UNIVERSIDAD AUTÓNOMA DE MADRID

ESCUELA POLITECNICA SUPERIOR

PROYECTO FIN DE CARRERA

CODIFICACIÓN DE AUDIO PARA SONIDOS CARDIACOS Y RESPIRATORIOS ADQUIRIDOS CON ESTETOSCOPIO DIGITAL

Ignacio Palacios Santos

Octubre 2012

CODIFICACIÓN DE AUDIO PARA SONIDOS CARDIACOS Y RESPIRATORIOS ADQUIRIDOS CON ESTETOSCOPIO DIGITAL

AUTOR: Ignacio Palacios Santos

TUTOR: Andrés Martínez Fernández

PONENTE: Doroteo Torre Toledano

Biometric Recognition Group - ATVS

Escuela Politécnica Superior

Universidad Autónoma de Madrid

Octubre de 2012

Fundación Enlace Hispano Americano de Salud

Departamento de Teoría de la Señal y Comunicaciones

Universidad Rey Juan Carlos

Octubre de 2012

RESUMEN

En 2011 finalizó un proyecto piloto de la Fundación EHAS (Enlace Hispano Americano de Salud) en el área peruana del rio Napo, cuyo objetivo era proveer a la población local de tele-diagnóstico sobre afecciones cardiorrespiratorias. Para ello, esta Fundación junto a la Fundación FUNDATEL de Argentina desarrollaron un estetoscopio digital bluetooth de bajo coste y un sistema de transmisión extremo-extremo del audio cardiorrespiratorio adquirido. El diagnóstico se realiza entre los puestos de salud rurales (sin médico) y los centros de salud de referencia (donde se encuentra el médico). A través de videoconferencia y con un estetoscopio digital en cada extremo el médico va guiando al técnico de salud remoto, transmitiéndose el audio cardiorrespiratorio mediante VoIP. Esta solución esta permitiendo a muchas personas aisladas tener un diagnóstico rápido, evitando así tener que trasladarse cientos de kilómetros hasta el hospital mas cercano transitando por zonas mal comunicadas. Tras este proyecto piloto nacieron otros, entre ellos éste, con el objetivo de mejorar los servicios ofrecidos y aprovechar la infraestructura de comunicaciones ya construida.

Este PFC presenta una propuesta para la codifiación en tiempo real de audios cardiacos y respiratorios obtenidos a través del estetoscopio digital bluetooth de bajo coste. Actualmente se utiliza el codificador de voz G.722 para la compresión y posterior transmisión de las señales cardio-respiratorias, pero éstas tienen unas características frecuenciales diferentes a las señales vocales. Por lo tanto se busca una alternativa para codificar estas señales de forma óptima, con un coste computacional bajo, buena calidad y un bajo uso del ancho de banda.

PALABRAS CLAVE

Sonidos Cardiorrespiratorios, Estetoscopio, Codificación ADPCM, Cooperación al Desarrollo, VoIP.

ABSTRACT

In 2011, a pilot project was conducted in the Peruvian area of River Napo, with the objective of providing cardiorespiratory tele-diagnosis for the local population. EHAS Foundation, together with Fundatel Foundation, developed a low cost bluetooth digital stethoscope and a peer-to-peer transmission system for the cardiorespiratory signals. The diagnosis is made between rural health posts (without a physician) and health centres of reference (with medical presence). With the audio encoding system, the physician is able to guide the patient via videoconference by means of a digital stethoscope at each end, transmitting the cardiorespiratory audio over VoIP. This solution will allow many isolated people to have a quick diagnosis, sparing them the travelling of hundreds of miles to get to the nearest hospital, going through poorly served areas.

This Thesis presents a real-time audio encoding system for cardiorespiratory sounds acquired with a low cost Bluetooth digital stethoscope developed by EHAS (Enlace Hispano Americano de Salud) Foundation and Fundatel Foundation. The system is currently working with a G.722 speech coder for compression and subsequent transmission of cardiorespiratory signals. However, these signals have different frequency characteristics from speech. Therefore, we sought a better adapted alternative to encode these signals optimally, with an encoder not subject to the payment of any license, with low computational cost, good quality and low bandwidth.

KEYWORDS

Cardiorespiratory Sounds, Stethoscopy, ADPCM Encoding, Development Cooperation, VoIP.

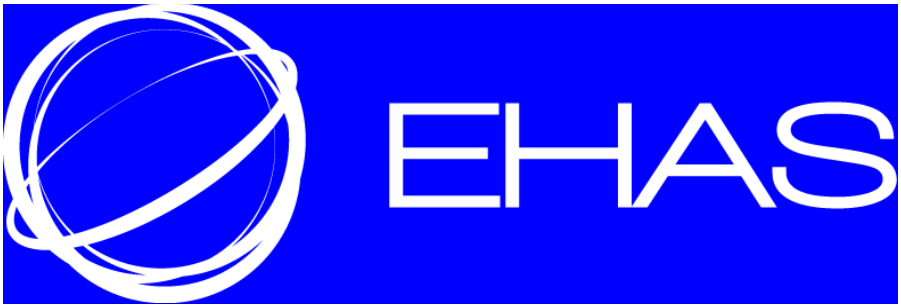
AGRADECIMIENTOS

Quería agradecer la realización de este proyecto a todos mis amigos de la carrera (Nicolás, Sergio, Pablo, Imanol, Marcos, Juanjo, Ana, Bogdana, Guido …) que han hecho que la estancia en esta escuela sea mucho más divertida y entretenida, y que me han ayudado en los estudios, en todas aquellas asignaturas en las que he tenido dificultades. También a todos a mis amigos del colegio, que me acompañan desde hace muchos años. Especialmente quiero agradecer la paciencia que han tenido y el apoyo que he recibido por parte de mi familia y Oihane.

Agradecer también a mi tutor Andrés Martínez por brindarme la oportunidad de realizar el PFC en la Fundación EHAS, orientando así este trabajo hacia la Cooperación al Desarrollo, y a los compañeros de la Fundación, especialmente a Nacho por su gran ayuda e interés. Por ultimo agradecer a mi ponente Doroteo Torre, por aceptar la colaboración con esta Fundación y tutorizar todo el desarrollo del PFC.

Ignacio Palacios Santos

Octubre, 2012



El presente Proyecto Fin de Carrera fue concebido y desarrollado en el Departamento de Teoría de la Señal y Comunicaciones de la Fundación Enlace Hispano Americano de Salud (EHAS), Universidad Rey Juan Carlos I de Madrid.

INDICE DE CONTENIDOS

INDICE DE FIGURAS

INDICE DE TABLAS

1. INTRODUCCIÓN

1.1 Motivación

1.1.1 La Fundación EHAS

La Fundación EHAS es una institución sin animo de lucro cuyo fin es promover el uso apropiado de las nuevas Tecnologías de la Información y la Comunicación (TIC) para mejorar los procesos de salud en zonas rurales aisladas de países en desarrollo. Para ello plantea cuatro grandes líneas de acción [1]:

1. La investigación y el desarrollo de nuevas tecnologías de comunicación y sistemas de acceso e intercambio de información adaptadas a las zonas rurales de países en desarrollo.

2. El asesoramiento, desarrollo y evaluación de protocolos de actuación para la mejora de los procesos de atención de salud en las zonas rurales, con especial atención en los relacionados con la salud materno-infantil.

3. El diseño y la ejecución de proyectos de cooperación para el desarrollo que permitan validar tanto la tecnología como los protocolos de actuación anteriores.

4. El desarrollo de actividades de formación, difusión, transferencia e incidencia política, para promover el uso adecuado de las TIC en el sector salud rural de países en desarrollo.

EHAS nace de un equipo de estudiantes del Grupo de Bioingeniería y Telemedicina (GBT) de la Universidad Politécnica de Madrid (UPM) y de la ONGD Ingeniería Sin Fronteras, que en 1997, arrancaron investigaciones para el diseño de sistemas y servicios de comunicación apropiados a las necesidades del personal sanitario rural de los países de AL (América Latina). A raíz de estos trabajos se diseña

y ejecuta el Programa Enlace Hispano Americano de Salud (EHAS), que desde la telemedicina pretende contribuir a la mejora de los sistemas públicos de asistencia sanitaria, en las zonas rurales de AL. Después de un periodo inicial de investigación realizado en Madrid por el GBT-UPM se obtiene una importante conclusión: el acceso a Internet a traves de la tecnología radio VHF/HF en zonas rurales aisladas de países en desarrollo es viable, tanto tecnológica como económicamente. EHAS fue constituida como fundación en el año 2004 por la Universidad Politécnica de Madrid (UPM) y la ONGD Ingeniería Sin Fronteras Asociación para el Desarrollo (ISF ApD), albergando su sede en la Escuela Técnica Superior de Ingenieros de Telecomunicación (ETSIT) de UPM. En 2008 se amplio el patronato con la Universidad del Cauca de Colombia, la Pontificia Universidad Católica del Perú (UPCH) y la Universidad Rey Juan Carlos. La PUCP, ha actuado desde 1997 en Perú como contraparte tecnológica, y la Facultad de Medicina de la Universidad Peruana Cayetano Heredia (UPCH), actuaría como contraparte médica durante los primeros años del Programa EHAS. Este equipo multidisciplinar comienza a trabajar en el desarrollo de dos líneas principales de acción: la tecnología y los teleservicios . Actualmente, la Fundación EHAS se encuentra investigando también las posibilidades que otras tecnologías inalámbricas pueden ofrecer, como por ejemplo los nuevos estándares de WiFi 802.11n y 802.11e, así como la tecnología certificada WiMAX, 802.16. Asimismo, la fundación se encuentra también inmersa en dos nuevas líneas que fueron recientemente emprendidas, que son: la línea del desarrollo de nuevos servicios de telemedicina, cuyo principal exponente en este momento es el teleestetoscopio digital Bluetooth, el desarrollo de un sistema de teleelectrocardiograma (tele-ECG) y en una fase preliminar del desarrollo de sistemas de telemicroscopía y teleecografia, todos ellos de bajo coste. EHAS trabaja desde hace años en Perú, Colombia, Cuba, Ecuador y Guatemala y tiene la intención de empezar a desarrollar sus actividades en Paraguay muy pronto [1].

1.1.2 La telemedicina en zonas rurales aisladas en países en desarrollo

La Secretaría de la ITU-D (Telecommunication Development Sector) es la BDT (Bureau du Developpement des Telecommunications), cuyas funciones son las de promover y ofrecer asistencia técnica a los países en desarrollo en el campo de las telecomunicaciones, promover la movilización de los recursos materiales y financieros necesarios para su implementación, y promover la extensión de los beneficios de las nuevas tecnologías de telecomunicación a todos los habitantes del planeta. Según una definición de la ITU-D/BDT, la "Telemedicina es la investigación, monitorización y gestión de pacientes y la educación de pacientes y personal médico, que permite un fácil acceso a la opinión de expertos e información de pacientes, sin importar donde los pacientes o la información relevante este localizada" [2]. Por su parte, la Organización Mundial de la Salud (OMS), describe la telemedicina como "la practica del cuidado medico usando comunicaciones audiovisuales e interactivas incluyendo atención médica a distancia, diagnóstico, consulta y tratamiento, así como la educación y transferencia de datos médicos". La Telemedicina interactúa con cualquier disciplina de la medicina, desde la cirugía hasta los análisis epidemiológicos en zonas endémicas, igual que contribuye a optimizar la cobertura sanitaria de regiones aisladas. Con ella se puede extender el alcance de las especialidades médicas y puede mejorar la manera en que interactúan los profesionales médicos con sus pacientes. Estas contribuciones pueden enmarcarse dentro de un conjunto de servicios básicos que nombramos a continuación [3]:

- **Teleformación:** Brinda la posibilidad de capacitar a distancia a médicos, enfermeras y demás personal sanitario, por medio de videoconferencias asistidas, formándolos médicamente y/o actualizándolos en los avances tecnológicos, permitiendo la ubicuidad del personal formador. De esta manera se resuelven problemas como la distancia, los costos altos de capacitación y la necesidad de presencialidad del profesorado de alta calidad.

- **Telediagnóstico**: Surge con la necesidad de reducir costos, molestias de traslado al hospital y mejorar la atención a los pacientes que necesiten revisiones y controles durante varios meses. La atención al paciente, se puede

realizar cuando éste se encuentre en su centro/puesto de salud, pero la emisión del diagnóstico se realiza a través de una conexión remota al hospital/centro de salud, en el cual el personal mas capacitado podrá emitir un diagnóstico o una segunda opinión. Una aplicación que surgió a partir del telediagnóstico es el almacenamiento de las historias clínicas en bases de datos, que facilitan al médico la labor de búsqueda de las mismas en el momento de la atención al paciente.

- **Teleconsulta.** Se basa en la realización de consultas a un especialista o médico general por parte del paciente desde su centro sanitario o desde su propio hogar; el doctor lo interroga y realiza toda la consulta a distancia, generalmente mediante videoconferencia. Posteriormente puede o no emitir un diagnóstico en función de si requiere la realización de alguna prueba que no pueda ser hecha a distancia. Un sistema de teleconsulta, generalmente aumenta el número de consultas recibidas por el médico en cuestión.

- **Historia Clínica Digital o Telealmacenamiento clínico**. La recopilación de la anamnesis, realizada o no mediante teleconsulta, por contacto directo, o por una solución mixta entre ambas, arroja toda una serie de datos que tradicionalmente se han venido almacenando en lo que constituye la historia clínica del paciente y que sirve al profesional médico para disponer del histórico de datos relevantes para el diagnóstico con respecto a un paciente. El almacenamiento en papel de estos datos generaba el problema de que la referencia/contrarreferencia de un paciente obligaba a remitir también todos estos datos junto al mismo. La disposición en una base de datos centralizada o distribuida de esta información, accesible para los profesionales médicos autorizados, facilita enormemente el diagnóstico, así como también ofrece mas garantía sobre la seguridad de los mismos (tanto contra perdida, deterioro, robo, o contra acceso no autorizado, empleando técnicas adecuadas de cifrado y autorización de acceso).

- **Telemonitorización.** Consiste en la utilización de un equipamiento de monitorización conectado a los pacientes, con el que son registrados los parámetros que se quiere controlar. Generalmente se monitorizan los signos vitales mediante un ECG (Electrocardiograma), pulsometro y oximetro, si bien

cualquier parámetro que pueda medirse y enviarse a una ubicación remota durante un periodo de tiempo, entraría en esta categoría. Toda la información recopilada es dirigida a un centro de monitorización, donde se encuentra personal atento para cualquier eventualidad, o a un sistema de procesamiento, que analiza las señales recibidas y genera las alertas correspondientes hacia el personal sanitario.

Por otra parte, hay que destacar tambien la jerarquía de establecimientos de salud rurales en los países en desarrollo, que generalmente se divide en [4]:

- **Puestos de Salud (PS):** Son el escalafón mas bajo en el sistema de atención primaria. Están localizados en las poblaciones mas aisladas, generalmente en poblaciones con pocos habitantes, sin línea telefónica y con sistemas de transporte muy deficientes (sin carreteras, o con pocas y de mala calidad). Dependen jerárquicamente de los Centros de Salud (CS) de referencia, constituyendo una red en la que varios PS dependen de un mismo CS. En la mayor parte de los casos son dirigidos por un técnico de enfermería con escasa formación que requiere, debido a que se enfrentan en algunas ocasiones a casos graves, de continua realimentación por parte de los médicos de referencia que se encuentran en los CS. El número de técnicos de enfermería que atienden un PS es muy reducido (en ocasiones una o dos personas), para hacer frente a las necesidades sanitarias de primer nivel, atendiendo al 70-80% de la demanda del sistema de salud. Debido a que los PS se enclavan en las regiones más aisladas de la geografía de los países en desarrollo, el aislamiento al que son sometidos estos profesionales es muy elevado, y en muchas ocasiones, este personal no procede de la población en la que ejerce. Este aislamiento no es el entorno apropiado para el desempeño de su profesión, por lo que frecuentemente la rotación de este tipo de personal es muy elevada (en ocasiones con rotaciones anuales), provocando en cada rotación de personal la pérdida de la experiencia adquirida y, por tanto, deteriorando la calidad de la atención, la relación con el paciente y derrochando los recursos formativos de la región.

- **Centros de Salud (CS):** Se sitúan jerárquicamente sobre los PS. Generalmente se ubican en capitales de distrito, donde existe mayor disponibilidad de sistemas de telecomunicación como la telefonía. Son dirigidos por médicos y presentan cierta infraestructura y equipamiento para la realización de algunas pruebas diagnósticas. En muchos casos permiten la hospitalización. Normalmente, en zonas aisladas, es en los CS donde se gestionan los informes epidemiológicos que se envían desde los PS, así como también se organizan batidas regulares para alcanzar aquella población que, por su ubicación y falta de movilidad, no puede desplazarse para ser atendida en un PS o CS. Con frecuencia en los CS se dispone de algún tipo de equipo de cómputo para digitalizar los informes que se envían desde los PS y suelen contar con algún personal responsable de esta tarea.

La telemedicina, entendida ésta como la suma de todas sus áreas, desde la tele formación hasta el telealmacenamiento clínico, ha tenido diversos niveles de implantacion en el mundo. Es importante resaltar que donde se observa la mayor disparidad de implantación de la telemedicina es realizando la distinción urbano/rural. El medio rural ha sufrido, especialmente en los países en desarrollo, el abandono sanitario, que ha concentrado todos los medios, y más concretamente todos los referentes a la telemedicina, en el sector urbano [4].

1.2 Objetivos y Métodos

Este trabajo pretende presentar una propuesta para la codificación en tiempo real de audios cardiacos y respiratorios obtenidos a través del estetoscopio digital bluetooth de bajo coste desarrollado por la fundación EHAS y la fundación Fundatel. Actualmente se utiliza el codificador de voz G.722 para la compresión y posterior transmisión de las señales cardio-respiratorias, pero estas tienen unas características frecuenciales diferentes a las señales vocales. Por lo tanto se busca una alternativa para codificar estas señales de forma óptima, con un coste computacional bajo, buena calidad y un bajo uso del ancho de banda.

Para la realización de este PFC se siguieron los siguientes pasos en busca de una alternativa al codificador G.722. En primer lugar se estudiaron las posibles

alternativas de codificación disponibles. Las patentes de estas codificaciones tenían que haber ya expirado o pertenecer al código libre. Una vez seleccionados los codificadores se realizaron codificaciones de audios adquiridos con estetoscopios digitales, generando varios Scripts de Linux, los cuales hacen uso del programa de manipulación de audio SoX (Sound eXchange). Gracias a esas pruebas se extrajeron diferentes figuras de mérito, las cuales ayudaron a seleccionar el codificador óptimo. El siguiente paso fue generar un programa en lenguaje C, capaz de ser integrado posteriormente en el programa de videoconferencia Softphone Ekiga. Tras la integración del codificador en Softphone Ekiga se establecieron llamadas entre dos ordenadores seleccionando el nuevo codificador como codificador de audio. Finalmente diferentes doctores del Hospital Clinico San Carlos de Madrid realizaron una evaluación subjetiva del codificador.

1.3 Organización del PFC

El presente PFC está organizado de la siguiente forma:

- Capítulo 1. Introducción, motivación y metas del PFC.
- Capítulo 2. Estado del arte de todas las tecnologías involucradas en la teleestetoscopia (características de las señales cardiorrespiratorias, posibles codificadores a aplicar, tecnología VoIP).
- Capítulo 3. Diseño y desarrollo de una codificador para señales cardiorrespiratorias.
- Capítulo 4. Test y pruebas del codificador propuesto, tanto objetivas como subjetivas, así como su implementación para llamadas VoIP.
- Capítulo 5. Conclusiones y trabajo futuro.
- ANEXO A. Script del Codec-Estetoscopia-Propuesto.
- ANEXO B. Códigos fuente del Codec-Estetoscopia-Propuesto.

- ANEXO C. Resultados detallados de las diferentes codificaciones testeadas.

- ANEXO D. Archivos generados para la librería OPAL.

- ANEXO E. Paper aceptado en la conferencía Iber Speech 2012 Audio Encoding for Heart and Breath Sounds Acquired with Digital Stethoscope

2. ESTADO DEL ARTE

2.1 Introducción

En esta sección se realizará un repaso de todos los temas involucrados en este proyecto. Se hablará de los principios básicos de la teleestetoscopia, los codificadores empleados para la compresión de las señales cardiorrespiratorias digitalizas, los estudios actuales sobre compresión de señales biomédicas y las características de la transmisión de señales mediante VoIP asi como el programa de videoconferencia empleado para ello.

2.2 La teleestetoscopia

2.2.1 Introducción

Un estetoscopio, también llamado fonendoscopio, es un aparato acústico usado en medicina, enfermería, kinesiología, fonoaudiología y veterinaria, para la auscultación o para oír los sonidos internos del cuerpo humano o animal. Generalmente se usa en la auscultación de los ruidos cardíacos o los ruidos respiratorios, aunque algunas veces también se usa para objetivar ruidos intestinales o soplos por flujos anómalos sanguíneos en arterias y venas. El estetoscopio que conocemos hoy en dia no es muy diferente al original aunque debemos resaltar que es de metal, biauricular y posee un receptor de sonidos junto con dos tubos de goma para los oídos**.** En conclusión decimos que el estetoscopio nos permite la ampliación de la energía sonora producida por los ruidos del organismo, de esta forma se consiguen mostrar los fenómenos producidos por el cuerpo sin deformación alguna, lo que permite al médico obtener datos vitales para luego elaborar un diagnóstico exacto. Actualmente, con la llegada de la era digital, han surgido un gran número de estetoscopios electronicos desarrollados por empresas privadas ligadas a la medicina como 3M y Zarguis Medical. Estos estetoscopios están ganando terreno respecto a los otros modelos tradicionales, ya que presentan mejoras significativas: posee una

mayor respuesta a la frecuencia, una mejor sensibilidad al sonido y un control de volumen para poder disminuir el nivel si el sonido es muy fuerte o molesto para el oído de los seres humanos. Gracias a las redes de comunicaciones y a los nuevos estetoscopios digitales ha surgido el nuevo término de teleestetocopía. La gran mayoría de los estetoscopios digitales no tienen función para la transmisión de las señales en tiempo real, su modo de funcionamiento es grabar las señales cardiorrespiratorias digitales y almacenarlas para su posterior envío si es necesario. Estos nuevos estetoscopios suelen ser utilizados para la docencia, ya que se puede mostrar el audio de una patología a un gran número de alumnos de manera simultánea, sin necesidad de una auscultación directa por parte de los alumnos y la presencia de un paciente con esa patología.

2.2.2 Caracterización de los sonidos respiratorios y cardiacos

Los sonidos cardiorespiratorios se obtienen con el estetoscopio mediante la auscultación. La auscultación es el procedimiento clínico de la exploración física que consiste en escuchar de manera directa o por medio de instrumentos como el estetoscopio, el área torácica o del abdomen, en busca de los sonidos normales o patológicos producidos por el cuerpo humano [5]. Existen cuatro zonas de auscultacion:

- **Mitral**: En el ápice cardiaco.
- **Tricúspide**: En el cuarto y quinto espacio intercostal izquierdo a lo largo del borde izquierdo del esternón.
- **Aórtica**: En el segundo espacio intercostal a lo largo del borde derecho del esternón.
- **Pulmonar**: En el segundo espacio intercostal a lo largo del borde izquierdo del esternón.

Los sonidos recogidos en estas 4 zonas tendrán unas características frecuenciales diferentes a las de la voz.

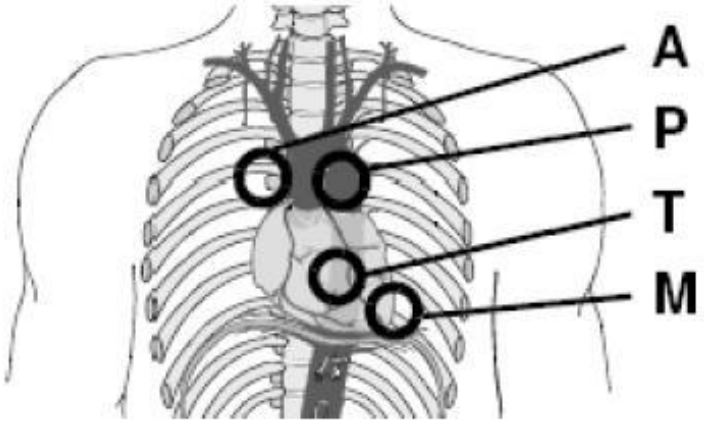


Figura 1 - Zonas de auscultación

Generalmente, las componenetes de los sonidos cardiacos y respiratorios útiles para el diagnóstico están en el rango de 20-1000 Hz. Los dos primeros de los cuatro sonidos cardiacos (S1 y S2), estan producidos por el cierre de la válvulas atrioventricular y semilunar respectivamente. Estos sonidos se encuentran en el rango de 20-115 Hz. Afecciones como los soplos cardiacos se producen en el rango de 140-600 Hz [6]. De esta forma, el rango de los sonidos cardiacos útiles para su diagnóstico se encuentra aproximadamente entre 20 Hz y los 600 Hz. Para los sonidos respiratorios, las frecuencias mas importantes de la señal se encuentran normalmente por debajo de los 100 Hz, aunque la señal puede tener componentes útiles hasta los 1.2 kHz [7].

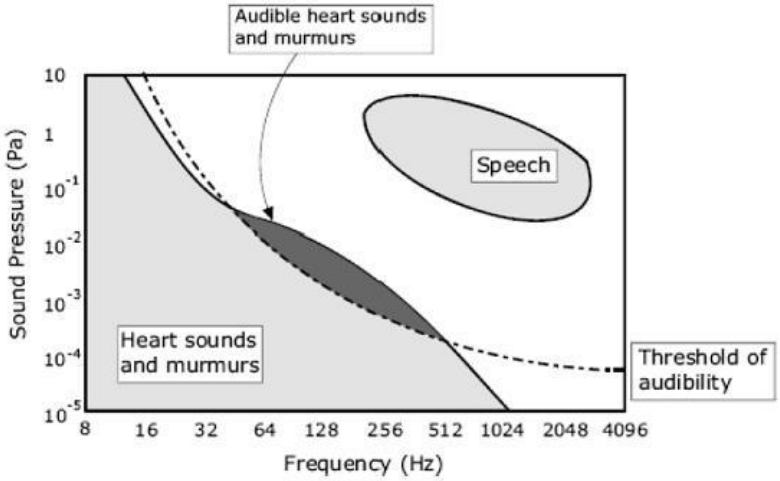


Figura 2 - Distribución frecuencial de los sonidos cardiorrespiratorios

Por lo tanto, mediante el teorema de Nyquist [8]:

$$f_s > 2 f_M \qquad (1)$$

podremos deducir la frecuencia de muestreo minima (f_s) con la que se pueden submuestrear las señales cardiorrespiratorias sin perder información relevante para el diagnóstico. Siendo f_M = 1200 Hz la frecuencia máxima, si (1) entonces la señal cardiorrespiratoria se podrá recuperar completamente mediante la interpolación de sus muestras. Esto significa que la señal tendrá que ser muestreada al menos con una f_s = 2400 Hz.

2.3 Codificación de señales biomédicas

Debido al desarrollo de la telemedicina en los últimos años surgió el interés en el procesamiento digital de señales biomédicas, para su posible transporte a través de las redes de comunicación. El objetivo principal es la compresión de las señales biomédicas sin perder las características útiles para un buen diagnóstico. Se realiza una compresión con pérdidas, ya que el canal de comunicaciones (ancho de banda) exige una tasa binaria baja.

2.3.1 Compresión de señales cardiacas

Existen principalmente dos tipos de señales biomédicas relacionadas con el corazón, el electrocardiograma (ECG, que consiste en la digitalización de las señales eléctricas producidas por el corazón) y el fonocardiograma (FCG, es la representación gráfica de los sonidos y soplos generados por el corazón, y podría pensarse en un principio en utilizar técnicas estándar de compresión de audio. Sin embargo, estas técnicas han sido desarrolladas principalmente para la compresión con gran calidad de música o habla, por lo que su aplicación a la compresión del FCG no es adecuada, dado que el rango de frecuencias de interés de la música o del habla es muy diferente al del FCG [9]) . En nuestro caso nos vamos a centrar en los FCG ya que es el tipo de señal biomédica digitalizada que nos compete en este PFC.

Hasta el momento se ha trabajado con tres métodos de compresión: algoritmo SPIHT [10] (Set Partitioning in Herarchical Trees), códigos run-lenght y Huffman, y transformada Wavalet. Mediante la transformada Wavelet se hace una detección de eventos del ciclo cardiaco, aplicando posteriormente un método de umbralización [9].

2.3.2 Compresión de señales respiratorias

Los principales métodos de compresión de señales respiratorias estan basados actualmente en en transformada DCT (Discrete Cosine Transform) y BAMs adaptativos (Bit Allocation Methods) [11]. Esto se debe a que las señales respiratorias son mas parecidas a las señales vocales en cuanto a sus caracterisiticas estadisticas, siendo éstas usualmente no estacionarias.

El método BAMs consiste en asignar bits a diferentes coeficientes de transformadas para la cuantización, la razón por la cual estos métodos deben ser adaptativos se debe a que las señales respiratorias son de tipo no estacionario. El método usado para la compresión es la segmentación de la señal en ventanas que no se superpongan, además de ser normalizada por la desviación estándar [11].

Teniendo en cuenta que las señales respiratorias tienen características similares a las del habla, una de las técnicas de compresión es la codificación en el dominio de la frecuencia, para lo cual se extraen los componentes de frecuencia de las señales mediante un banco de filtros, para luego codificar mediante cuantización escalar [12]. Este sería el caso de la codificación DCT, donde se realizan compresiones de señales estacionarias y no estacionarias.

2.4 Codificadores de forma de Onda

2.4.1 Introducción

Los codificadores de forma de onda son aquellos que no hacen ningún tipo de suposición acerca de la naturaleza de la señal a codificar. Simplemente consideran que deben codificar una forma de onda de modo que la forma de onda decodificada se parezca lo más posible a la original [13]. El método más sencillo de codificación de forma de onda es el Pulse Code Modulation (PCM). Básicamente el codificador PCM transforma una señal analógica en una secuencia de bits mediante muestreo uniforme y cuantificación. El decodificador opera de manera inversa.

