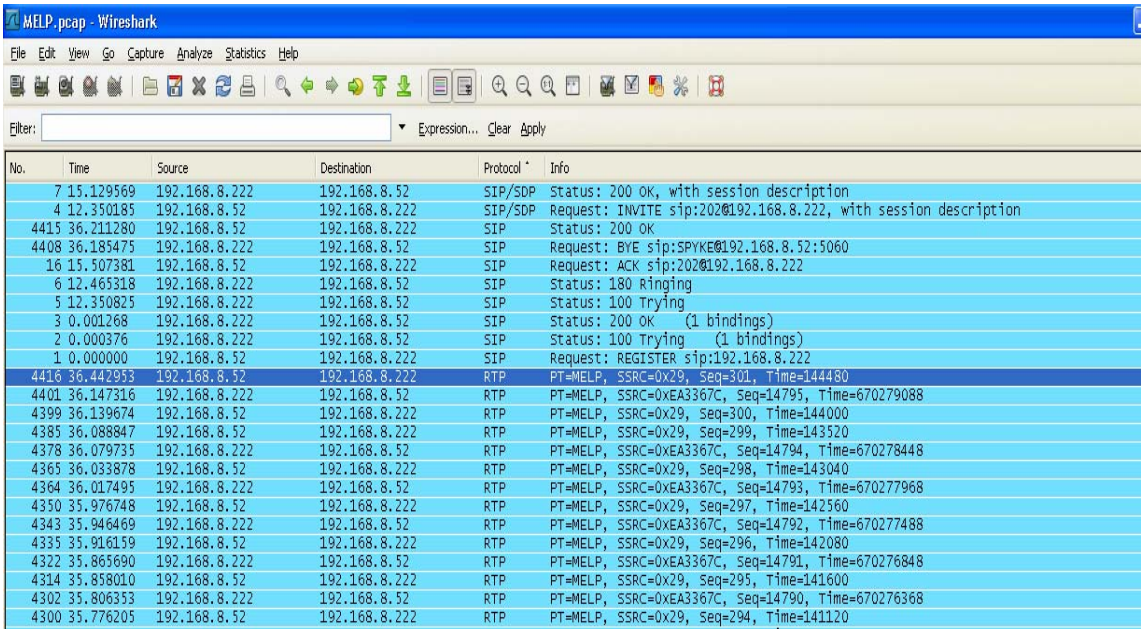

Figura 4.5: Transmisión de paquetes durante la llamada

2. La segunda parte del programa muestra las tramas según los diferentes encapsulados que lleva la trama final por la red de Internet:
    - Cabecera Ethernet
    - Cabecera IP
    - Cabecera UDP
    - Cabecera RTP

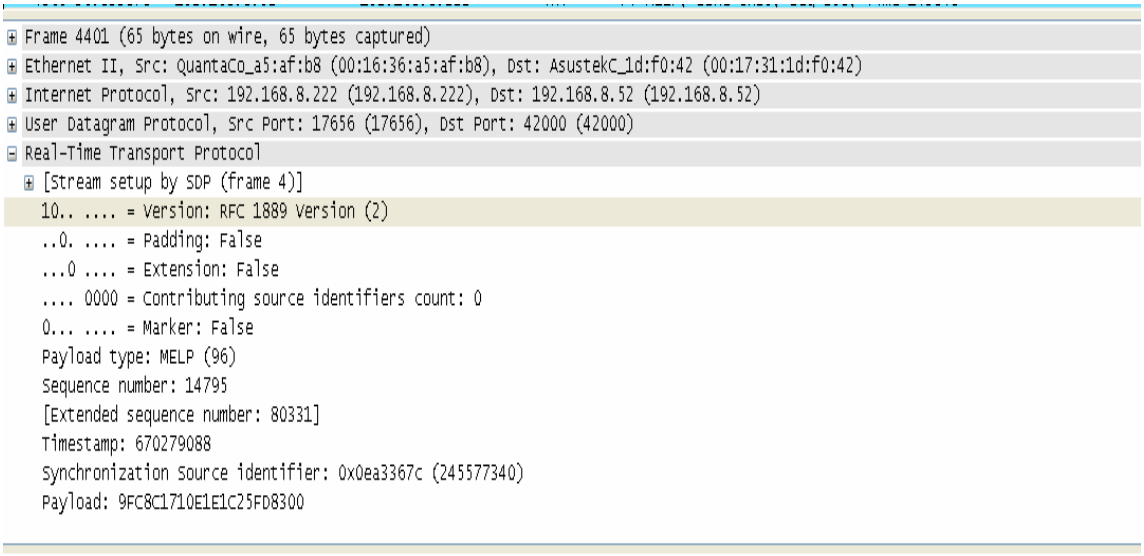

Figura 4.6: Composición de los paquetes

3. La tercera parte del programa muestra el payload de la trama RTP, donde comprobamos que efectivamente se transmiten los 11 bytes codificados del traductor MELP.

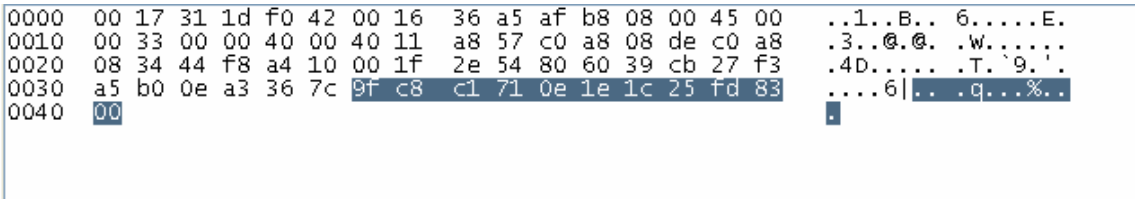

Figura 4.7: Trama MELP

Debido a la imposibilidad de realizar las llamadas sin un retraso evidente en los paquetes a causa del microprocesador y del sistema operativo Windows, no se han podido realizar los diferentes test de calidad existentes.

5. Conclusiones y Trabajo Futuro

5.1 Conclusiones

Cuando inicié este proyecto en Octubre en la empresa InTecDom, entre mis principales objetivos, se encontraban su realización con éxito y aprender todo lo que pudiera durante el proceso. El desarrollo del sistema ha seguido una serie de fases lógicas. En primer lugar, se ha llevado a cabo un proceso de investigación y aprendizaje de los diferentes componentes que tuvo como resultado la adquisición del conocimiento necesario para la aplicación práctica. De esta fase, se ha sacado básicamente el "estado del arte" en la actualidad de las aplicaciones y dispositivos utilizados para el proyecto. Durante esta etapa, han aparecido una serie de dificultades relacionadas sobre todo con la falta de información en el tema de aplicaciones con MELP e integraciones de codecs en Asterisk.

A este período, le siguió una segunda fase de desarrollo técnico, es decir, de programación de todas las aplicaciones para la inserción y configuración de los dispositivos para probar el funcionamiento del sistema.

En etapa han aparecido las mayores dificultades para la realización del presente sistema, ya que la fase de integración en Asterisk era un campo desconocido totalmente y unido a la complejidad de algoritmo MELP han dificultado su realización bastante. El desconocimiento de Linux y cómo utilizar su entorno de programación ha sido otro obstáculo a la hora de la realización. No obstante, a pesar de todas estas dificultades, el aprendizaje y la técnica adquirida durante los años de carrera, me ha permitido asimilar con relativa rapidez la formación necesaria para la finalización del PFC:

También ha sido necesario, para la realización y comprobación del sistema final, un importante trabajo en equipo, centrado sobre todo en el desarrollo del softphone a partir de productos creados por la empresa Intecdom, que permitiese la integración del codec MELP y su posterior uso para la realización de llamadas y comprobación del funcionamiento de Asterisk.

Todo el proceso, desde la primera fase de investigación, hasta la última fase de integración y pruebas ha supuesto una importante experiencia personal, ya que me ha permitido adquirir la suficiente experiencia y conocimiento para generar un producto por mi misma y para integrarme en la dinámica de trabajo de una empresa de telecomunicaciones.

5.2 Trabajo Futuro

Durante la realización del proyecto han surgido diferentes situaciones y problemas que se pueden mejoran en un futuro. Los problemas básicos que se han encontrado durante el proceso, son básicamente problemas de rendimiento y optimización:

- Carga de la CPU: El codec MELP es un algoritmo muy complejo que consume muchos recursos del microprocesador. Una posible mejora en este campo sería aligerar o dividir los procesos a través de los cuales se realiza la codificación y decodificación para no dejar al ordenador sobrecalentado cada vez que queremos usarlo.
- Tasa variable de bits: El código fuente del codec MELP da la posibilidad de hacer una conversión entre sus dos modos de funcionamiento: de 1200 bps a 2400 bps y de 24000bps a 1200bps. En un futuro, se puede realizar la integración de este modo en Asterisk, para que la plataforma en sí misma sea capaz de elegir que tasa binaria es la más adecuada en cada situación.
- Una posible aplicación suplementaria con el codec MELP, sería la inserción del codec MELP en Asterisk pero no para la realización de llamadas, si no para la grabación de mensajes. Esta interfaz se denominaría *format_melp.c*, y es parecida a la interfaz *codec_melp.c* pero en vez de hacer la traducción en tiempo real en ambos sentidos, el mensaje se codificaría en melp y se guardaría.
- Implementación física en una DSP del algoritmo para poder utilizar el hardware en teléfonos IP y muchas más aplicaciones. En la actualidad se pueden encontrar implementaciones en Internet sobre DSPs del codec MELP. Entre las más conocidas se encuentran la implementación del MELP a 2.4 kbps en una DSP TI fixed-point, o las implementaciónes en tiempo real SigC5xxx desarrolladas por Texas Instruments.

Referencias

[1] *Alberto M. Vilas Barros. Introducción a la VoIP Asterisk.* Curso Aje – Vigo: http://www.quobis.com/

[2] Steven W. Smith; *The Scientist and Engineer's Guide to Digital Signal Processing Second Edition.* ISBN 0-9660176-6-8. Page site: http://www.DSPguide.com

[3] Vigai K. Madisetti, Douglas B. Williams: Digital Signal Processing Handbook. Ed. Chapman & Hall/ CRCnetBase.

[4] DSPedia: http://www.dspedia.com

[5] WEB *Asterisk.* http://www.asterisk.org

[6] Van Meggelen J., Smith J., Madsen L.; *Asterisk. The Future of Telephony. Ed. O'Reilly (2005)* ISBN: 0-596-00962-3.

[7] IRONTEC: *CURSO VOZ SOBRE IP Y ASTERISK v1.0. Módulo I, II y III.* http://www.irontec.com

[8] Protocolo SIP: *IRIS-MMEDIA XV Grupos de Trabajo de RedIRIS Mayo 2003.*

[9] SIP: http://voip.megawan.com.ar/doku.php/sip

[10] Manuales Debian: http://cslu.cse.ogi.edu/HLTsurvey/ch5node3.html#SECTION51

[11] E.C. Tan, T.T. Teo; *REAL-TIME IMPLEMENTATION OF MELP VOCODER.* Journal of The Institution of Engineers, Singapore Vol. 44 Issue 3 2004

[12] M. A. Kohler; *A comparison of the New 2400 BPS MELP Federal Standard with Other Standard Coders.* U.S. Department of Defense, 9800 Savage Road STE 6515, Ft. Meade, MD 20755-6516

[13] Wikipedia. http://wikipedia.org. 09.06.2008.

Glosario

VoIP	**Voice over Internet Protocol**
IP	**Internet Protocol**
PBX	**Private Branch eXchange**
GPL	**General Public Licence**
MELP	**Mixed Excitation Linear Prediction**
SIP	**Session Initation Protocol**
SDP	**Session Description Protocol**
RTP	**Real-time Transport Protocol**
PSTN	**Public switch Telephone Network**
IAX	**Inter Asterisk eXchange**
FXS	**Foreign Exchange Station**
FXO	**Foreign Exchange Office**
DSP	**Digital Signal Processing**
LPC	**Linear Prediction Coding**
API	**Application Programming Interface**
LAN	**Local Area Network**

ANEXO A: Protocolo SIP

1. Introducción

El protocolo SIP (Session Initation Protocol), es un protocolo de iniciación de sesión creado por el grupo IETF para VoIP, para texto y sesiones multimedia,

El protocolo SIP es principalmente un protocolo de señalización de capa de aplicación para la iniciación, modificación y terminación de sesiones de comunicación multimedia entre usuarios. Estas sesiones incluyen llamadas telefónicas por Internet, distribución de datos multimedia y conferencias multimedia.

Las capacidades de señalización que ofrece el protocolo SIP son:

- Localización de los usuarios, lo que conlleva una movilidad
- Gestión del conjunto de participantes: disponibilidad del usuario y determinación de la voluntad del receptor para participar en las comunicaciones.
- Negociación de la capacidad del usuario: Determinación del medio y de sus parámetros
- Gestión de la sesión: Descripción de los componentes y de las características de las sesiones así como la transferencia, modificación y terminación de la sesión desde el propio usuario.

2. Características Principales

Las principales características que hacen que SIP se un protocolo en alza son:

- Simplicidad: Está basado en texto para una implementación y depuración simples, utiliza primitivas (métodos y respuestas basadas en el modelo HTTP) para el establecimiento de sesiones. No se definen servicios o funciones.
- Escalabilidad y flexibilidad: Tiene funcionalidades Proxy, de redireccionamiento, localización/registro que pueden residir en un único servidor o en varios distribuidos.
- Simplicidad de las URIs de usuario basadas en DNS.
- No es necesario un control centralizado: Tiene un funcionamiento Peer to Peer de forma nativa.

- SIP ofrece todas las potencialidades y las características comunes de la telefonía de Internet como: llamada o transferencia de medios, conferencia de llamada y llamada en espera.
- SIP funciona por encima de varios diversos protocolos del transporte.

El protocolo SIP posee una sintaxis similar a HTTP o SMTP, mediante el uso de URIs. Los métodos básicos que implementa son: INVITE, ACK, BYE, CANCEL, REGISTER, OPTIONS. Los mensajes se agrupan en transacciones y llamadas. Generalmente, el cuerpo de los mensajes contiene descripciones de sesiones multimedia y los códigos de respuesta similares a los de HTTP. La localización de usuarios está basada en el DNS y posee cabeceras como método de ampliación.

Los mensajes consisten de encabezados y un cuerpo de mensaje. Los cuerpos de mensaje de SIP para las llamadas telefónicas se definen en SDP- protocolo de descripción de la sesión.

- El SIP es un protocolo basado en texto que utiliza la codificación Utf-8
- Las aplicaciones SIP usan el puerto 5060 para ambos UDP y TCP. SIP puede utilizar otros transportes

Puesto que SIP es un protocolo flexible, es posible agregar más características y mantener la interoperabilidad hacia atrás

3. Arquitectura y Elementos

La arquitectura del protocolo SIP está integrada en la infraestructura web y sigue básicamente un modelo cliente-servidor, reutilizando conceptos de otros servicios (web, correo, dns).Los principales elementos que constituyen la arquitectura de este protocolo son:

- Agentes de usuario:
    - Agentes de usuario clientes (UAC).
    - Agentes de usuario servidores (UAS).
- Servidores:
    - Proxys, de registro y de redirección.

Un Proxy Servidor es una aplicación intermedia que actúa tanto como servidor y cliente, generando mensajes SIP a nombre del cliente que generó el mensaje original.

- Los mensajes pueden ser respondidos o encaminados a otros servidores donde se encuentra la localización actual del usuario

  -Interpreta, re-escribe o traduce los mensajes antes de encaminarlos.

-Autentifica y autoriza a los usuarios para los servicios

-Implementa políticas de encaminamiento.

-Hay dos tipos de Proxy Server:

- Outbound Proxy: Permite simplificar la administración de los usuarios de un dominio, aplica políticas, tarifica, etc.
- Inbound Proxy: Permite independizar al usuario del dispositivo que utiliza y de su localización.

Un mismo servidor puede funcionar como Proxy saliente o entrante de un dominio.