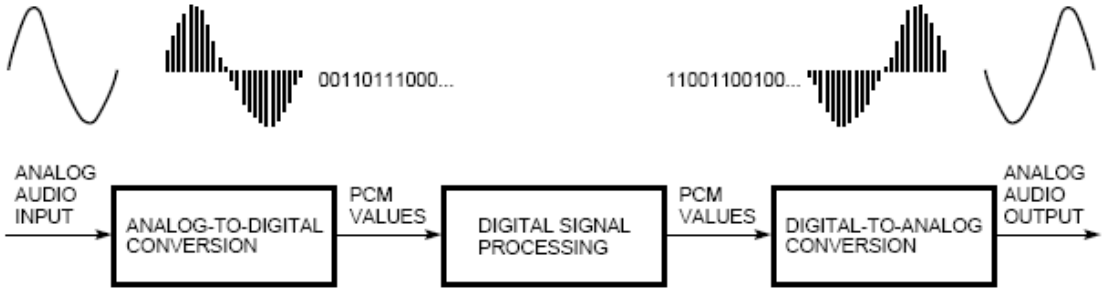


Figura 3 - Codificación Pulse Code Modulation (PCM)

Existe una variante al PCM denominado DPCM (PCM diferencial) que explota el hecho de que las muestras de las señales de voz no cambian mucho de una muestra a otra, sino que están correladas. Basándose en la muestra anterior de la señal, predice el valor actual cometiendo un error de predicción, al ser el margen dinámico de la señal de error menor que el de la señal original se cuantifica dicha señal de error, empleando un número inferior de bits manteniendo el ruido de codificación bajo [13].

2.4.2 G.722

Esta Recomendación de la UIT-T describe las características de un sistema de codificación audio (banda de 50 a 7000 Hz) que puede utilizarse en diversas aplicaciones para señales vocales de alta calidad. El sistema de codificación utiliza

modulación por impulsos codificados diferencial adaptativa de subbanda (MICDA-SB) a una velocidad binaria de hasta 64 kbit/s. El sistema se denominará en adelante sistema de codificación audio (7 kHz) a 64 kbit/s. En la técnica MICDA-SB utilizada, la banda de frecuencias se divide en dos subbandas (superior e inferior) y las señales de cada una se codifican utilizando la MICDA [14].

El sistema tiene tres modos básicos de funcionamiento, correspondientes a las velocidades binarias utilizadas para la codificación de audio de 7 kHz: 64, 56 y 48 kbit/s. Los dos últimos modos permiten obtener, respectivamente, un canal de datos auxiliar de 8 kbit/s o de 16 kbit/s, que se proporciona dentro de los 64 kbit/s mediante el uso de bits de la subbanda inferior (como se explicará más adelante). En la Figura 4 pueden verse las principales partes funcionales del códec audio (7 kHz) a 64 kbit/s. Éstas son las siguientes [14]:

1. Codificador audio (7 kHz) a 64 kbit/s, que comprende:

- Una parte audio de emisión que convierte una señal audio en una señal digital uniforme, que se codifica utilizando 14 bits con muestreo a 16 kHz.

- Un codificador MICDA-SB que reduce la velocidad binaria a 64 kbit/s;

2. Decodificador audio (7 kHz) a 64 kbit/s, que comprende:

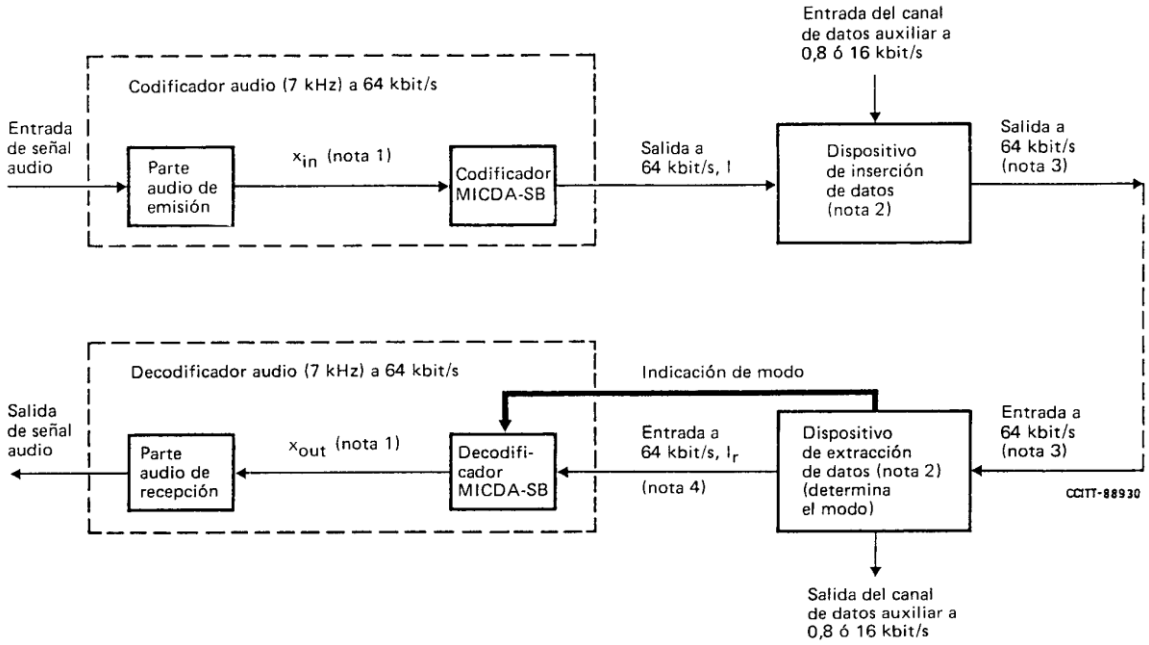
- Un decodificador MICDA-SB que realiza la operación inversa a la del codificador, considerando que la velocidad binaria efectiva de codificación audio a la entrada del decodificador puede ser de 64, 56 ó 48 kbit/s, según el modo de funcionamiento.

- Una parte audio de recepción que reconstruye la señal audio a partir de la señal digital uniforme, que se ha codificado utilizando 14 bits con muestreo a 16 kHz.

Las dos partes siguientes, que se indican en la Figura 4 para mayor claridad, serán necesarias en aplicaciones que exijan un canal de datos auxiliar en la señal a 64 kbit/s:

- Un dispositivo de inserción de datos en el extremo emisión, que utiliza, si es necesario, uno o dos bits por octeto en la banda audio según el modo de funcionamiento y los sustituye por bits de datos para proporcionar un canal de datos auxiliar a 8 ó 16 kbit/s respectivamente.

- Un dispositivo de extracción de datos en el extremo recepción que determina el modo de funcionamiento según una estrategia de control de modos y extrae los bits de datos según proceda.



Nota 1 – x_{in} y x_{out} son señales digitales codificadas uniformemente con 14 bits y muestreadas a 16 kHz.
Nota 2 – Estos dispositivos sólo son necesarios en aplicaciones que requieren un canal de datos auxiliar dentro de los 64 kbit/s.
Nota 3 – Comprende 64, 56 ó 48 kbit/s para la codificación de las señales audio y 0,8 ó 16 kbit/s para datos.
Nota 4 – Señal a 64 kbit/s que comprende 64, 56 ó 48 kbit/s para la codificación de audio, según el modo de funcionamiento.

Figura 4 - Codificación/Decodificación G.722 [14]

2.4.3 G.726

Es un codificador de voz ADPCM bajo la recomendación de la UIT-T para la transmisión de audios a 16, 24, 32 y 40 kbit/s. Fue introducido para reemplazar el codec ADPCM G.721 a 32 kbit/s y al codificador ADPCM G.723 a 24 y 40 kbit/s. Las cuatro tasas binarias de este codificador estan relacionadas con el tamaño de las muestras, las cuales son 2, 3, 4 y 5 bits respectivamente. La aplicación principal de los

canales a 24 y 16 kbit/s es para canales de sobrecarga que transportan señal vocal en equipos de multiplicación de circuitos digitales (EMCD), mientras que la aplicación principal de los canales de 40 kbit/s es la del transporte de señales de módem de datos en EMCD, especialmente en módems que funcionan a velocidades superiores a 4800 bit/s [15].

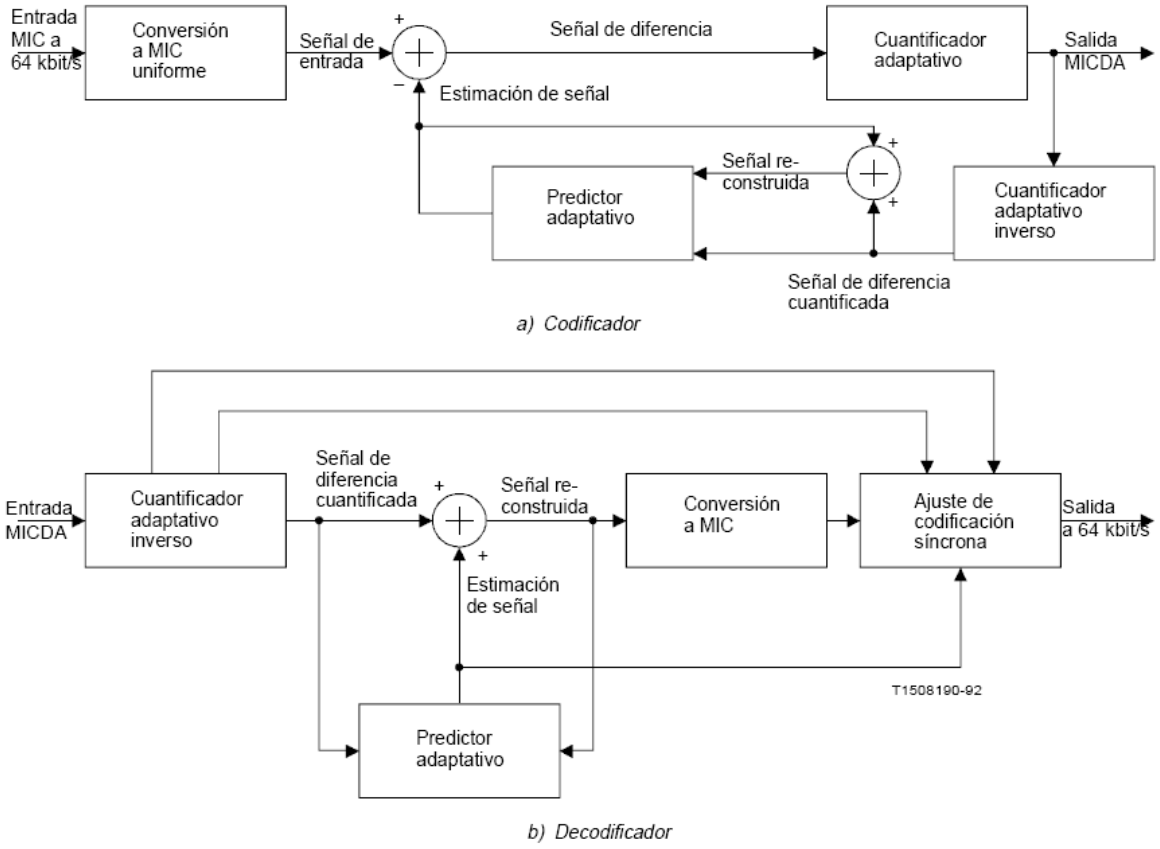


Figura 5 - Codificación/Decodificación G.726 [15]

Como se puede observar en la Figura 5, se pueden destacar dos etapas fundamentales del codec G.726 [15]:

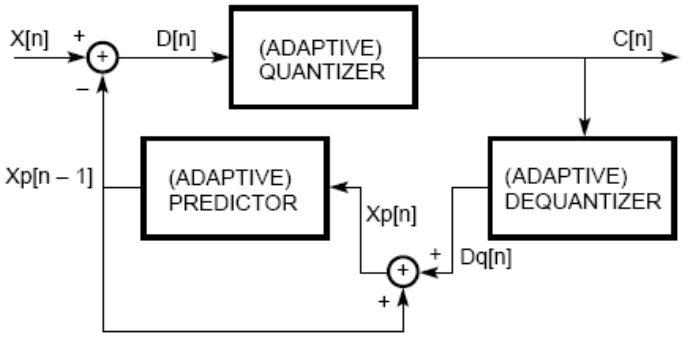
- **Codificador MICDA:** Tras la conversión de la señal de entrada MIC, codificada según la ley A o la ley μ en una MIC uniforme, se obtiene una señal de diferencia sustrayendo, de la señal de entrada, una estimación de dicha señal. Se utiliza un cuantificador adaptativo de 31, 15, 7 ó 4 niveles para asignar 5, 4, 3 ó 2 dígitos binarios, respectivamente, al valor de la señal de diferencia, para su transmisión a decodificador. Un cuantificador inverso produce la señal de diferencia cuantificada a partir de estos mismos 5, 4, 3, ó 2

dígitos binarios, respectivamente. El valor estimado de la señal (estimación de señal) se añade a esta señal de diferencia cuantificada para producir una versión reconstruida de la señal de entrada. Tanto la señal reconstruida como la señal de diferencia cuantificada se aplican a un predictor adaptativo, que produce la estimación de la señal de entrada completando así el bucle de realimentación.

- **Decodificador MICDA:** El decodificador consiste en una parte idéntica a la que constituye el bucle de realimentación del codificador, y una conversión de MIC uniforme a ley A o ley μ más un ajuste de codificación síncrona. El ajuste de codificación síncrona evita la distorsión acumulativa que se produce en las codificaciones síncronas en cascada (conexiones digitales MICDA-MIC-MICDA, etc.) en determinadas condiciones. El ajuste de codificación síncrona se consigue ajustando los códigos de salida MIC de tal manera que se elimine la distorsión de cuantificación en la siguiente etapa de codificación MICDA.

2.4.4 IMA-ADPCM

El codificador IMA-ADPCM es un codificador de audio de tipo ADPCM. La codificación ADPCM (Adapatative Differential Pulse Code Modulation) es un tipo de codificación diferencial con pérdidas, en la que la diferencia de la muestra codificada respecto a la muestra anterior se cuantifica con un paso de cuantización (*step*) adaptativo. Este paso de cuantización es adaptativo porque se va incrementando o decrementando en función de la magnitud de las diferencias previamente codificadas. Los codificadores ADPCM se aprovechan del hecho de que las muestras de sonido vecinas son generalmente muy similares entre ellas.



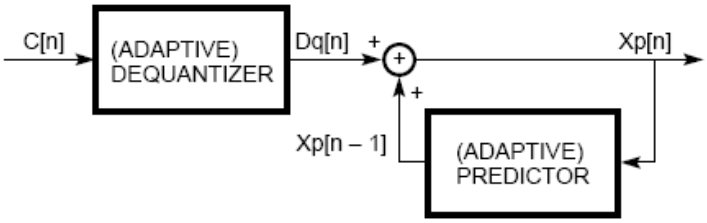


Figura 6 - Codificación ADPCM [16]

Este codificador de tipo ADPCM fue desarrollado por la extinta Asociación de Multimedia Interactiva (*Interactive Multimedia Association*, IMA). Esta versión de ADPCM, que a veces también es denominada como Intel DVI ADPCM, es capaz de comprimir muestras usando 4 bits por muestra, con lo que el nivel de compresión alcanzado será de 4:1 teniendo en cuenta la codificacion de muestras de 16 bits. IMA tenía como objetivo desarrollar un compresor de audio de dominio publico de buena calidad, con un factor de compresión igualmente bueno y lo suficientemente sencillo para que el algoritmo de codificacion fuese unicamente software y en tiempo real.

Algoritmo:

Como se ha comentado previamente este algoritmo realiza una codificación diferencial respecto a la muestra anterior, pero usando una cuantización adaptativa, de forma que si el resultado de cuantizar las muestras anteriores ha dado un valor muy grande, se aumenta el paso de cuantización, y si ha sido más bien pequeño se disminuye. Para los posibles pasos de cuantización a aplicar se dispone de una tabla (cuyos pasos se incrementan siguiendo una escala logarítmica) [16].

Para la compresión de una muestra m codificada en formato PCM (es decir, sin compresión) se parte de uno de los pasos de esta tabla (en concreto de aquel indexado por la variable global Index), y de la muestra que se codificó en el paso anterior, tal y como la leerá el decodificador (variable global signed short PredictedValue). Por tanto, lo primero que se calcula es la diferencia (delta, Δ) entre esta muestra y la anterior.

$$\Delta = m - \text{PredictedValue}$$

Siendo este valor de delta lo que vamos a codificar con el paso de cuantización (*Step Size*) que nos indique la tabla anterior.

A continuación vamos a ver exactamente cómo se codifica esta diferencia usando los cuatro bits de ADPCM (del bit 3 al bit 0):

- El bit 3 tiene un tratamiento especial ya que indica el signo de la diferencia. Se codifica 1 para signo negativo, 0 en otro caso. Además, para continuar con la codificación, se aplicará valor absoluto a la diferencia delta: $\Delta = |\Delta|$

- A continuación, el bit 2 se codifica con un 1, si $\Delta >$ *Step Size* . En ese caso, Se actualiza delta como sigue: $\Delta = \Delta -$ *Step Size*

- Ahora, el bit 1 se codifica con un 1, si $\Delta >$ *Step Size/2* . En ese caso, también se actualiza delta como sigue: $\Delta = \Delta -$ *Step Size/2*

- Por último, el bit 0 se codifica con un 1, si $\Delta >$ *Step Size/4*

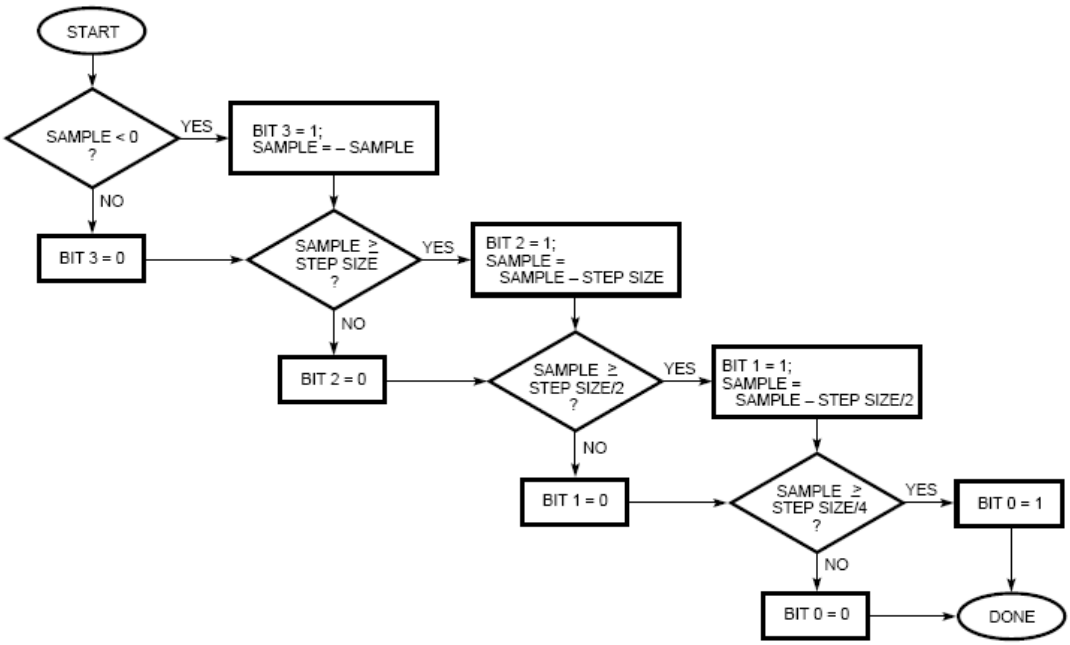


Figura 7 - Codificación IMA ADPCM [16]

A continuación se decodifica el valor que se acaba de codificar para asignarse a la variable global *PredictedValue,* de forma que la próxima muestra se pueda codificar diferencialmente a partir de este valor. Observar que usamos el valor de la muestra tal y como lo leerá el decodificador (es decir, después de la cuantización y la decuantización) y no la muestra original. Esto se hace así porque el decodificador no dispone de los valores de estas muestras originales sino tan sólo de su decodificación. Por último, hay que realizar la adaptación de la cuantización. Para esto se debe de actualizar el índice del paso de cuantización (la variable global *Index*) según el resultado de la codificación ADPCM que se acaba de realizar. Para esta actualización se usaran los tres bits de menor peso del resultado de la muestra codificada como índice a otra tabla que se usa para incrementar o decrementar la variable global *Index*.

First Table Lookup for the IMA ADPCM Quantizer Adaptation

Three Bits Quantized Magnitude	Index Adjustment
000	-1
001	-1
010	-1
011	-1
100	2
101	4
110	6
111	8

Tabla 1 – First Table lookup for the IMA ADPCM quantizer adaptation [16]

Como se puede observar, resultados entre 0 y 3 provocan que el índice del paso de cuantización (*Index*) se decremente en uno, mientras que un resultado de 4, 5, 6 ó 7 hace que este índice aumente en 2, 4, 6 u 8 respectivamente (lo que resultará en aumentos en el paso de cuantización mucho mayores). De esta forma es como se adapta el paso de cuantización al resultado de previas codificaciones.

Algo a tener en cuenta es que la tabla sólo contiene 89 valores, y por tanto la variable *Index* debe de mantenerse siempre dentro del rango de 0 a 88.

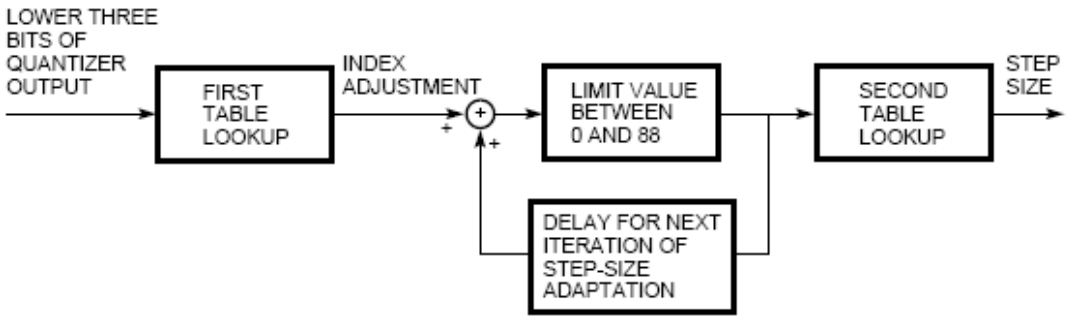


Figura 8 - Cuantificación IMA ADPCM [16]

Es importante destacar que todas las operaciones que se realizan en la codificación IMA ADPCM son siempre enteras, para lo que bastará con usar tipos de datos enteros (o sea, operandos enteros).

2.5 Tecnología VoIP

2.5.1 Introducción

La telefonía IP tambien llamada Voz sobre IP (VoIP) se puede definir como la transmisión de paquetes de voz utilizando redes de datos. La comunicación se realiza por medio del protocolo IP (Internet Protocol), permitiendo establecer llamadas de voz y fax sobre conexiones IP, obteniendo de esta manera una reducción de costos considerable en telefonía. Para que los datos multimedia circulen por la red IP primero hay que encapsularlos y esto se consigue digitalizando la voz. Como Internet no es una red orientada a la comunicación en tiempo real, la voz sobre IP plantea un reto importante: conseguir digitalizar la voz y transmitirla en tiempo real. Esto se consigue gracias a los protocolos de transporte en tiempo real, RTP y RTCP, combinados con el protocolo de transporte no fiable, UDP. 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.

2.5.2 Características principales

La integración de la voz en redes IP mediante la tecnología VoIP aporta múltiples ventajas [17]:

- 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, WLAN,..) 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.

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 el retardo 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 [17]:

• 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

2.5.3 Protocolos

Vamos a describir los dos protocolos implicados en este proyecto: SIP y H.323. Existen otros muchos protocolos de señalización sobre VoIP como IAX (Inter Asterisk eXchange) y MGCP (Media Gateway Control Protocol).

- **SIP**: Session Initiation Protocol (SIP o Protocolo de Inicialización de Sesiones) es un protocolo de señalización simple, utilizado para telefonía y videoconferencia por Internet. Basado en el Protocolo de Transporte de correo simple (SMTP) y en el Protocolo de Transferencia Hipertexto (HTTP) fue desarrollado por el IETF MMUSIC Working Group con la intención de ser el estándar para la iniciación, modificación y finalización de sesiones interactivas de usuario donde intervienen elementos multimedia como el vídeo, voz, mensajería instantánea, juegos online y realidad virtual. SIP es uno de los protocolos de señalización para voz sobre IP, acompañado por H.323. SIP es definido completamente en la RFC 2543 y en la RFC 3261. SIP es un protocolo de la capa de aplicación independiente de los protocolos de paquetes subadyacentes (TCP, UDP, ATM, X.25). SIP esta basado en una arquitectura cliente servidor en la cual los clientes inician las llamadas y los servidores responden las llamadas. Es un protocolo abierto basado en estándares, ampliamente soportado y no es dependiente de un solo fabricante de equipos [18].

- **H.323**: H323 es un estándar de la UIT-T que define los protocolos necesarios para establecer una comunicación audiovisual a través de redes no orientadas a conexión, como es el caso de las redes IP. Este estándar es compatible con los sistemas tradicionales de telefonía e incluso permite la señalización extremo a extremo conectados a redes distintas (recomendaciones H.320 y H.324) [19]. La primera versión del protocolo H.323 se publicó en 1996 para permitir comunicaciones multimedia en redes IP de área local y hasta 2009 se han publicado 7 versiones que se han ido adaptando a las nuevas exigencias del mercado. El sistema H.323 puede constar de cuatro componentes [20]:

  • **Terminal**: elemento hardware (teléfono IP) o software (softphone) que permite la comunicación H.323.

  • **Gateway**: componente encargado de conectar redes o protocolos distintos entre sí para establecer la comunicación.

  • **MCU** (Multipoint Control Unit): equipo que se encarga se las conferencias dónde hay más de dos usuarios. Se compone de un Multipoint Processor, que se encarga de mezclar los distintos flujos multimedia, y un Multipoint Controller, encargado de la señalización y control de los terminales.

  • **Gatekeeper**: es el equipo que hace de centralita telefónica y que proporciona el acceso al servicio H.323. Todos los gateways, terminales y MCU los gestiona el gatekeeper, formando una zona por cada gatekeeper.

2.6 Softphone Ekiga

Ekiga es una aplicación software de videoconferencia. Ésta nos permite escuchar y ver a otro usuario gratuitamente usando un ordenador o smart phone con conexión a internet. Ademas permite chatear con los otros usuarios independientemente de que estén en línea o no. Este software está disponible tanto en GNU/Linux como en Windows [21].

Más específicamente, Ekiga es una aplicación VoIP que permite hacer llamadas de audio y vídeo a usuarios remotos mediante los protocolos SIP y H.323. Ekiga soporta varios codecs de audio y vídeo y todas las necesidades requeridas por los dos protocolos anteriormente mencionados, siendo la primera aplicación de codigo abierto que soporta ambos dos.

Actualmente, en su versión 3.2, Ekiga soporta los siguiente codecs de audio: G.711-Alaw, G.711-uLaw, Speex (NarrowBand and WideBand), G.722 (WideBand), iLBC, GSM-06.10, MS-GSM, G.726, G.721, CELT ultra-low delay (32 kHz or 48 kHz); y los siguientes codecs de video: THEORA, H.264, H.263, H.263+, H.261, MPEG4.

3. DISEÑO Y DESARROLLO

3.1 Selección del codificador

Para la selección de un codificador se establecieron varios requisitos [1]:

- Que no fuese un codec paramétrico, sino de forma de onda, pues no existen codecs paramétricos pensados para la transmisión de audio cardiorrespiratorio (es de esperar que la extracción de parámetros de una señal de voz, sea sensiblemente distinta a la que sería necesaria para capturar lo imprescindible de una señal cardiorrespiratoria).

- Que no tuviese licencias de uso, ni patentes.

- Que estuviese implementado en alguna de las herramientas de software libre disponibles para GNU/Linux

- Que fuese una alternativa a las actuales investigaciones de codificación de señales biomédicas mediante el uso de wavelets (sección 2.3), ya que este estudio se estaba realizando paralelamente para la Fundación EHAS a través de un Proyecto Fin de Máster en la Universidad Técnica Particular de Loja, Ecuador.

- Que tuviese un bajo coste computacional para ser aplicable en tiempo real.

- Que respetase el ancho de banda de las señales cardiorrespiratorias.

En los primeros meses de 2009, cuando el equipo de desarrollo de EHAS estaba evaluando el codec a utilizar, expiró la patente del codec G.722, cuyas características eran prometedoras. El nuevo codec realizaba una codificación diferencial que permitía, con el mismo número de bits/muestra, obtener en la

práctica un nivel de resolución muy superior al codec G.711 utilizado anteriormente. Lo que se codificaba no era el valor de la amplitud, sino la diferencia entre esta amplitud y la que habia sido predicha por un predictor lineal. En poco tiempo la comunidad del software libre realizó buenas implementaciones del codec y las versiones 1.6 de Asterisk y 3.2 de Ekiga incluyeron las implementaciones del nuevo codec, por lo que finalmente el equipo de desarrollo escogió el codificador de voz G.722 para la transmisión de las señales cardiorespitatorias [1].

Por lo tanto, nuestro primer paso fue evaluar el codificador G.722 utilizado actualmente en el sistema de teleestetoscopia ya establecido. Tras los resultados de este codificador se exploraron las virtudes de otros codificadores de forma de onda, en concreto el G.726 en sus cuatro modos 16, 24, 32 y 40 kbit/s y el codec IMA-ADPCM. Se compararon los resultados en busca de una mejora, tanto en calidad como en factor de compresión.

3.2 Contexto del codificador (Sistema teleestetoscopia EHAS-Fundatel)

En esta sección explicaremos las diferentes transformaciones que sufre el audio cardiorrespiratorio recogido por el estetoscopio digital EHAS/Fundatel y en qué parte de esa cadena de transformaciones y codificaciones/decodificaciones se encuentra nuestro codificador software.

Como se cita en [1] el sonido analógico cardiorrespiratorio captado por la campana del estetoscopio digital EHAS-Fundatel Bluetooth es recogido por un micrófono, el cual convierte el movimiento mecánico de las ondas de presión sonora en una variación de voltaje:

1. Este voltaje se transmite, tras atravesar una etapa de amplificación analógica, hasta un codec hardware TLV320AIC33. El codec hardware, en su primera etapa, realiza la conversión de la señal analógica emitida por el microfono, a una señal digital submuestreada que es posible procesar por otros dispositivos electrónicos. Esta señal queda submuestreada a 8kHz y 16 bits por muestra.

2. La señal resultante se envía hacia un chip Bluetooth que manda, empleando el perfil Serial Port Profile (SPP) del protocolo Bluetooth, el audio hacia un PC. Este audio se envía codificado en un formato que adapta el sonido a las caracteríticas de la transmisión Bluetooth. En esta comunicación se envía también información de control de flujo que permite al PC asegurar que el audio que está recibiendo no ha sufrido alteraciones provocadas por una mala gestión de la velocidad de transmisión/recepción, así como la información necesaria para permitir que el audio pueda ser reconstruido.

3. Cuando el sonido digital alcanza el PC, reside en éste un Software Estetoscopia desarrollado por EHAS que realiza una codificación previa antes de mandarlo a una aplicación VoIP (Softphone Ekiga). El objetivo es permitir la compatibilidad con cualquier servidor de sonido, funcionando a cualquier frecuencia de muestreo y ajustando la tasa de envío de datos a 44.100 muestras por segundo y 32 bits por muestra (que es el caso mas frecuente).