Las invitaciones de SIP son usadas para crear sesiones y llevan las descripciones de la sesión que permiten que los participantes convengan en un sistema de tipos de medios

Un Servidor de redirección es un agente de usuario servidor (UAS) que genera respuestas a las peticiones que recibe, ordenando al cliente entrar en contacto con un sistema alterno de URIs. En algunas arquitecturas puede ser deseable reducir la carga de los servidores Proxy que son responsables de las peticiones de encaminamiento, y mejorar la robustez del recorrido de los mensajes de señalización, mediante redirecciones.

La redirección permite que los servidores envíen la información de encaminamiento para una petición como respuesta al cliente, de tal modo quitándose del camino de los subsiguientes mensajes para una transacción mientras que ayudan en la localización del blanco de la petición. Cuando el autor de la petición recibe el cambio de dirección, enviará una nueva petición basada en la URI (s) que ha recibido. Propagando URIs desde el núcleo de la red hacia sus extremos, el cambio de dirección permite obtener una escalabilidad considerable en la red.

Un servidor de registro es un servidor que acepta peticiones de tipo REGISTER y pone la información que recibe de esas peticiones en el servicio de localización para el dominio que maneja (IP + Puerto) ya que los usuarios pueden tener múltiples localizaciones. Normalmente es el mismo servidor que el Proxy.

El protocolo SIP permite implementar dos tipos de movilidad diferentes:

- La movilidad personal donde el usuario puede ser alcanzado en un dispositivo cualquiera, registrándose en el servidor de registro.
- La movilidad propia al protocolo IP (VPN).

El registro permite mantener las localizaciones actuales del usuario de manera dinámica. Se basa en la localización actual. El servidor Proxy encaminará las llamadas al destino. SIP ofrece

la capacidad de descubrimiento. Si un usuario desea iniciar una sesión con otro usuario, el SIP debe descubrir el host actual en el cual el usuario de destinación es accesible. Este proceso de descubrimiento es logrado por elementos de la red SIP tales como servidores proxy y servidores de redirección que son los responsables de recibir una petición, determinar dónde enviarla basados en el conocimiento de localización de usuarios, y después enviarla allí. Para hacer esto, los elementos de la red SIP consultan un servicio abstracto conocido como servicio de localización, que proporciona los mapeos de dirección para un dominio particular. Éstos mapeos de direcciones mapean SIP URIs entrantes.

Las principales desventajas que conlleva el uso de este protocolo son:

- Problemas de Red: La utilización de un canal PtP para el streaming de audio RTP plantea numerosos problemas a nivel de red: NAT routers, firewall, etc.
- Interoperabilidad con PSTN: El protocolo H.323 ofrece mayores ventajas.

ANEXO B: Funcionalidades e Interfaces de una PBX y Asterisk

1. Funcionalidades y Servicios de una PBX

Las principales funcionalidades de este tipo de centralitas son:

1 El establecimiento de una comunicación entre dos terminales, ya sean llamadas internas o externas

2 El mantenimiento de dicha llamada durante el tiempo requerido por la transmisión.

3 La facturación de la duración de la llamada y los costes.

Entre los servicios más extendidos en las centralitas telefónicas se encuentran:

- Operadora automática/virtual: Permite pasar la llamada entrante a la extensión deseada mediante menús interactivos sin la necesidad de un operador físico. Esto es posible gracias a un sistema basado en reconocimiento de voz y/o tonos DTMF (Dual Tone Multi Frequency) generados al marcar en el teclado del teléfono. Este servicio automatizado permite atender a múltiples llamadas simultáneas.
- Buzón de Voz: Es un servicio de almacenamiento de mensajes de voz.
- Marcación rápida a números de emergencia como el 112, policía o bomberos.
- Desvío de llamada a otro Terminal en el caso de que la extensión se encuentre ocupada o no responda.
- Transferencia de llamadas a distintas extensiones.
- Follow-me: Listado de número para redireccionar en caso de que la extensión esté ocupada.
- Llamada en espera: Mantener una conversación en espera para atender nuevas llamadas.
- Música en espera: Servicio de reproducción de música antes de atender llamadas entrantes.
- Tarificación de llamadas: Sistema para el cálculo de los costes de las llamadas.
- CallerID: Servicio para la identificación del número de llamada entrante.
- Salas de conferencia: Servicio que permite mantener una conversación entre más de dos terminales.
- Listas negras: Restricción del acceso a ciertos números.
- Registro y listado de llamadas.
- Envío o recepción automática de faxes.
- Grabación de llamadas.

- Integración con bases de datos.
- Mensajería SMS.
- Monitorización de llamadas en curso.
- Distribuidor automático de tráfico de llamadas.

El funcionamiento de las PBX a gran escala en empresas es como un dispositivo físico que administra el tráfico de llamadas, realiza la contabilización del número de llamadas y la facturación. Actualmente, una PBX es como una computadora especializada que el usuario configura para permitir llamadas entrantes y salientes. Generalmente la PBX se conecta mediante un único enlace digital (E1/T1) utilizando 2 pares de cables con capacidad de 30 líneas. El funcionamiento para empresas pequeñas y medianas es por líneas comunes de la compañía telefónica, usando tantas líneas conectadas a la central telefónica como requiera el usuario.

2. Interfaces

Una interfaz es un circuito físico que establece una conexión entre dos sistemas permitiendo su comunicación. Dependiendo del estándar se establecen especificaciones técnicas concretas. Los puertos más comunes que se pueden encontrar en una centralita PBX son:

- FXO (Foreign eXchange Office): Dispositvo conmutador que permite la conexión analógica mediante un cable RJ-11 entre la PBX y la central local RTB (Red Telefónica Básica). La interfaz debe informar a la central local cuando se cierra el bucle de abonado. También permite crea extensiones a otras centralitas.
- FXS (Foreign eXchange Station): Tarjetas que permiten la conexión de teléfonos analógicos tradicionales a un ordenador y realizar y recibir llamadas por software hacia el exterior, hacia otros interfaces FXS o faxes. Utilizan un cable RJ-11
- E&M: Utiliza un cable RJ-48 para conectar llamadas remotas de una conexión de datos con las troncales de una PBX para ser distribuidas localmente. Se usan teléfonos de dos y cuatro hilos e interfaces troncales.
- BRI (Basic Rate Interface): Acceso básico RDSI
- PRI (Primary Rate Interface): Acceso primario RDSI. Proporciona mayor ancho de banda.
- 703/G.704 (E&M digital): Especifica las conexiones a centralitas a 2 Mbps.

La plataforma Asterisk puede funcionar como diferentes elementos:

- Asterisk como un conmutador (PBX): Se puede configurar como el núcleo de una IP PBX o una PBX híbrida, permitiendo cambio de llamadas, gestión de rutas, estableciendo características y realizando conexiones de los usuarios con el mundo exterior sobre IP, líneas analógicas (POTS) y digitales (T1/E1).
- Asterisk como gateway (pasarela): También puede ser configurado como el corazón de una pasarela de medios, introducir la red PTSN en la expansión para telefonía IP. Gracias a su arquitectura modular le permite la traducción entre una amplia gama de protocolos de comunicación y codecs.
- Asterisk como servidor de medios o características: Se puede configurar como un IVR (respuesta de voz interactiva), como un puente para conferencias, como asistente automatizado, buzón de voz o interfaz de telefonía para aplicaciones web.
- Asterisk en un call center (centralita): Asterisk ha sido adoptado por muchos call centres debido a su flexibilidad. Los desarrolladores de centralitas y centros de contacto han construido sistemas ACD completos basados en esta plataforma. Asterisk ha añadido nuevas soluciones dando la posibilidad de capacidades de agentes IP remotos, capacidades avanzadas de enrutamiento, marcación predictiva y mucho más.
- Asterisk en la red: Los proveedores de servicios de telefonía (ITSP), las compañías proveedoras de telecomunicaciones (CLECS) e incluso los operadores incumbentes tradicionales han descubierto el poder del código abierto y las facilidades de Asterisk. Las soluciones pre-pago, los sistemas de buzón de voz, los servidores "feature servers"y los servicios por grupos han ayudado a reducir costes y permitido una mayor flexibilidad.

Asterisk se ejecuta en una amplia variedad de sistemas operativos, incluyendo Linux, Mac OSX, OPenBSC, FreeBSD y Solaris de Sun y ofrece todas las características avanzadas de una PBX propietaria. Está diseña para proporcionar facilidad, flexibilidad e interoperación con casi todos los estándares de telefonía utilizando para ello un equipo hardware relativamente barato.

La arquitectura de Asterisk está cuidadosamente diseñada y se basa en APIs (Application Programming Interface) específicas en torno a un núcleo avanzado y centralizado PBX. El avanzado núcleo interior se ocupa de la interconexión de las PBX, abstraídas de los protocolos específicos, codecs y las interfaces hardware para las aplicaciones telefónicas. Esto permite un uso adecuado de la tecnología disponible para desempeñar funciones esenciales.

Las APIs se dividen en cuatro módulos recargables lo que facilita la abstracción del hardware y de los protocolos. Mediante el uso de este sistema de módulos recargables, el núcleo de Asterisk no tiene porque preocuparse de los detalles:

- Channel API: Maneja el tipo de conexión de una llamada entrante, ya que puede ser mediante VoIP, RSDI, PRI o por señalización por robo de bits. Los módulos dinámicos son cargados para manejar la capa inferior de las conexiones.
- Application API: Esta aplicación permite a diversos módulos separados ejecutarse para realizar varias funciones. Conferencias, listado en directorio, buzón de voz, transmisión de datos en línea y cualquier tarea que se pueda realizar por una PBX.
- Codec Translator API: Carga los módulos de los codec que permiten la tarea de codificar y decodificar diferentes formatos de audio como por ejemplo: GSM, Mu-law, A-law o en el caso de mi proyecto MELP.
- File Format API: Maneja la lectura y la escritura de diversos formatos de archivo para el almacenamiento de datos en el sistema de ficheros.

Los elementos tratados internamente por el núcleo son:

- Conmutación de la PBX: Al ser un sistema conmutador permite conectar llamadas entre varios usuarios y tareas automatizadas. El núcleo de conmutación conecta de manera transparente llamadas entrantes en diversas interfaces hardware y software.
- Application Launcher: Lanza aplicaciones que realizan servicios tales como buzón de voz, reproducción de ficheros y listado de directorios.
- Codec Translator: Usa los módulos de la API para la codificación y decodificación de diversos formatos de compresión de audio. Por defecto, hay una serie de codec disponibles para satisfacer las diversas necesidades y conseguir el mejor equilibro entre calidad y ancho de banda disponible.
- Scheduler and I/O Manager: Maneja tareas de bajo nivel de programación y la gestión del sistema para un rendimiento óptimo bajo todas las condiciones de carga.

Mediante el uso de estas APIs, Asterisk consigue una abstracción completa entre las funciones del núcleo como sistema PBX y las diversas tecnologías existentes en el campo de la telefonía. La estructura modular permite integrar sin problemas la conmutación hardware actual y las crecientes tecnologías de voz empaquetada. La habilidad para cargar los módulos que contienen los codec permite dar soporte a conexiones lentas que necesitan grandes compresiones y proveer gran calidad de audio en conexiones menos exigentes.

La API de aplicación proporciona un uso flexible de los módulos para realizar cualquier función de flexibilidad bajo demanda y permite un desarrollo abierto de nuevas aplicaciones para necesitadas y situaciones especificas y únicas. Además, la carga de los módulos permite

tener a los administradores la habilidad para diseñar el camino adecuado para los usuarios y a su vez poder modificar los caminos según las necesidades de la comunicación.

Entre las funcionalidades generales que ofrece Asterisk, se encuentran:

- Soporte de los estándares de telefonía tradicionales: líneas analógicas, líneas digitales (E1/T1) y accesos básicos.
- Soporte de la mayoría de protocolos VoIP: SIP, IAX, H.323, MGCP, Cisco Skinny.
- Funcionalidades de centralita: Transferencia, Llamada en espera, Caller ID, Música en espera, Buzón de voz personal y por mail, colas de llamadas y colas con prioridad, desvíos, registro de llamadas en BD, etc.

Funcionalidades avanzadas: IVR (Interactive Voice Response) para la gestión de llamadas con menús interactivos, LCR (Least Cost Routing) para el encaminamiento de llamadas por el proveedor VoIP más económico, AGI (Asterisk Gateway Interface) para la integración con todo tipo de aplicaciones externas, AMI (Asterisk Management Interface) para la gestión y el control remoto de Asterisk y Configuración en bases de datos

ANEXO C: Codificadores de voz

Los codificadores de la voz se dividen en dos categorías: Codificadores basados en modelos y codificadores basados en la forma de la onda:

1. Codificadores basados en modelos

Están basados en modelos de producción del habla. Entre los más comunes podemos encontrar:

- *Codificador de voz LPC:* El análisis es realizado en una trama del discurso y la información es cuantificada y transmitida. Se determinan los silencios mediante la señal residual LPC o la original pero siempre basándose en la periodicidad de la misma. Si la trama es sonora, el periodo del pitch es transmitido y la excitación de la señal es convertida a un tren de impulsos periódicos.
- *Codificadores de Excitación Multibanda (MBE):* La premisa más importante de estos codificadores es que la onda que representa la voz puede ser modelada como una combinación de onda sinusoidales armónicas y ruido en banda estrecha. Dentro del ancho de banda, la voz es clasificada como periódica o aperiódica. Las sinusoides armónicamente relacionadas generan los componentes periódicos mientras que el ruido genera los componentes no periódicos. La trama transmitida consiste en un conjunto de decisiones sonoras o no.
- *Codificadores de interpolación de la forma de onda:* En estos codificadores se asume que la voz está compuesta por formas de onda periódicas que evolucionan lentamente (SEW) y formas de onda de ruido que evolucionan rápidamente (REW). La trama se analiza para extraer su forma característica. Estas formas se filtran para separarlas. Se transmiten de forma independiente el LPC, el pitch, el espectro de las formas SEW y REW y la energía global.

2. Codificadores de voz según la forma de onda en el dominio del tiempo

Tienen la propiedad de que no hay error en la cuantificación y la señal original puede ser reproducida con total exactitud. Entre los más comunes se encuentran:

- *Modulación Adaptativa Diferencial de los Pulsos (ADPCM):* Se basa en la cuantificación muestra por muestra y la predicción del error. Las dos partes que se pueden adaptar del codificador son el cuantificador y el filtro de predicción. La adaptación se puede ser progresiva o regresiva. En la regresiva la cuantificación se basa sólo en las muestras previas. En la progresiva los valores adaptados se cuantifican y se transmiten.
- *Codificadores basados en la modulación de las deltas:* El cuantificador es sólo el signo del bit. El tamaño del paso en la cuantificación es adaptativo. La calidad de esta codificación tiende a ser directamente proporcional a la frecuencia de muestreo.
- *Codificación de Predicción Adaptativa*: Se basa en la teoría de que cuánto mejor es la realización del filtro de predicción, menor es la tasa de bits que se necesita para codificar la señal de voz. Se usa un filtro de predicción lineal adaptativo de gran orden. La voz es cuantificada trama por trama.
- *Codificadores de Predicción lineal con análisis por medio de síntesis:* Se basa en la codificación muestra por muestra. Se empieza por un análisis LPC progresivamente adaptativo. La señal cuantificada se pasa por un filtro con peso perceptual. Las mejores señales excitadas son las escogidas. Entre los codificadores más comunes se encuentran el MPLPC (Multipulse Linear Predictive Coding), el CELP (Codebook Excited Linear Predictive Coding) y el VSELP (Vector Sum Excitation Linear Predictive Coding)

3. Codificadores de voz según la forma de onda en el dominio de la frecuencia

- *Codificadores de Sub-bandas*: El concepto se basa en dividir la señal de voz en un número de bandas de frecuencias y cuantificar cada banda por separado. De esta manera el ruido de cuantificación se mantiene dentro de la banda. Normalmente se usan bancos de filtros y espejos de cuadratura. Tiene las propiedades de que en ausencia de error de cuantificación todo el aliasing causado por la diezmación en el análisis por banco de filtros se elimina en la síntesis y las bandas pueden ser muestreadas de manera crítica.
- *Codificadores de transformación adaptativa (ATC):* Son como una extensión de los codificadores de sub-bandas. El banco de filtros es reemplazado por una transformación como la FFT o la DCT o cualquier otro banco de filtros que haga una transformación. Proporciona una mayor resolución en el análisis permitiendo explotar la estructura armónica espectral del pitch. Se usa una cuantificación muy sofistica que recolocan dinámicamente los bits para conseguir el mejor rendimiento.

ANEXO D: Tecnologías para la síntesis de voz

Las dos tecnologías clave para la generación del habla sintética son la síntesis por concatenación y la síntesis por picos de frecuencias:

- *Síntesis por concatenación*: Se basa en la concatenación de los segmentos del habla grabada. Generalmente, este método produce la voz sintetizada más natural. Sin embargo, las diferencias entre las variaciones naturales en el habla y la naturaleza de las técnicas automatizadas para segmentar las formas de onda algunas veces se perciben en los fallos de sistema a la salida. Hay tres sub-tipos básicos de síntesis por concatenación:

    - Síntesis por selección de unidad: Esta tecnología usa grandes bases de datos que contienen el habla grabada. Durante la creación de la base de datos, cada sonido grabado se divide en varias o en todas las clases siguientes: fonemas individuales, sílabas, morfemas, palabras, locuciones y frases. Normalmente, la división en segmentos se hace usando un reconocedor de voz especialmente modificado fijado con un "alineamiento forzado" y con alguna corrección manual al finalizar, usando representaciones visuales tales como la forma de onda y el espectrograma. Una vez realizado todo este proceso se crea un índice en la base de dato basado en la segmentación y los parámetros acústicos como la frecuencia fundamental o pitch, la duración, la posición en la sílaba y los fonemas vecinos. En tiempo de ejecución, el enunciado final es creado mediante la determinación de la mejor cadena de unidades candidatas seleccionadas desde la base de datos (selección de unidades). Este proceso es logrado mediante un árbol de decisión ponderado. Esta tecnología proporciona una gran naturalidad y normalmente la salida es indistinguible con respecto a voces reales humanas, especialmente cuando se usa el método TtS. Sin embargo, para obtener la máxima naturalidad se necesitan bases de datos enormes que pueden llegar a ocupar gigabytes de espacio.
    - Síntesis por difonos: Esta tecnología usa una base de datos de voz muy pequeña que contiene todos los difonos (transiciones de sonido a sonido) que ocurren en el lenguaje. El número de difonos depende de las tácticas fonéticas del lenguaje. En esta síntesis, sólo se almacena un ejemplo de cada difono en la base de datos. En tiempo de ejecución, la prosodia objetivo de una frase es superponer en estas unidades mínimas por medio de las técnicas de procesamiento digital de señales tales como codificación predictiva lineal, PSOLA o MBROLA. La calidad de la voz sintetizada es generalmente peor que en los sistemas anteriores, ya que esta

técnica sufre de fallos sónicos de la síntesis por concatenación y de sonidos de naturaleza robótica.

- Síntesis por dominio específico: Esta tecnología concatena palabras pregrabadas y frases para crear sentencias completas. Se usa en aplicaciones donde la variedad de textos está limitada a un dominio en particular. La tecnología es muy simple a la hora de implementarla. El nivel de naturalidad puede ser muy alto debido a que el número de frases es limitado y encajan muy bien con la prosodia y la entonación de las grabaciones originales.

- *Síntesis por resonancia*: Esta tecnología no usa muestras de la voz humana en tiempo de ejecución. En su lugar, la voz sintetizada se crea usando un modelo acústico. La frecuencia fundamental, los niveles sonoros y de ruido son variados en el tiempo para crear la forma de onda del habla artificial. Muchos de los sistemas que usan esta tecnología tienen como resultado una voz artificial con sonoridad robótica que se distingue perfectamente de la humana. Sin embargo, este método puede ser bastante inteligible, incluso a velocidades muy elevadas, evitando los fallos acústicos. Normalmente ocupan poco espacio y no usan bases de datos. Esta síntesis controla todos los aspectos de la voz, como una gran variedad de prosodias y entonaciones, así como gran variedad de emociones y tonos de voz.
- *Síntesis por articulación de la voz*: Son técnicas computacionales basadas en modelos del tracto vocal humano y los procesos de articulación que ahí ocurren.
- *Síntesis basada en HMM:* Este método se basa en modelos ocultos de Markov. En este sistema, el espectro de las frecuencias (tracto vocal), la frecuencia fundamental y la duración (prosodia) del habla son modelados simultáneamente por HMM. Las formas de onda son generadas basándose en el criterio de máxima igualdad.
- *Síntesis por onda sinusoidal*: Es una técnica de síntesis mediante el reemplazo de las resonancias por tonos puros.

ANEXO E. Codificador y Decodificador MELP

1. Codificador

El codificador está dividido en dos partes: El analizador y el cuantificador. La señal de entrada es muestreada 8000 veces por segundo a 16 bits por muestra. Cada muestra se guarda para formar una trama de 180 muestras (22.5ms con una frecuencia de muestreo de 8000Hz).

Los parámetros necesarios que son extraídos de la señal de entrada son: el pitch, los coeficientes de décimo orden de la predicción linear, sonidos en paso banda, la ganancia y el indicador de pulsos aperiódicos.

Mediante un buffer, el codec es capaz de almacenar la trama previa de la señal de voz en memoria y así poder tener la trama actual y la anterior disponibles para ser analizadas. Esto permite el análisis de una trama de mayor tamaño lo que conduce a un estudio más preciso de las características del habla. Además, los cambios repentinos y abruptos pueden ser analizados dentro del contexto de un número mayor de muestras. Sin embargo, este "buffering" implica un retraso a tener en cuenta. Los tamaños de las ventanas para analizar los diferentes parámetros son diferentes.

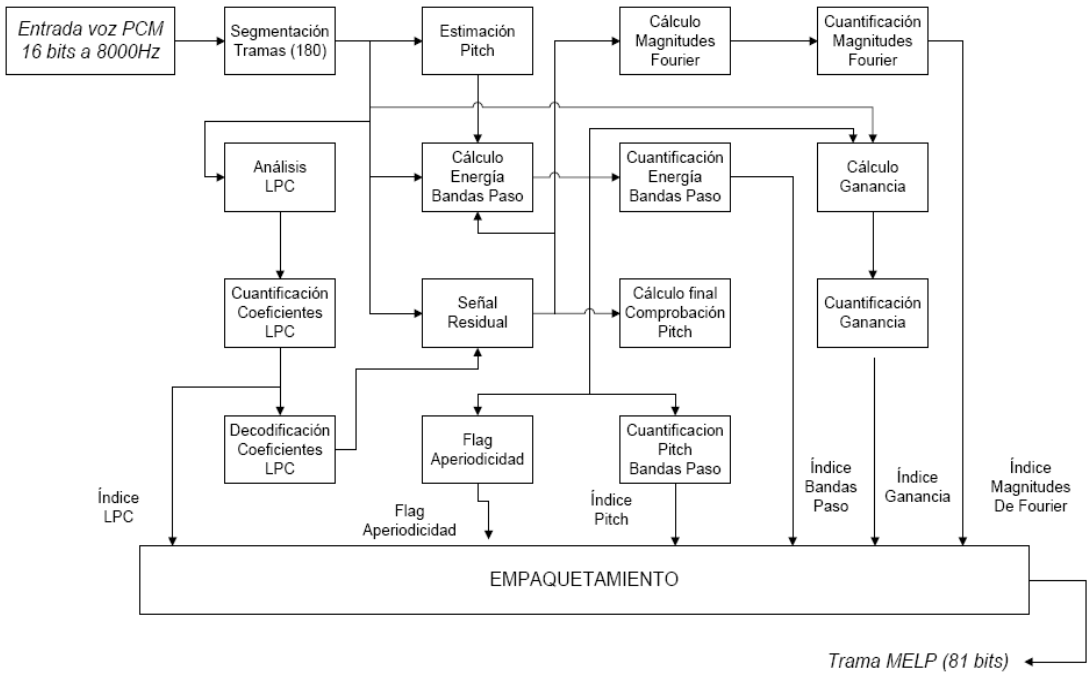

Figura E.1: Diagrama de bloques funcionales del codificador MELP

A continuación se detallan los pasos a seguir para la codificación del habla:

1.1 Análisis del Pitch y de las Tramas Sonoras

El primer paso en la codificación es la eliminación de las bajas frecuencias que se encuentran en la señal de entrada. Este procedimiento se implementa mediante un filtro paso alto Chebyshev de tipo II de cuarto orden, con una frecuencia de corte de 60Hz y 30dB de rechazo en la banda de supresión.

Durante todo el proceso de codificación, en el buffer se mantienen las muestras más recientes de la señal de entrada una vez filtrada para eliminar las bajas frecuencias. Así mismo, el buffer se extiende para poder contener las muestras necesarias del pasado y el futuro para el proceso de codificación. La última muestra en el actual marco de análisis sirve como punto de referencia para muchos de los cálculos.

Para proceder al cálculo del pitch primero es necesario procesar la señal de entrada mediante un filtro paso bajo Butterworth. El valor entero del pitch es aquel que maximiza la función de autocorrelación normalizada:

$$r(\tau) = \frac{c_\tau(0, \tau)}{\sqrt{c_\tau(0, 0)c_\tau(\tau, \tau)}},$$

$$c_\tau(m, n) = \sum_{k=-\lfloor \tau/2 \rfloor - 80}^{-\lfloor \tau/2 \rfloor + 79} s_{k+m} s_{k+n}$$

El centro de la ventana de análisis del pitch es determinado por la muestra s_0. El índice temporal k permite que la ventana esté continuamente alineada con el punto medio y la normalización compensa los cambios de amplitud de la señal permitiendo calcular el valor de $r(\tau)$ sin que decaiga la magnitud.

La intensidad de la autocorrelación es usada para determinar las muestras sonoras. Si $r(\tau)$ tiene un valor alto se puede decir con bastante certeza que la trama de voz tiene fuertes propiedades periódicas y puede ser clasificada como sonora.

La determinación de las intensidades sonoras se lleva a cabo mediante el filtrado de la señal de entrada en cinco bandas de frecuencias separadas (Vbp_i =1,2,…,5). Este procedimiento se

realiza mediante filtros Butterworth paso banda de sexto orden. Las bandas de paso comprenden las siguientes frecuencias: 0-500, 500-1000, 1000-2000, 2000-3000 y 3000-4000 Hz. También se realiza un análisis con una autocorrelación y una búsqueda del valor de pico teniendo como referencia la demora del pitch encontrado en la primera banda.

El valor de pico de la señal residual es calculado sobre una ventana de 160 muestras centrada en la última muestra de la trama actual, donde r_n es la muestra n de la señal residual.

$$Peakiness = \left[\left(\sum_{n=1}^{160} r_n^2\right)/160\right]^{1/2} / \left[\left(\sum_{n=1}^{160} |r_n|\right)/160\right],$$

Valores altos en los picos de la señal aun cuando los valores de la autocorrelación son pequeños puede significar que la señal de voz es poco periódica o que hay jitter. Este fenómeno puede ocurrir en regiones de transición en el habla donde hay cambios en el pitch bruscos.

Si el valor de pico excede 1.34, entonces el valor Vbp_1 se fuerza a 1. Si excede 1.6 entonces las tres bandas más bajas son forzadas todas a 1.

El Indicador Aperiódico sirve para decir al decodificador cuando los pulsos resultantes de la excitación son periódicos o aperiódicos. Se basa en el valor obtenido en el análisis del sonido en paso banda Vbp_1. Si este valor es menor que 0.5 el indicador se pone a 1 en caso contrario se pone a 0.

A continuación se realiza un perfeccionamiento del valor del pitch obtenido mediante la señal residual y un mejoramiento fraccional del pitch, usando para ello la señal obtenida en la primera banda de frecuencias en el banco de filtros (0-500Hz) a la que se pasa por una autocorrelación para llevar a cabo una búsqueda del valor entero del pitch sobre un rango de 5 muestras menos y 5 muestras más del valor actual. Esta medida está centrada en la salida del filtro cuando ésta es la última muestra de la trama actual.

La señal residual se obtiene mediante un filtro lineal de predicción de síntesis de orden p:

$$A(z) = H^{-1}(z) = 1 + \sum_{i=1}^{p} \alpha_i z^{-i}$$

donde los coeficientes α_i son los coeficientes de la predicción lineal.

El mejoramiento fraccional del pitch usa una fórmula de interpolación para incrementar la exactitud del valor entero del pitch.

$$\Delta = \frac{c_T(0,T+1)c_T(T,T) - c_T(0,T)c_T(T,T+1)}{c_T(0,T+1)\left[c_T(T,T) - c_T(T,T+1)\right] + c_T(0,T)\left[c_T(T+1,T+1) - c_T(T,T+1)\right]}$$

$$r(T+\Delta) = \frac{(1-\Delta)c_T(0,T) + \Delta c_T(0,T+1)}{\sqrt{c_T(0,0)\left[(1-\Delta)^2 c_T(T,T) + 2\Delta(1-\Delta)c_T(T,T+1) + \Delta^2 c_T(T+1,T+1)\right]}}$$

Primero se calcula el valor de offset del pitch fraccional Δ mediante la fórmula de interpolación. Este valor se usa en la autocorrelación normalizada para hallar el valor final. Si el valor obtenido es inferior a 0.55 se procede a calcular el valor medio. En caso contrario se procede a ejecutar un doble chequeo de los resultados obtenidos.

El valor medio a largo plazo del pitch P_{avg} es actualizado si el valor de la autocorrelación es superior a 0.8 y la Ganancia G_2 es superior a 30db y se deposita en el buffer con los valores del pitch más recientes que se encuentran en el buffer. Por contra si no se superan esas cotas, los valores del buffer son cambiados a un valor por defecto del pitch de 50 muestras mediante la fórmula:

$$p_i = 0.95p_i + 0.05P_{default} \ , \ i = 1, 2, 3$$

Una vez obtenidos estos valores se hace la media.

Si el pitch mejorado se pasa a través de un procedimiento de Doble Chequeo, éste busca y corrige los valores del pitch que son múltiplos del valor actual. Si el valor obtenido de la autocorrelación es superior a 0.6 entonces se efectúa el chequeo con un valor del umbral duplicado. El chequeo devuelve un valor nuevo del pitch y de su correspondiente autocorrelación.

La Ganancia de la señal de voz entrante es medida dos veces por ventana analizada usando para ello la ventana de longitud adaptable del pitch. La longitud es la misma para ambas ganancias

$$G_i = 10\log_{10}\left(0.01 + \frac{1}{L}\sum_{n=1}^{L} s_n^2\right)$$

El cálculo para obtener la primera ventana produce G_1 y está centrada 90 muestras antes de la última muestra del marco actual. El cálculo para la segunda ventana produce G_2 y se centra en la última muestra del marco actual. La ganancia es el valor RMS, medido en dB de la señal en la ventana.

1.2 Análisis por Predicción Lineal

Las frecuencias contenidas en el habla humana pueden ser estimadas con precisión usando el análisis de la predicción lineal sobre un segmento del habla que contenga características que puedan ser asumidas como estacionarias. Existen dos procedimientos para obtener los coeficientes: La covarianza y la autocorrelación.

En el caso del codec MELP se utiliza el método de la autocorrelación. El método consta en un análisis predictivo lineal de décimo orden sobre la señal de entrada usando una ventana Hamming de 200 muestras (25ms) centrada en la última muestra de la ventana que se está codificando en el momento. La autocorrelación es implementada mediante la recursión Levinson-Durbin. Además se introduce un coeficiente de expansión del ancho de banda de 0.994 (15 Hz) a los coeficientes de predicción.

La señal residual predictiva lineal es calculada filtrando la señal de entrada con el filtro de predicción lineal cuyos coeficientes se han predeterminado anteriormente. La ventana residual se centra en la última muestra del marco actual y es lo suficientemente amplia como para hacer el cálculo final del valor del pitch.

En un sistema de predicción lineal que codifica/decodifica el habla los coeficientes tienen que ser cuantificados para poder transmitirlos. Debido a la aproximación basada en tramas, los coeficientes son transmitidos una vez por trama por lo que es necesario interpolar los valores de los valores transmitidos entre tramas para la síntesis del habla. Debido a que los coeficientes son el resultado de un filtro basado en el cálculo de los polos, la interpolación puede dar resultados inestables y una cuantificación directa puede producir una pérdida de calidad inaceptable y afectar también a la estabilidad.