4. Una vez que la señal llega al softphone Ekiga, éste adapta de nuevo este audio digital a las necesidades de entrada del codec de audio utilizado (en nuestro caso de ejemplo se utiliza el G.722, que obliga a tener una entrada con 16.000 muestras por segundo y 14 bits por muestra, ofreciendo una salida de 64 Kbps).

5. Tras ser enviado el audio final digitalizado a la red, éste es recibido en el ordenador remoto, produciendose el proceso inverso (decodificación).

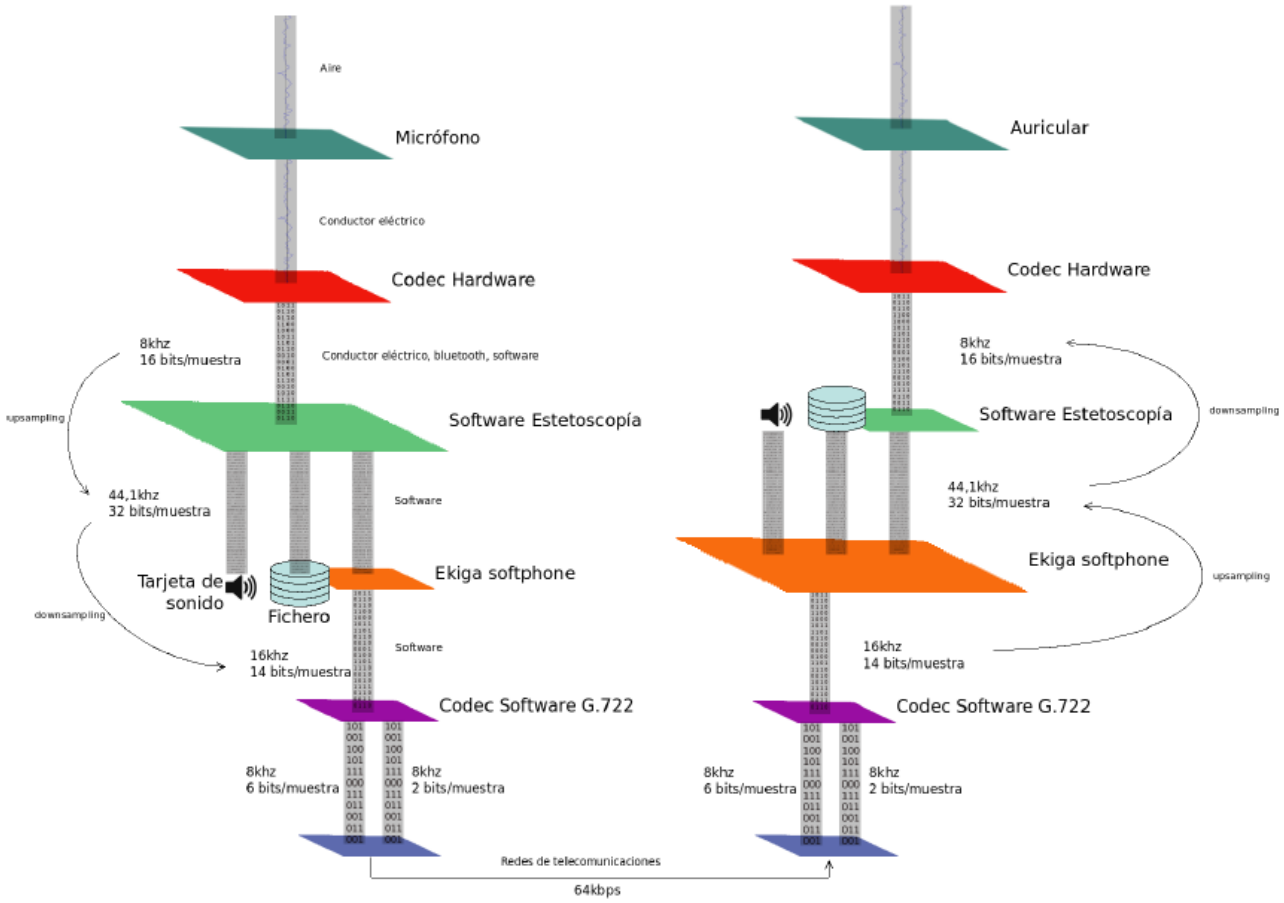


Figura 9 - Esquema del Sistema Teleestetoscopia EHAS-Fundatel [1]

3.2 Creación de scripts para automatizar las pruebas de los codecs

3.2.1 Introducción

Con el objetivo de buscar una alternativa al codificador G.722 se realizaron simulaciones, con diferentes codificadores, de las dos últimas etapas del sistema EHAS-Fundatel, tomando como audio fuente el audio de salida del Software Estetoscopia (punto donde entraría nuestro codificador). Se utilizaron 33 muestras de audios cardiorrespiratorios, 15 de ellas obtenidas con el estetoscopio bluetooth EHAS-Fundatel (3 de tipo mitral, 3 aórtica, 3 tricúspide, 6 pulmonares), y 18 descargadas de [22] (10 cardiacas, 8 respiratorias). Dado el gran número de audios a analizar se desarrollaron una serie de scripts de Linux para automatizar las pruebas con los distintos codificadores, y no tener que ir cofidicando uno, por uno, cada

muestra de audio cardiorrespiratorio. Al ser nuestra fuente de audio la salida del Software Estetoscopia las muestras EHAS-Fundatel tendrán un tamaño de 32 bits/muestra y estarán muestreadas a 44.1 kHz, mientras que los sonidos cardiacos de [22] están digitalizadas a 16khz y 16 bits/muestra y las respiratoria a 11.025 Khz y 16 bits/muestras. Los audios se encuentran en formato '.wav' y se decodificaran con el mismo formato, tras pasar por formato '.raw' en algunas de las etapas. Una vez realizado todo el proceso de codificación/decodificación, la señal de audio recuperará sus características iniciales, siendo así posible la comparación entre las muestras de audio codificadas y decodificadas, observando los errores producidos a causa de una codificación con pérdidas. Para realizar la simulación nos apoyamos en el programa de manipulación de audio SoX (Sound eXchange) [23] y los códigos fuente programados en C de los codecs G.722, G.726 y G.711 obtenidos de [24]. SoX es capaz de leer y escribir archivos de audio en los formatos más populares y puede opcionalmente aplicar efecto a ellos. También puede combinar multiples fuentes de entrada, sintetizar audio, y, en varios sistemas, actuar como reproductor y grabador de audio con múltiples canales.

3.2.2 Desarrollo

Inicialmente se realizaron pruebas con los codificadores G.722, G.726 e IMA-ADPCM, y submuestreos a 1, 2 y 4 kHz. Para ello se crearon 3 scripts que realizaban las codificacaciones a G.722 y G.726, 3 scripts para la codificación a IMA-ADPCM y otros 3 para los submuestreos a 1, 2 y 4 kHz. La razón de crear grupos formados por 3 scripts se debe a que las características de la fuente de audio no eran homogeneas. El primero de los scripts de cada grupo codifica los audios EHAS-Fundatel, otro los audios cardiacos y el último los audios respiratorios.

Tras observar los resultados que pueden verse en el ANEXO C, se optó por diseñar un codec compuesto por una codificador IMA ADPCM con un submuestreo previo a 4 kHz, asegurándonos de esta forma respetar el ancho de banda de las señales cardiorrespiratorias. A esta codificación se la llamará a partir de ahora Codec-Estetoscopia Propuesto. Siguiendo la estructura anterior se crearon otros 3 nuevos scripts en los que se realizaba la codificación y decodificación del Codec-Estetoscopia Propuesto.

3.2.3 Estructura de los Scrips

- Petición por pantalla del nombre del audio en formato .wav a codificar.

- Adaptación de la fuente de audio a la entrada requerida por los codificadores. En todos los casos se adapta la entrada a 16 kHz y 16 bits/muestra, incluyendo las codificaciones por submuestreo e interpolación, aunque en este último caso este aspecto no influya.

- Codificación/Decodificación del audio de entrada al codificador. En esta etapa se utiliza el comando **‘ time –f ’** en la línea de código en la que se produce la codificación. Este comando permite obtener el tiempo total de ejecución de un programa. Este valor es guardado en el archivo **tiempos.txt**.

- Readaptación del audio decodificado a las características del audio fuente.

- Mediante el comando **‘cksum’** se guarda en el archivo **tamaños.txt** el tamaño de el audio original, el adaptado y el codificado.

- Conversión del audio fuente y el audio decodificado en formato .wav a formato .dat, y recolección de las características principales de ambos audios gracias al comando **‘stats’** perteneciente al software SoX. Estos valores son guardados en los archivos **original.txt** y **decodificado.txt** respectivamente. Los archivos resultantes en formato .dat están compuestos por dos columnas, conteniendo la primera de ellas los tiempos para cada muestra, y la columna restante los valores de éstas. Los archivos .dat generados a partir del audio fuente y audio decodificado sirven de entrada al programa **rms.c** desarrollado. Este programa resta los valores de ambos archivos generando un nuevo archivo resta.dat a partir del cual el programa calcula dos figuras de mérito, Root Mean Square Error (ERMS) y Percent RMS Difference (PRD). Estos dos parámetros son guardados en el archivo **rms.dat**.

Finalmente, apoyandonos en los archivos generados podremos calcular diferentes figuras de mérito, las cuales nos permitirán la comparación objetiva de los diferentes codificadores. Estas figuras son:

- Factor de Compresión (CF). Calculado a partir de los datos recogidos en **tamaños.txt**.

$$CF = \frac{source\_data\_size}{encoded\_data\_size} \qquad (2)$$

- Factor en tiempo Real (RTF). Calculado a partir de **tiempos.txt** y la duración del audio original.

$$RTF = \frac{computation\_time}{audio\_duration} \qquad (3)$$

- Root Mean Square Error (RMSE) y Percent RMS Difference (PRD). Valores guardados en rms.dat. Siendo $x[n]$ la señal original, $\tilde{x}[n]$ la señal obtenida tras la codificación y decodificación del audio, y N el número de muestras del audio:

$$RMSE = \sqrt{\frac{\sum_{n=1}^{N}\left(x[n]-\tilde{x}[n]\right)^2}{N}} \qquad (4)$$

$$PRD = \sqrt{\frac{\sum_{n=1}^{N}\left(x[n]-\tilde{x}[n]\right)^2}{\sum_{n=1}^{N}x^2[n]}} \times 100\% \qquad (5)$$

- Tasa Binaria (R). Figura de mérito característica de cada codificador. Se calcula multiplicando la frecuencia de muestreo por el número de bits por muestra.

3.3 Programación en C del Codec-Estetoscopia-Propuesto

3.3.1 Introducción

Tras la generación de los scripts se decidió programar en lenguaje C el Codec-Estetoscopia-Propuesto, que vendría a ser nuestra solución para la codificación de audios cardiorrespiratorios partiendo de la salida del Software Estetoscopia. Este programa sería el paso previo a la integración de este codificador en el softphone Ekiga. Para facilitar el trabajo de programación se descargaron las librerías en C libsamplerate-0.1.8 [25] y libsndfile-1.0.25 [26]. Libsamplerate permite aplicar submuestreos e interpolaciones a audios, mientras que la librería libsndfile permite leer y escribir archivos de audio con datos en formato big-endian o little-endian, y aplicar a estos audios una larga lista de codificaciones. Ambas librerías están bajo la licencia GNU General Public License, por lo que se nos permite la libre utilización de éstas.

3.3.2 Estructura

El programa esta formado por 12 archivos: 7 archivos fuente y 5 archivos de cabecera.

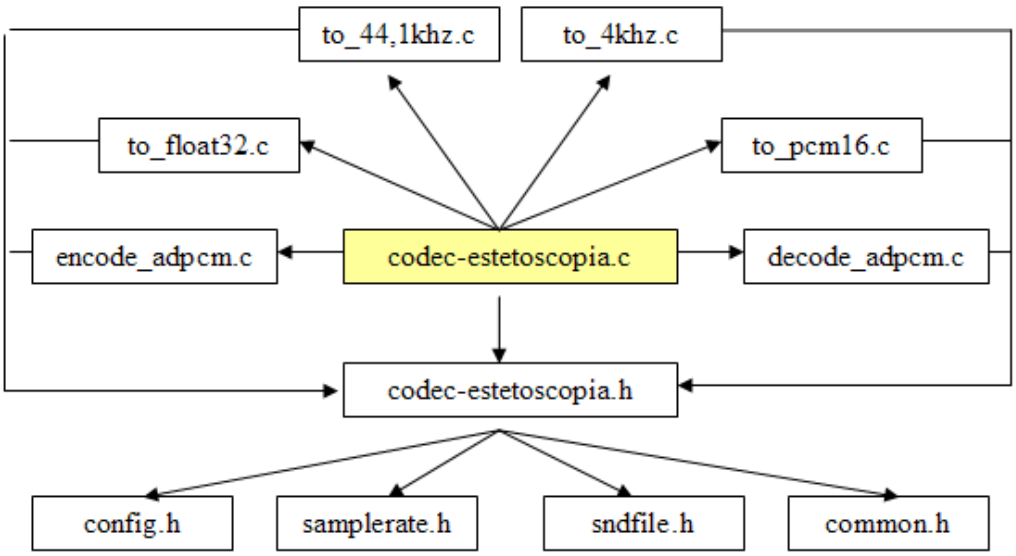


Figura 10 - Dependencias del Programa codec-estetoscopia

codec-estetoscopia.c: recibe como argumento el nombre de un archivo de audio en formato .wav, y va llamando en serie al resto de funciones. Estas funciones reciben por referencia el nombre del audio generado por su función predecesora, modificando sus características, generando un nuevo audio y renombrándolo. Este audio será pasado a su vez a la siguiente función hasta llegar al último paso de la codificación/decodificación del sistema de teleEstetoscopia EHAS-FUNDATEL. A continuación se muestran las funciones ordenadas en función del orden de llamada:

1. **to_4khz.c**: realiza un submuestreo del audio fuente a 4 kHz.
2. **to_pcm16.c**: modifica la codificacion del audio, pasando de una codificacion de 32 bits/muestra a una PCM 16 bits/muestra.
3. **encode_adpcm.c**: codificacion del audio a IMA-ADPCM.
4. **decode_adpcm.c**: decodificacion del audio a PCM 16 bits/muestra.
5. **to_float32.c**: modifica la codificación de PCM 16 bits/muestra a 32 bits/muestra.
6. **to_44,1khz.c**: realiza una interpolación a 44,1 kHz del audio recibido generando en audio final decodificado.

3.4 Integración del Codec-Estetoscopia-Propuesto en Softphone Ekiga

3.4.1 Introducción

Tras la programación en C del Codec-Estetoscopia-Propuesto, se comenzó a estudiar el funcionamiento del softphone Ekiga. El objetivo era integrar el codificador en el software, para su posterior selección en el menú de codificadores disponibles.

3.4.2 Instalación del Softphone Ekiga en Linux

Ekiga depende de dos librerías PTLIB [27] y OPAL [28], por lo que habrá que instalarlas previamente. Es necesario compilar primero PTLIB y después OPAL, ya que OPAL mantiene dependencias con PTLIB. En nuestro caso fueron instaladas las siguientes versiones: ptlib-2.6-7, opal-3.6.8 y ekiga-3.2.7; sobre la versión 10.10 de Ubuntu. Antes de comenzar la compilación y posterior instalación de estas librerías hay que comprobar que se cumplen todas las dependencias.

Dependecias obligatorias y opcionales:

- PTLIB: autotools-dev, doxygen, pkg-config, bison, flex, libssl-dev, libldap2-dev, libsasl2-dev, libkrb5-dev, libexpat1-dev, libsdl1.2-dev, libdv4-dev, libv4l-dev, libasound2-dev.
- OPAL: doxygen, autotools-dev, pkg-config, libpt-dev, libcelt-dev, libtheora-dev, libgsm1-dev, libspeex-dev, libspeexdsp-dev, libcapi20-dev.
- EKIGA: libsasl2-dev, gettext, libldap2-dev, libpt-dev, libopal-dev, libgconf2-dev, libxv-dev, autotools-dev, gnome-pkg-tools, rarian-compat, intltool, libxml-parser-perl, evolution-data-server-dev, gnome-doc-utils, libavahi-client-dev, libavahi-glib-dev, libgtk2.0-dev, libdbus-glib-1-dev, libnotify-dev, libebook1.2-dev, libxext-d, libsigc++-2.0-dev.

Instalaciones:

- PTLIB: Por defecto esta librería se instalará en el directorio /usr/local. Con el objetivo de que otras librerías como OPAL localicen PTLIB, el directorio /usr/local/lib/pkconfig debe ser añadido al camino de búsqueda de pkg-config. Esto se consigue actualizando la variable PKG_CONFIG_PATH. También será necesario actualizar la variable LD_LIBRARY_PATH de tal forma que los programas que lo requieran puedan encontrar la librería PTLIB.

  ```
  export PKG_CONFIF_PATH=/usr/local/lib/pkgconfig
  export LD_LIBRARY_PATH=/usr/local/lib
  ```

  Finalmente, situandonos en el directorio donde hayamos guardado la librería, ejecutamos los siguientes comandos:

  ```
  ./configure --prefix=/usr
  make
  ```

```
make install
```

- OPAL y EKIGA: Ejecutamos los siguientes comandos una vez situados en el directorio donde hayamos guardado las librerías.

```
./configure --prefix=/usr
make
make install
```

3.4.3 Integración del Codec-Estetoscopia-Propuesto

En esta librería existe un subdirectorio (~/plugins/audio) donde se encuentran los diferentes codificadores desarrollados para su uso. Hasta hace poco, añadir un nuevo codificador de audio a la librería OPAL requería modificar y recompilar el código fuente de la librería, exigiendose de esta forma un detallado conocimiento de C++ y de las capacidades del protocolo H.323. Desde la versión 1.14 de esta librería se permiten cargar codificadores de audio desde archivos binarios .so. Esto quiere decir que añadir nuevos codificadores en OPAL es más fácil que con el antiguo y monolítico diseño [29]. Lamentable existe muy poca documentación sobre este sistema de carga de nuevos codificadores, por lo que tras varios intentos fallidos se decidió añadir el codificador de forma estática, adentrándonos en el código fuente de la librería y modificando ésta.

Al explorar la subcarpeta ~/plugins/audio, descubrimos que se había desarrollado un codificador IMA-ADPCM para esta librería (ima_adpcm.c) , por lo que se decidió partir de este codificador para la generación del Codec-Estetoscopia-Propuesto, y no del codificador en C codec-estetoscopia.c previamente desarrollado por nosotros. Se tomó esta decisión ya que en el archivo ima_adpcm.c el codificador IMA ADPCM ya se encontraba adaptado a las exigencias de la librería OPAL. En principio solo se tendría que cambiar la frecuencia a la que trabaja el codificador, pasando de los 8 kHz a los 4 kHz exigidos por nuestro codec. Esto nos ahorraría el trabajo de adaptación del codificador a las estructuras de la librería, y la transformación de éste a un codificador en tiempo real.

El primer paso fue crear una nueva carpeta llamada Codec-Estetoscopia dentro del subdirectorio ~/plugins/audio. Esta carpeta contenía el codificador ima_adpcm.c renombrado a codec_estetoscopia.c y un makefile que permitía la compilación de éste

y la generación del plugin .so requerido. Tras generarse el plugin codec_estetoscopia_audio_pwplugin.so, éste fue guardado en el directorio /usr/local/lib/opal-3.6.8/codecs/audio, directorio de donde Ekiga lee los codificadores disponibles. El problema fue que este codificador no es soportado por Ekiga tal y como se especifica en la sección 2.6, por lo que este codificador no apareció en el menú de selección de codificadores de audio. Por lo tanto, como se comentó anteriormente, habría que añadir el codificador de forma estática, adentrándose en el código fuente de la librería y modificando ésta.

El objetivo era que previamente fuese reconocido por Ekiga el codec IMA-ADPCM, transformando posteriormente el archivo ima_adpcm.c a un codec_estetoscopia.c actualizado en función de los cambios realizados en la librería, generándose de nuevo el plugin codec_estetoscopia_audio_pwplugin.so.

A continuación se listan los archivos de la librería opal-3.6.8 donde se realizaron modificaciones y cuáles fueron éstas:

h323caps.cxx (~/src/h323): Se añade un nuevo miembro a la clase H245_AudioMode.

```
PBoolean H323AudioCapability::OnSendingPDU(H245_AudioMode & pdu) const
{
  static const H245_AudioMode::Choices AudioTable[] = {
    ...
    H245_AudioMode::e_ima_adpcm,
    ...
  };

  unsigned subType = GetSubType();
  if (subType >= PARRAYSIZE(AudioTable))
    return PFalse;

  pdu.SetTag(AudioTable[subType]);
  return PTrue;
}
```

opalplugin.h (~/include/codec): Se añade una nueva capacidad para H.323 creando un nuevo tipo de audio codec plugin.

```
enum {
  ...
  PluginCodec_H323AudioCodec_ima_adpcm,
  ...
};
```

h245.h (~/include/asn): Se añade una nueva variable publica a la clase H245_AudioCapability y H245_AudioMode.

```
class H245_AudioCapability : public PASN_Choice
{
#ifndef PASN_LEANANDMEAN
    PCLASSINFO(H245_AudioCapability, PASN_Choice);
#endif
  public:
    H245_AudioCapability(unsigned tag = 0, TagClass tagClass =
UniversalTagClass);

    enum Choices {
      ...
      e_ima_adpcm,
      ...
    };

class H245_AudioMode : public PASN_Choice
{
#ifndef PASN_LEANANDMEAN
    PCLASSINFO(H245_AudioMode, PASN_Choice);
#endif
  public:
    H245_AudioMode(unsigned tag = 0, TagClass tagClass =
UniversalTagClass);

    enum Choices {
      ...
      e_ima_adpcm,
      ...
    };
```

opalpluginmgr.cxx (~/src/codec): Asociamos el nuevo tipo de audio codec plugin con un nuevo miembro para la clase H245_AudioCapability.

```
static H323CodecPluginCapabilityMapEntry audioMaps[] = {

...

{PluginCodec_H323AudioCodec_ima_adpcm,
H245_AudioCapability::e_ima_adpcm },

...

};
```

mediafmt.h (~/include/opal):

```
#define OPAL_IMA_ADPCM  "IMA_ADPCM"
extern const OpalAudioFormat & GetOpalIMA_ADPCM();

#define OpalIMA_ADPCM   GetOpalIMA_ADPCM()
```

codec_estetoscopiammf.cxx (src/codec): generación del archivo (ANEXO C).

makefile (~/): Modificamos el makefile para que tenga en cuenta el nuevo archivo fuente codec_estetoscopiammf.cxx

```
SOURCES += $(OPAL_SRCDIR)/codec/codec_estetoscopiammf.cxx \
```

makefile (~/plugins): Añadimos el nuevo subdirectorio creado para el nuevo codificador.

```
SSUBDIRS = audio/Codec-Estetoscopia
```

Una vez integrado estáticamente el codificador IMA-ADPCM se modificó el archivo codec_estetoscopia.c generado previamente (ANEXO C).

codec_estetoscopia.c (~/plugins/audio/Codec-Estetoscopia):

- Definimos el Payload Type de la codificación. Este valor identifica el formato del data que sigue después del encabezado RTP. En este caso le damos el valor 5 correspondiente a DIV4 audio (8kHz), payload de la codificación IMA-ADPCM, al ser el más parecido a nuestra codificación: `#define PAYLOAD_CODE 5`

- Eliminación de la estructura `PluginCodec_H323NonStandardCodecData`, al no ser ya necesaria al habernos creado previamente el nuevo tipo de audio codec plugin `PluginCodec_H323AudioCodec_ima_adpcm` en el archivo opalplugin.h;

  ~~`static struct PluginCodec_H323NonStandardCodecData imaADPCM_Cap`~~

  y la eliminación de la función `imaCompareFunc` dependiente de ella

  ~~`int imaCompareFunc(struct PluginCodec_H323NonStandardCodecData * data)`~~

- Modificación de la estructura `PluginCodec_Definition`, modificando las variables: `sampleRate` (sustituyendo 8000 por 4000 muestras/segundo), `rtpPayload` (añadimos el Payload_Code anteriormente definido),

h323CapabilityData (lo eliminamos), h323CapabilityType (sustituimos `PluginCodec_H323 NonStandardCodecData`), y `sdpFormat`.

```
static struct PluginCodec_Definition imaADPCMCodecDefn[] =
{
  {
...

    4000,                                   // samples per second
...
    PAYLOAD_CODE,                           // no IANA RTP
                                            payload code
    "DVI4_8k",                              // RTP payload name
...
    PluginCodec_H323AudioCodec_ima_adpcm,  // h323CapabilityType
    NULL                                   // h323CapabilityData
  },
```

Tras recompilar e instalar de nuevo la librería modificada podremos abrir softphone Ekiga y comprobar cómo este programa ya reconoce nuestro codificador. Para ello, en el menú del programa, seleccionamos Editar-> Preferencias-> Sonido-> Códecs y seleccionamos el nuevo codec como codificador de audio.

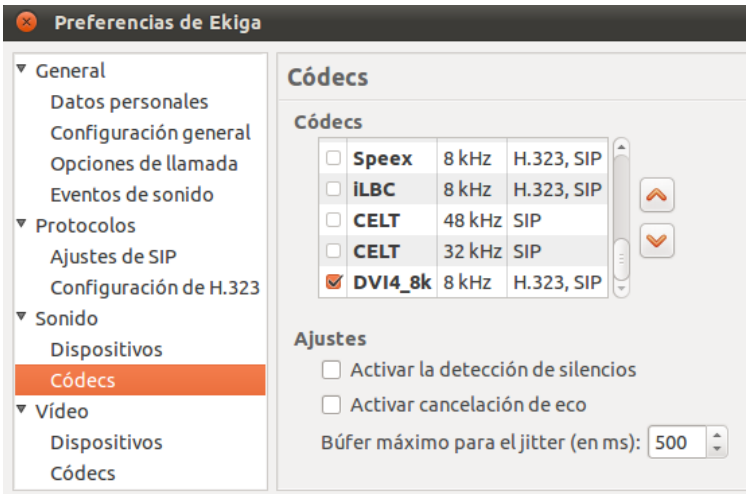


Figura 11 - Selección del tipo de Codificador de Audio

4. TEST Y PRUEBAS

4.1 Resultados objetivos

A continuación se muestran los resultados obtenidos al ejecutar los scripts previamente programados. Como se comentó en la sección 3.2.3 los codificadores son comparados de manera objetiva a través de las siguientes figuras de mérito: Factor de Compresión (CF), Tasa binaria (R), Root Mean Square Error (RMSE), Percent RMS Difference (PRD) y Factor en Tiempo Real (RTF).

En el ANEXO C se muestran los resultados obtenidos de los audios adquiridos con el estetoscopio EHAS-FUNDATEL y los audios cardiados y respiratorios descargados de [22] :

- **Audios EHAS-Fundatel**: se encuentran muestrados a 44,1 kHz y 32 bits por muestra y tienen una duración de 5 segundos y un tamaño de 1048634 bytes.

- **Audios Cardiacos**: se encuentran muestreados a 16 kHz y 16 bits por muestra y tienen una duración de 122 segundos y un tamaño de 3925516 bytes.

- **Audios Respiratorios**: se encuentran muestrados a 11,025 kHz y 16 bits por muestra y tienen una duración de 10 segundos y un tamaño de 222744 bytes.

Una vez extraidos los resultados de todas las muestras de audio se realizó una tabla comparativa de cada codificación. La tasa binaria (R), como se comentó en la sección 3.2.3 se calcula a partir de la frecuencia de muestreo y el número de bits por muestra. Los submuestreos a 4, 2 y 1 kHz se realizan con 16 bits/muestra:

- G.722: 8 kHz * 8 bits/muestra = 64 Kbps
- G.726-2: 8 kHz * 2 bits/muestra = 16 Kbps
- G.726-3: 8 kHz * 3 bits/muestra = 24 Kbps
- G.726-4: 8 kHz * 4 bits/muestra = 32 Kbps
- G.726-5: 8 kHz * 5 bits/muestra = 40 Kbps
- IMA-ADPCM: 16 kHz * 4 bits/muestra = 64 Kbps

	CF	R (Kbps)	RMSE	PRD (%)	RTF
G.722	4	64	0,039405267	22,7856303	0,2122
G.726-2	16	16	0,004458867	3,22418553	0,03503
G.726-3	10.66	24	0,002614943	1,93012648	0,03636
G.726-4	8	32	0,002327377	1,71252903	0,03420
G.726-5	6.4	40	0,003868983	2,84869895	0,03633
4 kHz	4	64	0,000347993	0,29957613	0,00448
2 kHz	8	32	0,002195643	1,98224353	0,00446
1 kHz	16	16	0,010389677	5,048618	0,00441
IMA-ADPCM	4	64	0,000396343	1,93784919	0,00613

Tabla 2 - Comparativa entre codificaciones en función de las figuras de mérito

Observando la Tabla 2, se aprecia que G.726 da buenos resultados objetivos, pero tenía el problema de necesitar un audio de entrada previamente codificado a ley-μ con G.711 y además el audio decodificado parecía tener peor calidad subjetiva. G.711 es un codec sencillo tipo PCM, con una mínima carga de procesamiento para la CPU del PC, y su uso no esta sometido al pago de ninguna licencia. Sin embargo la calidad del audio recibido es muy dependiente del nivel de presión que se aplique a la campana de captación del teleestetoscopio. Este codec emplea una cuantificación de 8 bits no uniforme, por lo que las señales de baja amplitud sufren menos distorsión que las que poseen gran amplitud. Debido a que la señal cardiorespiratoria presenta una gran amplitud de sus componentes de baja frecuencia, esto obligaba a auscultar con un nivel de presión bajo, para evitar que el codificador trabajase en su región menos lineal, distorsionando la señal. Asimismo, trabajar en niveles de presión bajos de la campana conllevaba estar, en la práctica, usando un menor numero de

bits/muestra para la codificación, por lo que la resolución se veía enormemente afectada [1].

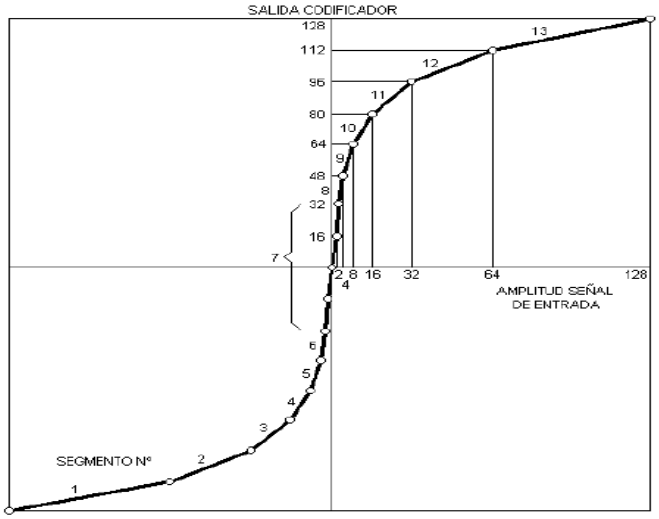


Figura 12 – Niveles de cuantificación G.711

Por otro lado, IMA-ADCPM y los submuestreos daban mejores resultados que G.722 en cuanto a ERMS y PRD, por lo que se optó por diseñar un Codec-Estetoscopia compuesto por un codificador IMA-ADPCM más un submuestreo previo a 4khz, asegurándonos de esta forma respetar el ancho de banda de las señales cardiorrespiratorias.

	CODEC-ESTETOSCOPIA		
	RMSE	**PRD (%)**	**RTF**
Mitral 1	0,000769	1,120503	0,00588
Mitral 2	0,000501	0,427449	0,006
Mitral 3	0,000506	0,376754	0,0054
Aórtica 1	0,000837	0,632204	0,00532
Aórtica 2	0,002574	1,716183	0,0063
Aórtica 3	0,000382	0,285643	0,0054
Tricúspide1	0,000695	0,494306	0,006
Tricúspide 2	0,000587	0,428879	0,00588
Tricúspide 3	0,000503	0,363285	0,0063
Pulmonar 1	0,000534	0,57704	0,0063
Pulmonar 2	0,000746	0,691908	0,0054
Pulmonar 3	0,000904	0,667252	0,00588
Pulmonar 4	0,000499	0,203866	0,00532
Pulmonar 5	0,001623	0,59876	0,0055
Pulmonar 6	0,001683	0,879413	0,0045
	0,00088953	**0,63089633**	**0,00553**

Tabla 3 - Resultados del codificador Codec-Estetoscopia Propuesto con los sonidos EHAS-Fundatel

	CODEC-ESTETOSCOPIA		
	RMSE	**PRD (%)**	**RTF**
Cardiaco 1	0,00061	0,491997	0,005
Cardiaco 2	0,000663	0,657136	0,0042
Cardiaco 3	0,000548	0,531861	0,0062
Cardiaco 4	0,000715	0,682541	0,0062
Cardiaco 5	0,000613	0,494859	0,0053
Cardiaco 6	0,000662	0,523603	0,0061
Cardiaco 7	0,000706	0,521473	0,006
Cardiaco 8	0,000645	0,630833	0,0054
Cardiaco 9	0,000591	0,608498	0,0045
Cardiaco 10	0,00077	0,623369	0,0058
	0,0006523	**0,576617**	**0,0059**

Tabla 4 - Resultados del codificador Codec-Estetoscopia Propuesto con los sonidos Cardiacos

	CODEC-ESTETOSCOPIA		
	RMSE	**PRD (%)**	**RTF**
Pulmonar 1	0,000766	0,264416	0,00688
Pulmonar 2	0,001351	0,845782	0,006
Pulmonar 3	0,011912	16,14164	0,0054
Pulmonar 4	0,006494	3,796678	0,00632
Pulmonar 5	0,009967	8,616585	0,0063
Pulmonar 6	0,000732	0,319646	0,0054
Pulmonar 7	0,001541	1,254112	0,006
Pulmonar 8	0,007109	5,609415	0,00588
	0,004984	**4,60603425**	**0,0062**

Tabla 5 - Resultados del codificador Codec-Estetoscopia Propuesto con los sonidos Respiratorios

Una vez extraídos los resultados de todas las muestras de audio se realizó una tabla comparativa entre el codificador actual en uso (G.722) y el Codec-Estetoscopia-Propuesto. La tasa binaria (R) del Codec-Estetoscopia-Propuesto es calculado de la misma forma que el resto de codificaciones. R = 4000 muestras/segundo * 4 bits/muestra = 16 Kbps.

	CF	**R (Kbps)**	**RMSE**	**PRD (%)**	**RTF**
G.722	4	64	0,039405267	22,7856303	0,02122
Codec-Estetoscopia-Propuesto	16	16	0,002175277	1,93784919	0,00588

Tabla 6 - Comparativa entre el codificador actual en uso (G.722) y el Codec-Estetoscopia Propuesto.

Como se puede observar en la Tabla 6, el nuevo Codec-Estetoscopia Propuesto da mejores resultados objetivos que el codificador G.722 comparando todas las figuras de mérito.

4.2 Resultados subjetivos, evaluación médica

Tras la valoración objetiva de los codificadores se realizó la valoración subjetiva de éstos, ya que se busca que la codificación de los audios cardiorrespiratorios no distorsione en exceso la calidad de la señal y que los audios decodificados sean útiles para un diagnóstico a distancia. Las 33 muestras originales y las respectivas muestras decodificadas con los codecs G.722 y Codec-Estetoscopia fueron evaluadas por un grupo de cuatro doctores pertenecientes al Hospital Clínico San Carlos de Madrid. Los doctores tenían que puntuar de 1 a 10 la calidad de los audios escuchados, tanto los originales como los decodificados, sin conocer el tipo de codificación de cada muestra.

- Evaluación de Dra. Asunción Nieto Barbero, neumóloga adjunta al Hospital Clínico San Carlos, Madrid.

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
Original	7	9	8	8	7	8	8	9	7	7	8
G.722	5	7	7	7	5	6	5	8	6	5	7
Codec-Estetoscopia	5	7	7	6	5	5	4	7	6	5	7
	12	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**	**22**
Original	5	8	8	8	8	8	7	8	6	6	8
G.722	4	8	6	7	8	7	7	6	4	4	6
Codec-Estetoscopia	4	6	6	7	7	7	6	6	4	4	5
	23	**24**	**25**	**26**	**27**	**28**	**29**	**30**	**31**	**32**	**33**
Original	8	6	4	3	6	6	6	5	8	7	6
G.722	6	6	5	3	4	4	4	4	7	7	5
Codec-Estetoscopia	6	6	5	3	4	4	4	4	6	6	5

Tabla 7 - Valoración de la Dra. Asunción Nieto Barbero

- Evaluación de Dra. Gema Rodríguez Trigo, neumóloga adjunta al Hospital Clínico San Carlos, Madrid.

	1	2	3	4	5	6	7	8	9	10	11
Original	8	10	10	10	7	6	6	9	10	9	6
G.722	8	10	10	10	7	6	6	9	9	9	6
Codec-Estetoscopia	8	10	10	9	8	7	7	9	9	9	6
	12	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**	**22**
Original	6	7	7	9	6	8	8	5	5	5	7
G.722	6	8	8	9	6	8	8	6	5	4	7
Codec-Estetoscopia	7	8	8	9	6	8	9	7	6	6	7
	23	**24**	**25**	**26**	**27**	**28**	**29**	**30**	**31**	**32**	**33**
Original	8	8	8	8	8	8	7	6	6	5	6
G.722	8	8	8	8	8	8	7	6	6	6	7
Codec-Estetoscopia	9	9	8	9	8	9	8	7	7	6	8

Tabla 8 - Valoración de la Dra. Gema Rodríguez Trigo

- Evaluación de Dra. Celia Pinedo Sierra, neumóloga adjunta al Hospital Clínico San Carlos, Madrid.

	1	2	3	4	5	6	7	8	9	10	11
Original	8	8	8	7	8	7	7	8	7	6	7
G.722	8	7	7	7	7	7	6	8	6	7	7
Codec-Estetoscopia	6	7	6	7	6	6	7	6	6	7	6
	12	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**	**22**
Original	7	8	7	8	6	8	9	5	6	6	7
G.722	8	7	6	7	5	8	9	5	5	5	6
Codec-Estetoscopia	7	7	6	6	5	7	8	5	5	5	5
	23	**24**	**25**	**26**	**27**	**28**	**29**	**30**	**31**	**32**	**33**
Original	5	6	5	4	7	7	6	7	6	5	6
G.722	5	5	5	4	5	6	6	6	5	5	6
Codec-Estetoscopia	5	5	5	4	5	5	5	6	5	5	5

Tabla 9 - Valoración de la Dra. Celia Pinedo Sierra

- Evaluación de Dra. Encarnación Borreguero Martínez, adjunta especialista en medicina interna, Hospital Clínico San Carlos, Madrid.

	1	2	3	4	5	6	7	8	9	10	11
Original	7	8	8	6	6	7	6	7	8	8	6
G.722	7	7	7	5	6	6	5	6	7	7	6
Codec-Estetoscopia	5	6	6	6	5	5	5	6	7	6	7
	12	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**	**22**
Original	7	7	7	8	6	6	8	6	6	5	5
G.722	7	6	7	8	6	5	7	6	6	6	6
Codec-Estetoscopia	6	6	6	7	6	5	6	4	5	5	5
	23	**24**	**25**	**26**	**27**	**28**	**29**	**30**	**31**	**32**	**33**
Original	6	6	5	6	6	7	6	6	7	6	7
G.722	6	6	6	5	5	7	6	7	6	5	7
Codec-Estetoscopia	6	5	6	5	4	6	6	6	5	5	6

Tabla 10 - Valoración de la Dra. Encarnación Borreguero Martínez

Solo la Dra. Gema Rodríguez Trigo puntuó, en términos generales, al Codec-Estetoscopia-Propuesto con los valores más altos, comentando que existía menos ruido en los audios escuchados, y que debido a esto, los sonidos cardiacos y respiratorios en los que se tenía que focalizar quedaban resaltados. El resto de médicos dieron, en general, una puntuación levemente superior a los audios codificados con el codec G.722 respecto a los codificados con el Codec-Estetoscopia-Propuesto. La Dra. Asunción Nieto Barbero explicó que en el caso de los audios en los que se escuchaban patologías con componentes muy agudas, se observaba una distorsión, escuchándose estas componentes más graves. Sin embargo aseguró que conociéndose la patología era capaz de diagnosticarla.

	Original	G.722	Codec-Estetoscopia
Dra. Barbero	7	5.75757576	5.42424242
Dra. Trigo	7.33333333	7.42424242	7.90909091
Dra. Pinedo	6.72727273	6.24242424	5.87878788
Dr. Borreguero	6.54545455	6.21212121	5.60606061
	6.90151515	**6.40909091**	**6.20454545**

Tabla 11 - Comparativa de los promedios de la valoración subjetiva entre G.722 y Codec-Estetoscopia

Observamos en la Tabla 11 como G.722 obtiene una puntuación levemente superior al Codec-Estetoscopia-Propuesto. La Figura 13 muestra una comparativa del

promedio de las evaluaciones de cada muestra. Se toma como referencia el valor ponderado de la muestra original.

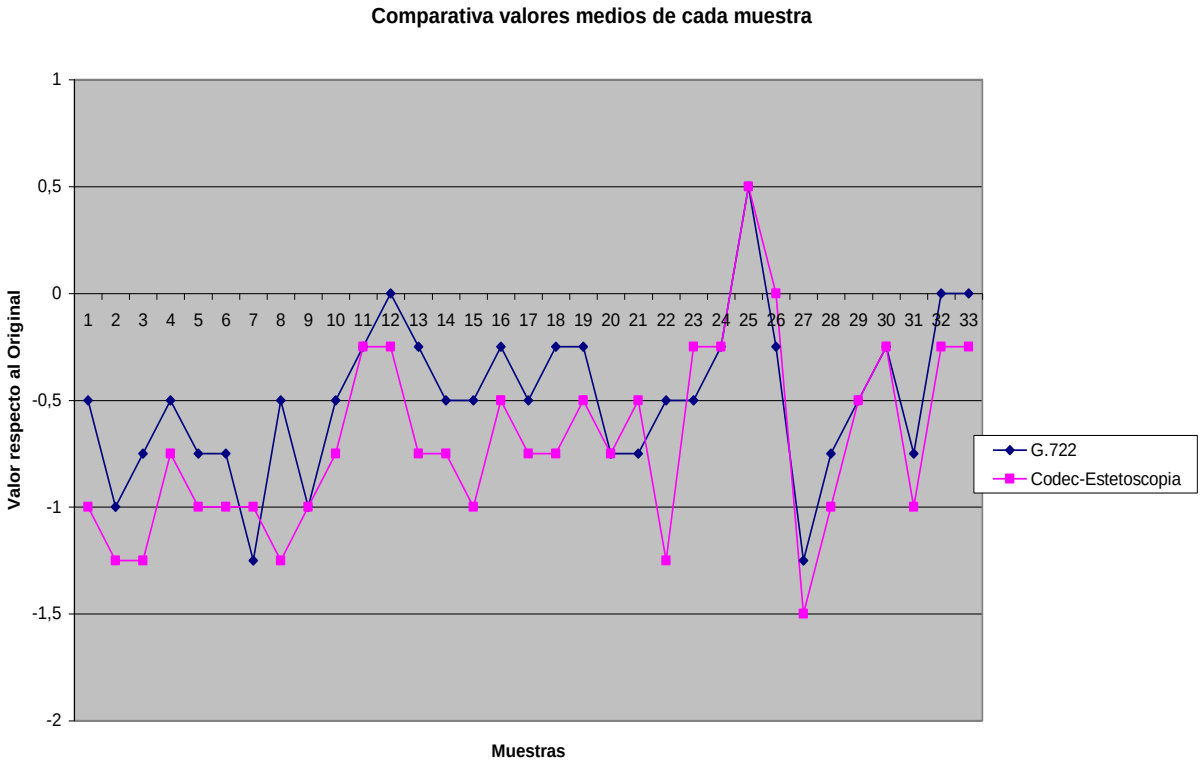


Figura 13 - Gráfica comparativa de las evaluaciones promedios de los codificadores G.722 y Codec-Estetoscopia

4.3 Pruebas con softphone Ekiga

4.3.1 Introducción

Una vez integrado el codec estetoscopia en la librería OPAL es posible realizar una prueba de conexión entre dos ordenadores a traves del Softphone Ekiga. Debido a que ningún servidor de VoIP soporta el Payload Type DVI4_8kHz tendremos que conectar dos ordenadores directamente configurando una red cableada utilizando un cable Cat5e crossover network, evitando así el paso de los paquetes IP por los servidores de VoIP SIP o H.323. De esta forma será posible establecer una llamada sin que se produzca un error en los servidores.

4.3.2 Configuración de una Red Cableada

El primer paso para configurar una red cableada entre los dos ordenadores es conocer la dirección IP de éstos. Para ello se ejecuta en una terminal el comando **ifconfig**. Una vez conocida la dirección IP accedemos a Conexiones de Red Cableadas a traves de: Sistema -> Preferencias ->Conexiones de red ->Cableada

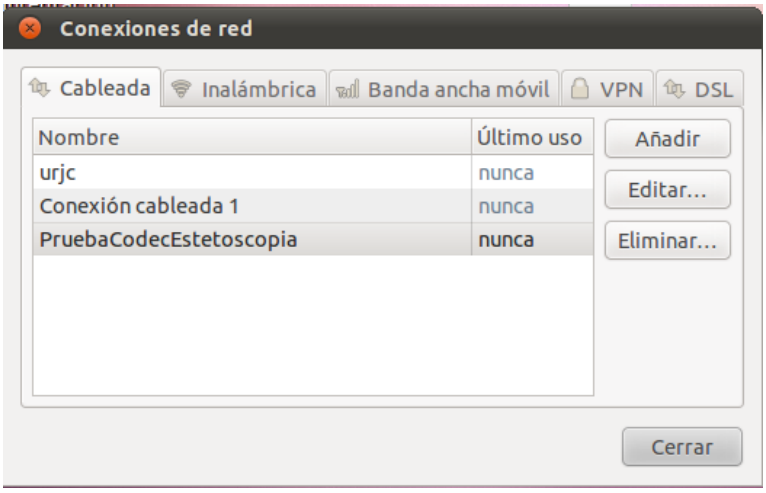


Figura 14 - Selección de un red cableada

Añadimos una nueva conexión, le damos un nombre, y en la pestaña Ajustes de IPV4 editamos los siguientes valores:

- **Método**: Manual
- **Dirección**: Nuestras direcciones IP (en nuestro caso 192.168.1.35 y 192.168.1.12)
- **Mascara de Red**: 255.255.255.0
- **Puerta e enlace**: 0.0.0.0

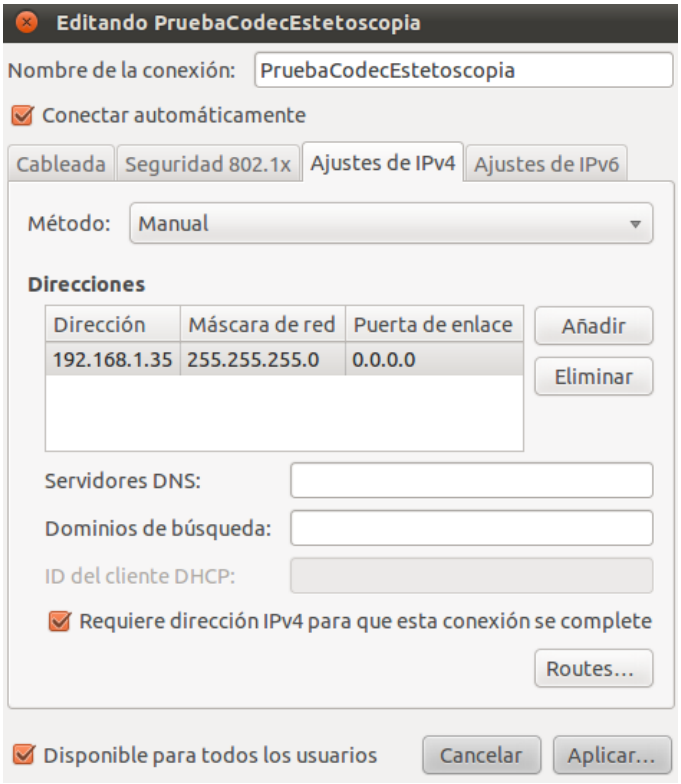


Figura 15 - Configuración de una red cableada

Una vez configurada la conexión cableada en ambos ordenadores, se procede a conectar ambos ordenadores.

4.3.3 Establecimiento y cierre de llamada, protocolo SIP

Abrimos el Softphone Ekiga y seleccionamos el codificador correspondiente a Codec-Estetoscopia (DIV4) como codificador de audio tal y como se describió en la sección 3.4.3 .

Observamos cómo en Ekiga nos aparece el otro terminal en la sección vecinos. Lo seleccionamos y le damos llamar. En un instante aparecerá un mensaje de llamada entrante.

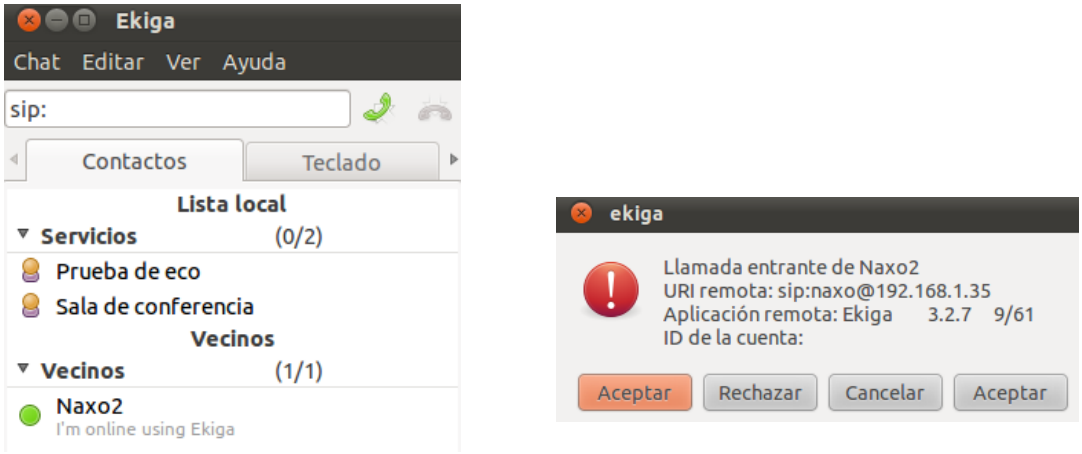


Figura 16 - Lamada con Ekiga Softphone

Mediante el uso del programa Ethereal captamos los paquetes IP que se envían entre los dos ordenadores al realizar la llamada. Podemos observar en la Figura 17 cómo se produce correctamente el protocolo SIP de establecimiento y cierre de la llamada VoIP.

No.	Time	Source	Destination	Protocol	Info
7	6.853381	192.168.1.12	192.168.1.35	SIP/SDP	Request: INVITE sip:neighbour@ubuntu.local:5060, with session description
8	6.855353	192.168.1.35	192.168.1.12	SIP	Status: 100 Trying
9	6.859511	192.168.1.35	192.168.1.12	SIP	Status: 180 Ringing
12	12.551186	192.168.1.35	192.168.1.12	SIP/SDP	Status: 200 OK, with session description
13	12.557707	192.168.1.12	192.168.1.35	SIP	Request: ACK sip:neighbour@192.168.1.35
1566	28.850206	192.168.1.12	192.168.1.35	SIP	Request: BYE sip:neighbour@192.168.1.35
1567	28.851735	192.168.1.35	192.168.1.12	SIP	Status: 200 OK

Figura 17 - Paquetes del protocolo SIP captados durante el establecimiento y cierre de llamada

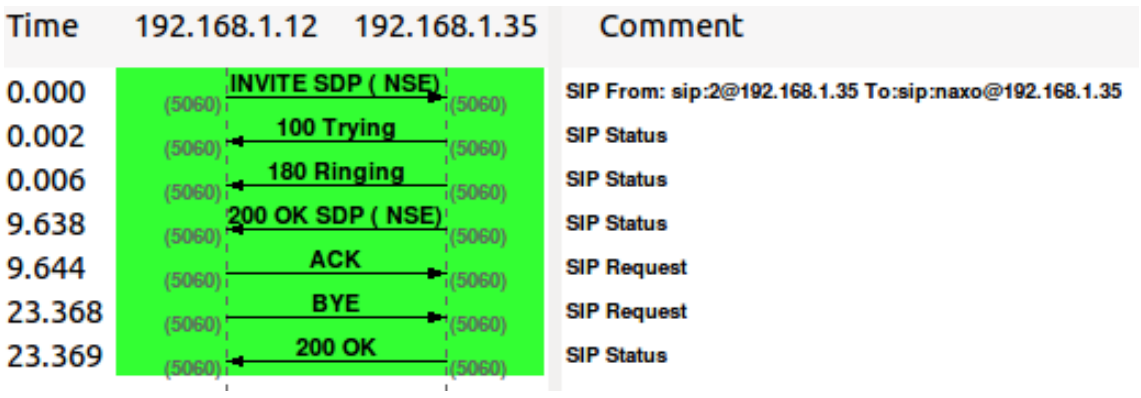


Figura 18 - Apertura y cierre del protocolo SIP

En la Figura 18 se puede comprobar el intercambio de mensajes de una llamada SIP típica. Para iniciar la sesión el llamante enviará un mensaje INVITE que

contendrá los parámetros necesarios para la comunicación. Estos parámetros se describen en el mensaje SDP que hay dentro del INVITE. Lo habitual es que este mensaje llegue al servidor proxy, cuya dirección se obtiene mediante peticiones DNS. Cuando la petición de llamada llega al proxy, determina si él es el encargado de ese dominio mediante el campo "destination address". En caso de que no sea el responsable, se enviarán una serie de mensajes DNS para averiguar la dirección IP de destino, el puerto y el tipo de transporte a utilizar. Una vez ya se conoce toda esta información, el servidor proxy abre una conexión UDP o TCP contra el servidor proxy de destino y envía el mensaje INVITE. El mensaje llega al servidor proxy de destino, éste se encarga de analizar, a través del campo "destination address", si es o no el encargado de ese dominio y si lo es, pregunta al servidor de ubicación (Location Server) por la dirección (Address of Record) del llamado y obtiene su "contact address". Finalmente hace llegar la petición al destinatario de la llamada. Cuando el mensaje INVITE llega al teléfono del destinatario, éste suena y mientras tanto, envía mensajes de información 100 Trying (se está procediendo la llamada) y 180 Ringing (el terminal está sonando). Los mensajes de información se envían haciendo el camino inverso que indica el campo "Via" de la cabecera, donde se almacena la dirección IP de los equipos por los que pasa el mensaje. Cada servidor proxy va leyendo y eliminando el valor correspondiente "Via" hasta llegar al teléfono del llamante. Cuando el destinatario acepta la llamada, su teléfono (el UA del teléfono) envía una respuesta del tipo 200 OK para indicar que se acepta la llamada. En esta respuesta va incluida la respuesta SDP que indica el tipo y las características multimedia con los codecs ofrecidos. Este mensaje se transmite al llamante de la misma manera que el mensaje 180, esto es, a través de la información del campo Via. Una vez el llamante recibe la confirmación, enviará un mensaje ACK ratificando la confirmación. En este momento, ya está todo listo para que empiece el flujo RTP/RTCP de voz y/o vídeo como podremos observar en la siguiente sección. Para terminar la llamada, uno de los dos extremos envía un mensaje BYE directamente a la dirección Contact que se obtuvo en la cabecera del INVITE inicial. El otro extremo responderá con un mensaje de aceptación 200 OK y la sesión finalizará. Hay que puntualizar que en nuestro caso al ser una conexión cableada directa entre dos terminales la transmisión de mensajes se hará directamente entre estos, obviando el tránsito por los servidores proxys [20].

A pesar de producirse correctamente la apertura y cierre de la llamada comprobamos cómo no se produce el envió de información, ya que no se escucha nada en ninguno de los dos terminales. Pasamos por tanto a analizar el envío de paquetes RTP donde se encuentran encapsulados los datos codificados del audio enviado.

4.3.4 Envio de paquetes RTP

Entre la apertura y cierre de llamada del protocolo SIP, Ethereal también captó la tansmisión de los paquetes RTP.

No. .	Time	Source	Destination	Protocol	Info
16	12.608291	192.168.1.35	192.168.1.12	RTP	Payload type=DVI4 8000 samples/s, SSRC=4005074012, Seq=10685, Time=0, Mark
19	12.678601	192.168.1.35	192.168.1.12	RTP	Payload type=DVI4 8000 samples/s, SSRC=4005074012, Seq=10686, Time=504
22	12.730794	192.168.1.12	192.168.1.35	RTP	Payload type=DVI4 8000 samples/s, SSRC=1982384854, Seq=60501, Time=0, Mark
25	12.739371	192.168.1.35	192.168.1.12	RTP	Payload type=DVI4 8000 samples/s, SSRC=4005074012, Seq=10687, Time=1008
28	12.793947	192.168.1.12	192.168.1.35	RTP	Payload type=DVI4 8000 samples/s, SSRC=1982384854, Seq=60502, Time=504
31	12.800049	192.168.1.35	192.168.1.12	RTP	Payload type=DVI4 8000 samples/s, SSRC=4005074012, Seq=10688, Time=1512
34	12.849502	192.168.1.12	192.168.1.35	RTP	Payload type=DVI4 8000 samples/s, SSRC=1982384854, Seq=60503, Time=1008
37	12.860375	192.168.1.35	192.168.1.12	RTP	Payload type=DVI4 8000 samples/s, SSRC=4005074012, Seq=10689, Time=2016
40	12.915385	192.168.1.12	192.168.1.35	RTP	Payload type=DVI4 8000 samples/s, SSRC=1982384854, Seq=60504, Time=1512
43	12.930722	192.168.1.35	192.168.1.12	RTP	Payload type=DVI4 8000 samples/s, SSRC=4005074012, Seq=10690, Time=2520
46	12.990607	192.168.1.12	192.168.1.35	RTP	Payload type=DVI4 8000 samples/s, SSRC=1982384854, Seq=60505, Time=2016

Figura 19 - Parquetes RTP captados durante el establecimiento de llamada

Al adentrarnos en la información encapsulada en los paquetes RTP, en concreto en el campo Payload (donde se encuentran los datos de audio codificados), observamos cómo este campo se encuentra vacio.

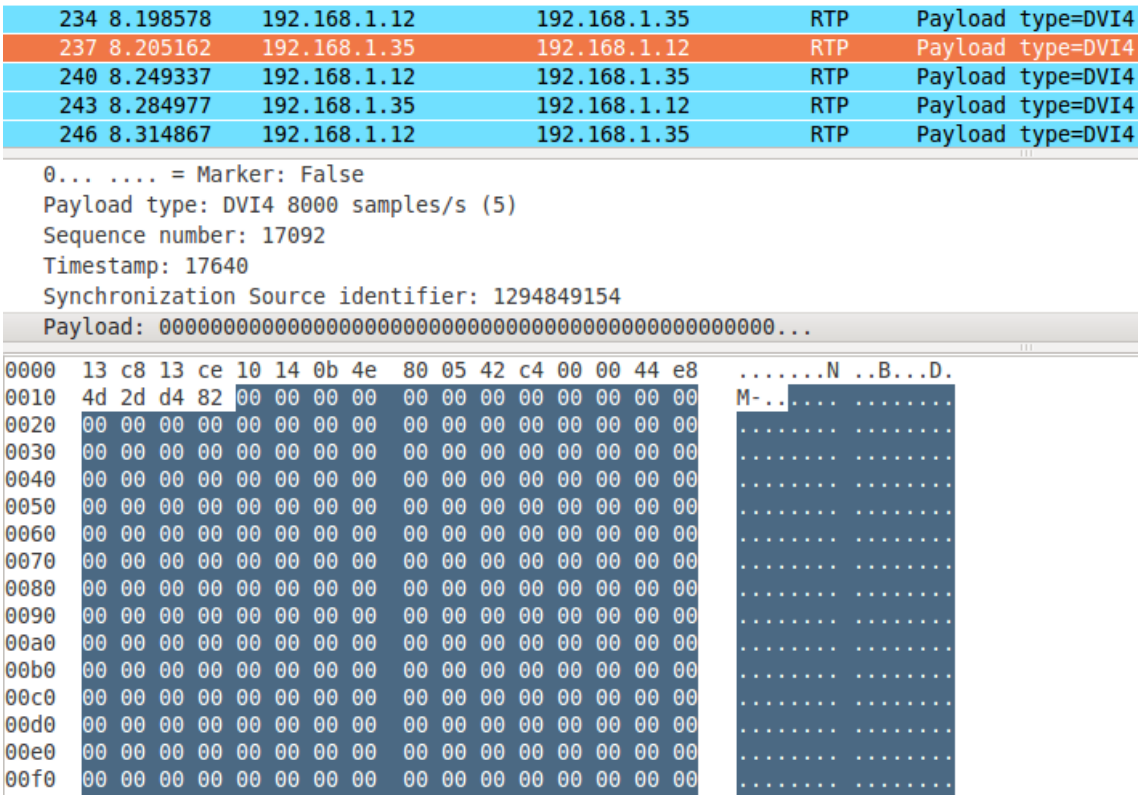


Figura 20 – Campo Payload del paquete RTP

Analizamos softphone Ekiga en busca de la razón por la cúal el Payload se encuentra vacío. Para ello ejecutamos el comando **ekiga -d 4 2>output.txt** mientras se produce la llamada, guardándose los datos en el archivo de texto output.txt. Al abrir el archivo observamos que se produce un error en la codificación, en concreto en el archivo patch.cxx:

```
2012/01/03 17:07:05.648   0:40.510   Media Patch:0xb21eeb70 Patch Media
                                     conversion (primary) failed

2012/01/03 17:07:05.648   0:40.510   Media Patch:0xb21eeb70 Patch
                                     WriteFrame failed

2012/01/03 17:07:05.648   0:40.510   Media Patch:0xb21eeb70 Patch
                                    Thread ended because all sink
                                     writes failed failed

2012/01/03 17:07:05.648   0:40.510   Media Patch:0xb21eeb70 Patch
                                      Thread ended for Patch
                                     OpalAudioMediaStream-Source-PCM-16
                                     -> OpalRTPMediaStream-Sink-MS-IMA-
                                     ADPCM
```

Tras meses intentando solucionar el error, debido al alto conocimiento en C++ exigido y a la pobre documentación de todas las librerías implicadas en el funcionamiento del Softphone Ekiga, se decidió que la resolución del error excedía el objetivo primordial del PFC: la proposición de una alternativa al codificador de voz G.722 para la codificación de señales cardiorrespiratorias obtenidas con un estetoscopio digital.