Los coeficientes LSF (Lineal Spectral Frequencies) son utilizados como método alternativo debido a la estabilidad que proporcionan a la hora de cuantificar y su buen comportamiento a la hora de interpolar en la tramas con cambios espectrales más bien suaves.

1.3 Cuantificación de los Parámetros

El módulo de cuantificación cuantifica todos los parámetros obtenidos del módulo de análisis: 10 coeficientes LSF, pitch, las 2 ganancias, las 5 intensidades sonoras, las magnitudes de Fourier y el Indicador aperiódico.

Los coeficientes LPC se transforman al dominio de la frecuencia (LSF) mediante el uso de los polinomios Chebyshev, forzando a los coeficientes a tener un orden ascendente con una separación mínima de 50Hz. El vector resultante es cuantificado usando un algoritmo multi-etapa (MSVQ). Los códigos de este algoritmo constan de cuatro etapas cuyos índices tienen 7, 6, 6 y 6 bits, respectivamente:

$$d^2(\mathbf{f},\hat{\mathbf{f}}) = \sum_{i=1}^{10} w_i (f_i - \hat{f}_i)^2 \text{ , where } \mathbf{f} = [f_1, f_2, \ldots, f_{10}],\ \hat{\mathbf{f}} = \left[\hat{f}_1, \hat{f}_2, \ldots, \hat{f}_{10}\right]$$

$$w_i = \begin{cases} P(f_i)^{0.3}, & 1 \le i \le 8 \\ 0.64P(f_i)^{0.3}, & i = 9 \\ 0.16P(f_i)^{0.3}, & i = 10 \end{cases}$$

El valor del pitch es cuantificado en una escala logarítmica mediante un cuantificador uniforme con 99 niveles y con un rango que va desde 20 muestras hasta 160. Los valores del pitch cuantificados son codificados en palabras de 7 bits de longitud gracias a tablas de traducción predeterminadas. Las palabras con todo ceros representan el estado no sonoro y depende de que el valor V_{bp1} sea menor que 0.6.

Los valores de las ganancias obtenidas previamente son cuantificados de la siguiente forma:

-La primera ganancia G_1 es cuantificada usando palabras de 3 bits usando un algoritmo adaptativo

-La segunda ganancia *G2* es cuantificada usando palabras de 5 bits y un cuantificador uniforme con un rango desde 10 hasta 77 dB.

Los valores de las intensidades sonoras se cuantifican de la siguiente manera:

-Si $V_{bp1} \leq 0.6$ (tramas no sonoras) el resto de intensidades sonoras (2, 3, 4 y 5) son cuantificadas a 0

-Si $V_{bp1} > 0.6$ las intensidades sonoras son cuantificadas a 1.

Hay una excepción: Si V_{bpi} para i = 2, 3, 4, 5 son 0001 entonces V_{bp5} es cuantificada a 0

Las magnitudes de Fourier utilizadas en el codificador MELP consisten en los diez primeros armónicos del pitch de la predicción residual generados por los coeficientes de predicción cuantificados. Las magnitudes son cuantificadas y pueden ser utilizadas en el sintetizador para generar pulsos de excitación. Para realizar el cálculo se usa la Transformada de Fourier (FFT) con 512 puntos sobre una ventana de 200 muestras centrada al final de la trama.

En primer lugar se calculan los coeficientes de cuantificación a partir del vector LSF cuantificado. A continuación se calcula la ventana residual con los coeficientes de cuantificación. Se aplica la ventana Hamming de 200 muestras, la señal es acolchada a cero con 512 muestras y se aplica la FFT compleja. El resultado de este proceso es transformado en magnitudes y los armónicos son el resultado de la búsqueda mediante un algoritmo de selección espectral.

1.4 Protección ante errores y Empaquetamiento de los bits

Para mejorar el comportamiento frente a errores durante la transmisión por el canal los parámetros que no son usados para el modo no sonoro son utilizados para la protección frente a errores (FEC). Se utilizan tres códigos Hamming (7,4) y un código Hamming (8,4). El código (7,4) corrige los errores simples en bits mientras que el código (8,4) corrige los errores dobles.

Codificador	**2.4 kbps**		**1.2 kbps**	
Parámetros	**Sonoros**	**No Sonoros**	**Sonoros**	**No Sonoros**
LSF	25	25	18	18
Magnitudes de Fourier	8	-	8	-
Ganancia (2 por trama)	8	8	8	8
Pitch	7	7	7	7
Banda de paso sonora	4	-	4	-
Bandera pulsos aperiódicos	1	-	1	-
Protección de errores	-	13	-	13
Bit de sincronización	1	1	1	1
Bits totales	54	54	48	48

Tabla E.1: Composición en bits de las tramas MELP

2. Decodificador

El sintetizador del codec MELP difiere de un decodificador clásico LPC en que incorpora control del pulso de ruido mezclado, estado del jitter sonoro y filtrado con intensidad adaptativa.

Los parámetros del esquema del sintetizador son linealmente interpolados para cada periodo del pitch de manera síncrona usando el conjunto de parámetros de la trama previa y aquellos de la trama actual que se está analizando. Entre los parámetros que se interpolan se encuentran: 2 Ganancias en decibelios, 10 Frecuencias Lineal LSF, el pitch, la intensidad del jitter y un coeficiente espectral de inclinación que se usa en el filtrado adaptativo de mejoramiento, y también las intensidades sonoras en paso banda.

El factor de interpolación se basa en el punto de inicio del nuevo periodo del pitch:

$$int = t_0 / 180 .$$

2.1 Desempaquetado de bits y Corrección del Error.

Los bits que llegan desde el canal de transmisión son desempaquetados en las correspondientes palabras de codificación. Los parámetros usados para la decodificación son diferentes para los modos sonoros y no sonoros. Lo primero que se decodifica es el pitch puesto que éste contiene el modo. Si la palabra correspondiente al pitch es todo ceros el modo correspondiente es el no sonoro. Si la trama tiene dos bits es indicativo para la eliminación de la trama. En caso contrario el valor del pitch es decodificado y se usa el modo sonoro.

En el modo no sonoro, el código Hamming (8,4) se decodifica para corregir errores simples de bit y detectar errores dobles. Si se detecta un error que no se puede corregir, se indica la eliminación de la trama. Si hay errores que se pueden corregir se utilizan los códigos Hamming (7,4) decodificados para los errores simples pero sin detección de errores dobles.

Si hay eliminación de tramas en la trama que se está analizando por el código Hamming, el código del pitch o directamente por señalización del canal, se implementa un mecanismo de retransmisión. Todos los parámetros de la trama actual son reemplazados con parámetros de la trama previa. Además el factor de Ganancia se pone igual que el segundo factor de Ganancia de tal manera que no hay transiciones permitidas.

Si no hay eliminación de tramas, los parámetros restantes son decodificados. Los coeficientes LSF son chequeados en orden ascendente y con separación mínima de 50Hz. En el modo no sonoro los parámetros toman valores por defecto: El pitch es puesto a 50 muestras, el jitter es puesto al 25%, todas las intensidades sonoras en paso banda son puestas a 0 y las magnitudes de Fourier son puestas a 1. En el modo sonoro V_{bp1} se asigna el valor 1; el jitter es puesto al 25% si el indicador aperiódico está a 1 y en caso contrario el jitter se pone a 0%. Las intensidades sonoras en paso banda para las cuatro bandas superiores son puestas a 1 y el bit correspondiente está a 1; en caso contrario las intensidades se ponen a 0. Hay una excepción: si se recibe la secuencia 0001 para V_{bpi} para i = 2, 3, 4, 5 respectivamente, V_{bp5} se pone a 0.

Cuando se recibe la palabra todo ceros para la primera ganancia G_1, algunos errores en la segunda ganancia G_2 pueden ser detectados y corregidos. Esta corrección mejora el comportamiento en cuanto a errores del canal se refiere.

2.2 Excitación Mixta y Pitch con Jitter

Un problema bastante común con el habla LPC clásica es la calidad con zumbidos en el habla sintetizada. Esto es básicamente producido por la excitación de los pulsos puros usados en el habla sintetizada. Muchas de las características del ruido y la información de fase en la señal original son ignoradas una vez que se extrae el valor final del pitch.

Un tren de impulsos simple es inadecuado a la hora de la sintetizar las diferentes características que se encuentran en el habla humana. En el método de síntesis del habla por excitación mixta, se crea una mezcla de los pulsos y del ruido para proporcionar una cercanía mayor con respecto a las características de la señal original.

La excitación de los pulsos es obtenida usando la Transformada Inversa Discreta de Fourier durante toda la longitud del periodo del pitch T, donde $M(k)$ son las magnitudes de Fourier.

$$e_p(n) = \frac{1}{T}\sum_{k=0}^{T-1} M(k)e^{j2\pi nk/T}.$$

El periodo del pitch es interpolado usando el valor de pitch más los valores de duración del jitter. Finalmente el periodo es redondeado al entero más cercano y anclado entre 20 y 160.Las fases son puestas a cero y como $M(k)$ es real el resto de magnitudes de Fourier pueden ser establecidas a 1.

El ruido es generado mediante un generador aleatorio uniforme con un valor RMS de 100 y un rango desde -1732 hasta 1732.

Finalmente, los pulsos y el ruido son filtrados y sumados para formar la excitación mixta. El filtro de pulsos para la ventana actual se consigue mediante la suma de todos los coeficientes filtrados en la zona de paso banda, para las bandas de frecuencias de paso sonoras, mientras que el filtro para el ruido se consigue mediante la suma de los coeficientes en las bandas de frecuencias no sonoras

El jitter usado en tramas débilmente sonoras proporciona un modelo más exacto de los pulsos glotales erróneos en el habla humana. El tercer estado sonoro evita el error a la hora de sintetizar tramas débilmente sonoras usando ruido puro o sintetizando tramas no sonoras mediante excitación de los pulsos puros. Este procedimiento elimina los ruidos sordos y el ruido tonal del habla LPC. El jitter pitch es implementado usando un generador aleatorio sobre el jitter del periodo del pitch actual hasta el 25%. Esto destruye la periodicidad en regiones donde el pitch cambia drásticamente y se deshace de los ruidos sordos los cuales son el resultado de utilizar la excitación en el ruido para generar el habla. Las tramas sonoras con jitter y las tramas sonoras son generadas usando la mezcla pulso/ruido. El habla no sonora es simplemente generada estableciendo las intensidades sonoras de las cinco bandas a cero.

Para señales bastante uniformes, se aplica una pequeña cantidad de ganancia en la atenuación en los dos parámetros decodificados de la ganancia usando una regla de sustracción de la potencia. Esta atenuación es un caso simplificado e invariante en la frecuencia del método de supresión “Smoothed Spectral Subtraction noise”.

2.3 Mejora Adaptativa y Filtros para la Dispersión de los Pulsos

El filtro de mejora adaptativa espectral es implementado directamente sobre la señal de excitación mixta antes obtenida. Este método es un filtro de polos y ceros de décimo orden, con una compensación adicional de primer orden de la inclinación.

Sus coeficientes son generados por la ampliación del ancho de banda de la función de transferencia del filtro de predicción lineal A(z) correspondiente a la interpolación de los parámetros LSF. La función de transferencia del filtro con el coeficiente de inclinación μ resulta:

$$H_{ase}(z) = \frac{A(\alpha z^{-1})}{A(\beta z^{-1})} \cdot (1 + \mu z^{-1}) \qquad \begin{matrix} \alpha = 0.5p \\ \beta = 0.8p \end{matrix}$$

El coeficiente de inclinación se calcula previamente como $max(0.5k1, 0)$, después se interpola y luego se multiplica por p la señal de probabilidad. El primer coeficiente de reflexión k_1 se calcula a través de los parámetros LSF decodificados. El factor p es estimado mediante comparación el valor actual interpolado de la ganancia G_{int} y el ruido estimado en segundo plano G_n mediante la fórmula:

$$p = \frac{G_{int} - G_n - 12}{18}$$

2.4 Síntesis y Ajustes

La síntesis de la Predicción lineal utiliza un filtro de forma directa, con los coeficientes correspondientes a la interpolación de los parámetros LSF.

Debido a que la excitación es generada en un nivel arbitrario, la ganancia del habla debe de ser introducida a la síntesis del discurso. El factor de corrección de la escala es introducido para cada periodo del pitch sintetizado.

$$S_{gain} = \frac{10^{G_{int}/20}}{\sqrt{\frac{1}{T}\sum_{n=1}^{T} \hat{s}_n^2}}.$$

Para evitar saltos en el habla sintetizada, el factor de escalado es linealmente intercalado entre los valores previos y actuales de las diez primeras muestras del pitch.

El filtro de dispersión del pulso es un filtro FIR de orden 65 derivado de un pulso triangular espectralmente aplanado.

Después del procesamiento de cada periodo del pitch, el decodificador actualizar t_0 añadiendo T, que es el número de muestras del periodo recién analizado. Si $t_0 < 180$, la síntesis continúa desde el procedimiento de interpolación. En caso contrario el decodificador almacena el resto de parámetro del periodo actual el cual se extiende más allá de la trama actual y resta 180 a t_0 para producir el valor inicial de la siguiente trama

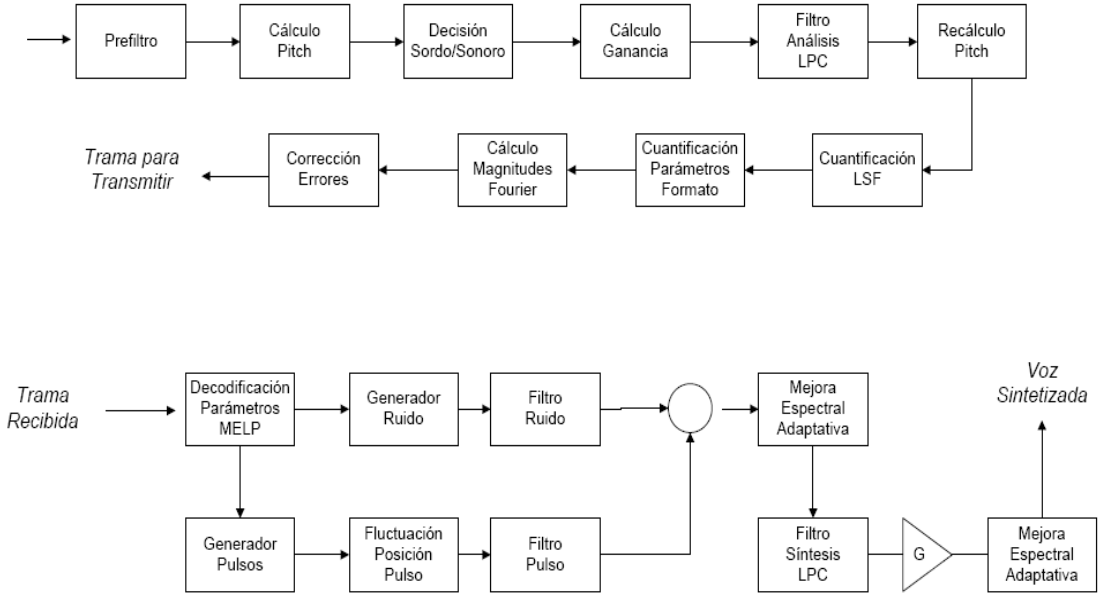

Figura E.2: Diagrama de funcionamiento global del codec MELP

ANEXO F: Presupuesto

1) Ejecución Material

- Consumo del ordenador personal (Software incluido)..... ..... 150 €
- Uso de impresora láser durante 10 meses.....50 €
- Material de oficina.....50 €
- Total de ejecución material..... 250 €