5. CONCLUSIONES Y TRABAJO FUTURO

5.1 Conclusiones

A pesar de que el Codec-Estetoscopia-Propuesto da mejores resultados objetivos que G.722 al comparar las figuras de merito como RMSE y PRD, la valoración subjetiva de este nuevo codificador es levemente peor. El Codec-Estetoscopia-Propuesto reduce la tasa binaria en una proporción de 4:1, pasando de 64 Kbps a 16 Kbps, y decrementa el RTF en una proporcion mayor de 3:1, pasando de 0,02122 a 0,00588. El Codec-Estetoscopia Propuesto podría ser una buena alternativa al codec G.722 en los casos donde las redes de comunicaciones son precarias y existe un bajo ancho de banda, caso típico de las zonas rurales en países subdesarrollados o en vías de desarrollo. Nuestro codificador podría ademas permitir un mayor número de auscultaciones al mismo tiempo, reduciendose la latencia y exigiendo una capacidad de procesamiento menor de los ordenadores utilizados debido a la reducción del RTF.

5.2 Trabajo Futuro

Las valoraciones médicas son solo orientativas debido al bajo número de médicos consultados. Se debería realizar un ensayo clínico con pacientes reales que sufran patologías cardiacas o respiratorias con el objetivo de validar médicamente el uso del Codec-Estetoscopia Propuesto.

También se tendrá que solucionar el problema de codificación aún no resuelto, producido en el archivo patch.cxx, y averiguar la razón de éste. Como último paso se tendría que configurar un servidor de VoIP para que soportase la codificación IMA-ADPCM. Seguramente sería aún más útil utilizar en la codificacion un Payload Type

soportado por todos los servidores de VoIP, como por ejemplo el del G.711, “engañando” así a los servidores y permitiendo la comunicación, ya que en el fondo los datos que transportan los paquetes RTP pertenecen a otra codificación.

REFERENCIAS

[1] Ignacio Foche Pérez: Desarrollo de un teleestetoscopio digital Bluetooth para zonas rurales aisladas de países en desarrollo, Proyecto Fin de Máster en Redes de Telecomunicación para Países en Desarrollo, Escuela Técnica Superior de Ingeniería de Telecomunicaciones, Universidad Rey Juan Carlos, 2011.

[2] Telecomumunication Development Bureau. Telemedicine and developing countries – lessons learned. http://www.itu.int/ITU-D/tech/telemedicine/Doc2-166e.pdf, Agosto 1999.

[3] Andrés Martínez Férnandez, Valentín Villarroel Ortega, Joaquín Seoane Pascual, Arnau Sánchez Sala, and Francisco del Pozo Guerrero. Sistemas de telemedicina rural para países en desarrollo. CASEIB, pp. 395-398, 2002.

[4] Ricardo Oña Martínez-Albelda: Evaluación del impacto en salud del proyecto de telemedicina EHAS-Napo, Proyecto Fin de Master en Redes de Telecomunicación para Países en Desarrollo, Escuela Técnica Superior de Ingeniería de Telecomunicaciones, Universidad Rey Juan Carlos, 2011.

[5] Wikipedia – "Auscultación".

[6] M. Abella and J. Formolo: Comparsion of the Acoustic Propierties os Six Popular Stethoscopes, Departamente os Internal Medicine, St Jhon Hospital, Detroit, Michigan 48236.

[7] I. Mazic, S. Sovilj and Magjarevic: Analysis od Respiratory Sounds in Asthmatic Infants, Measurement Science Review, Vol. 3, Section 2, 2003.

[8] Alan V. Oppenheim, Alans S. Willsky, S. Hamid Nawad: Señales y Sistemas, pp. 527-528, Segunda edición, Pearson Education.

[9] Juan Martínez Alajarín, Sistema de Ayuda al Telediagnóstico de Enfermedades Cardiovasculares basado en el Análisis de Fonocardiogramas.

[10] Mohammad Pooyan, Ali Taheri, Morteza Moazami-Goudarzi, Iman Saboori, Wavelet Compression of ECG Signals Using SPIHT Algorithm, World Academy of Science, Engineering and Technology, 2005.

[11] Yodollahi, Senior Member, Respiratory Sounds Compression, IEEE Transactions on Biomedical Engineering, Vol. 5 No. 4, April 2008.

[12] A. Yadollahi, Z. Moussavi, P. Yahampath. Adaptive Compression of Respiratory and Swallowing Sounds, Proceedings of the 28th IEEE EMBS Annual International Conference, New York City, USA, Aug 30 - Sep 3, 2006.

[13] Jesús Marcos Zamarreño, Imeplementación de un Códec de Voz Propietario para Aplicaciones de Codificaciones de Voz en Banda Ancha con Baja Relación Señal a Ruido, Proyecto Fin de Carrera de Telecomunicaciones, Escuela Politécnica Superior, Universidad Autónoma, Julio 2009.

[14] UIT-T, Recomendación UIT-T G.722, Aspectos generales de los sitemas de transmisión digital, Codificación de Audio de 7 kHz Dentro de 64 kbit/s, (Extracto del libro Azul), Melburne 1988.

[15] UIT-T, Recomendación UIT-T G.726, Aspectos generales de los sitemas de transmisión digital, Modificación por Impulsos Codificados Diferencial Adaptativa (MICDA) a 40, 32, 24, y 16 kbit/s, Ginebra 1990.

[16] Davis Yen Pan, Digital Audio Compression, Digital Technical Journal Vol. 5 No. 2, 1933.

[17] Cristina Romero Macías, Introducción del Codec MELP en la plataforma IP PBX Asterisk ®, Proyecto Fin de Carrera de Telecomunicaciones, Escuela Politécnica Superior, Universidad Autónoma, Junio 2008.

[18] Francois Bounoure, Session Initation Protocol, Maestría en Ingeniería de Telecomunicaciones, Facultad de Telecomunicaciones, Universidad de Buenos Aires, Diciembre 2006.

[19] ITU-T Recommendation H.323 v7, Packet-based multimedia communications systems, 2009.

[20] Francesc Sanz Pagés, Despliegue y analisis de un escenario de telefonía IP con la aplicación docente, Escuela de Ingeniería de Telecomunicación, Universidad Politécnica de Barcelona, Septiembre 2011.

[21] Ekiga, http://www.ekiga.org .

[22] University of Michigan Heart Sound and Murmur Library, http://www.med.umich.edu/lrc/psb/heartsounds/index.htm .

[23] Sound eXchange, http://sox.sourceforge.net/ .

[24] UIT-T, http://www.itu.int/net/itu-t/sigdb/speaudio/Gseries.htm .

[25] Erik Castro de Lopo, Librería Libsamplerate, Secret Rabbit Code, http://www.mega-nerd.com/SRC/download.html .

[26] Erik Castro de Lopo, Librería Libsndfile, http://www.mega-nerd.com/libsndfile .

[27] PTLIB, Opal VoIP wiki, http://www.opalvoip.org/wiki/index.php?n=Main.BuildingPTLibUnix .

[28] OPAL, Opal VoIP wiki, http://www.opalvoip.org/wiki/index.php?n=Main.BuildingOPALUnix .

[29] Andreas Damien, Códec Plugins, http://voxgratia.org/docs/codec_plugins.html , 2003.

GLOSARIO

TIC – Tecnologías de la Información y la Comunicación

Agrupación de los elementos y las técnicas usadas en el tratamiento y la transmisión de la información, principalmente la informática, Internet y las telecomunicaciones.

WiMAX – Worldwide Interoperability for Microwave Access

Es un estandar de comunicación wireless diseñado para proveer de 30 a 40 Mbps.

ITU-T – ITU Telecommunication Standardization Sector

Es uno de los tres sectores en los que se encuentra dividida la International Telecommunication Union (ITU); coordina los estandares de telecomunicaciones.

ECG – Electrocardiograma

Es la representación gráfica de la actividad eléctrica del corazón, que se obtiene con un electrocardiógrafo en forma de cinta continua.

FCG – Fonocardiograma

Curva obtenida por el registro gráfico de los ruidos del corazón.

VoIP – Voice Over IP

Es un grupo de recursos que hacen posible que la señal de voz viaje a través de Internet empleando un protocolo IP.

Transformada Wavelet

Es un tipo especial de transformada de Fourier que representa una señal en términos de versiones trasladadas y dilatadas de una onda finita (denominada óndula madre).

DCT – Discrete Cosine Transform

Es una transformada basada en la Transformada de Fourier discreta, pero utilizando únicamente números reales.

RTP – Real-time Transport Protocol

Es un protocolo de nivel de sesión utilizado para la transmisión de información en tiempo real, como por ejemplo audio y vídeo en una video-conferencia.

RTCP – RTP Control Protocol

Es un protocolo hermano del protocolo RTP. Su funcionalidad y la estructura del paquete esta definido en la especificación RTP RFC 3550. Da información estadísitica sobre el transporte de paquetes RTP.

PSTN – Public Switched Telephone Networ

Es la red pública de servicios telefonicos por circuitos conmutados.

QoS – Quality of Service

Son las tecnologías que garantizan la transmisión de cierta cantidad de información en un tiempo dado (throughput). Calidad de servicio es la capacidad de dar un buen servicio. Es especialmente importante para ciertas aplicaciones tales como la transmisión de vídeo o voz.

LAN – Red de Area Local

Es la interconexión de una o varias computadoras y periféricos. Su extensión está limitada físicamente a un edificio o a un entorno de 200 metros, con repetidores podría llegar a la distancia de un campo de 1 kilómetro. Su aplicación más extendida es la interconexión de computadoras personales y estaciones de trabajo en oficinas, fábricas, etc.

SPP – Serial Port Profile

Puerto serie. Basado en la especificación 07.10 de ETSI por medio del protocolo RFCOMM. Emula una línea serie y provee una interfaz de reemplazo de comunicaciones basadas en RS-232, con las señales de control típicas. Base de DUN, FAX, HSP y AVRCP.

RMSE – Root-Mean-Square Error

Es un habitual medida de las diferencias entre valores predichos por un modelo o un estimador y los valores reales.

ANEXOS

A. SCRIPT PARA CODEC-ESTETOSCOPIA-PROPUESTO

```
#!/bin/sh
#
cd /
cd home/naxo/Escritorio/Script/Muestras_EHAS

echo "Nombre del audio en .wav a codificar a ima adpcm tras submuestrear a
4Khz: "
read audio

#=====================================================================
#======================== CODEC-ESTETOSCOPIA =========================
#=====================================================================

#------------ (32bits a 44.1kHz) ----> (PCM-16bits a 4kHz) -----------

sox -r 44.1k -b 32 -e floating-point $audio.raw $audio.wav

(time -f "De 44.1k-32 a 4k-16 ---> %E" sox -r 44.1k -b 32 -e floating-point
$audio.raw -r 4k -b 16 $audio-4k-16.raw) 2> tiempos.txt

mkdir $audio-4KHz-ima_adpcm
cp tiempos.txt /home/naxo/Escritorio/Script/Muestras_EHAS/$audio-4KHz-
ima_adpcm/
cp $audio.raw /home/naxo/Escritorio/Script/Muestras_EHAS/$audio-4KHz-
ima_adpcm/
cp $audio-4k-16.raw /home/naxo/Escritorio/Script/Muestras_EHAS/$audio-4KHz-
ima_adpcm/
cp $audio.wav /home/naxo/Escritorio/Script/Muestras_EHAS/$audio-4KHz-
ima_adpcm/
rm $audio-4k-16.raw
rm $audio.wav
rm tiempos.txt
cd /home/naxo/Escritorio/Script/Muestras_EHAS/$audio-4KHz-ima_adpcm/


#----------------- CODIFICACION - DECODIFICACION --------------------

sox -r 4k -b 16 -e signed-integer $audio-4k-16.raw $audio-4k-16.wav

(time -f "De 4k-16 a Ima_ADPCM ---> %E" sox $audio-4k-16.wav -e ima-adpcm
$audio-4k-ima_adpcm.wav) 2>> tiempos.txt

(time -f "De Ima_ADPCM a 4k-16 ---> %E" sox $audio-4k-ima_adpcm.wav -r 4k -
b 16 -e signed-integer $audio-4k-16-ima_adpcm_deco.wav) 2>> tiempos.txt

sox $audio-4k-16-ima_adpcm_deco.wav -r 4k -b 16 -e signed-integer $audio-
4k-16-ima_adpcm_deco.raw
```

```
#----------- (PCM-16bits a 16kHz) ----> (32bits a 44.1kHz) -----------

(time -f "De 4k-16 a 44.1k-32 ---> %E" sox -r 4k -b 16 -e signed-integer
$audio-4k-16-ima_adpcm_deco.raw -r 44.1k -b 32 -e floating-point $audio-
44.1k-32-ima_adpcm_deco.raw) 2>> tiempos.txt

sox -r 44.1k -b 32 -e floating-point $audio-44.1k-32-ima_adpcm_deco.raw
$audio-44.1k-32-ima_adpcm_deco.wav


#----------------- EXTRACCION DE DATOS IMA-ADPCM --------------------

cksum $audio-4k-16.wav > $audio-tamaÃ±os-ima_adpcm.txt
cksum $audio-4k-ima_adpcm.wav >> $audio-tamaÃ±os-ima_adpcm.txt

sox $audio-4k-16.wav $audio-4k-16.dat stats 2> $audio-no_codificado-
ima_adpcm.txt
sox $audio-4k-16.wav -n stat 2>> $audio-no_codificado-ima_adpcm.txt

sox $audio-4k-16-ima_adpcm_deco.wav $audio-4k-16-ima_adpcm_deco.dat stats
2> $audio-decodificado-ima_adpcm.txt
sox $audio-4k-16-ima_adpcm_deco.wav -n stat 2>> $audio-decodificado-
ima_adpcm.txt

sox $audio-4k-ima_adpcm.wav $audio-4k-ima_adpcm.dat stats 2> $audio-
codificado-ima_adpcm.txt
sox $audio-4k-ima_adpcm.wav -n stat 2>> $audio-codificado-ima_adpcm.txt

cp $audio-4k-16.dat /home/naxo/Escritorio/Script/RMS/Programa/
cp $audio-4k-16-ima_adpcm_deco.dat
/home/naxo/Escritorio/Script/RMS/Programa/

cd /home/naxo/Escritorio/Script/RMS/Programa/

./rms $audio-4k-16.dat $audio-4k-16-ima_adpcm_deco.dat

cp RMS.dat /home/naxo/Escritorio/Script/Muestras_EHAS/$audio-4KHz-
ima_adpcm/

rm $audio-4k-16.dat
rm $audio-4k-16-ima_adpcm_deco.dat
rm RMS.dat
rm resta.dat


#------------- EXTRACCION DE DATOS CODEC-ESTETOSCOPIA ----------------

cd /home/naxo/Escritorio/Script/Muestras_EHAS/$audio-4KHz-ima_adpcm/

cksum $audio.wav > $audio-tamaÃ±os.txt
cksum $audio-4k-16.wav >> $audio-tamaÃ±os.txt
cksum $audio-4k-ima_adpcm.wav >> $audio-tamaÃ±os.txt
#cksum $audio-44.1k-32-ima_adpcm_deco.wav >> $audio-tamaÃ±os.txt

sox $audio-44.1k-32-ima_adpcm_deco.wav $audio-44.1k-32-ima_adpcm_deco.dat
stats 2> $audio-decodificado.txt
sox $audio-44.1k-32-ima_adpcm_deco.wav -n stat 2>> $audio-decodificado.txt

sox $audio.wav $audio.dat stats 2>$audio-original.txt
sox $audio.wav -n stat 2>>$audio-original.txt

cp $audio.dat /home/naxo/Escritorio/Script/RMS/Programa/
```

```
cp $audio-44.1k-32-ima_adpcm_deco.dat
/home/naxo/Escritorio/Script/RMS/Programa/

cd /home/naxo/Escritorio/Script/RMS/Programa/

./rms $audio.dat $audio-44.1k-32-ima_adpcm_deco.dat

rename 'y/A-Z/a-z/' RMS.dat

cp rms.dat /home/naxo/Escritorio/Script/Muestras_EHAS/$audio-4KHz-
ima_adpcm/

rm $audio.dat
rm $audio-44.1k-32-ima_adpcm_deco.dat
rm rms.dat
rm resta.dat

#--------------------------------------------------------------------
```

B. CODIGO FUENTE DEL CODIFICADOR CODEC-ESTETOSCOPIA-PROPUESTO

Common.h

```
/*
** Copyright (C) 1999-2011 Erik de Castro Lopo <erikd@mega-nerd.com>
**
** All rights reserved.
**
** Redistribution and use in source and binary forms, with or without
** modification, are permitted provided that the following conditions are
** met:
**
**     * Redistributions of source code must retain the above copyright
**       notice, this list of conditions and the following disclaimer.
**     * Redistributions in binary form must reproduce the above copyright
**       notice, this list of conditions and the following disclaimer in
**       the documentation and/or other materials provided with the
**       distribution.
**     * Neither the author nor the names of any contributors may be used
**       to endorse or promote products derived from this software without
**       specific prior written permission.
**
** THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
** "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED
** TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR
** PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR
** CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
** EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
** PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
PROFITS;
** OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
** WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR
** OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
** ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*/


#define     ARRAY_LEN(x)      ((int) (sizeof (x) / sizeof (x [0])))
#define     MAX(a,b)          ((a) > (b) ? (a) : (b))

typedef struct
{     const char * title ;
      const char * copyright ;
      const char * artist ;
      const char * comment ;
      const char * date ;
      const char * album ;
      const char * license ;
```

```
	/* Stuff to go in the 'bext' chunk of WAV files. */
	int has_bext_fields ;
	int coding_hist_append ;

	const char * description ;
	const char * originator ;
	const char * originator_reference ;
	const char * origination_date ;
	const char * origination_time ;
	const char * umid ;
	const char * coding_history ;
	const char * time_ref ;
} METADATA_INFO ;

typedef SF_BROADCAST_INFO_VAR (2048) SF_BROADCAST_INFO_2K ;

void sfe_apply_metadata_changes (const char * filenames [2], const
METADATA_INFO * info) ;

void sfe_copy_data_fp (SNDFILE *outfile, SNDFILE *infile, int channels) ;

void sfe_copy_data_int (SNDFILE *outfile, SNDFILE *infile, int channels) ;

int sfe_file_type_of_ext (const char *filename, int format) ;

void sfe_dump_format_map (void) ;

const char * program_name (const char * argv0) ;
```

config.h

```
/* src/config.h.  Generated from config.h.in by configure.  */
/* src/config.h.in.  Generated from configure.ac by autoheader.  */

/* Set to 1 if the compile is GNU GCC. */
#define COMPILER_IS_GCC 1

/* Target processor clips on negative float to int conversion. */
#define CPU_CLIPS_NEGATIVE 1

/* Target processor clips on positive float to int conversion. */
#define CPU_CLIPS_POSITIVE 0

/* Target processor is big endian. */
#define CPU_IS_BIG_ENDIAN 0

/* Target processor is little endian. */
#define CPU_IS_LITTLE_ENDIAN 1

/* Set to 1 to enable experimental code. */
#define ENABLE_EXPERIMENTAL_CODE 0

/* Define to 1 if you have the <alsa/asoundlib.h> header file. */
#define HAVE_ALSA_ASOUNDLIB_H 1
```

```
/* Define to 1 if you have the <byteswap.h> header file. */
#define HAVE_BYTESWAP_H 1

/* Define to 1 if you have the `calloc' function. */
#define HAVE_CALLOC 1

/* Define to 1 if you have the `ceil' function. */
#define HAVE_CEIL 1

/* Set to 1 if S_IRGRP is defined. */
#define HAVE_DECL_S_IRGRP 1

/* Define to 1 if you have the <dlfcn.h> header file. */
#define HAVE_DLFCN_H 1

/* Define to 1 if you have the <endian.h> header file. */
#define HAVE_ENDIAN_H 1

/* Will be set to 1 if flac, ogg and vorbis are available. */
#define HAVE_EXTERNAL_LIBS 1

/* Set to 1 if the compile supports the struct hack. */
#define HAVE_FLEXIBLE_ARRAY 1

/* Define to 1 if you have the `floor' function. */
#define HAVE_FLOOR 1

/* Define to 1 if you have the `fmod' function. */
#define HAVE_FMOD 1

/* Define to 1 if you have the `free' function. */
#define HAVE_FREE 1

/* Define to 1 if you have the `fstat' function. */
#define HAVE_FSTAT 1

/* Define to 1 if you have the `fsync' function. */
#define HAVE_FSYNC 1

/* Define to 1 if you have the `ftruncate' function. */
#define HAVE_FTRUNCATE 1

/* Define to 1 if you have the `getpagesize' function. */
#define HAVE_GETPAGESIZE 1

/* Define to 1 if you have the `gettimeofday' function. */
#define HAVE_GETTIMEOFDAY 1

/* Define to 1 if you have the `gmtime' function. */
#define HAVE_GMTIME 1

/* Define to 1 if you have the `gmtime_r' function. */
#define HAVE_GMTIME_R 1

/* Define to 1 if you have the <inttypes.h> header file. */
#define HAVE_INTTYPES_H 1

/* Define to 1 if you have the `m' library (-lm). */
#define HAVE_LIBM 1

/* Define to 1 if you have the <locale.h> header file. */
#define HAVE_LOCALE_H 1
```

```
/* Define to 1 if you have the `localtime' function. */
#define HAVE_LOCALTIME 1

/* Define to 1 if you have the `localtime_r' function. */
#define HAVE_LOCALTIME_R 1

/* Define if you have C99's lrint function. */
#define HAVE_LRINT 1

/* Define if you have C99's lrintf function. */
#define HAVE_LRINTF 1

/* Define to 1 if you have the `lseek' function. */
#define HAVE_LSEEK 1

/* Define to 1 if you have the `malloc' function. */
#define HAVE_MALLOC 1

/* Define to 1 if you have the <memory.h> header file. */
#define HAVE_MEMORY_H 1

/* Define to 1 if you have the `mmap' function. */
#define HAVE_MMAP 1

/* Define to 1 if you have the `open' function. */
#define HAVE_OPEN 1

/* Define to 1 if you have the `pipe' function. */
#define HAVE_PIPE 1

/* Define to 1 if you have the `pread' function. */
#define HAVE_PREAD 1

/* Define to 1 if you have the `pwrite' function. */
#define HAVE_PWRITE 1

/* Define to 1 if you have the `read' function. */
#define HAVE_READ 1

/* Define to 1 if you have the `realloc' function. */
#define HAVE_REALLOC 1

/* Define to 1 if you have the `setlocale' function. */
#define HAVE_SETLOCALE 1

/* Define to 1 if you have the <sndio.h> header file. */
/* #undef HAVE_SNDIO_H */

/* Define to 1 if you have the `snprintf' function. */
#define HAVE_SNPRINTF 1

/* Set to 1 if you have libsqlite3. */
#define HAVE_SQLITE3 0

/* Define to 1 if the system has the type `ssize_t'. */
#define HAVE_SSIZE_T 1

/* Define to 1 if you have the <stdint.h> header file. */
#define HAVE_STDINT_H 1

/* Define to 1 if you have the <stdlib.h> header file. */
#define HAVE_STDLIB_H 1
```

```
/* Define to 1 if you have the <strings.h> header file. */
#define HAVE_STRINGS_H 1

/* Define to 1 if you have the <string.h> header file. */
#define HAVE_STRING_H 1

/* Define to 1 if you have the <sys/stat.h> header file. */
#define HAVE_SYS_STAT_H 1

/* Define to 1 if you have the <sys/time.h> header file. */
#define HAVE_SYS_TIME_H 1

/* Define to 1 if you have the <sys/types.h> header file. */
#define HAVE_SYS_TYPES_H 1

/* Define to 1 if you have <sys/wait.h> that is POSIX.1 compatible. */
#define HAVE_SYS_WAIT_H 1

/* Define to 1 if you have the <unistd.h> header file. */
#define HAVE_UNISTD_H 1

/* Define to 1 if you have the `vsnprintf' function. */
#define HAVE_VSNPRINTF 1

/* Define to 1 if you have the `waitpid' function. */
#define HAVE_WAITPID 1

/* Define to 1 if you have the `write' function. */
#define HAVE_WRITE 1

/* Define to the sub-directory in which libtool stores uninstalled
libraries.
   */
#define LT_OBJDIR ".libs/"

/* Define to 1 if your C compiler doesn't accept -c and -o together. */
/* #undef NO_MINUS_C_MINUS_O */

/* Set to 1 if compiling for MacOSX */
#define OS_IS_MACOSX 0

/* Set to 1 if compiling for Win32 */
#define OS_IS_WIN32 0

/* Name of package */
#define PACKAGE "libsndfile"

/* Define to the address where bug reports for this package should be sent.
*/
#define PACKAGE_BUGREPORT "sndfile@mega-nerd.com"

/* Define to the full name of this package. */
#define PACKAGE_NAME "libsndfile"

/* Define to the full name and version of this package. */
#define PACKAGE_STRING "libsndfile 1.0.25"

/* Define to the one symbol short name of this package. */
#define PACKAGE_TARNAME "libsndfile"

/* Define to the home page for this package. */
#define PACKAGE_URL "http://www.mega-nerd.com/libsndfile/"
```

```
/* Define to the version of this package. */
#define PACKAGE_VERSION "1.0.25"

/* Set to maximum allowed value of sf_count_t type. */
#define SF_COUNT_MAX 0x7FFFFFFFFFFFFFFFLL

/* The size of `double', as computed by sizeof. */
#define SIZEOF_DOUBLE 8

/* The size of `float', as computed by sizeof. */
#define SIZEOF_FLOAT 4

/* The size of `int', as computed by sizeof. */
#define SIZEOF_INT 4

/* The size of `int64_t', as computed by sizeof. */
#define SIZEOF_INT64_T 8

/* The size of `loff_t', as computed by sizeof. */
#define SIZEOF_LOFF_T 8

/* The size of `long', as computed by sizeof. */
#define SIZEOF_LONG 4

/* The size of `long long', as computed by sizeof. */
#define SIZEOF_LONG_LONG 8

/* The size of `off64_t', as computed by sizeof. */
#define SIZEOF_OFF64_T 0

/* The size of `off_t', as computed by sizeof. */
#define SIZEOF_OFF_T 8

/* Set to sizeof (long) if unknown. */
#define SIZEOF_SF_COUNT_T 8

/* The size of `short', as computed by sizeof. */
#define SIZEOF_SHORT 2

/* The size of `size_t', as computed by sizeof. */
#define SIZEOF_SIZE_T 4

/* The size of `ssize_t', as computed by sizeof. */
#define SIZEOF_SSIZE_T 4

/* The size of `void*', as computed by sizeof. */
#define SIZEOF_VOIDP 4

/* The size of `wchar_t', as computed by sizeof. */
#define SIZEOF_WCHAR_T 4

/* Define to 1 if you have the ANSI C header files. */
#define STDC_HEADERS 1

/* Set to long if unknown. */
#define TYPEOF_SF_COUNT_T int64_t

/* Set to 1 to use the native windows API */
#define USE_WINDOWS_API 0

/* Version number of package */
#define VERSION "1.0.25"
```

```
/* Set to 1 if windows DLL is being built. */
#define WIN32_TARGET_DLL 0

/* Target processor is big endian. */
#define WORDS_BIGENDIAN 0

/* Number of bits in a file offset, on hosts where this is settable. */
#define _FILE_OFFSET_BITS 64

/* Define to make fseeko etc. visible, on some hosts. */
#define _LARGEFILE_SOURCE 1

/* Define for large files, on AIX-style hosts. */
/* #undef _LARGE_FILES */

/* Set to 1 to use C99 printf/snprintf in MinGW. */
/* #undef __USE_MINGW_ANSI_STDIO */
```

codec_estetoscopia.h

```
#ifndef milibreria_H
#include "config.h"

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <math.h>
#include <ctype.h>
#include <stdint.h>

#include <samplerate.h>
#include <sndfile.h>

#define DEFAULT_CONVERTER SRC_SINC_MEDIUM_QUALITY

#define     BUFFER_LEN        4096  /*-(1<<16)-*/

char file_pass[50];

typedef     struct
{     char  *infilename, *outfilename ;
      SF_INFO     infileinfo, outfileinfo ;
} OptionData ;

static void copy_metadata (SNDFILE *outfile, SNDFILE *infile, int
channels) ;


static void usage_exit (const char *progname) ;
static sf_count_t sample_rate_convert (SNDFILE *infile, SNDFILE *outfile,
int converter, double src_ratio, int channels, double * gain) ;
static double apply_gain (float * data, long frames, int channels, double
max, double gain) ;

char *to_4khz (char * audio_in_name);
char *to_pcm16 (char * audio_in_name2);
char *encode_adpcm(char *audio_in_name3);
char *decode_adpcm(char *audio_in_name4);
char *to_float32(char *audio_in_name5);
```

```
char *to_44khz (char *audio_in_name6);

#endif /* milibreria_H */
```

codec_estetoscopia.c

```
#include "codec-estetoscopia.h"

int main (int argc, char *argv [])
{
      char *audio_name= NULL, *audio_16 = NULL, *audio_4khz = NULL,
      *audio_adpcm = NULL, *audio_adpcm_deco = NULL, *audio_32 = NULL,
      *audio_44khz = NULL;
      int a;

      audio_name = argv[argc -1];

      audio_4khz = to_4khz (audio_name);
      audio_16 = to_pcm16 (audio_4khz);
      audio_adpcm = encode_adpcm(audio_16);
      audio_adpcm_deco = decode_adpcm(audio_adpcm);
      audio_32 = to_float32(audio_adpcm_deco);
      audio_44khz = to_44khz(audio_32);


      return 0;
}
```

to_4khz.c

```
#include "codec-estetoscopia.h"

char *to_4khz (char *audio_in_name)
{     SNDFILE     *infile = NULL, *outfile = NULL ;
      SF_INFO sfinfo ;

      sf_count_t  count ;
      double      src_ratio = -1.0, gain = 1.0 ;
      int         new_sample_rate = -1, k, converter, max_speed =
                  SF_FALSE, i;
      char *outfilename = NULL, *infilename = NULL, *pos2 = NULL, car
            = '.';

      /* Set default converter. */
      converter = DEFAULT_CONVERTER ;

      new_sample_rate = 4000;

      if (src_get_name (converter) == NULL)
      {     printf ("Error : bad converter number.\n") ;
            } ;

      if ((infile = sf_open (audio_in_name, SFM_READ, &sfinfo)) ==
            NULL)
      {     printf ("Error : Not able to open input file '%s'\n",
                        audio_in_name) ;
            exit (1) ;
            } ;
```

```
printf ("Input File    : %s\n", audio_in_name) ;
printf ("Sample Rate   : %d\n", sfinfo.samplerate) ;
printf ("Input Frames  : %ld\n\n", (long) sfinfo.frames) ;

if (new_sample_rate > 0)
{     src_ratio = (1.0 * new_sample_rate) / sfinfo.samplerate ;
      sfinfo.samplerate = new_sample_rate ;
      }
else if (src_is_valid_ratio (src_ratio))
      sfinfo.samplerate = (int) floor (sfinfo.samplerate *
      src_ratio) ;
else
{     printf ("Not able to determine new sample rate.
      Exiting.\n") ;
      sf_close (infile) ;
      exit (1) ;
      } ;

if (fabs (src_ratio - 1.0) < 1e-20)
{     printf ("Target samplerate and input samplerate are the
      same. Exiting.\n") ;
      sf_close (infile) ;
      exit (0) ;
      } ;

printf ("SRC Ratio     : %f\n", src_ratio) ;
printf ("Converter     : %s\n\n", src_get_name (converter)) ;

if (src_is_valid_ratio (src_ratio) == 0)
{     printf ("Error : Sample rate change out of valid
      range.\n") ;
      sf_close (infile) ;
      exit (1) ;
      } ;

outfilename = audio_in_name;

pos2 = strchr(outfilename, car);

*pos2 = '_'; pos2++; *pos2 = '4'; pos2++; *pos2 = 'k'; pos2++;
*pos2 = 'h'; pos2++; *pos2 = 'z'; pos2++; *pos2 = '.'; pos2++;
*pos2 = 'w'; pos2++; *pos2 = 'a'; pos2++; *pos2 = 'v'; pos2 ++;
*pos2 = '\0';

/* Delete the output file length to zero if already exists. */
//remove (argv [argc - 1]) ;

printf ("Output file   : %s\n", audio_in_name) ;
printf ("Sample Rate   : %d\n", sfinfo.samplerate) ;

do
{     sf_close (outfile) ;

      if ((outfile = sf_open (outfilename, SFM_RDWR, &sfinfo)) ==
          NULL)
      {     printf ("Error : Not able to open output file '%s'\n",
                  outfilename) ;
            sf_close (infile) ;
            exit (1) ;
            } ;

      if (max_speed)
```

```
            {
                  sf_command (outfile, SFC_SET_ADD_PEAK_CHUNK, NULL,
                              SF_FALSE) ;
                  }
            else
            {     /* Update the file header after every write. */
                  sf_command (outfile, SFC_SET_UPDATE_HEADER_AUTO, NULL,
                  SF_TRUE) ;
                  } ;

            sf_command (outfile, SFC_SET_CLIPPING, NULL, SF_TRUE) ;

            count = sample_rate_convert (infile, outfile, converter,
                    src_ratio, sfinfo.channels, &gain) ;
            }
      while (count < 0) ;

      printf ("Output Frames : %ld\n\n", (long) count) ;



      /*for (i=0; i< 16 ; i++)
      {
            file_pass[i] = *outfilename;
            outfilename ++;
      }

      for (i=0; i< 16 ; i++)
      {

            outfilename --;
      }*/


      sf_close (infile) ;
      sf_close (outfile) ;
      return outfilename;
} /* main */

/*==========================================================================*/

static sf_count_t
sample_rate_convert (SNDFILE *infile, SNDFILE *outfile, int converter,
double src_ratio, int channels, double * gain)
{     static float input [BUFFER_LEN] ;
      static float output [BUFFER_LEN] ;

      SRC_STATE   *src_state ;
      SRC_DATA    src_data ;
      int               error ;
      double            max = 0.0 ;
      sf_count_t  output_count = 0 ;

      sf_seek (infile, 0, SEEK_SET) ;
      sf_seek (outfile, 0, SEEK_SET) ;

      /* Initialize the sample rate converter. */
      if ((src_state = src_new (converter, channels, &error)) == NULL)
      {     printf ("\n\nError : src_new() failed : %s.\n\n",
                  src_strerror (error)) ;
            exit (1) ;
            } ;
```

```
	src_data.end_of_input = 0 ; /* Set this later. */

	/* Start with zero to force load in while loop. */
	src_data.input_frames = 0 ;
	src_data.data_in = input ;

	src_data.src_ratio = src_ratio ;

	src_data.data_out = output ;
	src_data.output_frames = BUFFER_LEN /channels ;

	while (1)
	{
		/* If the input buffer is empty, refill it. */
		if (src_data.input_frames == 0)
		{	src_data.input_frames = sf_readf_float (infile, input,
									BUFFER_LEN / channels) ;
			src_data.data_in = input ;

			/* The last read will not be a full buffer, so
			snd_of_input. */

			if (src_data.input_frames < BUFFER_LEN / channels)
				src_data.end_of_input = SF_TRUE ;
			} ;

		if ((error = src_process (src_state, &src_data)))
		{	printf ("\nError : %s\n", src_strerror (error)) ;
			exit (1) ;
			} ;

		/* Terminate if done. */
		if (src_data.end_of_input && src_data.output_frames_gen ==
	    0)
			break ;

		max = apply_gain (src_data.data_out,
		src_data.output_frames_gen, channels, max, *gain) ;

		/* Write output. */
		sf_writef_float (outfile, output,
		src_data.output_frames_gen) ;
		output_count += src_data.output_frames_gen ;

		src_data.data_in += src_data.input_frames_used * channels ;
		src_data.input_frames -= src_data.input_frames_used ;
		} ;

	src_state = src_delete (src_state) ;

	if (max > 1.0)
	{	*gain = 1.0 / max ;
		printf ("\nOutput has clipped. Restarting conversion to
				prevent clipping.\n\n") ;
		return -1 ;
		} ;

	return output_count ;
} /* sample_rate_convert */

static double
apply_gain (float * data, long frames, int channels, double max, double
gain)
```

```
{
      long k ;

      for (k = 0 ; k < frames * channels ; k++)
      {     data [k] *= gain ;

            if (fabs (data [k]) > max)
                  max = fabs (data [k]) ;
            } ;

      return max ;
} /* apply_gain */

/*==============================================================================*/
```

to_44,1k.c

```
#include "codec-estetoscopia.h"


char *to_44khz (char *audio_in_name6)
{     SNDFILE     *infile, *outfile = NULL ;
      SF_INFO sfinfo ;

      sf_count_t  count ;
      double      src_ratio = -1.0, gain = 1.0 ;
      int         new_sample_rate = -1, k, converter, max_speed =
                  SF_FALSE,i ;
      char *outfilename = NULL, *infilename = NULL, *pos2 = NULL, car
            = '.';


      /* Set default converter. */
      converter = DEFAULT_CONVERTER ;

      new_sample_rate = 44100;


      if (src_get_name (converter) == NULL)
      {     printf ("Error : bad converter number.\n") ;

            } ;

      if ((infile = sf_open (audio_in_name6, SFM_READ, &sfinfo)) ==
         NULL)
      {     printf ("Error : Not able to open input file '%s'\n",
                        audio_in_name6) ;
            exit (1) ;
            } ;

      printf ("Input File    : %s\n", audio_in_name6) ;
      printf ("Sample Rate   : %d\n", sfinfo.samplerate) ;
      printf ("Input Frames  : %ld\n\n", (long) sfinfo.frames) ;

      if (new_sample_rate > 0)
      {     src_ratio = (1.0 * new_sample_rate) / sfinfo.samplerate ;
            sfinfo.samplerate = new_sample_rate ;
            }
      else if (src_is_valid_ratio (src_ratio))
```

```
      sfinfo.samplerate = (int) floor (sfinfo.samplerate *
                           src_ratio) ;
else
{     printf ("Not able to determine new sample rate.
               Exiting.\n") ;
      sf_close (infile) ;
      exit (1) ;
      } ;

if (fabs (src_ratio - 1.0) < 1e-20)
{     printf ("Target samplerate and input samplerate are the
             same. Exiting.\n") ;
      sf_close (infile) ;
      exit (0) ;
      } ;

printf ("SRC Ratio     : %f\n", src_ratio) ;
printf ("Converter     : %s\n\n", src_get_name (converter)) ;

if (src_is_valid_ratio (src_ratio) == 0)
{     printf ("Error : Sample rate change out of valid
               range.\n") ;
      sf_close (infile) ;
      exit (1) ;
      } ;

outfilename = audio_in_name6;

pos2 = strchr(outfilename, car);

*pos2 = '_'; pos2++; *pos2 = '4'; pos2++; *pos2 = '4'; pos2++;
*pos2 = 'k'; pos2++; *pos2 = 'h'; pos2++; *pos2 = 'z'; pos2++;
*pos2 = '.'; pos2++; *pos2 = 'w'; pos2++; *pos2 = 'a'; pos2 ++;
*pos2 = 'v'; pos2 ++; *pos2 = '\0';


/* Delete the output file length to zero if already exists. */
//remove (argv [argc - 1]) ;

printf ("Output file   : %s\n", audio_in_name6) ;
printf ("Sample Rate   : %d\n", sfinfo.samplerate) ;

do
{     sf_close (outfile) ;

      if ((outfile = sf_open (outfilename, SFM_WRITE, &sfinfo))
          == NULL)
      {     printf ("Error : Not able to open output file '%s'\n",
            outfilename) ;
            sf_close (infile) ;
            exit (1) ;
            } ;

      if (max_speed)
      {
            sf_command (outfile, SFC_SET_ADD_PEAK_CHUNK, NULL,
            SF_FALSE) ;
            }
      else
      {     /* Update the file header after every write. */
            sf_command (outfile, SFC_SET_UPDATE_HEADER_AUTO, NULL,
                        SF_TRUE) ;
            } ;
```

```
            sf_command (outfile, SFC_SET_CLIPPING, NULL, SF_TRUE) ;

            count = sample_rate_convert (infile, outfile, converter,
                                src_ratio, sfinfo.channels, &gain) ;
            }
      while (count < 0) ;

      printf ("Output Frames : %ld\n\n", (long) count) ;

      sf_close (infile) ;
      sf_close (outfile) ;

      return outfilename;
} /* main */

/*==============================================================*/

static sf_count_t
sample_rate_convert (SNDFILE *infile, SNDFILE *outfile, int converter,
double src_ratio, int channels, double * gain)
{     static float input [BUFFER_LEN] ;
      static float output [BUFFER_LEN] ;

      SRC_STATE   *src_state ;
      SRC_DATA    src_data ;
      int               error ;
      double            max = 0.0 ;
      sf_count_t  output_count = 0 ;

      sf_seek (infile, 0, SEEK_SET) ;
      sf_seek (outfile, 0, SEEK_SET) ;

      /* Initialize the sample rate converter. */
      if ((src_state = src_new (converter, channels, &error)) == NULL)
      {     printf ("\n\nError : src_new() failed : %s.\n\n",
                  src_strerror (error)) ;
            exit (1) ;
            } ;

      src_data.end_of_input = 0 ; /* Set this later. */

      /* Start with zero to force load in while loop. */
      src_data.input_frames = 0 ;
      src_data.data_in = input ;

      src_data.src_ratio = src_ratio ;

      src_data.data_out = output ;
      src_data.output_frames = BUFFER_LEN /channels ;

      while (1)
      {
            /* If the input buffer is empty, refill it. */
            if (src_data.input_frames == 0)
            {     src_data.input_frames = sf_readf_float (infile, input,
                                          BUFFER_LEN / channels) ;
                  src_data.data_in = input ;

                  /* The last read will not be a full buffer, so s
                        nd_of_input. */
                  if (src_data.input_frames < BUFFER_LEN / channels)
                        src_data.end_of_input = SF_TRUE ;
```

```
                } ;

            if ((error = src_process (src_state, &src_data)))
            {     printf ("\nError : %s\n", src_strerror (error)) ;
                  exit (1) ;
                  } ;

            /* Terminate if done. */
            if (src_data.end_of_input && src_data.output_frames_gen ==
          0)
                  break ;

            max = apply_gain (src_data.data_out,
            src_data.output_frames_gen, channels, max, *gain) ;

            /* Write output. */
            sf_writef_float (outfile, output,
            src_data.output_frames_gen) ;
            output_count += src_data.output_frames_gen ;

            src_data.data_in += src_data.input_frames_used * channels ;
            src_data.input_frames -= src_data.input_frames_used ;
            } ;

      src_state = src_delete (src_state) ;

      if (max > 1.0)
      {     *gain = 1.0 / max ;
            printf ("\nOutput has clipped. Restarting conversion to
                    prevent clipping.\n\n") ;
            return -1 ;
            } ;

      return output_count ;
} /* sample_rate_convert */

static double
apply_gain (float * data, long frames, int channels, double max, double
gain)
{
      long k ;

      for (k = 0 ; k < frames * channels ; k++)
      {     data [k] *= gain ;

            if (fabs (data [k]) > max)
                  max = fabs (data [k]) ;
            } ;

      return max ;
} /* apply_gain */
```

to_pcm16.c

```
#include "codec-estetoscopia.h"


char *to_pcm16(char *audio_in_name2)
{     char  *progname, *infilename, *outfilename;
      SNDFILE     *infile = NULL, *outfile = NULL ;
      SF_INFO     sfinfo ;
```

```
int          k, outfilemajor, outfileminor = 0, infileminor, i ;
char   nombre_cadena[30];

char        car = '.', *pos2= NULL;


infilename = audio_in_name2 ;


outfileminor = SF_FORMAT_PCM_16;


memset (&sfinfo, 0, sizeof (sfinfo)) ;

if ((infile = sf_open (audio_in_name2, SFM_READ, &sfinfo)) ==
    NULL)
{     printf ("Not able to open input file %s.\n", infilename) ;
      puts (sf_strerror (NULL)) ;
      return  ;
      } ;

infileminor = sfinfo.format & SF_FORMAT_SUBMASK ;


for (i=0; audio_in_name2[i];i++)
nombre_cadena[i] = audio_in_name2[i];

outfilename = nombre_cadena;

pos2 = strchr(outfilename, car);

*pos2 = '_'; pos2++; *pos2 = '1'; pos2++; *pos2 = '6'; pos2++;
*pos2 = '.'; pos2++; *pos2 = 'w'; pos2++;
*pos2 = 'a'; pos2++; *pos2 = 'v'; pos2 ++; *pos2 = '\0';


if ((sfinfo.format = sfe_file_type_of_ext (outfilename,
      sfinfo.format)) == 0)
{     printf ("Error : Not able to determine output file type
      for %s.\n", outfilename) ;
      return ;
      } ;

outfilemajor = sfinfo.format & (SF_FORMAT_TYPEMASK |
SF_FORMAT_ENDMASK) ;

if (outfileminor == 0)
      outfileminor = sfinfo.format & SF_FORMAT_SUBMASK ;

if (outfileminor != 0)
      sfinfo.format = outfilemajor | outfileminor ;
else
      sfinfo.format = outfilemajor | (sfinfo.format &
                        SF_FORMAT_SUBMASK) ;

if ((sfinfo.format & SF_FORMAT_TYPEMASK) == SF_FORMAT_XI)
      switch (sfinfo.format & SF_FORMAT_SUBMASK)
      {     case SF_FORMAT_PCM_16 :
                        sfinfo.format = outfilemajor |
                        SF_FORMAT_DPCM_16 ;
                        break ;

            case SF_FORMAT_PCM_S8 :
```

```
                case SF_FORMAT_PCM_U8 :
                            sfinfo.format = outfilemajor |
                            SF_FORMAT_DPCM_8 ;
                            break ;
                } ;

      if (sf_format_check (&sfinfo) == 0)
      {     printf ("Error : output file format is invalid (0x%08X).\n",
                  sfinfo.format) ;
            return  ;
            } ;

      /* Open the output file. */
      if ((outfile = sf_open (outfilename, SFM_RDWR, &sfinfo)) ==
         NULL)
      {     printf ("Not able to open output file %s : %s\n",
                   outfilename, sf_strerror (NULL)) ;
            return  ;
            } ;

      /* Copy the metadata */
      copy_metadata (outfile, infile, sfinfo.channels) ;

      if ((outfileminor == SF_FORMAT_DOUBLE) || (outfileminor ==
      SF_FORMAT_FLOAT) || (infileminor == SF_FORMAT_DOUBLE) ||
      (infileminor == SF_FORMAT_FLOAT)|| (infileminor ==
      SF_FORMAT_VORBIS) || (outfileminor == SF_FORMAT_VORBIS))
            sfe_copy_data_fp (outfile, infile, sfinfo.channels) ;
      else
            sfe_copy_data_int (outfile, infile, sfinfo.channels) ;

      sf_close (infile) ;
      sf_close (outfile) ;

      printf("Codificacion a PCM 16 bits realizada con exito del
      audio: %s\n", infilename);

      for (i=0; i< 19 ; i++)
      {
            file_pass[i] = *outfilename;
            outfilename ++;
      }

      for (i=0; i< 19 ; i++)
      {

            outfilename --;
      }

      return file_pass ;
} /* main */

static void
copy_metadata (SNDFILE *outfile, SNDFILE *infile, int channels)
{     SF_INSTRUMENT inst ;
      SF_BROADCAST_INFO_2K binfo ;
      const char *str ;
      int k, chanmap [256] ;

      for (k = SF_STR_FIRST ; k <= SF_STR_LAST ; k++)
      {     str = sf_get_string (infile, k) ;
            if (str != NULL)
                  sf_set_string (outfile, k, str) ;
```

```
        } ;
    memset (&inst, 0, sizeof (inst)) ;
    memset (&binfo, 0, sizeof (binfo)) ;

    if (channels < ARRAY_LEN (chanmap))
    {   size_t size = channels * sizeof (chanmap [0]) ;

        if (sf_command (infile, SFC_GET_CHANNEL_MAP_INFO, chanmap,
        size) == SF_TRUE)
            sf_command (outfile, SFC_SET_CHANNEL_MAP_INFO,
            chanmap, size) ;
        } ;

    if (sf_command (infile, SFC_GET_INSTRUMENT, &inst, sizeof
        (inst)) == SF_TRUE)
        sf_command (outfile, SFC_SET_INSTRUMENT, &inst, sizeof
        (inst)) ;

    if (sf_command (infile, SFC_GET_BROADCAST_INFO, &binfo, sizeof
        (binfo)) == SF_TRUE)
        sf_command (outfile, SFC_SET_BROADCAST_INFO, &binfo, sizeof
        (binfo)) ;

} /* copy_metadata */
```

to_float32.c

```
#include "codec-estetoscopia.h"


char *to_float32(char *audio_in_name5)
{   char  *progname, *infilename, *outfilename ;
    SNDFILE     *infile = NULL, *outfile = NULL ;
    SF_INFO     sfinfo ;
    int         k, outfilemajor, outfileminor = 0, infileminor, i ;
    char   nombre_cadena[50];

    char  car = '.', *pos2= NULL;


    infilename = audio_in_name5 ;


    outfileminor = SF_FORMAT_FLOAT;


    memset (&sfinfo, 0, sizeof (sfinfo)) ;

    if ((infile = sf_open (infilename, SFM_READ, &sfinfo)) == NULL)
    {   printf ("Not able to open input file %s.\n", infilename) ;
        puts (sf_strerror (NULL)) ;
        return ;
        } ;

    infileminor = sfinfo.format & SF_FORMAT_SUBMASK ;

    for (i=0; audio_in_name5[i];i++)
    nombre_cadena[i] = audio_in_name5[i];

    outfilename = nombre_cadena;
```

```
pos2 = strchr(outfilename, car);

*pos2 = '_'; pos2++; *pos2 = '3'; pos2++; *pos2 = '2'; pos2++;
*pos2 = '.'; pos2++; *pos2 = 'w'; pos2++;
*pos2 = 'a'; pos2++; *pos2 = 'v'; pos2 ++; *pos2 = '\0';


if ((sfinfo.format = sfe_file_type_of_ext (outfilename,
      sfinfo.format)) == 0)
{     printf ("Error : Not able to determine output file type
      for %s.\n", outfilename) ;
      return ;
      } ;

outfilemajor = sfinfo.format & (SF_FORMAT_TYPEMASK |
SF_FORMAT_ENDMASK) ;

if (outfileminor == 0)
      outfileminor = sfinfo.format & SF_FORMAT_SUBMASK ;

if (outfileminor != 0)
      sfinfo.format = outfilemajor | outfileminor ;
else
      sfinfo.format = outfilemajor | (sfinfo.format &
      SF_FORMAT_SUBMASK) ;

if ((sfinfo.format & SF_FORMAT_TYPEMASK) == SF_FORMAT_XI)
      switch (sfinfo.format & SF_FORMAT_SUBMASK)
      {     case SF_FORMAT_PCM_16 :
                        sfinfo.format = outfilemajor |
                        SF_FORMAT_DPCM_16 ;
                        break ;

                        case SF_FORMAT_PCM_S8 :
                        case SF_FORMAT_PCM_U8 :
                        sfinfo.format = outfilemajor |
                        SF_FORMAT_DPCM_8 ;
                        break ;
            } ;

if (sf_format_check (&sfinfo) == 0)
{     printf ("Error : output file format is invalid (0x%08X).\n",
            sfinfo.format) ;
      return  ;
      } ;

/* Open the output file. */
if ((outfile = sf_open (outfilename, SFM_WRITE, &sfinfo)) ==
      NULL)
{     printf ("Not able to open output file %s : %s\n",
            outfilename, sf_strerror (NULL)) ;
      return  ;
      } ;

/* Copy the metadata */
copy_metadata (outfile, infile, sfinfo.channels) ;

if ((outfileminor == SF_FORMAT_DOUBLE) || (outfileminor ==
SF_FORMAT_FLOAT)  || (infileminor == SF_FORMAT_DOUBLE) ||
(infileminor == SF_FORMAT_FLOAT)|| (infileminor ==
SF_FORMAT_VORBIS) || (outfileminor == SF_FORMAT_VORBIS))
      sfe_copy_data_fp (outfile, infile, sfinfo.channels) ;
else
```

```
            sfe_copy_data_int (outfile, infile, sfinfo.channels) ;

      sf_close (infile) ;
      sf_close (outfile) ;

      printf("Codificacion a Float 32 bits realizada con exito del
            audio %s\n", infilename);

      for (i=0; i< 33 ; i++)
      {
            file_pass[i] = *outfilename;
            outfilename ++;
      }

      for (i=0; i< 33 ; i++)
      {

            outfilename --;
      }

      return file_pass ;

} /* main */

static void
copy_metadata (SNDFILE *outfile, SNDFILE *infile, int channels)
{     SF_INSTRUMENT inst ;
      SF_BROADCAST_INFO_2K binfo ;
      const char *str ;
      int k, chanmap [256] ;

      for (k = SF_STR_FIRST ; k <= SF_STR_LAST ; k++)
      {     str = sf_get_string (infile, k) ;
            if (str != NULL)
                  sf_set_string (outfile, k, str) ;
            } ;

      memset (&inst, 0, sizeof (inst)) ;
      memset (&binfo, 0, sizeof (binfo)) ;

      if (channels < ARRAY_LEN (chanmap))
      {     size_t size = channels * sizeof (chanmap [0]) ;

            if (sf_command (infile, SFC_GET_CHANNEL_MAP_INFO, chanmap,
      size) == SF_TRUE) sf_command (outfile,
      SFC_SET_CHANNEL_MAP_INFO, chanmap, size) ;
            } ;

      if (sf_command (infile, SFC_GET_INSTRUMENT, &inst, sizeof
            (inst)) == SF_TRUE)
            sf_command (outfile, SFC_SET_INSTRUMENT, &inst, sizeof
            (inst)) ;

      if (sf_command (infile, SFC_GET_BROADCAST_INFO, &binfo, sizeof
            (binfo)) == SF_TRUE)
            sf_command (outfile, SFC_SET_BROADCAST_INFO, &binfo, sizeof
            (binfo)) ;

} /* copy_metadata */
```

encode_adpcm.c

```
#include "codec-estetoscopia.h"
#include "common.h"


char *encode_adpcm(char *audio_in_name3)
{     char  *progname, *infilename, *outfilename ;
      SNDFILE     *infile = NULL, *outfile = NULL ;
      SF_INFO     sfinfo ;
      int         k, outfilemajor, outfileminor = 0, infileminor,i ;
      char   nombre_cadena[30];
      char        car = '.', *pos2= NULL;


      infilename = audio_in_name3;
      //outfilename = argv [2] ;


      outfileminor = SF_FORMAT_IMA_ADPCM;


      memset (&sfinfo, 0, sizeof (sfinfo)) ;

      if ((infile = sf_open (infilename, SFM_READ, &sfinfo)) == NULL)
      {     printf ("Not able to open input file %s.\n", infilename) ;
            puts (sf_strerror (NULL)) ;
            return ;
            } ;

      infileminor = sfinfo.format & SF_FORMAT_SUBMASK ;

      for (i=0; audio_in_name3[i];i++)
      nombre_cadena[i] = audio_in_name3[i];

      outfilename = nombre_cadena;

      pos2 = strchr(outfilename, car);

      *pos2 = '_'; pos2++; *pos2 = 'a'; pos2++; *pos2 = 'd'; pos2++;
      *pos2 = 'p'; pos2++; *pos2 = 'c'; pos2++;
      *pos2 = 'm'; pos2++; *pos2 = '.'; pos2 ++; *pos2 = 'w'; pos2 ++;
      *pos2 = 'a';pos2 ++; *pos2 = 'v';pos2 ++; *pos2 = '\0';


      if ((sfinfo.format = sfe_file_type_of_ext (outfilename,
      sfinfo.format)) == 0)
      {     printf ("Error : Not able to determine output file type
      for %s.\n", outfilename) ;
            return ;
            } ;

      outfilemajor = sfinfo.format & (SF_FORMAT_TYPEMASK |
      SF_FORMAT_ENDMASK) ;

      if (outfileminor == 0)
            outfileminor = sfinfo.format & SF_FORMAT_SUBMASK ;

      if (outfileminor != 0)
            sfinfo.format = outfilemajor | outfileminor ;
      else
```

```
            sfinfo.format = outfilemajor | (sfinfo.format &
                              SF_FORMAT_SUBMASK) ;

      if ((sfinfo.format & SF_FORMAT_TYPEMASK) == SF_FORMAT_XI)
            switch (sfinfo.format & SF_FORMAT_SUBMASK)
            {     case SF_FORMAT_PCM_16 :
                              sfinfo.format = outfilemajor |
                              SF_FORMAT_DPCM_16 ;
                              break ;

                  case SF_FORMAT_PCM_S8 :
                  case SF_FORMAT_PCM_U8 :
                              sfinfo.format = outfilemajor |
                              SF_FORMAT_DPCM_8 ;
                              break ;
                  } ;

      if (sf_format_check (&sfinfo) == 0)
      {     printf ("Error : output file format is invalid (0x%08X).\n",
                  sfinfo.format) ;
            return  ;
            } ;

      /* Open the output file. */
      if ((outfile = sf_open (outfilename, SFM_WRITE, &sfinfo)) ==
            NULL)
      {     printf ("Not able to open output file %s : %s\n",
            outfilename, sf_strerror (NULL)) ;
            return  ;
            } ;

      /* Copy the metadata */
      copy_metadata (outfile, infile, sfinfo.channels) ;

      if ((outfileminor == SF_FORMAT_DOUBLE) || (outfileminor ==
      SF_FORMAT_FLOAT)|| (infileminor == SF_FORMAT_DOUBLE) ||
      (infileminor == SF_FORMAT_FLOAT)|| (infileminor ==
      SF_FORMAT_VORBIS) || (outfileminor == SF_FORMAT_VORBIS))
            sfe_copy_data_fp (outfile, infile, sfinfo.channels) ;
      else
            sfe_copy_data_int (outfile, infile, sfinfo.channels) ;

      sf_close (infile) ;
      sf_close (outfile) ;

      printf("Codificacion IMA_ADPCM realizada con exito del
            audio: %s\n", infilename);

      for (i=0; i< 25 ; i++)
      {
            file_pass[i] = *outfilename;
            outfilename ++;
      }

      for (i=0; i< 25 ; i++)
      {

            outfilename --;
      }

      return file_pass ;

} /* main */
```

```
static void
copy_metadata (SNDFILE *outfile, SNDFILE *infile, int channels)
{     SF_INSTRUMENT inst ;
      SF_BROADCAST_INFO_2K binfo ;
      const char *str ;
      int k, chanmap [256] ;

      for (k = SF_STR_FIRST ; k <= SF_STR_LAST ; k++)
      {     str = sf_get_string (infile, k) ;
            if (str != NULL)
                  sf_set_string (outfile, k, str) ;
            } ;

      memset (&inst, 0, sizeof (inst)) ;
      memset (&binfo, 0, sizeof (binfo)) ;

      if (channels < ARRAY_LEN (chanmap))
      {     size_t size = channels * sizeof (chanmap [0]) ;

            if (sf_command (infile, SFC_GET_CHANNEL_MAP_INFO, chanmap,
            size) == SF_TRUE)
                  sf_command (outfile, SFC_SET_CHANNEL_MAP_INFO,
                  chanmap, size) ;
            } ;

      if (sf_command (infile, SFC_GET_INSTRUMENT, &inst, sizeof
            (inst)) == SF_TRUE)
            sf_command (outfile, SFC_SET_INSTRUMENT, &inst, sizeof
            (inst)) ;

      if (sf_command (infile, SFC_GET_BROADCAST_INFO, &binfo, sizeof
            (binfo)) == SF_TRUE)
            sf_command (outfile, SFC_SET_BROADCAST_INFO, &binfo, sizeof
            (binfo)) ;

} /* copy_metadata */
```