2) Gastos generales

- 16 % sobre Ejecución Material ..... 40 €

3) Beneficio Industrial

- 6 % sobre Ejecución Material ..... 15 €

4) Honorarios Proyecto

- 400 horas a 5 € / hora ..... 2000 €

5) Material fungible

- Gastos de impresión ..... 50€
- Encuadernación ..... 10€

6) Subtotal del presupuesto

- Subtotal Presupuesto ..... 2365 €

7) I.V.A. aplicable

- 16% Subtotal Presupuesto..... 378.4 €

8) Total presupuesto

- Total Presupuesto ..... 2743.4 €

Madrid, Junio de 2008

ANEXO G: Cógido Fuente del Proyecto

CODEC_MELP.C

```
/*
 * Asterisk -- An open source telephony toolkit.
 *
 *
 * Mark Spencer <markster@digium.com>
 *
 * See http://www.asterisk.org for more information about
 * the Asterisk project. Please do not directly contact
 * any of the maintainers of this project for assistance;
 * the project provides a web site, mailing lists and IRC
 * channels for your use.
 *
 * This program is free software, distributed under the terms of
 * the GNU General Public License Version 2. See the LICENSE file
 * at the top of the source tree.
 */

/*! \file
 *
 * \brief Translate between signed linear and codec MELP
 *
 * \ingroup codecs
 */

/*** MODULEINFO***/

#include "asterisk.h"

ASTERISK_FILE_VERSION(__FILE__, "$Revision$")

#include <fcntl.h>
#include <stdlib.h>
#include <unistd.h>
#include <netinet/in.h>
#include <string.h>
#include <stdio.h>
#include <math.h>

#include "asterisk/lock.h"
#include "asterisk/translate.h"
#include "asterisk/config.h"
#include "asterisk/options.h"
#include "asterisk/module.h"
#include "asterisk/logger.h"
#include "asterisk/channel.h"
#include "asterisk/utils.h"


/* Sample frame data */
#include "slin_melp_ex.h"
#include "melp_slin_ex.h"

//#include "melp/melp12.h"
```

```
 #include "melp/sc1200.h"

#define BUFFER_SAMPLES  8000
#define MELP_SAMPLES    540 /*Tres ventanas de 180 muestras a 22,5 ms
por ventana, total de 67,5ms*/
#define     MELP_FRAME_LEN    11 /*En bytes*/

struct melp_translator_pvt {  /* both melp2lin and lin2melp */

      int16_t buf[BUFFER_SAMPLES];  /* lin2melp, temporary storage */

};


static int melp_new(struct ast_trans_pvt *pvt)
{
      struct melp_translator_pvt *tmp = pvt->pvt;

      return 0;
}

static struct ast_frame *lintomelp_sample(void)
{
      static struct ast_frame f;
      f.frametype = AST_FRAME_VOICE;
      f.subclass = AST_FORMAT_SLINEAR;
      f.datalen = sizeof(slin_melp_ex);/*540*16bits*/
      /* Assume 8000 Hz */
      f.samples = sizeof(slin_melp_ex)/2;/*540*/
      f.mallocd = 0;
      f.offset = 0;
      f.src = __PRETTY_FUNCTION__;
      f.data = slin_melp_ex;
      return &f;
}

static struct ast_frame *melptolin_sample(void)
{
      static struct ast_frame f;
      f.frametype = AST_FRAME_VOICE;
      f.subclass = AST_FORMAT_MELP;
      f.datalen = sizeof(melp_slin_ex);/*11*8bits*/
      /* All frames are 67,5 ms long */
      f.samples =MELP_SAMPLES;
      f.mallocd = 0;
      f.offset = 0;
      f.src = __PRETTY_FUNCTION__;
      f.data = melp_slin_ex;
      return &f;
}

/*! \brief decode and store in outbuf. */
static int melptolin_framein(struct ast_trans_pvt *pvt, struct
ast_frame *f)
{
      struct melp_translator_pvt *tmp = pvt->pvt;
      int x;
      int16_t *dst = (int16_t *)pvt->outbuf;


      for (x=0; x < f->datalen; x += MELP_FRAME_LEN) {
```

```
            //unsigned char data[2 * MELP_FRAME_LEN];
            unsigned char *src;
            int len;

            len = MELP_SAMPLES;
            src = f->data + x;

            /* XXX maybe we don't need to check */
            if (pvt->samples + len > BUFFER_SAMPLES) {
                  ast_log(LOG_WARNING, "Out of buffer space\n");
                  return -1;
            }
            if (melp_decode(src, dst + pvt->samples)==0) {
                  ast_log(LOG_WARNING, "Invalid MELP data (1)Cris\n");
                  return -1;
            }
            pvt->samples += MELP_SAMPLES;
            pvt->datalen += 2 * MELP_SAMPLES;

      }
      return 0;
}



/*! \brief store samples into working buffer for later decode */
static int lintomelp_framein(struct ast_trans_pvt *pvt, struct
ast_frame *f)
{
      struct melp_translator_pvt *tmp = pvt->pvt;

      /* XXX We should look at how old the rest of our stream is, and
if it
         is too old, then we should overwrite it entirely, otherwise we
can
         get artifacts of earlier talk that do not belong */
      if (pvt->samples + f->samples > BUFFER_SAMPLES) {
            ast_log(LOG_WARNING, "Out of buffer space\n");
            return -1;
      }
      memcpy(tmp->buf + pvt->samples, f->data, f->datalen);
      pvt->samples += f->samples;
      return 0;
}

/*! \brief encode and produce a frame */
static struct ast_frame *lintomelp_frameout(struct ast_trans_pvt *pvt)
{
      struct melp_translator_pvt *tmp = pvt->pvt;
      int datalen = 0;
      int samples = 0;

      /* We can't work on anything less than a frame in size */
      if (pvt->samples < MELP_SAMPLES)
            return NULL;
      while (pvt->samples >= MELP_SAMPLES) {
            /* Encode a frame of data */

            if (melp_encode(tmp->buf + samples, pvt->outbuf + datalen
)==0) {
```

```
                        ast_log(LOG_WARNING, "Invalid MELP data (2)Cris\n");
                        return -1;
                }

                datalen += MELP_FRAME_LEN;
                samples += MELP_SAMPLES;
                pvt->samples -= MELP_SAMPLES;
        }

        /* Move the data at the end of the buffer to the front */
        if (pvt->samples)
                memmove(tmp->buf, tmp->buf + samples, pvt->samples * 2);

        return ast_trans_frameout(pvt, datalen, samples);
}

/*static void melp_destroy_stuff(struct ast_trans_pvt *pvt)
{
        struct melp_translator_pvt *tmp = pvt->pvt;
}*/

static struct ast_translator melptolin = {
        .name = "melptolin",
        .srcfmt = AST_FORMAT_MELP,
        .dstfmt = AST_FORMAT_SLINEAR,
        .newpvt = melp_new,
        .framein = melptolin_framein,
//      .destroy = melp_destroy_stuff,
        .sample = melptolin_sample,
        .buffer_samples = BUFFER_SAMPLES,
        .buf_size = BUFFER_SAMPLES * 2,
        .desc_size = sizeof (struct melp_translator_pvt ),
        .plc_samples = MELP_SAMPLES,
};

static struct ast_translator lintomelp = {
        .name = "lintomelp",
        .srcfmt = AST_FORMAT_SLINEAR,
        .dstfmt = AST_FORMAT_MELP,
        .newpvt = melp_new,
        .framein = lintomelp_framein,
        .frameout = lintomelp_frameout,
//      .destroy = melp_destroy_stuff,
        .sample = lintomelp_sample,
        .desc_size = sizeof (struct melp_translator_pvt ),
        .buf_size = (BUFFER_SAMPLES * 2),
//(BUFFER_SAMPLES * MELP_FRAME_LEN + MELP_SAMPLES -1)/MELP_SAMPLES,
};


static void parse_config(void)
{
        struct ast_variable *var;
        struct ast_config *cfg = ast_config_load("codecs.conf");
        if (!cfg)
                return;
        for (var = ast_variable_browse(cfg, "plc"); var; var = var->next)
{
                if (!strcasecmp(var->name, "genericplc")) {
                        melptolin.useplc = ast_true(var->value) ? 1 : 0;
                        if (option_verbose > 2)
```

```
                        ast_verbose(VERBOSE_PREFIX_3 "codec_melp:
 %susing generic PLC\n", melptolin.useplc ? "" : "not ");
            }
      }
      ast_config_destroy(cfg);
}

/*! \brief standard module glue */
static int reload(void)
{
      parse_config();
      return 0;
}

static int unload_module(void)
{
      int res;

      res = ast_unregister_translator(&lintomelp);
      if (!res)
            res = ast_unregister_translator(&melptolin);

      return res;
}

static int load_module(void)
{
      int res;

      parse_config();
      res = ast_register_translator(&melptolin);
      if (!res)
            res=ast_register_translator(&lintomelp);
      else
            ast_unregister_translator(&melptolin);

      return res;
}

AST_MODULE_INFO(ASTERISK_GPL_KEY, AST_MODFLAG_DEFAULT, "MELP
Coder/Decoder",
            .load = load_module,
            .unload = unload_module,
            .reload = reload,
             );
```

MELP12_ENC.C

```
#include <fcntl.h>
#include <stdlib.h>
#include <unistd.h>
#include <netinet/in.h>
#include <string.h>
#include <stdio.h>
#include <inttypes.h>


#include "sc1200.h"
#include "mat_lib.h"
#include "global.h"
#include "macro.h"
#include "mathhalf.h"
#include "dsp_sub.h"
#include "melp_sub.h"
#include "constant.h"
#include "math_lib.h"
#include "math.h"
#include "transcode.h"

#include "melp12.h"

#if NPP
#include "npp.h"
#endif

#define X05_Q7                      64          /* 0.5 * (1 << 7) */
#define THREE_Q7              384         /* 3 * (1 << 7) */

/* ====== External memory =_u61 ?=== */

Shortword   mode;
Shortword   chwordsize;


/* ========== Static Variables ========== */

char in_name[100], out_name[100];

/*****************************************************************
 *
 *     FUNCTION NAME:      melp_encode
 *     PURPOSE:            Implements the analysis function.
 *     INPUTS:
 *
 *     buf_in             16 bit short signed integer (Shortword).
 *
 *     buf_out            8  bit  signed integer codeword
 *
 *
 *     OUTPUTS:          none
 *
 *     RETURN VALUE:     BOOLEAN

*****************************************************************/
```

```
 BOOLEAN melp_encode(int16_t *buf_in, char *buf_out )
 {
      Longword    length;
      Shortword   speech_in[BLOCK];
      Shortword   bitBufSize, bitBufSize12, bitBufSize24;


 /* size of the bitstream buffer */

      rate=RATE1200;
      mode=ANALYSIS;
      chwordsize=8;

/* ====== Initialize MELP analysis and synthesis ====== */

      frameSize = (Shortword) BLOCK;

/* Computing bitNum = rate * frameSize / FSAMP.  Note that bitNum*/
/*computes the n of bytes written to the channel and it has to be */
/* exact.  */

      bitNum12 = 81;
      bitNum24 = 54;
      bitBufSize12 = 11;
      bitBufSize24 = 7;

      frameSize = BLOCK;
      bitBufSize = bitBufSize12;
      melp_ana_init();

/* ====== Run MELP coder on input signal ====== */

      frame_count = 0;

/* Perform MELP analysis */

      memcpy(speech_in, buf_in, (2*frameSize));//leer frameSize
elementos de tamaño Shortword (2 bytes)
      length=540;


/* ---- Noise Pre-Processor ---- */
#if NPP
      npp(speech_in, speech_in);
      npp(&(speech_in[FRAME]), &(speech_in[FRAME]));
      npp(&(speech_in[2*FRAME]), &(speech_in[2*FRAME]));
#endif

      analysis(speech_in, melp_par);
      //fprintf(stderr, "cris analysis");

/* ---- Write channel output if needed ---- */

      memcpy(buf_out,chbuf,bitBufSize);

      frame_count ++;


      return TRUE;
}
```

MELP12_DEC.C

```
#include <fcntl.h>
#include <stdlib.h>
#include <unistd.h>
#include <netinet/in.h>
#include <string.h>
#include <stdio.h>
#include <inttypes.h>


#include "sc1200.h"
#include "mat_lib.h"
#include "global.h"
#include "macro.h"
#include "mathhalf.h"
#include "dsp_sub.h"
#include "melp_sub.h"
#include "constant.h"
#include "math_lib.h"
#include "math.h"
#include "transcode.h"

#include "melp12.h"


#if NPP
#include "npp.h"
#endif

#define X05_Q7                  64          /* 0.5 * (1 << 7) */
#define THREE_Q7                384         /* 3 * (1 << 7) */

/* ====== External memory ====== */

Shortword   mode;
Shortword   chwordsize;


/*****************************************************************
 *
 *      FUNCTION NAME: melp_decode
 *
 *      PURPOSE:        Implements the synthesis function.
 *
 *
 *      INPUTS:
 *
 *       src           unsigned char buffer input
 *       dst           int_16_t buffer output speech
 *
 *      OUTPUTS:       none
 *
 *      RETURN VALUE:  BOOLEAN
 *
 *
*****************************************************************/

BOOLEAN melp_decode(unsigned char *src, int16_t *dst )
{
```

```
        Longword    length;
        Shortword   speech_in[BLOCK], speech_out[BLOCK];
       Shortword   bitBufSize, bitBufSize12, bitBufSize24;

 /* size of the bitstream buffer */

       mode=SYNTHESIS;
       rate=RATE1200;
       chwordsize=8;

/* ====== Initialize MELP synthesis ====== */
       frameSize = (Shortword) BLOCK;

/* Computing bitNum = rate*frameSize /FSAMP. Note that bitNum        */
/* computes the number of bytes written to the channel and it has to be
exact*/

       bitNum12 = 81;
       bitNum24 = 54;
       bitBufSize12 = 11;
       bitBufSize24 = 7;

       bitBufSize = bitBufSize12;
       melp_syn_init();


/* ====== Run MELP coder on input signal ====== */

       memcpy(chbuf, src, bitBufSize);

       length=11;

       synthesis(melp_par, speech_out);
       //writebl(speech_out, fp_out, frameSize);
       memcpy(dst,speech_out,frameSize);

       frame_count ++;

       return TRUE;
}
```