decode_adpcm.c

```
#include "codec-estetoscopia.h"


char *decode_adpcm(char *audio_in_name4)
{     char  *progname, *infilename, *outfilename ;
      SNDFILE     *infile = NULL, *outfile = NULL ;
      SF_INFO     sfinfo ;
      int         k, outfilemajor, outfileminor = 0, infileminor,i ;
      char        nombre_cadena[30];
      char        car = '.', *pos2= NULL;

      infilename = audio_in_name4 ;
      //outfilename = argv [2] ;


      outfileminor = SF_FORMAT_PCM_16;


      memset (&sfinfo, 0, sizeof (sfinfo)) ;
```

```
if ((infile = sf_open (infilename, SFM_READ, &sfinfo)) == NULL)
{     printf ("Not able to open input file %s.\n", infilename) ;
      puts (sf_strerror (NULL)) ;
      return  ;
      } ;

infileminor = sfinfo.format & SF_FORMAT_SUBMASK ;

for (i=0; audio_in_name4[i];i++)
nombre_cadena[i] = audio_in_name4[i];

outfilename = nombre_cadena;

pos2 = strchr(outfilename, car);

*pos2 = '_'; pos2++; *pos2 = 'd'; pos2++; *pos2 = 'e'; pos2++;
*pos2 = 'c'; pos2++; *pos2 = 'o'; pos2++;
*pos2 = '.'; pos2++; *pos2 = 'w'; pos2 ++; *pos2 = 'a'; pos2 ++;
*pos2 = 'v';pos2 ++; *pos2 = '\0';


if ((sfinfo.format = sfe_file_type_of_ext (outfilename,
      sfinfo.format)) == 0)
{     printf ("Error : Not able to determine output file type
                  for %s.\n", outfilename) ;
      return  ;
      } ;

outfilemajor = sfinfo.format & (SF_FORMAT_TYPEMASK |
                  SF_FORMAT_ENDMASK) ;

if (outfileminor == 0)
      outfileminor = sfinfo.format & SF_FORMAT_SUBMASK ;

if (outfileminor != 0)
      sfinfo.format = outfilemajor | outfileminor ;
else
      sfinfo.format = outfilemajor | (sfinfo.format &
                        SF_FORMAT_SUBMASK) ;

if ((sfinfo.format & SF_FORMAT_TYPEMASK) == SF_FORMAT_XI)
      switch (sfinfo.format & SF_FORMAT_SUBMASK)
      {     case SF_FORMAT_PCM_16 :
                        sfinfo.format = outfilemajor |
                                    SF_FORMAT_DPCM_16 ;
                        break ;

            case SF_FORMAT_PCM_S8 :
            case SF_FORMAT_PCM_U8 :
                        sfinfo.format = outfilemajor |
                                    SF_FORMAT_DPCM_8 ;
                        break ;
            } ;

if (sf_format_check (&sfinfo) == 0)
{     printf ("Error : output file format is invalid (0x%08X).\n",
      sfinfo.format) ;
      return  ;
      } ;

/* Open the output file. */
if ((outfile = sf_open (outfilename, SFM_WRITE, &sfinfo)) ==
      NULL)
```

```
	{	printf ("Not able to open output file %s : %s\n",
		outfilename, sf_strerror (NULL)) ;
		return ;
		} ;

	/* Copy the metadata */
	copy_metadata (outfile, infile, sfinfo.channels) ;

	if ((outfileminor == SF_FORMAT_DOUBLE) || (outfileminor ==
	SF_FORMAT_FLOAT)|| (infileminor == SF_FORMAT_DOUBLE) ||
	(infileminor == SF_FORMAT_FLOAT)|| (infileminor ==
	SF_FORMAT_VORBIS) || (outfileminor == SF_FORMAT_VORBIS))
		sfe_copy_data_fp (outfile, infile, sfinfo.channels) ;
	else
		sfe_copy_data_int (outfile, infile, sfinfo.channels) ;

	sf_close (infile) ;
	sf_close (outfile) ;

	printf("Decodificacion IMA_ADPCM realizada con exito del
	audio: %s\n",infilename);

	for (i=0; i< 30 ; i++)
	{
		file_pass[i] = *outfilename;
		outfilename ++;
	}

	for (i=0; i< 30 ; i++)
	{

		outfilename --;
	}

	return file_pass;
} /* main */

static void
copy_metadata (SNDFILE *outfile, SNDFILE *infile, int channels)
{	SF_INSTRUMENT inst ;
	SF_BROADCAST_INFO_2K binfo ;
	const char *str ;
	int k, chanmap [256] ;

	for (k = SF_STR_FIRST ; k <= SF_STR_LAST ; k++)
	{	str = sf_get_string (infile, k) ;
		if (str != NULL)
			sf_set_string (outfile, k, str) ;
		} ;

	memset (&inst, 0, sizeof (inst)) ;
	memset (&binfo, 0, sizeof (binfo)) ;

	if (channels < ARRAY_LEN (chanmap))
	{	size_t size = channels * sizeof (chanmap [0]) ;

		if (sf_command (infile, SFC_GET_CHANNEL_MAP_INFO, chanmap,
		size) == SF_TRUE)
			sf_command (outfile, SFC_SET_CHANNEL_MAP_INFO,
			chanmap, size) ;
		} ;
```

```
        if (sf_command (infile, SFC_GET_INSTRUMENT, &inst, sizeof
              (inst)) == SF_TRUE)
              sf_command (outfile, SFC_SET_INSTRUMENT, &inst, sizeof
              (inst)) ;

        if (sf_command (infile, SFC_GET_BROADCAST_INFO, &binfo, sizeof
              (binfo)) == SF_TRUE)
              sf_command (outfile, SFC_SET_BROADCAST_INFO, &binfo, sizeof
              (binfo)) ;

} /* copy_metadata */
```

C. RESULTADOS DETALLADOS DE LAS DIFERENTES CODIFICACIONES

- Resultados obtenidos de los audios adiquiridos con el estetoscopio EHAS-FUNDATEL.

	G.722		
	RMSE	**PRD (%)**	**RTF**
Mitral 1	0,014858	21,645073	0,2112
Mitral 2	0,01725	14,71721	0,1912
Mitral 3	0,013987	10,418758	0,262
Aórtica 1	0,014439	10,910855	0,2226
Aórtica 2	0,019815	13,210282	0,2124
Aórtica 3	0,013831	10,353948	0,2113
Tricúspide 1	0,01473	10,471964	0,1993
Tricúspide 2	0,014212	10,376623	0,2115
Tricúspide 3	0,019584	14,140274	0,222
Pulmonar 1	0,00998	10,793248	0,2014
Pulmonar 2	0,013019	12,069352	0,2441
Pulmonar 3	0,014925	11,011109	0,231
Pulmonar 4	0,024687	10,087489	0,2021
Pulmonar 5	0,030245	11,159951	0,2112
Pulmonar 6	0,02605	13,612425	0,2215
	0,017441	**12,331904**	**0,2169**

Tabla 12 - Resultados figuras de mérito codificador G.722

	G.726-2			G.726-3		
	RMSE	**PRD (%)**	**RTF**	**RMSE**	**PRD (%)**	**RTF**
Mitral 1	0,002393	3,486325	0,03401	0,001227	1,787823	0,0441
Mitral 2	0,004456	3,801875	0,03221	0,001227	2,683048	0,0407
Mitral 3	0,004644	3,459411	0,03053	0,003226	2,402624	0,03988
Aórtica 1	0,004677	3,533877	0,0288	0,00237	1,791049	0,03988
Aórtica 2	0,00592	3,946906	0,02971	0,004195	2,79682	0,03201
Aórtica 3	0,004379	3,278166	0,03055	0,003147	2,355568	0,05033
Tricúspide1	0,005234	3,720728	0,02788	0,003322	2,361797	0,03988
Tricúspide 2	0,004777	3,487964	0,03166	0,003007	2,195563	0,0381
Tricúspide 3	0,004568	3,297875	0,0305	0,002713	1,95851	0,0421
Pulmonar 1	0,00319	3,449782	0,03411	0,001694	1,832644	0,03988
Pulmonar 2	0,005449	5,051981	0,02874	0,004132	3,830783	0,03804
Pulmonar 3	0,004736	3,493964	0,03221	0,003146	2,321366	0,03402
Pulmonar 4	0,008116	3,316241	0,03053	0,003636	1,48593	0,03909
Pulmonar 5	0,008813	3,252022	0,0288	0,006984	2,576972	0,03636
Pulmonar 6	0,006638	3,468687	0,03166	0,002892	1,511126	0,03988
	0,005199	**3,6030536**	**0,03079**	**0,003128**	**2,259442**	**0,03960**

Tabla 13 - Resultados figuras de mérito con el codificador G.726 con dos y tres bits por muestra

	G.726-4			G.726-5		
	RMSE	**PRD (%)**	**RTF**	**RMSE**	**PRD (%)**	**RTF**
Mitral 1	0,001114	1,62353	0,03044	0,00121	1,762765	0,03909
Mitral 2	0,002334	1,991302	0,0407	0,002516	2,146613	0,03636
Mitral 3	0,002787	2,075608	0,03988	0,002994	2,230104	0,042
Aórtica 1	0,002565	1,938077	0,03404	0,00281	2,123256	0,03988
Aórtica 2	0,003214	2,1426	0,03804	0,004302	2,868066	0,03988
Aórtica 3	0,00241	1,803904	0,03402	0,003037	2,273617	0,0388
Tricúspide1	0,002774	1,971928	0,03045	0,003962	2,816432	0,03402
Tricúspide 2	0,002235	1,631864	0,02988	0,002901	2,11808	0.03305
Tricúspide 3	0,002391	1,726431	0,03301	0,003152	2,215926	0,03201
Pulmonar 1	0,001508	1,631267	0,0381	0,001692	1,82984	0,03622
Pulmonar 2	0,003527	3,269703	0,03909	0,003782	3,505943	0,03988
Pulmonar 3	0,002483	1,831523	0,03636	0,002735	2,017523	0,04102
Pulmonar 4	0,003205	1,309495	0,03988	0,005561	2,272277	0,03877
Pulmonar 5	0,005759	2,124921	0,02988	0,008709	3,213561	0,03304
Pulmonar 6	0,002587	1,35178	0,02755	0,003315	1,732094	0,03633
	0,002726	**1,8949289**	**0,03475**	**0,003512**	**2,34174**	**0,03515**

Tabla 14 - Resultados figuras de mérito con el codificador G.726 con cuatro y cinco bits por muestra

	IMA ADPCM		
	RMSE	**PRD (%)**	**RTF**
Mitral 1	0,000402	0,586311	0,00613
Mitral 2	0,000254	0,216516	0,006
Mitral 3	0,00029	0,216302	0,0062
Aórtica 1	0,000344	0,259764	0,0058
Aórtica 2	0,002021	1,347089	0,0054
Aórtica 3	0,000207	0,155276	0,006
Tricúspide1	0,000378	0,268625	0,00613
Tricúspide 2	0,000336	0,245215	0,0062
Tricúspide 3	0,00026	0,187513	0,0059
Pulmonar 1	0,000343	0,371323	0,0059
Pulmonar 2	0,000438	0,40645	0,006
Pulmonar 3	0,000411	0,303575	0,0062
Pulmonar 4	0,000245	0,100233	0,00613
Pulmonar 5	0,000578	0,213247	0,0055
Pulmonar 6	0,000647	0,337923	0,006
	0,000477	**0,3476908**	**0,0061**

Tabla 15 - Resultados figuras de mérito con el codificador IMA ADPCM

	4kHz		
	RMSE	**PRD (%)**	**RTF**
Mitral 1	0,000755	1,10008	0,00448
Mitral 2	0,001169	0,997285	0,0045
Mitral 3	0,000907	0,67595	0,00465
Aórtica 1	0,00036	0,272235	0,0044
Aórtica 2	0,000894	0,596004	0,00448
Aórtica 3	0,000612	0,457928	0,00464
Tricúspide1	0,000619	0,439814	0,0043
Tricúspide 2	0,000412	0,300631	0,0043
Tricúspide 3	0,000417	0,301157	0,00435
Pulmonar 1	0,000272	0,2941	0,004
Pulmonar 2	0,000785	0,727877	0,00448
Pulmonar 3	0,001758	1,296744	0,0044
Pulmonar 4	0,000316	0,128989	0,0043
Pulmonar 5	0,000774	0,285649	0,0045
Pulmonar 6	0,000555	0,28977	0,00465
	0,000707	**0,5442809**	**0,00455**

Tabla 16 - Resultados figuras de mérito con un submuestreo a 4 kHz

	2kHz			1kHz		
	RMSE	**PRD (%)**	**RTF**	**RMSE**	**PRD (%)**	**RTF**
Mitral 1	0,000553	0,805537	0,00446	0,001556	2,26609	0,0044
Mitral 2	0,000793	0,676318	0,00448	0,002403	2,049765	0,00439
Mitral 3	0,000643	0,478583	0,00446	0,001836	1,367887	0,00435
Aórtica 1	0,000584	0,441504	0,0044	0,000836	0,631642	0,00441
Aórtica 2	0,001357	0,904616	0,00435	0,002258	1,505193	0,00446
Aórtica 3	0,000476	0,356242	0,00433	0,001276	0,955242	0,00435
Tricúspide1	0,000701	0,49867	0,0044	0,001313	0,933583	0,0044
Tricúspide 2	0,000518	0,377862	0,00446	0,000955	0,697089	0,00441
Tricúspide 3	0,000532	0,384366	0,00448	0,000968	0,69897	0,00441
Pulmonar 1	0,000342	0,36906	0,0045	0,000559	0,604367	0,00433
Pulmonar 2	0,000862	0,799448	0,0044	0,001541	1,428767	0,00438
Pulmonar 3	0,001163	0,858283	0,00435	0,003606	2,660228	0,00438
Pulmonar 4	0,000333	0,136212	0,00433	0,00075	0,306619	0,00441
Pulmonar 5	0,000976	0,360152	0,0044	0,001577	0,581805	0,0044
Pulmonar 6	0,001169	0,611113	0,00446	0,002039	1,065398	0,00441
	0,000733	**0,5371977**	**0,00442**	**0,001565**	**1,18351**	**0,00441**

Tabla 17 - Resultados figuras de mérito con submuestreos a 2 kHz y 1 kHz

- Resultados obtenidos de los audios cardiados descargados de [3].

	G.722		
	RMSE	**PRD (%)**	**RTF**
Cardiaco 1	0,04298	34,36549	0,2122
Cardiaco 2	0,04451	44,105347	0,1889
Cardiaco 3	0,037163	36,082541	0,2001
Cardiaco 4	0,04735	45,219857	0,2001
Cardiaco 5	0,042598	34,365462	0,1955
Cardiaco 6	0,045491	35,954777	0,1955
Cardiaco 7	0,048508	35,838717	0,221
Cardiaco 8	0,037038	36,219976	0,2211
Cardiaco 9	0,039465	40,610034	0,1889
Cardiaco 10	0,046315	37,48522	0,1966
	0,043142	**38,024742**	**0,20199**

Tabla 18 - Resultados figuras de mérito codificador G.722

	G.726-2			G.726-3		
	RMSE	**PRD (%)**	**RTF**	**RMSE**	**PRD (%)**	**RTF**
Cardiaco 1	0,003799	3,065199	0,03055	0,00199	1,605611	0,03402
Cardiaco 2	0,003255	3,195267	0,02788	0,001746	1,729674	0,0399
Cardiaco 3	0,003199	3,105636	0,03166	0,001776	1,724401	0,03705
Cardiaco 4	0,003326	3,175997	0,0305	0,001848	1,764402	0,03801
Cardiaco 5	0,003751	3,0257	0,03411	0,001977	1,594596	0,03804
Cardiaco 6	0,004014	3,172702	0,02788	0,002046	1,616776	0,03402
Cardiaco 7	0,004198	3,101569	0,03166	0,002324	1,716999	0.03305
Cardiaco 8	0,003296	3,222909	0,0298	0,001645	1,608253	0,03201
Cardiaco 9	0,00318	3,271941	0,03055	0,001613	1,659609	0,03622
Cardiaco 10	0,003988	3,227532	0,03042	0,002032	1,644926	0,03877
	0,003601	**3,1564452**	**0,030501**	**0,0019**	**1,666525**	**0,0328**

Tabla 19 - Resultados figuras de mérito con el codificador G.726 con dos y tres bits por muestra

	G.726-4			G.726-5		
	RMSE	**PRD (%)**	**RTF**	**RMSE**	**PRD (%)**	**RTF**
Cardiaco 1	0,001766	1,424646	0,03402	0,001859	1,499771	0,04041
Cardiaco 2	0,001594	1,579134	0,03305	0,004069	4,031848	0,04303
Cardiaco 3	0,001622	1,575194	0,03201	0,004012	3,895107	0,03305
Cardiaco 4	0,00157	1,499406	0,03122	0,004204	4,015029	0,03988
Cardiaco 5	0,001769	1,427274	0,03402	0,001864	1,503952	0,03988
Cardiaco 6	0,002059	1,627358	0,03988	0,004586	3,624671	0,03201
Cardiaco 7	0,002198	1,623966	0,0381	0,005454	4,029353	0,03988
Cardiaco 8	0,001501	1,467342	0,0355	0,0031	3,031601	0,03622
Cardiaco 9	0,001359	1,398235	0,03636	0,00213	2,192174	0,03877
Cardiaco 10	0,001762	1,426456	0,03303	0,004499	3,64092	0,0455
	0,003601	**3,1564452**	**0,03471**	**0,003578**	**3,146443**	**0,03886**

Tabla 20 - Resultados figuras de mérito con el codificador G.726 con cuatro y cinco bits por muestra

	IMA ADPCM		
	RMSE	**PRD (%)**	**RTF**
Cardiaco 1	0,000196	0,157772	0,00613
Cardiaco 2	0,000205	0,202642	0,00611
Cardiaco 3	0,000182	0,176442	0,0061
Cardiaco 4	0,000216	0,206648	0,0062
Cardiaco 5	0,000196	0,15802	0,0059
Cardiaco 6	0,000206	0,163022	0,00585
Cardiaco 7	0,000214	0,158369	0,006
Cardiaco 8	0,000198	0,193684	0,00613
Cardiaco 9	0,000191	0,196893	0,0062
Cardiaco 10	0,000225	0,181958	0,0061
	0,000203	**0,179545**	**0,00615**

Tabla 21 - Resultados figuras de mérito con el codificador IMA ADPCM

	4kHz		
	RMSE	**PRD (%)**	**RTF**
Cardiaco 1	0,000026	0,021377	0,0044
Cardiaco 2	0,00003	0,029375	0,00448
Cardiaco 3	0,000029	0,028605	0,00446
Cardiaco 4	0,00003	0,02856	0,0045
Cardiaco 5	0,000026	0,021375	0,00445
Cardiaco 6	0,000028	0,021759	0,0044
Cardiaco 7	0,00028	0,020816	0,00444
Cardiaco 8	0,000343	0,335228	0,00446
Cardiaco 9	0,000027	0,027981	0,00448
Cardiaco 10	0,000257	0,288543	0,00448
	0,000108	**0,0823619**	**0,00441**

Tabla 22 - Resultados figuras de mérito con submuestreo a 4 kHz

	2kHz			1kHz		
	RMSE	**PRD (%)**	**RTF**	**RMSE**	**PRD (%)**	**RTF**
Cardiaco 1	0,000027	0,022014	0,00448	0,000036	0,028785	0,0044
Cardiaco 2	0,000031	0,030997	0,00446	0,000042	0,041624	0,00441
Cardiaco 3	0,000031	0,030167	0,00444	0,000043	0,041979	0,00444
Cardiaco 4	0,000032	0,03034	0,00442	0,000072	0,068634	0,0044
Cardiaco 5	0,000027	0,022039	0,0044	0,000036	0,028797	0,00442
Cardiaco 6	0,000031	0,024693	0,0045	0,000049	0,038776	0,00444
Cardiaco 7	0,00005	0,036834	0,00448	0,000175	0,129055	0,0044
Cardiaco 8	0,000379	0,370495	0,00446	0,000401	0,392054	0,00442
Cardiaco 9	0,000029	0,02958	0,00444	0,000037	0,038248	0,0045
Cardiaco 10	0,000396	0,320822	0,00442	0,000433	0,35024	0,0044
	0,000103	**0,0917981**	**0,00448**	**0,000132**	**0,115819**	**0,00442**

Tabla 23 - Resultados figuras de mérito con submuestreos a 2 kHz y 1 kHz

- Resultados obtenidos de los audios respiratorios descargados de [3].

	G.722		
	RMSE	**PRD (%)**	**RTF**
Pulmonar 1	0,041551	14,346854	0,2322
Pulmonar 2	0,060219	37,691633	0,2022
Pulmonar 3	0,059135	33,04555	0,2344
Pulmonar 4	0,073357	42,879422	0,2112
Pulmonar 5	0,07803	44,00873	0,1976
Pulmonar 6	0,049254	21,517068	0,2221
Pulmonar 7	0,05291	30,046453	0,2102
Pulmonar 8	0,046608	17,34005	0,2334
	0,057633	**30,10947**	**0,2179125**

Tabla 24 - Resultados figuras de mérito con el codificador G.722

	G.726-2			G.726-3		
	RMSE	**PRD (%)**	**RTF**	**RMSE**	**PRD (%)**	**RTF**
Pulmonar 1	0,008243	2,846046	0,03411	0,004466	1,542001	0,03201
Pulmonar 2	0,004319	2,703278	0,02788	0,003767	2,358045	0,03622
Pulmonar 3	0,002463	3,337283	0,03166	0,002112	2,862272	0,03988
Pulmonar 4	0,0045	2,630685	0,0298	0,002592	1,515114	0,03402
Pulmonar 5	0,004202	3,632336	0,02788	0,00186	1,6077	0,0399
Pulmonar 6	0,005953	2,600704	0,03056	0,003188	1,39251	0,03705
Pulmonar 7	0,00331	2,692812	0,03001	0,001873	1,52372	0,03801
Pulmonar 8	0,003626	2,861315	0,02982	0,002679	2,11394	0,03622
	0,004577	**2,9130574**	**0,030215**	**0,002817**	**1,864413**	**0,03666**

Tabla 25 - Resultados figuras de mérito con el codificador G.726 con dos y tres bits por muestra

	G.726-4			G.726-5		
	RMSE	**PRD (%)**	**RTF**	**RMSE**	**PRD (%)**	**RTF**
Pulmonar 1	0,003773	1,302784	0,02988	0,006529	2,254446	0,03402
Pulmonar 2	0,003278	2,051549	0,03301	0,005937	3,716233	0,033
Pulmonar 3	0,00262	3,549578	0,0381	0,003283	4,447942	0,03201
Pulmonar 4	0,002483	1,451388	0,03305	0,004188	2,448362	0,03201
Pulmonar 5	0,001358	1,173841	0,03201	0,00292	2,523934	0,03988
Pulmonar 6	0,002875	1,256066	0,02988	0,004198	1,833827	0,03622
Pulmonar 7	0,001551	1,262214	0,03301	0,002935	2,387558	0,0306
Pulmonar 8	0,002351	1,854636	0,0381	0,006148	4,851012	0,042
	0,002536	**1,737757**	**0,03338**	**0,004517**	**3,057914**	**0,03496**

Tabla 26 - Resultados figuras de merito con el codificador G.726 con cuatro y cinco bits por muestra

	IMA ADPCM		
	ERMS	**PRD (%)**	**RTF**
Pulmonar 1	0,000235	0,081136	0,00613
Pulmonar 2	0,000256	0,160298	0,0061
Pulmonar 3	0,000638	0,865067	0,0062
Pulmonar 4	0,000393	0,229484	0,006
Pulmonar 5	0,001429	1,235028	0,0059
Pulmonar 6	0,000185	0,080962	0,0061
Pulmonar 7	0,000287	0,2331	0,00613
Pulmonar 8	0,00089	0,708286	0,0061
	0,000539	**0,4491701**	**0,00615**

Tabla 27 - Resultados figuras de mérito con el codificador IMA ADPCM

	4kHz		
	ERMS	**PRD (%)**	**RTF**
Pulmonar 1	0,000395	0,136259	0,0044
Pulmonar 2	0,000329	0,206043	0,0045
Pulmonar 3	0,000109	1,148125	0,00444
Pulmonar 4	0,000399	0,23337	0,00442
Pulmonar 5	0,000175	0,151286	0,00448
Pulmonar 6	0,000111	0,048566	0,00448
Pulmonar 7	0,000121	0,098276	0,0044
Pulmonar 8	0,000196	0,154759	0,00442
	0,000229	**0,2720855**	**0,00446**

Tabla 28 - Resultados figuras de mérito con submuestreo a 4 kHz

	2kHz			1kHz		
	ERMS	**PRD (%)**	**RTF**	**ERMS**	**PRD (%)**	**RTF**
Pulmonar 1	0,000812	0,280306	0,0044	0,002008	0,693192	0,0044
Pulmonar 2	0,001026	0,642264	0,00443	0,004469	2,797435	0,00441
Pulmonar 3	0,009362	12,686313	0,00446	0,023008	31,17686	0,00443
Pulmonar 4	0,002997	1,75222	0,00448	0,007727	4,517044	0,00438
Pulmonar 5	0,028366	24,522241	0,00449	0,05331	45,9056	0,00439
Pulmonar 6	0,00024	0,104632	0,0045	0,000534	0,233489	0,0044
Pulmonar 7	0,001119	0,91033	0,00452	0,006924	5,633517	0,00442
Pulmonar 8	0,002083	1,64357	0,0044	0,025113	19,81506	0,0044
	0,005751	**5,3177345**	**0,00445**	**0,015387**	**13,84652**	**0,00442**