MAKEFILE

```
# ====================================================================

#--------------------------------------------------------------------
#
#  Makefile for SC1200 1200 bps speech coder
#
#--------------------------------------------------------------------

#--------------------------------------------------------------------
#   Turn on dependency tracking (for include files).
#--------------------------------------------------------------------
.KEEP_STATE:

#--------------------------------------------------------------------
#   Specify names of source files and executable.
#--------------------------------------------------------------------
OBJS = melp12_dec.o melp12_enc.o classify.o coeff.o dsp_sub.o
fec_code.o fft_lib.o \
       fs_lib.o fsvq_cb.o global.o harm.o lpc_lib.o mathdp31.o \
       mathhalf.o math_lib.o mat_lib.o melp_ana.o melp_chn.o melp_sub.o
\
       melp_syn.o msvq_cb.o npp.o pitch.o pit_lib.o postfilt.o qnt12.o
\
       qnt12_cb.o transcode.o vq_lib.o


SCMAIN      = sc1200.o

#--------------------------------------------------------------------
#   Specify options for compiling, linking, and archiving.
#--------------------------------------------------------------------

CC         = gcc
CFLAGS     = -O3 -Wall
LIBS     = -lm
COMPILE.c  = $(CC) $(CFLAGS) -c
LINK.c     = $(CC) $(CFLAGS)

all: sc

sc: ${SCMAIN} ${OBJS}
      ${CC} -o $@ ${CFLAGS} ${SCMAIN} ${OBJS} ${LIBS}


clean:
      rm *.o
# *~ sc

#--------------------------------------------------------------------
#     Header Dependences
#--------------------------------------------------------------------

      classify.o : classify.c sc1200.h cprv.h global.h mat_lib.h
      macro.h mathhalf.h constant.h math_lib.h dsp_sub.h coeff.h
      melp_sub.h
coeff.o : coeff.c sc1200.h
```

```
 dsp_sub.o : dsp_sub.c sc1200.h macro.h mathhalf.h mat_lib.h math_lib.h
 dsp_sub.h constant.h global.h fec_code.o : fec_code.c sc1200.h
mathhalf.h
fft_lib.o : fft_lib.c sc1200.h mathhalf.h mat_lib.h math_lib.h
constant.h global.h fft_lib.h
fs_lib.o : fs_lib.c sc1200.h mathhalf.h mat_lib.h math_lib.h fs_lib.h
constant.h global.h dsp_sub.h macro.h fft_lib.h
fsvq_cb.o : fsvq_cb.c sc1200.h
global.o : global.c sc1200.h
harm.o : harm.c sc1200.h constant.h mathhalf.h mat_lib.h math_lib.h
lpc_lib.o : lpc_lib.c sc1200.h macro.h mathhalf.h mathdp31.h math_lib.h
mat_lib.h lpc_lib.h constant.h global.h dsp_sub.h
mat_lib.o : mat_lib.c sc1200.h mathhalf.h mat_lib.h
math_lib.o : math_lib.c sc1200.h mathhalf.h mathdp31.h math_lib.h
constant.h global.h macro.h
mathdp31.o : mathdp31.c sc1200.h mathhalf.h
mathhalf.o : mathhalf.c sc1200.h constant.h mathhalf.h mathdp31.h
global.h macro.h
melp_ana.o : melp_ana.c sc1200.h mathhalf.h macro.h lpc_lib.h mat_lib.h
vq_lib.h fs_lib.h fft_lib.h pit_lib.h math_lib.h constant.h global.h
cprv.h pitch.h qnt12_cb.h qnt12.h msvq_cb.h fsvq_cb.h melp_sub.h
dsp_sub.h coeff.h
melp_chn.o : melp_chn.c sc1200.h vq_lib.h melp_sub.h math_lib.h
constant.h global.h qnt12_cb.h mathhalf.h mat_lib.h qnt12.h msvq_cb.h
fsvq_cb.h dsp_sub.h
melp_sub.o : melp_sub.c sc1200.h mathhalf.h math_lib.h mat_lib.h
dsp_sub.h melp_sub.h pit_lib.h constant.h coeff.h
melp_syn.o : melp_syn.c sc1200.h mathhalf.h macro.h lpc_lib.h mat_lib.h
vq_lib.h fs_lib.h math_lib.h constant.h global.h harm.h fsvq_cb.h
dsp_sub.h melp_sub.h coeff.h postfilt.h
msvq_cb.o : msvq_cb.c sc1200.h
npp.o : npp.c npp.h sc1200.h mathhalf.h math_lib.h mathdp31.h mat_lib.h
dsp_sub.h fs_lib.h global.h
pit_lib.o : pit_lib.c sc1200.h mathhalf.h mathdp31.h mat_lib.h
math_lib.h dsp_sub.h pit_lib.h constant.h global.h coeff.h
pitch.o : pitch.c sc1200.h cprv.h macro.h mat_lib.h pitch.h mathhalf.h
math_lib.h constant.h dsp_sub.h melp_sub.h
postfilt.o : postfilt.c sc1200.h mathhalf.h macro.h constant.h
mat_lib.h dsp_sub.h math_lib.h lpc_lib.h
qnt12.o : qnt12.c sc1200.h lpc_lib.h vq_lib.h global.h macro.h
qnt12_cb.h mat_lib.h math_lib.h qnt12.h constant.h mathhalf.h msvq_cb.h
fsvq_cb.h dsp_sub.h melp_sub.h
qnt12_cb.o : qnt12_cb.c sc1200.h
sc1200.o : sc1200.c sc1200.h mat_lib.h global.h macro.h mathhalf.h
dsp_sub.h melp_sub.h constant.h npp.h math_lib.h mathdp31.h
transcode.o : transcode.c sc1200.h mathhalf.h mat_lib.h math_lib.h
constant.h global.h dsp_sub.h qnt12_cb.h qnt12.h msvq_cb.h fsvq_cb.h
melp_sub.h dsp_sub.h coeff.h macro.h transcode.h lpc_lib.h fs_lib.h
cprv.h vq_lib.h
vq_lib.o : vq_lib.c sc1200.h macro.h mathhalf.h mat_lib.h math_lib.h
lpc_lib.h constant.h
melp12_dec.o : melp12_dec.c sc1200.h mat_lib.h global.h macro.h
mathhalf.h dsp_sub.h melp_sub.h constant.h math_lib.h transcode.h
melp12_enc.o : melp12_enc.c sc1200.h mat_lib.h global.h macro.h
mathhalf.h dsp_sub.h melp_sub.h constant.h math_lib.h transcode.h


#-----------------------------------------------------------------
#   End of Makefile.
#-----------------------------------------------------------------
```

MELP_PATCH.TXT

```
Index: build_tools/embed_modules.xml
===================================================================
--- build_tools/embed_modules.xml  (revision 81434)
+++ build_tools/embed_modules.xml  (working copy)
@@ -8,7 +8,7 @@
            <member name="channels" displayname="Channels"
remove_on_change="channels/*.o channels/misdn/*.o">
                  <depend>gnu_ld</depend>
            </member>
-           <member name="codecs" displayname="Coders/Decoders"
remove_on_change="codecs/*.o codecs/gsm/src/*.o codecs/gsm/src/*.o
codecs/gsm/src/*.o codecs/ilbc/*.o codecs/lpc10/*.o
codecs/gsm/lib/libgsm.a codecs/lpc10/liblpc10.a codecs/ilbc/libilbc.a
codecs/melp/*.o ">
+           <member name="codecs" displayname="Coders/Decoders"
remove_on_change="codecs/*.o codecs/gsm/src/*.o codecs/ilbc/*.o
codecs/lpc10/*.o codecs/gsm/lib/libgsm.a codecs/lpc10/liblpc10.a
codecs/ilbc/libilbc.a codecs/melp/*.o ">
                  <depend>gnu_ld</depend>
            </member>
            <member name="formats" displayname="File Formats"
remove_on_change="formats/*.o">
Index: include/asterisk/frame.h
===================================================================
--- include/asterisk/frame.h  (revision 81434)
+++ include/asterisk/frame.h  (working copy)
@@ -242,6 +242,8 @@
 #define AST_FORMAT_G726           (1 << 11)
 /*! G.722 */
 #define AST_FORMAT_G722           (1 << 12)
+/*! AMR-NB */
+#define AST_FORMAT_MELP      (1 << 13)
 /*! Maximum audio format */
 #define AST_FORMAT_MAX_AUDIO (1 << 15)
 /*! Maximum audio mask */
Index: main/channel.c
===================================================================
--- main/channel.c      (revision 81434)
+++ main/channel.c      (working copy)
@@ -695,6 +695,8 @@
            /*! Okay, we're down to vocoders now, so pick GSM because
it's small and easier to
                translate and sounds pretty good */
            AST_FORMAT_GSM,
+           /*! Try MELP */
+           AST_FORMAT_MELP,
            /*! iLBC is not too bad */
            AST_FORMAT_ILBC,
            /*! Speex is free, but computationally more expensive than
GSM */
Index: main/translate.c
===================================================================
--- main/translate.c    (revision 81434)
+++ main/translate.c    (working copy)
@@ -492,7 +492,7 @@
 /*! \brief CLI "show translation" command handler */
 static int show_translation_deprecated(int fd, int argc, char *argv[])
 {
-#define SHOW_TRANS 13
```

```
 +#define SHOW_TRANS 14
       int x, y, z;
      int curlen = 0, longest = 0;

Index: main/rtp.c
===================================================================
--- main/rtp.c    (revision 81434)
+++ main/rtp.c    (working copy)
@@ -1341,6 +1341,7 @@
      {{1, AST_FORMAT_ILBC}, "audio", "iLBC"},
      {{1, AST_FORMAT_G722}, "audio", "G722"},
      {{1, AST_FORMAT_G726_AAL2}, "audio", "AAL2-G726-32"},
+     {{1, AST_FORMAT_MELP}, "audio", "MELP"},
      {{0, AST_RTP_DTMF}, "audio", "telephone-event"},
      {{0, AST_RTP_CISCO_DTMF}, "audio", "cisco-telephone-event"},
      {{0, AST_RTP_CN}, "audio", "CN"},
@@ -1378,6 +1379,7 @@
      [31] = {1, AST_FORMAT_H261},
      [34] = {1, AST_FORMAT_H263},
      [103] = {1, AST_FORMAT_H263_PLUS},
+     [96] = {1, AST_FORMAT_MELP},
      [97] = {1, AST_FORMAT_ILBC},
      [99] = {1, AST_FORMAT_H264},
      [101] = {0, AST_RTP_DTMF},
Index: main/frame.c
===================================================================
--- main/frame.c  (revision 81434)
+++ main/frame.c  (working copy)
@@ -117,10 +117,10 @@
      { 1, AST_FORMAT_ILBC, "ilbc", "iLBC", 50, 30, 30, 30, 30 },
      /*!< 11: codec_ilbc.c */ /* inc=30ms - workaround */
      { 1, AST_FORMAT_G726_AAL2, "g726aal2", "G.726 AAL2", 40, 10, 300,
10, 20 },   /*!<  12: codec_g726.c */
      { 1, AST_FORMAT_G722, "g722", "G722"},
      /*!< 13 */
+     { 1, AST_FORMAT_MELP, "melp", "MELP", 11, 68, 300, 68, 68},
/*!<  codec_melp.c */
      { 0, 0, "nothing", "undefined" },
      { 0, 0, "nothing", "undefined" },
      { 0, 0, "nothing", "undefined" },
-     { 0, 0, "nothing", "undefined" },
      { 0, AST_FORMAT_MAX_AUDIO, "maxaudio", "Maximum audio format" },
      { 1, AST_FORMAT_JPEG, "jpeg", "JPEG image"},   /*!< 17: See
format_jpeg.c */
      { 1, AST_FORMAT_PNG, "png", "PNG image"}, /*!< 18: Image format
*/
@@ -631,7 +631,7 @@
      ast_cli(fd, "----------------------------------------------------
---------------------------\n");
      if ((argc == 2) || (!strcasecmp(argv[1],"audio"))) {
            found = 1;
-           for (i=0;i<13;i++) {
+           for (i=0;i<14;i++) {
                  snprintf(hex,25,"(0x%x)",1<<i);
                  ast_cli(fd, "%11u (1 << %2d) %10s  audio   %8s
(%s)\n",1 << i,i,hex,ast_getformatname(1<<i),ast_codec2str(1<<i));
            }
@@ -675,7 +675,7 @@
      ast_cli(fd, "----------------------------------------------------
---------------------------\n");
      if ((argc == 3) || (!strcasecmp(argv[3],"audio"))) {
```

```
             found = 1;
 -           for (i=0;i<13;i++) {
+           for (i=0;i<14;i++) {
                 snprintf(hex,25,"(0x%x)",1<<i);
                 ast_cli(fd, "%11u (1 << %2d) %10s  audio   %8s
(%s)\n",1 << i,i,hex,ast_getformatname(1<<i),ast_codec2str(1<<i));
             }
@@ -1462,6 +1462,36 @@
      return cnt;
 }

int ast_codec_get_samples(struct ast_frame *f)
 {
      int samples=0;
@@ -1499,6 +1529,9 @@
      case AST_FORMAT_G726_AAL2:
             samples = f->datalen * 2;
             break;
+     case AST_FORMAT_MELP:
+           samples = 540*(f->datalen / 11);
+           break;
      default:
             ast_log(LOG_WARNING, "Unable to calculate samples for
format %s\n", ast_getformatname(f->subclass));
      }
Index: codecs/Makefile
===================================================================
--- codecs/Makefile     (revision 81434)
+++ codecs/Makefile     (working copy)
@@ -25,6 +25,7 @@

 LIBILBC:=ilbc/libilbc.a
 LIBLPC10:=lpc10/liblpc10.a

 all: _all

@@ -39,6 +40,7 @@
      $(MAKE) -C gsm clean
      $(MAKE) -C lpc10 clean
      $(MAKE) -C ilbc clean
+     $(MAKE) -C melp clean

 gsm/lib/libgsm.a:
      @mkdir -p gsm/lib
@@ -53,3 +55,9 @@
      @$(MAKE) -C ilbc all

 $(if $(filter
codec_ilbc,$(EMBEDDED_MODS)),modules.link,codec_ilbc.so): $(LIBILBC);
```

SIP.CONF

```
; SIP Configuration example for Asterisk
;
; Syntax for specifying a SIP device in extensions.conf is
; SIP/devicename where devicename is defined in a section below.
;
; You may also use
; SIP/username@domain to call any SIP user on the Internet
; (Don't forget to enable DNS SRV records if you want to use this)
;
; If you define a SIP proxy as a peer below, you may call
; SIP/proxyhostname/user or SIP/user@proxyhostname
; where the proxyhostname is defined in a section below
;
; Useful CLI commands to check peers/users:
;   sip show peers            Show all SIP peers (including friends)
;   sip show users            Show all SIP users (including friends)
;   sip show registry         Show status of hosts we register with
;
;   sip debug                 Show all SIP messages
;
;   reload chan_sip.so        Reload configuration file
;                         Active SIP peers will not be reconfigured
;

[general]
context=default               ; Default context for incoming calls
;allowguest=no                ; Allow or reject guest calls (default is
yes)
allowoverlap=no               ; Disable overlap dialing support.
(Default is yes)
;allowtransfer=no       ; Disable all transfers (unless enabled in
peers or users)
                       ; Default is enabled
;realm=mydomain.tld           ; Realm for digest authentication
                       ; defaults to "asterisk". If you set a system
name in
                       ; asterisk.conf, it defaults to that system
name
                       ; Realms MUST be globally unique according to
RFC 3261
                       ; Set this to your host name or domain name
bindport=5060                 ; UDP Port to bind to (SIP standard port
is 5060)
                       ; bindport is the local UDP port that Asterisk
will listen on
bindaddr=0.0.0.0       ; IP address to bind to (0.0.0.0 binds to all)
srvlookup=yes                 ; Enable DNS SRV lookups on outbound
calls
                       ; Note: Asterisk only uses the first host
                       ; in SRV records
                       ; Disabling DNS SRV lookups disables the
                       ; ability to place SIP calls based on domain
                       ; names to some other SIP users on the Internet

;domain=mydomain.tld          ; Set default domain for this host
                       ; If configured, Asterisk will only allow
                       ; INVITE and REFER to non-local domains
                       ; Use "sip show domains" to list local domains
;pedantic=yes                 ; Enable checking of tags in headers,
```

```
                        ; international character conversions in URIs
                        ; and multiline formatted headers for strict
                       ; SIP compatibility (defaults to "no")

; See doc/README.tos for a description of these parameters.
;tos_sip=cs3                    ; Sets TOS for SIP packets.
;tos_audio=ef                   ; Sets TOS for RTP audio packets.
;tos_video=af41                 ; Sets TOS for RTP video packets.

;maxexpiry=3600              ; Maximum allowed time of incoming
registrations
                       ; and subscriptions (seconds)
;minexpiry=60                ; Minimum length of
registrations/subscriptions (default 60)
;defaultexpiry=120           ; Default length of incoming/outgoing
registration
;t1min=100              ; Minimum roundtrip time for messages to
monitored hosts
                       ; Defaults to 100 ms
;notifymimetype=text/plain    ; Allow overriding of mime type in MWI
NOTIFY
;checkmwi=10                 ; Default time between mailbox checks for
peers
;buggymwi=no                 ; Cisco SIP firmware doesn't support the
MWI RFC
                       ; fully. Enable this option to not get error
messages
                       ; when sending MWI to phones with this bug.
;vmexten=voicemail           ; dialplan extension to reach mailbox
sets the
                       ; Message-Account in the MWI notify message
                       ; defaults to "asterisk"
;disallow=all                ; First disallow all codecs
;allow=ulaw             ; Allow codecs in order of preference
;allow=ilbc             ; see doc/rtp-packetization for framing options
;
; This option specifies a preference for which music on hold class this
channel
; should listen to when put on hold if the music class has not been set
on the
; channel with Set(CHANNEL(musicclass)=whatever) in the dialplan, and
the peer
; channel putting this one on hold did not suggest a music class.
;
; This option may be specified globally, or on a per-user or per-peer
basis.
;
;mohinterpret=default
;
; This option specifies which music on hold class to suggest to the
peer channel
; when this channel places the peer on hold. It may be specified
globally or on
; a per-user or per-peer basis.
;
;mohsuggest=default
;
;language=en                 ; Default language setting for all
users/peers
                       ; This may also be set for individual
users/peers
```

```
 ;relaxdtmf=yes                ; Relax dtmf handling
 ;trustrpid = no               ; If Remote-Party-ID should be trusted
;sendrpid = yes               ; If Remote-Party-ID should be sent
;progressinband=never         ; If we should generate in-band ringing
always
                        ; use 'never' to never use in-band signalling,
even in cases
                        ; where some buggy devices might not render it
                        ; Valid values: yes, no, never Default: never
;useragent=Asterisk PBX       ; Allows you to change the user agent
string
;promiscredir = no            ; If yes, allows 302 or REDIR to non-
local SIP address
                             ; Note that promiscredir when redirects
are made to the
                             ; local system will cause loops since
Asterisk is incapable
                             ; of performing a "hairpin" call.
;usereqphone = no       ; If yes, ";user=phone" is added to uri that
contains
                        ; a valid phone number
;dtmfmode = rfc2833           ; Set default dtmfmode for sending DTMF.
Default: rfc2833
                        ; Other options:
                        ; info : SIP INFO messages
                        ; inband : Inband audio (requires 64 kbit codec
-alaw, ulaw)
                        ; auto : Use rfc2833 if offered, inband
otherwise

;compactheaders = yes         ; send compact sip headers.
;
;videosupport=yes       ; Turn on support for SIP video. You need to
turn this on
                        ; in the this section to get any video support
at all.
                        ; You can turn it off on a per peer basis if
the general
                        ; video support is enabled, but you can't
enable it for
                        ; one peer only without enabling in the general
section.
;maxcallbitrate=384           ; Maximum bitrate for video calls
(default 384 kb/s)
                        ; Videosupport and maxcallbitrate is settable
                        ; for peers and users as well
;callevents=no                ; generate manager events when sip ua
                        ; performs events (e.g. hold)
;alwaysauthreject = yes       ; When an incoming INVITE or REGISTER is
to be rejected,
                        ; for any reason, always reject with '401
Unauthorized'
                        ; instead of letting the requester know whether
there was
                        ; a matching user or peer for their request

;g726nonstandard = yes        ; If the peer negotiates G726-32 audio,
use AAL2 packing
                        ; order instead of RFC3551 packing order (this
is required
```

```
                         ; for Sipura and Grandstream ATAs, among
 others). This is
                        ; contrary to the RFC3551 specification, the
peer _should_
                        ; be negotiating AAL2-G726-32 instead :-(

;matchexterniplocally = yes     ; Only substitute the externip or
externhost setting if it matches
                                ; your localnet setting. Unless you
have some sort of strange network
                                ; setup you will not need to enable
this.

;
; If regcontext is specified, Asterisk will dynamically create and
destroy a
; NoOp priority 1 extension for a given peer who registers or
unregisters with
; us and have a "regexten=" configuration item.
; Multiple contexts may be specified by separating them with '&'. The
; actual extension is the 'regexten' parameter of the registering peer
or its
; name if 'regexten' is not provided.  If more than one context is
provided,
; the context must be specified within regexten by appending the
desired
; context after '@'.  More than one regexten may be supplied if they
are
; separated by '&'.  Patterns may be used in regexten.
;
;regcontext=sipregistrations
;
;--------------------------- RTP timers ------------------------------
---------------------
; These timers are currently used for both audio and video streams. The
RTP timeouts
; are only applied to the audio channel.
; The settings are settable in the global section as well as per device
;
;rtptimeout=60               ; Terminate call if 60 seconds of no RTP
or RTCP activity
                        ; on the audio channel
                        ; when we're not on hold. This is to be able to
hangup
                        ; a call in the case of a phone disappearing
from the net,
                        ; like a powerloss or grandma tripping over a
cable.
;rtpholdtimeout=300          ; Terminate call if 300 seconds of no RTP
or RTCP activity
                        ; on the audio channel
                        ; when we're on hold (must be > rtptimeout)
;rtpkeepalive=<secs>         ; Send keepalives in the RTP stream to
keep NAT open
                        ; (default is off - zero)
;--------------------------- SIP DEBUGGING ---------------------------
----------------------
;sipdebug = yes              ; Turn on SIP debugging by default, from
                        ; the moment the channel loads this
configuration
;recordhistory=yes           ; Record SIP history by default
```

```
                         ; (see sip history / sip no history)
 ;dumphistory=yes        ; Dump SIP history at end of SIP dialogue
                        ; SIP history is output to the DEBUG logging
channel


;--------------------------- STATUS NOTIFICATIONS (SUBSCRIPTIONS) -----
-----------------------
; You can subscribe to the status of extensions with a "hint" priority
; (See extensions.conf.sample for examples)
; chan_sip support two major formats for notifications: dialog-info and
SIMPLE
;
; You will get more detailed reports (busy etc) if you have a call
limit set
; for a device. When the call limit is filled, we will indicate busy.
Note that
; you need at least 2 in order to be able to do attended transfers.
;
; For queues, you will need this level of detail in status reporting,
regardless
; if you use SIP subscriptions. Queues and manager use the same
internal interface
; for reading status information.
;
; Note: Subscriptions does not work if you have a realtime dialplan and
use the
; realtime switch.
;
;allowsubscribe=no           ; Disable support for subscriptions.
(Default is yes)
;subscribecontext = default   ; Set a specific context for SUBSCRIBE
requests
                        ; Useful to limit subscriptions to local
extensions
                        ; Settable per peer/user also
;notifyringing = yes         ; Notify subscriptions on RINGING state
(default: no)
;notifyhold = yes       ; Notify subscriptions on HOLD state (default:
no)
                        ; Turning on notifyringing and notifyhold will
add a lot
                        ; more database transactions if you are using
realtime.
;limitonpeers = yes          ; Apply call limits on peers only. This
will improve
                        ; status notification when you are using
type=friend
                        ; Inbound calls, that really apply to the user
part
                        ; of a friend will now be added to and compared
with
                        ; the peer limit instead of applying two call
limits,
                        ; one for the peer and one for the user.
                        ; "sip show inuse" will only show active calls
on
                        ; the peer side of a "type=friend" object if
this
                        ; setting is turned on.
```

```
 ;---------------------------------------- T.38 FAX PASSTHROUGH
 SUPPORT -----------------------
;
; This setting is available in the [general] section as well as in
device configurations.
; Setting this to yes, enables T.38 fax (UDPTL) passthrough on SIP to
SIP calls, provided
; both parties have T38 support enabled in their Asterisk configuration
; This has to be enabled in the general section for all devices to
work. You can then
; disable it on a per device basis.
;
; T.38 faxing only works in SIP to SIP calls, with no local or agent
channel being used.
;
; t38pt_udptl = yes            ; Default false
;
;---------------------------------------- OUTBOUND SIP REGISTRATIONS
-----------------------
; Asterisk can register as a SIP user agent to a SIP proxy (provider)
; Format for the register statement is:
;       register => user[:secret[:authuser]]@host[:port][/extension]
;
; If no extension is given, the 's' extension is used. The extension
needs to
; be defined in extensions.conf to be able to accept calls from this
SIP proxy
; (provider).
;
; host is either a host name defined in DNS or the name of a section
defined
; below.
;
; Examples:
;
;register => 1234:password@mysipprovider.com
;
;     This will pass incoming calls to the 's' extension
;
;
;register => 2345:password@sip_proxy/1234
;
;    Register 2345 at sip provider 'sip_proxy'.  Calls from this
provider
;    connect to local extension 1234 in extensions.conf, default
context,
;    unless you configure a [sip_proxy] section below, and configure a
;    context.
;    Tip 1: Avoid assigning hostname to a sip.conf section like
[provider.com]
;    Tip 2: Use separate type=peer and type=user sections for SIP
providers
;           (instead of type=friend) if you have calls in both
directions

;registertimeout=20          ; retry registration calls every 20
seconds (default)
;registerattempts=10         ; Number of registration attempts before
we give up
                       ; 0 = continue forever, hammering the other
server
```

```
                        ; until it accepts the registration
                        ; Default is 0 tries, continue forever

;------------------------------------- NAT SUPPORT ---------------
--------
; The externip, externhost and localnet settings are used if you use
Asterisk
; behind a NAT device to communicate with services on the outside.

;externip = 200.201.202.203   ; Address that we're going to put in
outbound SIP
                        ; messages if we're behind a NAT

                        ; The externip and localnet is used
                        ; when registering and communicating with other
proxies
                        ; that we're registered with
;externhost=foo.dyndns.net    ; Alternatively you can specify an
                        ; external host, and Asterisk will
                        ; perform DNS queries periodically.  Not
                        ; recommended for production
                        ; environments!  Use externip instead
;externrefresh=10       ; How often to refresh externhost if
                        ; used
                        ; You may add multiple local networks.  A
reasonable
                        ; set of defaults are:
;localnet=192.168.0.0/255.255.0.0; All RFC 1918 addresses are local
networks
;localnet=10.0.0.0/255.0.0.0  ; Also RFC1918
;localnet=172.16.0.0/12       ; Another RFC1918 with CIDR notation
;localnet=169.254.0.0/255.255.0.0 ;Zero conf local network

; The nat= setting is used when Asterisk is on a public IP,
communicating with
; devices hidden behind a NAT device (broadband router).  If you have
one-way
; audio problems, you usually have problems with your NAT configuration
or your
; firewall's support of SIP+RTP ports.  You configure Asterisk choice
of RTP
; ports for incoming audio in rtp.conf
;
;nat=no                       ; Global NAT settings  (Affects all peers
and users)
                                ; yes = Always ignore info and assume
NAT
                                ; no = Use NAT mode only according to
RFC3581 (;rport)
                                ; never = Never attempt NAT mode or
RFC3581 support
                        ; route = Assume NAT, don't send rport
                        ; (work around more UNIDEN bugs)

;---------------------------------- MEDIA HANDLING ------------------
-------------
; By default, Asterisk tries to re-invite the audio to an optimal path.
If there's
; no reason for Asterisk to stay in the media path, the media will be
redirected.
```

```
 ; This does not really work with in the case where Asterisk is outside
 and have
; clients on the inside of a NAT. In that case, you want to set
canreinvite=nonat
;
;canreinvite=yes        ; Asterisk by default tries to redirect the
                        ; RTP media stream (audio) to go directly from
                        ; the caller to the callee.  Some devices do
not
                        ; support this (especially if one of them is
behind a NAT).
                        ; The default setting is YES. If you have all
clients
                        ; behind a NAT, or for some other reason wants
Asterisk to
                        ; stay in the audio path, you may want to turn
this off.

                        ; In Asterisk 1.4 this setting also affect
direct RTP
                        ; at call setup (a new feature in 1.4 - setting
up the
                        ; call directly between the endpoints instead
of sending
                        ; a re-INVITE).

;directrtpsetup=yes           ; Enable the new experimental direct RTP
setup. This sets up
                        ; the call directly with media peer-2-peer
without re-invites.
                        ; Will not work for video and cases where the
callee sends
                        ; RTP payloads and fmtp headers in the 200 OK
that does not match the
                        ; callers INVITE. This will also fail if
canreinvite is enabled when
                        ; the device is actually behind NAT.

;canreinvite=nonat            ; An additional option is to allow media
path redirection
                        ; (reinvite) but only when the peer where the
media is being
                        ; sent is known to not be behind a NAT (as the
RTP core can
                        ; determine it based on the apparent IP address
the media
                        ; arrives from).

;canreinvite=update           ; Yet a third option... use UPDATE for
media path redirection,
                        ; instead of INVITE. This can be combined with
'nonat', as
                        ; 'canreinvite=update,nonat'. It implies 'yes'.

;---------------------------------------- REALTIME SUPPORT -----------
-------------
; For additional information on ARA, the Asterisk Realtime
Architecture,
; please read realtime.txt and extconfig.txt in the /doc directory of
the
; source code.
```

```
 ;
 ;rtcachefriends=yes           ; Cache realtime friends by adding them
to the internal list
                        ; just like friends added from the config file
only on a
                        ; as-needed basis? (yes|no)

;rtsavesysname=yes            ; Save systemname in realtime database at
registration
                        ; Default= no

;rtupdate=yes                 ; Send registry updates to database using
realtime? (yes|no)
                        ; If set to yes, when a SIP UA registers
successfully, the ip address,
                        ; the origination port, the registration
period, and the username of
                        ; the UA will be set to database via realtime.
                        ; If not present, defaults to 'yes'.
;rtautoclear=yes        ; Auto-Expire friends created on the fly on the
same schedule
                        ; as if it had just registered?
(yes|no|<seconds>)
                        ; If set to yes, when the registration expires,
the friend will
                        ; vanish from the configuration until requested
again. If set
                        ; to an integer, friends expire within this
number of seconds
                        ; instead of the registration interval.

;ignoreregexpire=yes          ; Enabling this setting has two
functions:
                        ;
                        ; For non-realtime peers, when their
registration expires, the
                        ; information will _not_ be removed from memory
or the Asterisk database
                        ; if you attempt to place a call to the peer,
the existing information
                        ; will be used in spite of it having expired
                        ;
                        ; For realtime peers, when the peer is
retrieved from realtime storage,
                        ; the registration information will be used
regardless of whether
                        ; it has expired or not; if it expires while
the realtime peer
                        ; is still in memory (due to caching or other
reasons), the
                        ; information will not be removed from realtime
storage

;------------------------------------------ SIP DOMAIN SUPPORT ---------
; Incoming INVITE and REFER messages can be matched against a list of
'allowed'
; domains, each of which can direct the call to a specific context if
desired.
; By default, all domains are accepted and sent to the default context
or the
; context associated with the user/peer placing the call.
```

```
 ; Domains can be specified using:
 ; domain=<domain>[,<context>]
; Examples:
; domain=myasterisk.dom
; domain=customer.com,customer-context
;
; In addition, all the 'default' domains associated with a server
should be
; added if incoming request filtering is desired.
; autodomain=yes
;
; To disallow requests for domains not serviced by this server:
; allowexternaldomains=no

;domain=mydomain.tld,mydomain-incoming
                        ; Add domain and configure incoming context
                        ; for external calls to this domain
;domain=1.2.3.4              ; Add IP address as local domain
                        ; You can have several "domain" settings
;allowexternaldomains=no     ; Disable INVITE and REFER to non-local
domains
                        ; Default is yes
;autodomain=yes              ; Turn this on to have Asterisk add local
host
                        ; name and local IP to domain list.

; fromdomain=mydomain.tld     ; When making outbound SIP INVITEs to
                            ; non-peers, use your primary domain
"identity"
                            ; for From: headers instead of just your
IP
                            ; address. This is to be polite and
                            ; it may be a mandatory requirement for
some
                            ; destinations which do not have a prior
                            ; account relationship with your server.

;------------------------------ JITTER BUFFER CONFIGURATION -----------
; jbenable = yes              ; Enables the use of a jitterbuffer on
the receiving side of a
                            ; SIP channel. Defaults to "no". An
enabled jitterbuffer will
                            ; be used only if the sending side can
create and the receiving
                            ; side can not accept jitter. The SIP
channel can accept jitter,
                            ; thus a jitterbuffer on the receive SIP
side will be used only
                            ; if it is forced and enabled.

; jbforce = no                ; Forces the use of a jitterbuffer on the
receive side of a SIP
                            ; channel. Defaults to "no".

; jbmaxsize = 200             ; Max length of the jitterbuffer in
milliseconds.

; jbresyncthreshold = 1000    ; Jump in the frame timestamps over which
the jitterbuffer is
                            ; resynchronized. Useful to improve the
quality of the voice, with
```

```
                                ; big jumps in/broken timestamps,
 usually sent from exotic devices
                               ; and programs. Defaults to 1000.

; jbimpl = fixed               ; Jitterbuffer implementation, used on
the receiving side of a SIP
                               ; channel. Two implementations are
currently available - "fixed"
                               ; (with size always equals to jbmaxsize)
and "adaptive" (with
                               ; variable size, actually the new jb of
IAX2). Defaults to fixed.

; jblog = no                   ; Enables jitterbuffer frame logging.
Defaults to "no".
;---------------------------------------------------------------------
-------------

[authentication]
; Global credentials for outbound calls, i.e. when a proxy challenges
your
; Asterisk server for authentication. These credentials override
; any credentials in peer/register definition if realm is matched.
;
; This way, Asterisk can authenticate for outbound calls to other
; realms. We match realm on the proxy challenge and pick an set of
; credentials from this list
; Syntax:
;     auth = <user>:<secret>@<realm>
;     auth = <user>#<md5secret>@<realm>
; Example:
;auth=mark:topsecret@digium.com
;
; You may also add auth= statements to [peer] definitions
; Peer auth= override all other authentication settings if we match on
realm

;---------------------------------------------------------------------
--------
; Users and peers have different settings available. Friends have all
settings,
; since a friend is both a peer and a user
;
; User config options:        Peer configuration:
; --------------------        -------------------
; context                     context
; callingpres                 callingpres
; permit                      permit
; deny                        deny
; secret                      secret
; md5secret                   md5secret
; dtmfmode                    dtmfmode
; canreinvite                 canreinvite
; nat                         nat
; callgroup                   callgroup
; pickupgroup                 pickupgroup
; language                    language
; allow                       allow
; disallow                    disallow
; insecure                    insecure
; trustrpid                   trustrpid
```

```
 ; progressinband              progressinband
 ; promiscredir                promiscredir
; useclientcode               useclientcode
; accountcode                 accountcode
; setvar                      setvar
; callerid              callerid
; amaflags              amaflags
; call-limit                  call-limit
; allowoverlap                allowoverlap
; allowsubscribe        allowsubscribe
; allowtransfer                     allowtransfer
; subscribecontext            subscribecontext
; videosupport                videosupport
; maxcallbitrate        maxcallbitrate
; rfc2833compensate            mailbox
;                              username
;                              template
;                              fromdomain
;                              regexten
;                              fromuser
;                              host
;                              port
;                              qualify
;                              defaultip
;                              rtptimeout
;                              rtpholdtimeout
;                              sendrpid
;                              outboundproxy
;                              rfc2833compensate


;---------------------------------------------------------------------
; Definitions of locally connected SIP devices
;
; type = user     a device that authenticates to us by "from" field to
place calls
; type = peer     a device we place calls to or that calls us and we
match by host
; type = friend two configurations (peer+user) in one
;
; For device names, we recommend using only a-z, numerics (0-9) and
underscore
;
; For local phones, type=friend works most of the time
;
; If you have one-way audio, you probably have NAT problems.
; If Asterisk is on a public IP, and the phone is inside of a NAT
device
; you will need to configure nat option for those phones.
; Also, turn on qualify=yes to keep the nat session open


[Cristina]
type=friend
host=dynamic
username=Cristina
secret=cristina
nat=yes
canreinvite=no
context=office
callerid="User1"
```

```
 allow=gsm
 allow=ulaw
allow=alaw
regexten=200

[201]
type=friend
host=dynamic
username=201
;secret=cristina
nat=no
canreinvite=no
;qualify=yes
context=skype
;callerid="User2"
disallow=all
;allow=gsm
allow=ulaw
allow=alaw
;regexten=201
port=5061

[SPYKE]
type=friend
host=dynamic
;username=gateway
;secret=cristina
nat=no
;callerid="User3"
canreinvite=no
;qualify=yes
context=skype
callerid="SkypeGW <5001>"
disallow=all
;allow=gsm
;allow=ulaw
;allow=alaw
allow=melp
;regexten=5001

[202]
type=friend
host=dynamic
;username=gateway
;secret=cristina
nat=no
callerid="202"
canreinvite=no
;qualify=yes
context=skype
;callerid="SkypeGW <5001>"
disallow=all
;allow=gsm
allow=ulaw
;allow=alaw
;allow=melp
```

EXTENSIONS.CONF

```
[office]

include => demo
include => VERBIO_TEST
include => pruebas
include => DV


exten => 200,1, Macro(stdexten,200,SIP/200)
exten => User1, 1, Goto(200|1)
exten => 201,1, Macro(stdexten,201,SIP/201)
exten => User2, 1, Goto(201|1)
exten => 202,1, Macro(stdexten,202,SIP/202)
exten => User3, 1, Goto(202|1)

;; Prueba nueva AGI
exten => 12,1,Answer()
exten =>
12,n,AGI(reconocedor.agi|es|es|pueblos.txt|ISOLATED|127.0.0.1|4|2|alaw)
exten => 12,n,NoOp(Numero de palabra del vocabulario ${VASR_INDEX})
exten => 12,n,NoOp(Resultado reconocimiento ${VASR_RESULT})
exten => 12,n,NoOp(Puntuacion reconocimiento ${VASR_SCORE})
;exten => 12,n,AGI(ConVoz|${VASR_RESULT})
exten => 12,n,AGI(verbio-tts.agi|es|laura|127.0.0.1|alaw|Usted ha dicho
${VASR_RESULT})
exten => 12,n,AGI(verbio-ftts.agi|es|laura|127.0.0.1|alaw|midemo.txt)

;exten => 12,n,AGI(/home/quique/Desktop/asterisk-c/agi|${REF})

;; Reconocimiento de Voz:
exten => 8,1,Answer()
exten => 8,n,AGI(reconocedor.agi|es|es|verbio-number-
es.bnf|ABNF|127.0.0.1|4|2|alaw)
exten => 8,n,NoOp(Resultado reconocimiento ${VASR_RESULT})
exten => 8,n,AGI(verbio-tts.agi|es|laura|127.0.0.1|alaw|Usted ha dicho
${VASR_RESULT})

;;Limpieza directorios al colgar.
exten => h,1,DeadAGI(vcleanup.agi)
exten => h,n,HangUp()


;exten => 802,1,Goto(VERBIO_TEST,s,1)

exten => h,1,VerbioUnloadVcb(-1,,)

;Pruebas IrCoNet
exten => 7,1,Answer()
exten => 7,n,System(php /var/www/apache2-
default/irconet/control/pruebaAPIrconet.php)
exten => 7,n,HangUp()



[pruebas]

exten => 200,1, Macro(stdexten,200,SIP/200)
exten => user1, 1, Goto(200|1)
exten => 201,1, Macro(stdexten,201,SIP/201)
```

```
 exten => user2, 1, Goto(201|1)
 exten => 202,1, Macro(stdexten,202,SIP/202)
exten => user3, 1, Goto(202|1)

;; ISOLATED:
exten => 7,1,Answer()
exten =>
7,n,AGI(reconocedor.agi|es|es|domotica_iso.txt|ISOLATED|127.0.0.1|4|2|a
law)
exten => 7,n,NoOp(Numero de palabra del vocabulario ${VASR_INDEX})
exten => 7,n,NoOp(Resultado reconocimiento ${VASR_RESULT})
exten => 7,n,NoOp(Puntuacion reconocimiento ${VASR_SCORE})
exten => 7,n,AGI(ConVoz|${VASR_RESULT})
exten => 7,n,AGI(verbio-tts.agi|es|laura|127.0.0.1|alaw|Usted ha dicho
${VASR_RESULT})
exten => 7,n,AGI(verbio-ftts.agi|es|laura|127.0.0.1|alaw|midemo.txt)

;exten => 7,n,AGI(/home/quique/Desktop/asterisk-c/agi|${REF})

;; ABNF:
exten => 6,1,Answer()
exten =>
6,n,AGI(reconocedor.agi|es|es|domotica_abnf.bnf|ABNF|127.0.0.1|4|2|alaw
)
exten => 6,n,NoOp(Resultado reconocimiento ${VASR_RESULT})
exten => 6,n,AGI(verbio-tts.agi|es|laura|127.0.0.1|alaw|Usted ha dicho
${VASR_RESULT} )

;;Limpieza directorios al colgar.
exten => h,1,DeadAGI(vcleanup.agi)
exten => h,n,HangUp()


;exten => 802,1,Goto(VERBIO_TEST,s,1)

exten => h,1,VerbioUnloadVcb(-1,,)


[


[DV]

exten => 1,1,Answer()
exten => 1,n,VerbioLoadVcb(vdemo.txt,ISOLATED,,,v)
exten => 1,n,VerbioPromptAndRec(Dime el nombre de la persona a la que
quieres llamar,1500,100,es,laura,es,es,vbgidn)
exten => 1,n,VerbioPrompt("Escor es ${VASR_SCORE0}",es,laura,v)
exten =>
1,n(comprobacion),GotoIf($[${VASR_SCORE0}>60]?${VASR_RESULT0},1:)
exten => 1,n,VerbioPromptAndRec(No te he entendido por favor repite
alto y claro,1500,100,es,laura,es,es,vbgidn)
exten => 1,n,Goto(comprobacion)

exten => 15,1,VerbioPrompt("Muy bien tio 1",es,laura,v)
exten => 10,1,VerbioPrompt("Muy bien tio 2",es,laura,v)
exten => 11,1,VerbioPrompt("Muy bien tio 3",es,laura,v)
exten => h,1,VerbioUnloadVcb(-1,,)
;(audio_file[|initsil][|maxsil][|asr_conf][|asr_lang][|opts])
```

```
 ;(fichero_o_texto,initsil,maxsil,tts_lang,tts_spkr,asr_conf,asr_lang,o
 pts)
;exten => 9,n,VerbioPromptAndRec(Que hago?,1500,100,es,laura,es,es,vgn)

[skype]
exten => 5001,1,Dial(SIP/jkoblents@SPYKE,60)
exten => 201,1,Dial(SIP/201,60)
exten => 202,1,Dial(SIP/202,60)

include => demo


[general]

static=yes
writeprotect=no

autofallthrough=yes

clearglobalvars=no

priorityjumping=no

[globals]
CONSOLE=Console/dsp                     ; Console interface for demo
IAXINFO=guest                           ; IAXtel username/password
TRUNK=Zap/g2                            ; Trunk interface
TRUNKMSD=1                          ; MSD digits to strip (usually 1 or
0)
[dundi-e164-canonical]
[dundi-e164-customers]
[dundi-e164-via-pstn]

[dundi-e164-local]
include => dundi-e164-canonical
include => dundi-e164-customers
include => dundi-e164-via-pstn

[dundi-e164-switch]
switch => DUNDi/e164

[dundi-e164-lookup]
include => dundi-e164-local
include => dundi-e164-switch
[macro-dundi-e164]
exten => s,1,Goto(${ARG1},1)
include => dundi-e164-lookup
[iaxtel700]
exten =>
_91700XXXXXX,1,Dial(IAX2/${IAXINFO}@iaxtel.com/${EXTEN:1}@iaxtel)

[iaxprovider]

[trunkint]

exten => _9011.,1,Macro(dundi-e164,${EXTEN:4})
exten => _9011.,n,Dial(${TRUNK}/${EXTEN:${TRUNKMSD}})

[trunkld]

exten => _91NXXNXXXXX,1,Macro(dundi-e164,${EXTEN:1})
```

```
 exten => _91NXXNXXXXXX,n,Dial(${TRUNK}/${EXTEN:${TRUNKMSD}})

[trunklocal]

exten => _9NXXXXXX,1,Dial(${TRUNK}/${EXTEN:${TRUNKMSD}})

[trunktollfree]

exten => _91800NXXXXXX,1,Dial(${TRUNK}/${EXTEN:${TRUNKMSD}})
exten => _91888NXXXXXX,1,Dial(${TRUNK}/${EXTEN:${TRUNKMSD}})
exten => _91877NXXXXXX,1,Dial(${TRUNK}/${EXTEN:${TRUNKMSD}})
exten => _91866NXXXXXX,1,Dial(${TRUNK}/${EXTEN:${TRUNKMSD}})

[international]

ignorepat => 9
include => longdistance
include => trunkint

[longdistance]

ignorepat => 9
include => local
include => trunkld

[local]

ignorepat => 9
include => default
include => parkedcalls
include => trunklocal
include => iaxtel700
include => trunktollfree
include => iaxprovider
[macro-stdexten];

exten => s,1,Dial(${ARG2},20)                          ; Ring the
interface, 20 seconds maximum
exten => s,2,Goto(s-${DIALSTATUS},1)                        ; Jump
based on status (NOANSWER,BUSY,CHANUNAVAIL,CONGESTION,ANSWER)

exten => s-NOANSWER,1,Voicemail(u${ARG1})      ; If unavailable, send
to voicemail w/ unavail announce
exten => s-NOANSWER,2,Goto(default,s,1)              ; If they press
#, return to start

exten => s-BUSY,1,Voicemail(b${ARG1})                ; If busy, send
to voicemail w/ busy announce
exten => s-BUSY,2,Goto(default,s,1)                  ; If they press
#, return to start

exten => _s-.,1,Goto(s-NOANSWER,1)                   ; Treat anything
else as no answer

exten => a,1,VoicemailMain(${ARG1})                  ; If they press
*, send the user into VoicemailMain

[macro-stdPrivacyexten];
```

```
 exten => s,1,Dial(${ARG2},20|p)                              ; Ring the
 interface, 20 seconds maximum, call screening option (or use P for
databased call screening)
exten => s,2,Goto(s-${DIALSTATUS},1)                         ; Jump
based on status (NOANSWER,BUSY,CHANUNAVAIL,CONGESTION,ANSWER)

exten => s-NOANSWER,1,Voicemail(u${ARG1})      ; If unavailable, send
to voicemail w/ unavail announce
exten => s-NOANSWER,2,Goto(default,s,1)              ; If they press
#, return to start

exten => s-BUSY,1,Voicemail(b${ARG1})                ; If busy, send
to voicemail w/ busy announce
exten => s-BUSY,2,Goto(default,s,1)                  ; If they press
#, return to start

exten => s-DONTCALL,1,Goto(${ARG3},s,1)               ; Callee chose to
send this call to a polite "Don't call again" script.

exten => s-TORTURE,1,Goto(${ARG4},s,1)                ; Callee chose to
send this call to a telemarketer torture script.

exten => _s-.,1,Goto(s-NOANSWER,1)                   ; Treat anything
else as no answer

exten => a,1,VoicemailMain(${ARG1})                  ; If they press
*, send the user into VoicemailMain

[demo]

exten => s,1,Wait,1                ; Wait a second, just for fun
exten => s,n,Answer                ; Answer the line
exten => s,n,Set(TIMEOUT(digit)=5) ; Set Digit Timeout to 5 seconds
exten => s,n,Set(TIMEOUT(response)=10)   ; Set Response Timeout to 10
seconds
exten => s,n(restart),BackGround(demo-congrats); Play a congratulatory
message
exten => s,n(instruct),BackGround(demo-instruct)     ; Play some
instructions
exten => s,n,WaitExten       ; Wait for an extension to be dialed.

exten => 2,1,BackGround(demo-moreinfo)   ; Give some more information.
exten => 2,n,Goto(s,instruct)

exten => 3,1,Set(LANGUAGE()=fr)          ; Set language to french
exten => 3,n,Goto(s,restart)             ; Start with the
congratulations


exten => 4,1,BackGround(demo-moreinfo)   ; Give some more information.
exten => 4,n,Goto(s,restart)             ; Start with the
congratulations

;
exten => 1000,1,Goto(default,s,1)
;
; We also create an example user, 1234, who is on the console and has
; voicemail, etc.
;
exten => 1234,1,Playback(transfer,skip)        ; "Please hold
while..."
```

```
                                ; (but skip if channel is not up)
 exten => 1234,n,Macro(stdexten,1234,${CONSOLE})

exten => 1235,1,Voicemail(u1234)         ; Right to voicemail

exten => 1236,1,Dial(Console/dsp)        ; Ring forever
exten => 1236,n,Voicemail(u1234)         ; Unless busy


exten => #,1,Playback(demo-thanks)       ; "Thanks for trying the
demo"
exten => #,n,Hangup                ; Hang them up.


exten => t,1,Goto(#,1)             ; If they take too long, give up
exten => i,1,Playback(invalid)           ; "That's not valid, try
again"

exten => 500,1,Playback(demo-abouttotry); Let them know what's going on
exten => 500,n,Dial(IAX2/guest@misery.digium.com/s@default); Call the
Asterisk demo
exten => 500,n,Playback(demo-nogo)  ; Couldn't connect to the demo site
exten => 500,n,Goto(s,6)           ; Return to the start over message.


exten => 600,1,Playback(demo-echotest)   ; Let them know what's going
on
exten => 600,n,Echo                ; Do the echo test
exten => 600,n,Playback(demo-echodone)   ; Let them know it's over
exten => 600,n,Goto(s,6)           ; Start over

exten => 8500,1,VoicemailMain
exten => 8500,n,Goto(s,6)
[default]

include => demo
```