Tabla 29 - Resultados figuras de mérito con submuestreos a 2 kHz y 1 kHz

D. ARCHIVOS GENERADOS PARA LA LIBRERÍA OPAL

codec_estetoscopia.c

```
* Copyright (C) 2004 Post Increment, All Rights Reserved
 *
 * The contents of this file are subject to the Mozilla Public License
 * Version 1.0 (the "License"); you may not use this file except in
 * compliance with the License. You may obtain a copy of the License at
 * http://www.mozilla.org/MPL/
 *
 * Software distributed under the License is distributed on an "AS IS"
 * basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See
 * the License for the specific language governing rights and limitations
 * under the License.
 *
 * The Original Code is Open H323 Library.
 *
 * The Initial Developer of the Original Code is Post Increment
 *
 * Contributor(s): _____________________________________.
 *
 * $Revision: 22675 $
 * $Author: rjongbloed $
 * $Date: 2009-05-19 23:23:29 -0500 (Tue, 19 May 2009) $
 */

#define _CRT_NONSTDC_NO_DEPRECATE 1
#define _CRT_SECURE_NO_WARNINGS 1

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#ifndef PLUGIN_CODEC_DLL_EXPORTS
#include "plugin-config.h"
#endif

#include <codec/opalplugin.h>

#define PAYLOAD_CODE 5

#define IMA_MAX_PACKET_SIZE        1
#define IMA_DESIRED_TRANSMIT_SIZE  1

#define IMA_SAMPLES_PER_FRAME      505
#define IMA_BYTES_PER_FRAME        256

#define IMA_NS_PER_FRAME    63100

#define PREF_FRAMES_PER_PACKET 1
#define MAX_FRAMES_PER_PACKET  1
```

```
#define IMA_BITS_SECOND 32443

static const unsigned char msIMAHeader[] = {

  // unknown data
  0x02, 0x00, 0x00, 0x00,
  0x00, 0x00, 0xf9, 0x01,
  0x00, 0x00, 0xf9, 0x01,
  0x01, 0x00, 0x04, 0x00,
  0x00, 0x00, 0x00, 0x00,

#define IMA_FIXED_START            20

  // standard MS waveformatex structure follows
  0x11, 0x00,                 //    WORD    wFormatTag;        /* format
type */
  0x01, 0x00,                 //    WORD    nChannels;         /* number of
channels (i.e. mono, stereo...) */
  0x40, 0x1f, 0x00, 0x00,     //    DWORD   nSamplesPerSec;    /* sample
rate */
  0xd7, 0x0f, 0x00, 0x00,     //    DWORD   nAvgBytesPerSec;   /* for
buffer estimation */
  0x00, 0x01,                 //    WORD    nBlockAlign;       /* block
size of data */
  0x04, 0x00,                 //    WORD    wBitsPerSample;    /* Number of
bits per sample of mono data */
  0x02, 0x00,                 //    WORD    cbSize;            /* The count
in bytes of the size of

#define IMA_FIXED_LEN              18

  // extra IMA information
  0xf9, 0x01,                //    WORD    numberOfSamples    /* 505 */

  // unknown data
  0x00, 0x00

};

struct adpcm_state {
  short valprev;        /* Previous output value */
  char  index;          /* Index into stepsize table */
};

/* Intel ADPCM step variation table */
static int indexTable[16] = {
    -1, -1, -1, -1, 2, 4, 6, 8,
    -1, -1, -1, -1, 2, 4, 6, 8
};

static int stepsizeTable[89] = {
    7, 8, 9, 10, 11, 12, 13, 14, 16, 17,
    19, 21, 23, 25, 28, 31, 34, 37, 41, 45,
    50, 55, 60, 66, 73, 80, 88, 97, 107, 118,
    130, 143, 157, 173, 190, 209, 230, 253, 279, 307,
    337, 371, 408, 449, 494, 544, 598, 658, 724, 796,
    876, 963, 1060, 1166, 1282, 1411, 1552, 1707, 1878, 2066,
    2272, 2499, 2749, 3024, 3327, 3660, 4026, 4428, 4871, 5358,
    5894, 6484, 7132, 7845, 8630, 9493, 10442, 11487, 12635, 13899,
    15289, 16818, 18500, 20350, 22385, 24623, 27086, 29794, 32767
};
```

```
static void adpcm_coder(short indata[], char outdata[], int len, struct
adpcm_state *state)
{

printf("Aqui llega");
    short *inp;                 /* Input buffer pointer */
    signed char *outp;          /* output buffer pointer */
    int val;                    /* Current input sample value */
    int sign;                   /* Current adpcm sign bit */
    int delta;                  /* Current adpcm output value */
    int diff;                   /* Difference between val and valprev */
    int step;                   /* Stepsize */
    int valpred;                /* Predicted output value */
    int vpdiff;                 /* Current change to valpred */
    char index;                 /* Current step change index */
    int outputbuffer = 0;       /* place to keep previous 4-bit value */
    int bufferstep;             /* toggle between outputbuffer/output */

    outp = (signed char *)outdata;
    inp = indata;

    //create header
    valpred = *inp;
    memcpy(outp, (char *)inp, 2);
    inp++;
    outp += sizeof(short);

    index = state->index;
    memcpy(outp, (char *)&index, 1);
    inp++;
    outp++;
    *outp = 0;
    outp++;
    //create header ends

    len--;

    step = stepsizeTable[(int)index];

    bufferstep = 1;

    for ( ; len > 0 ; len-- ) {
        val = *inp++;

        /* Step 1 - compute difference with previous value */
        diff = val - valpred;
        sign = (diff < 0) ? 8 : 0;
        if ( sign ) diff = (-diff);

        /* Step 2 - Divide and clamp */
        /* Note:
        ** This code *approximately* computes:
        **    delta = diff*4/step;
        **    vpdiff = (delta+0.5)*step/4;
        ** but in shift step bits are dropped. The net result of this is
        ** that even if you have fast mul/div hardware you cannot put it to
        ** good use since the fixup would be too expensive.
        */
        delta = 0;
        vpdiff = (step >> 3);

        if ( diff >= step ) {
            delta = 4;
```

```
            diff -= step;
            vpdiff += step;
        }
        step >>= 1;
        if ( diff >= step  ) {
            delta |= 2;
            diff -= step;
            vpdiff += step;
        }
        step >>= 1;
        if ( diff >= step ) {
            delta |= 1;
            vpdiff += step;
        }

        /* Step 3 - Update previous value */
        if ( sign )
          valpred -= vpdiff;
        else
          valpred += vpdiff;

        /* Step 4 - Clamp previous value to 16 bits */
        if ( valpred > 32767 )
          valpred = 32767;
        else if ( valpred < -32768 )
          valpred = -32768;

        /* Step 5 - Assemble value, update index and step values */
        delta |= sign;
        index = (char)(index + indexTable[delta]);
        if ( index < 0 ) index = 0;
        if ( index > 88 ) index = 88;
        step = stepsizeTable[(int)index];

        /* Step 6 - Output value */
        if ( bufferstep ) {
            outputbuffer = (delta << 4) & 0xf0;
        } else {
            *outp++ = (char)((delta & 0x0f) | outputbuffer);
        }
        bufferstep = !bufferstep;
    }

    /* Output last step, if needed */
    if ( !bufferstep )
      *outp++ = (char)outputbuffer;

    state->valprev = (short)valpred;
    state->index = index;

}

static void adpcm_decoder(char indata[], short outdata[], int len)
{
    signed char *inp;           /* Input buffer pointer */
    short *outp;                /* output buffer pointer */
    int sign;                   /* Current adpcm sign bit */
    int delta;                  /* Current adpcm output value */
    int step;                   /* Stepsize */
    int valpred;                /* Predicted value */
    int vpdiff;                 /* Current change to valpred */
    int index;                  /* Current step change index */
    int inputbuffer = 0;        /* place to keep next 4-bit value */
```

```
int bufferstep;             /* toggle between inputbuffer/input */

outp = outdata;
inp = (signed char *)indata;

valpred = 0;
index = 0;
memcpy((char *)&valpred, (char *)inp, 2);
inp += 2; //skip first 16 bits sample
index = (int)(unsigned char)*inp;
inp += 2; //skip index

step = stepsizeTable[index];
len -= 4; //skip header

bufferstep = 0;

len *= 2;

for ( ; len > 0 ; len-- ) {
    /* Step 1 - get the delta value */
    if ( bufferstep ) {
        delta = inputbuffer & 0xf;
    } else {
        inputbuffer = *inp++;
        delta = (inputbuffer >> 4) & 0xf;
    }
    bufferstep = !bufferstep;

    /* Step 2 - Find new index value (for later) */
    index += indexTable[delta];
    if ( index < 0 ) index = 0;
    if ( index > 88 ) index = 88;

    /* Step 3 - Separate sign and magnitude */
    sign = delta & 8;
    delta = delta & 7;
    /* Step 4 - Compute difference and new predicted value */
    /*
    ** Computes 'vpdiff = (delta+0.5)*step/4', but see comment
    ** in adpcm_coder.
    */
    vpdiff = step >> 3;
    if ( delta & 4 ) vpdiff += step;
    if ( delta & 2 ) vpdiff += step>>1;
    if ( delta & 1 ) vpdiff += step>>2;

    if ( sign )
      valpred -= vpdiff;
    else
      valpred += vpdiff;

    /* Step 5 - clamp output value */
    if ( valpred > 32767 )
      valpred = 32767;
    else if ( valpred < -32768 )
      valpred = -32768;

    /* Step 6 - Update step value */
    step = stepsizeTable[index];

    /* Step 7 - Output value */
    *outp++ = (char)valpred;
```

```
    }
}

////////////////////////////////////////////////////////////////////////
//

static void * create_codec(const struct PluginCodec_Definition * codec)
{
  struct adpcm_state * s_adpcm = malloc(sizeof (struct adpcm_state));

  s_adpcm->valprev = 0;
  s_adpcm->index   = 0;

  return s_adpcm;
}

static int codec_encoder(const struct PluginCodec_Definition * codec,
                                        void * _context,
                                  const void * from,
                                    unsigned * fromLen,
                                        void * to,
                                    unsigned * toLen,
                                unsigned int * flag)
{
  struct adpcm_state * s_adpcm = (struct adpcm_state *)_context;

  if (*fromLen < (IMA_SAMPLES_PER_FRAME*2) || (*toLen <
IMA_BYTES_PER_FRAME))
    return 0;

  adpcm_coder((short *)from, (char *)to, IMA_SAMPLES_PER_FRAME, s_adpcm);

  return 1;
}

static int codec_decoder(const struct PluginCodec_Definition * codec,
                                        void * _context,
                                  const void * from,
                                    unsigned * fromLen,
                                        void * to,
                                    unsigned * toLen,
                                unsigned int * flag)
{
  //struct adpcm_state * s_adpcm = (struct adpcm_state *)_context;

  if (*toLen < (IMA_SAMPLES_PER_FRAME*2) || (*fromLen <
IMA_BYTES_PER_FRAME))
    return 0;

  adpcm_decoder((char *)from, (short *)to, IMA_BYTES_PER_FRAME);

  return 1;
}


static void destroy_codec(const struct PluginCodec_Definition * codec, void
* _context)
{
  free(_context);
}

////////////////////////////////////////////////////////////////////////
//
```

```
static struct PluginCodec_information licenseInfo = {
  1084190703,                                 // timestamp = Mon 10 May 2004
12:05:03 PM UTC =

  "Craig Southeren, Post Increment",                              // source
code author
  "1.0",                                                          // source
code version
  "craigs@postincrement.com",                                     // source
code email
  "http://www.postincrement.com",                                 // source
code URL
  "Copyright (C) 2004 by Post Increment, All Rights Reserved", // source
code copyright
  "MPL 1.0",                                                      // source
code license
  PluginCodec_License_MPL,                                        // source
code license

  NULL,                                                           // codec
description
  NULL,                                                           // codec
author
  NULL,                                                           // codec
version
  NULL,                                                           // codec
email
  NULL,                                                           // codec URL
  NULL,                                                           // codec
copyright information
  NULL,                                                           // codec
license
  PluginCodec_Licence_None                                        // codec
license code
};

////////////////////////////////////////////////////////////////////////////
//

static const char L16Desc[]  = {"PCM-16-16kHz"}; //{ "L16" };

static const char imaADPCM[] = "MS-IMA-ADPCM";

////////////////////////////////////////////////////////////////////////////
//

static struct PluginCodec_Definition imaADPCMCodecDefn[] =
{
  {
    // encoder
    PLUGIN_CODEC_VERSION,               // codec API version
    &licenseInfo,                       // license information

    PluginCodec_MediaTypeAudio |        // audio codec
    PluginCodec_InputTypeRaw |          // raw input data
    PluginCodec_OutputTypeRaw |         // raw output data
    PluginCodec_RTPTypeExplicit,        // specified RTP type

    imaADPCM,                           // text decription
    L16Desc,                            // source format
    imaADPCM,                           // destination format
```

```
    0,                                    // user data

    4000,                                 // samples per second
    IMA_BITS_SECOND,                      // raw bits per second
    IMA_NS_PER_FRAME,                     // nanoseconds per frame
    IMA_SAMPLES_PER_FRAME,                // samples per frame
    IMA_BYTES_PER_FRAME,                  // bytes per frame
    PREF_FRAMES_PER_PACKET,               // recommended number of
                                             frames per packet
    MAX_FRAMES_PER_PACKET,                // maximum number of frames
                                             per packe
    PAYLOAD_CODE,                          // no IANA RTP payload code
    "DVI4_8k",                             // RTP payload name

    create_codec,                         // create codec function
    destroy_codec,                        // destroy codec
    codec_encoder,                        // encode/decode
    NULL,                                 // codec controls

    PluginCodec_H323AudioCodec_ima_adpcm,     // h323CapabilityType
    NULL                                 // h323CapabilityData
  },

  {
    // decoder
    PLUGIN_CODEC_VERSION,                 // codec API version
    &licenseInfo,                         // license information

    PluginCodec_MediaTypeAudio |          // audio codec
    PluginCodec_InputTypeRaw |            // raw input data
    PluginCodec_OutputTypeRaw |           // raw output data
    PluginCodec_RTPTypeDynamic,           // dynamic RTP type

    imaADPCM,                             // text decription
    imaADPCM,                             // source format
    L16Desc,                              // destination format

    0,                                    // user data

    4000,                                 // samples per second
    IMA_BITS_SECOND,                      // raw bits per second
    IMA_NS_PER_FRAME,                     // nanoseconds per frame
    IMA_SAMPLES_PER_FRAME,                // samples per frame
    IMA_BYTES_PER_FRAME,                  // bytes per frame
    PREF_FRAMES_PER_PACKET,               // recommended number of
                                             frames per packet
    MAX_FRAMES_PER_PACKET,                // maximum number of frames
                                             per packe
    PAYLOAD_CODE,                         // no IANA RTP
                                             payload code
    "DVI4_8k",                            // RTP payload name

    create_codec,                         // create codec function
    destroy_codec,                        // destroy codec
    codec_decoder,                        // encode/decode
    NULL,                                 // codec controls

    PluginCodec_H323AudioCodec_ima_adpcm,     // h323CapabilityType
    NULL                                 // h323CapabilityData
  }
};
```

```
PLUGIN_CODEC_IMPLEMENT_ALL(IMA_ADPCM, imaADPCMCodecDefn,
PLUGIN_CODEC_VERSION)

/////////////////////////////////////////////////////////////////////////////
//
```

codec_estetoscopiammf.cxx

```
#include <ptlib.h>
#include <opal/buildopts.h>

#include <opal/mediafmt.h>
#include <h323/h323caps.h>
#include <asn/h245.h>


#define new PNEW


#if OPAL_H323
class H323_IMA_ADPCMCapability : public H323AudioCapability
{
  public:
    virtual PObject * Clone() const
    {
      return new H323_IMA_ADPCMCapability(*this);
    }

    virtual unsigned GetSubType() const
    {
      return H245_AudioCapability::e_ima_adpcm;
    }

    virtual PString GetFormatName() const
    {
      return OpalIMA_ADPCM;
    }
};
#endif

const OpalAudioFormat & GetOpalIMA_ADPCM()
{
  static const OpalAudioFormat IMA_ADPCM_Format(OPAL_IMA_ADPCM,
RTP_DataFrame::DVI4_8k,  "DVI4_8k", 2, 16, 50, 10, 256, 8000);

#if OPAL_H323
  static H323CapabilityFactory::Worker<H323_IMA_ADPCMCapability>
IMA_ADPCM_Factory(OPAL_IMA_ADPCM, true);
#endif // OPAL_H323

  return IMA_ADPCM_Format;
}
```

makefile

```
AC_PLUGIN_DIR=opal-3.6.8/codecs/audio
prefix=/usr
exec_prefix=${prefix}
libdir=${exec_prefix}/lib
```

```
target_os=linux-gnu

SONAME      = codec_esteteo

SRCDIR      = ./src
OBJDIR      = ./obj
PLUGINDIR=../..

CC          =gcc
CFLAGS            = -Os
CXX         =g++
LDSO        =-shared -Wl,-soname,$(SONAME)
PLUGINEXT   =so
STDCCFLAGS  = -fPIC
LDFLAGS           =
EXTRACFLAGS     =-I$(PLUGINDIR)
SRCS  += codec_esteteo.c

vpath %.o $(OBJDIR)
vpath %.c $(SRCDIR)

ifeq ($(VERBOSE),)
Q_CC = @echo [CC] `echo $< | sed s^@OPALDIR@/^^` ;
Q_LD = @echo [LD] `echo $(PLUGIN) | sed s^@OPALDIR@/^^` ;
endif


$(OBJDIR)/%.o : %.c
      @mkdir -p $(OBJDIR) >/dev/null 2>&1
      $(Q_CC)$(CC) -I../../../include $(STDCCFLAGS) $(OPTCCFLAGS)
$(EXTRACFLAGS) $(CFLAGS) -c $< -o $@

PLUGIN      = ./codec_esteteo_audio_pwplugin.$(PLUGINEXT)

OBJECTS = $(addprefix $(OBJDIR)/,$(patsubst %.c,%.o,$(notdir $(SRCS))))

$(PLUGIN): $(OBJECTS)

ifeq (solaris,$(findstring solaris,$(target_os)))

      $(Q_LD)$(CC) $(LDSO) $@ -o $@ $^ $(EXTRALIBS)
else
      $(Q_LD)$(CC) $(LDSO) -o $@ $^ $(EXTRALIBS)

endif

install:
      mkdir -p $(DESTDIR)$(libdir)/$(AC_PLUGIN_DIR)
      install $(PLUGIN) $(DESTDIR)$(libdir)/$(AC_PLUGIN_DIR)

uninstall:
      rm -f $(DESTDIR)$(libdir)/$(AC_PLUGIN_DIR)/$(PLUGIN)

clean:
      rm -f $(OBJECTS) $(PLUGIN)

##########################################
```

E. PAPER ACEPTADO EN EL CONGRESO IBER SPEECH 2012

Audio Encoding for Heart and Breath Sounds Acquired with Digital Stethoscope

Ignacio Palacios Santos[1], Doroteo Torre Toledano[1] and Andrés Martínez Fernández[2]

[1] ATVS, Escuela Politécnica Superior, Universidad Autónoma de Madrid, Madrid, SPAIN
ignacio.palacios@estudiante.uam.es, doroteo.torre@uam.es
[2] Universidad Rey Juan Carlos, Fundación Enlace Hispano Americano de Salud, Fuenlabrada, SPAIN
andres.martinez@urjc.es

Abstract. This paper presents a real-time audio encoding system for cardiorespiratory sounds acquired with a low cost Bluetooth digital stethoscope developed by EHAS (Enlace Hispano Americano de Salud) Foundation and Fundatel Foundation. The system is currently working with a G.722 speech coder for compression and subsequent transmission of cardiorespiratory signals. However, these signals have different frequency characteristics from speech. Therefore, we sought a better adapted alternative to encode these signals optimally, with an encoder not subject to the payment of any license, with low computational cost, good quality and low bandwidth. We have evaluated the proposed solution both using objective measures such as root mean squared error and using subjective opinions from four expert physicians.

Keywords. Cardiorespiratory Sounds, Stethoscopy, ADPCM Encoding, Development Cooperation, VoIP.

1 INTRODUCTION

EHAS Foundation is a non-profit organization which aims to promote the appropriate use of new Information and Communication Technology (ICT) to improve health processes in isolated rural areas of developing countries. In 2011, a pilot project was conducted in the Peruvian area of River Napo, with the objective of providing cardiorespiratory tele-diagnosis for the local population. The diagnosis is made between rural health posts (without a physician) and health centres of reference (with medical presence). With the audio encoding system, the physician is able to guide the patient via videoconference by means of a digital stethoscope at each end, transmitting the cardiorespiratory audio over VoIP. This solution will allow many isolated people to have a quick diagnosis, sparing them the travelling of hundreds of miles to get to the nearest hospital, going through poorly served areas. EHAS Foundation, together with Fundatel Foundation, developed a low cost bluetooth digital stethoscope and a peer-to-peer transmission system for the cardiorespiratory signals. This paper sets forth a cardiorespiratory real-time audio encoding alternative for digitalized signals that is able to send these signals over VoIP with good quality, in real-time and using much less bandwidth than previous solutions available at EHAS Foundation. The development of a digital tele-stethoscope is not a new idea in itself [1, 2]. This work does not focus in the development of the digital tele-stethoscope, but in the optimization of the audio coding for the cardiorespiratory signals involved in digital tele-stethoscopy.

2 CHARACTERIZATION OF CARDIORESPIRATORY SIGNALS

Cardiorespiratory sounds are obtained through auscultation which is the term used for the listening of internal sounds of the body, usually by means of a stethoscope. Auscultation is performed for the purpose of examining the circulatory system and respiratory system (heart and breath sounds). There are four auscultation areas: mitral,

aortic, tricuspid and pulmonary. The sounds detected in these four areas will have different frequency characteristics than voice signals.

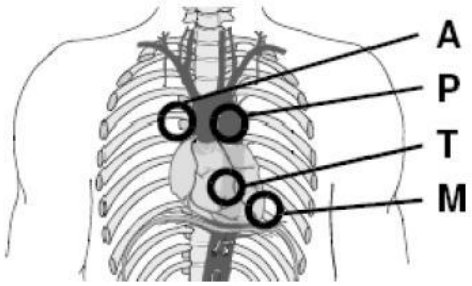


Fig. 1. Auscultation areas [3]

Generally, the components of heartbeat and lung sounds that are useful for diagnostic purposes are in the range of 20-1000 Hz. The first heart sounds (S1 and S2), are produced by the closing of the atrioventricular valves and semilunar valves respectively. These sounds fall in the range of 20-115 Hz. Disorders such as heart murmurs occur in the range of 140-600 Hz [4]. Thus, a suitable listening range for heart sounds would be approximately 20-600 Hz. For respiratory sounds, the strongest part of the signal is typically under 100 Hz, although the signal can have useful components up to 1.2 kHz [5].

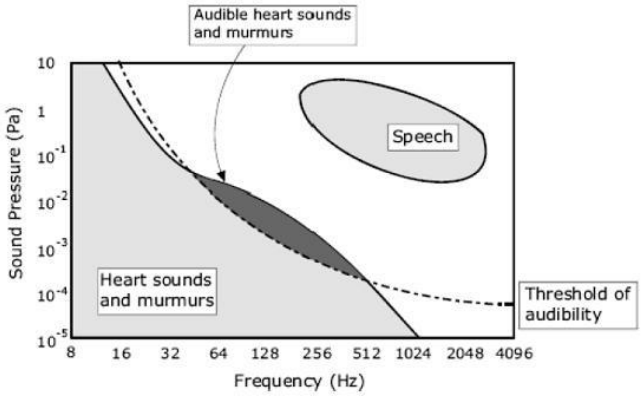


Fig. 2. Cardiorespiratory sound frequency distribution [3]

Hence, by the Nyquist theorem [6]:

$$f_s > 2 f_M \tag{3}$$

we could derive the minimum sampling frequency (f_s) to which we can downsample cardiorespiratory signals without losing relevant information for the diagnosis. In our case the maximum frequency present in the signal (f_M) is 1200 Hz. Hence by Nyquist theorem the cardiorespiratory signas can be completely recovered from samples at a sampling frequency (f_s) higher that 2400 Hz.

3 EHAS-FUNDATEL SYSTEM ARCHITECTURE

As mentioned in [3], the analogical cardiorespiratory sound captured by the chestpiece of the digital Bluetooth stethoscope EHAS-Fundatel is picked up by a microphone, which converts the mechanical motion of the sound pressure waves into a voltage variation. After the analog signal is captured, the following process is applied:

1. This voltage is transmitted, after an analogical amplification stage, to a TLV320AIC33 hardware codec. The codec hardware, at its first stage, performs the conversion of the analog signal emitted by the microphone to a digital signal that can be processed by other electronic devices. This signal is downsampled to 8 kHz and 16 bits per sample.

2. The resulting signal is sent to a OEMSPA310 Bluetooth chip that sends the audio to a PC using the Serial Port Profile (SPP) Bluetooth protocol. This audio is sent encoded in a format that adapts the sound to the features of Bluetooth transmission. During this communication, flow control information is also sent, allowing the PC to ensure it is receiving the audio without any alteration caused by a poor management of the speed of transmission/reception, as well as the necessary information to ensure the audio can be rebuilt.

3. There is a Stethoscope's software [3] that performs a pre-encoding protocol before sending it to a VoIP application (Ekiga Softphone [7]) when the digital sound reaches the PC. The aim is to support any sound server, running at any sample rate and adjust the sending rate of data at 44100 samples per second and 32 bits per sample (which is the most common case).

4. Once the signal reaches the softphone, the latter adapts the digital audio input to the needs of the audio codec used (G.722 in our example, which requires an entry with 16000 samples per second and 14 bits per sample, providing an output of 64 Kbps).

After sending the final digitalized audio to the network, the audio is received by the remote computer, starting the reverse process (decoding).

4 ENCODERS TEST

In order to find an alternative to G.722 codec several simulations of EHAS-Fundatel system architecture were performed with different encoders. We used 33 audio cardiorespiratory samples, 15 of which were obtained with the digital stethoscope's bluetooth EHAS-Fundatel (3 mitral-type, 3 aortic-type, 3 tricuspid-type and 6 pulmonary-type), and 18 downloaded from [8] (10 cardiac-type, 8 pulmonary-type). We created several Linux scripts where the audio samples were encoded and decoded using the audio manipulation software SoX [9] (Sound eXchange), obtaining different figures of merit as results that allow objective comparison of the encoders. These figures of merit are Compression Factor (CF), Bit Rate (R), Root Mean Square Error (RMSE), Percent RMS Difference (PRD) and Real Time Factor (RTF):

$$CF = \frac{source\_data\_size}{encoded\_data\_size} \tag{4}$$

$$RMSE = \sqrt{\frac{\sum_{n=1}^{N}\left(x[n]-\tilde{x}[n]\right)^2}{N}} \tag{5}$$

$$PRD = \sqrt{\frac{\sum_{n=1}^{N}\left(x[n]-\tilde{x}[n]\right)^2}{\sum_{n=1}^{N}x^2[n]}} \times 100\% \tag{6}$$

$$RTF = \frac{computation\_time}{audio\_duration} \tag{7}$$

Where $x[n]$ is the original signal and $\tilde{x}[n]$ is the signal obtained after coding and decoding the audio, and N is the number of samples in the audio.

The EHAS-Fundatel audio samples are digitalized at 44.1 kHz and 32 bits per sample, while the heart sounds [8] are digitalized at 16 kHz and 16 bits per sample; finally respiratory signals are digitalized at 11.025 kHz and 16 bits per sample. Initially we tested with G.722, G.726, IMA-ADPCM encoders and downsampled at 1, 2 and 4 kHz.

Table 1. Comparison encoders assuming a 16 bit per sample at 16 kHz input.

	CF	R (Kbps)	RMSE	PRD (%)	RTF
G.722	4	64	0.039405267	22.7856303	0.2122
G.726-2	16	16	0.004458867	3.22418553	0.03503
G.726-3	10.66	24	0.002614943	1.93012648	0.03636
G.726-4	8	32	0.002327377	1.71252903	0.03420
G.726-5	6.4	40	0.003868983	2.84869895	0.03633
4 kHz	4	64	0.000347993	0.29957613	0.00448
2 kHz	8	32	0.002195643	1.98224353	0.00446
1 kHz	16	16	0.010389677	5.048618	0.00441
IMA-ADPCM	4	64	0.000396343	1.93784919	0.00613

The G.726 codec worked well, but it requires as input audio previously encoded with G.711 μ-law or A-law. Furthermore, the decoded audio seemed to have worse subjective quality. The G.711 codec is a simple PCM with minimum processing load on the CPU of the PC, and its use is not subject to the payment of any license. However, the quality of the received audio is very dependent on the level of pressure applied to the chestpiece. This codec uses an 8-bit not uniform quantification, so that low amplitude signals suffer less distortion than those with large amplitude. As the cardiorespiratory signal presents large amplitude on low frequency components, this requires auscultation under low pressure levels, preventing signal distortion caused by the encoder working in its less linear region. In addition, working under low chestpiece pressure levels means, in practice, using a smaller number of bits per sample encoded, so the resolution is severely affected.

IMA-ADCPM codec and downsamplings had better results than the G.722 regarding the RMSE, PRD and RTF figures of merit, so we decided to design an integrated codec consisting of an IMA-ADPCM codec previously downsamped at 4 kHz, thus respecting the bandwidth of cardiorespiratory signals.

Table 2. Comparison between the current codec (G.722) and alternative (Proposed Stethoscope-Codec)

	CF	R (Kbps)	RMSE	PRD (%)	RTF
G.722	4	64	0.039405267	22.7856303	0.02122
Proposed Stethoscope-Codec	16	16	0.002175277	1.93784919	0.00588

The new Proposed Stethoscope-Codec provides better objective results than the G.722 codec when comparing all figures of merit. In particular, provides better objective quality with four times less bit rate and less computational complexity.

5 MEDICAL EVALUATION

After performing the objective assessment of the encoders, a subjective evaluation of these was carried out, since it is intended that the encoding of the cardiorespiratory audio does not distort too much the quality of the audio signal and that the decoded audio is useful for remote diagnosis. The 33 original samples and the respective samples decoded with the G.722 codec and Stethoscope-Codec were evaluated by a group of four physicians of the Hospital Clínico San Carlos in Madrid. The doctors were asked to grade from 1 to 10 the quality of the audio heard, both the original and decoded without knowing each sample's encoding. These physicians were three pulmonologist and one internal medicine doctor.

Only one doctor graded, in general, the new Proposed Stethoscope-Codec with a higher grade, stating that there was less noise in the audio heard and that, in such a way, the cardiac and respiratory audio where she needed to focus were highlighted. The rest of the physicians gave, in general, a slightly better grade to the G.722 codec with respect to the new Proposed Stethoscope-Codec. Other doctor explained that in the case of samples related to pathologies where very high-frequency noises were detected, a distortion was observed with such noises taking a deeper register. Nevertheless, being familiar with the pathology, it was possible to diagnose it.

Table 3. Average grades for 33 samples (1 to 15 EHAS-Fundatel samples, 16 to 25 cardiac samples [8], 26 to 33 pulmonary samples [8])

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
Original	7.5	8.75	8.5	7.75	7	7	6.75	8.25	8	7.5	6.75
G.722	7	7.5	7.5	7.25	6.25	6.25	5.5	7.75	7	7	6.5
Stethoscope-Codec	6.5	7.5	7.25	7	6	6	5.75	7	7	6.75	6.5
	12	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**	**22**
Original	6.25	7.5	**7.25**	8.25	6.5	7.5	8	6	5.75	5.5	6.75
G.722	6.25	7.25	6.75	7.75	6.25	7	7.75	5.75	5	4.75	6.25
Stethoscope-Codec	6	6.75	6.5	7.25	6	6.75	7.25	5.5	5	5	5.5
	23	**24**	**25**	**26**	**27**	**28**	**29**	**30**	**31**	**32**	**33**
Original	6.75	6.5	5.5	5.25	6.75	7	6.25	6	6.75	5.75	6.25
G.722	6.25	6.25	6	5	5.5	6.25	5.75	5.75	6	5.75	6.25
Stethoscope-Codec	6.5	6.25	6	5.25	5.25	6	5.75	5.75	5.75	5.5	6

Table 4. Average grades per physician

	Original	**G.722**	**Stethoscope-Codec**
Dr. 1	7.33333333	7.42424242	7.90909091
Dr. 2	7	5.75757576	5.42424242
Dr. 3	6.72727273	6.24242424	5.87878788
Dr. 4	6.54545455	6.21212121	5.60606061
Average	**6.90151515**	**6.40909091**	**6.20454545**

6 CONCLUSION AND FUTURE WORK

Despite the new Proposed Stethoscope-Codec provides better objective results than G.722 when comparing figures of merit such as RMSE and PRD, subjective medical assessment of the former is slightly lower. The new Proposed Stethoscope-Codec reduces the bit rate in a 4:1 proportion, going from 64 Kbps to 16 Kbps, and decreasing the RTF in a greater than 3:1 proportion, going from 0.02122 to 0.00588. The Proposed Stethoscope-Codec could be a good alternative to G.722 in cases where communication networks are precarious and with small bandwidth. Such is the case of most rural areas in developing countries. Our codec would therefore allow a higher number of auscultations at the same time, reducing the latency and requiring lower-processing-capacity computers thanks to the RTF reduction. The medical assessment presented should be taken with care, since the number of physicians consulted is still small. A more in-depth clinical study with patients suffering of a certain kind of pathology would be needed in order to validate the medical use of the Proposed Stethoscope-Codec.

REFERENCES

1. Hedayioglu FL, Mattos SS, Moser L, de Lima ME. 2007. Development of a tele-stethoscope and its application in pediatric cardiology. in Indian J Exp Biol. 2007 Jan;45(1):86-92.
2. Sankaran, P. RMK Eng. Coll., Chennai, India Chandrasekaran, K.;Baig, A.H.; Moll, C.L. 2010. Development of a tele-stethoscope: Indian perspective, 2010, in Proc. IEEE/ICME International Conference on Complex Medical Engineering (CME), 2010.
3. Ignacio Foche Pérez: Desarrollo de un teleestetoscopio digital Bluetooth para zonas rurales aisladas de países en desarrollo, Proyecto Fin de Master en Redes de Telecomunicación para Países en Desarrollo, Escuela Técnica Superior de Ingeniería de Recomunicaciones, Universidad Rey Juan Carlos.
4. M. Abella and J. Formolo: Comparsion of the Acoustic Properties of Six Popular Stethoscopes, Department of Internal Medicine, St. John Hospital, Detroit, Michigan 48236.
5. I. Mazic, S. Sovilj and R. Magjarevic: Analysis od Respiratory Sounds in Asthmatic Infants, Measurement Science Review, Vol. 3, Section 2, 2003.
6. Alan V. Oppenhheim, Alans S. Willsky, S. Hamid Nawab: Señales y Sistemas, pp. 527-528, Segunda edición, Pearson Educación.
7. http://www.ekiga.org/, Ekiga Softphone.

8. http://www.med.umich.edu/lrc/psb/heartsounds/index.htm, University of Michigan Heart Sound and Murmur Library.
9. http://sox.sourceforge.net/, Sound eXchange.

PRESUPUESTO

1) Ejecución Material

- Compra de ordenador personal (Software incluido) ..... 1.500 €

- Alquiler de impresora láser durante 18 meses ..... 100 €

- Material de oficina ..... 100 €

- Total de ejecución material ..... 1.700 €

2) Gastos generales
- 16 % sobre Ejecución Material ..... 272 €

3) Beneficio Industrial
- 6 % sobre Ejecución Material ..... 102 €

4) Honorarios Proyecto
- 1440 horas a 10 € / hora ..... 14400 €

5) Material fungible
- Gastos de impresión ..... 50 €
- Encuadernación .....200 €

6) Subtotal del presupuesto
- Subtotal Presupuesto ..... 16622 €

7) I.V.A. aplicable
- 21% Subtotal Presupuesto ..... 3491 €

8) Total presupuesto
- Total Presupuesto ..... 20113 €

Madrid, Octubre de 2012

El Ingeniero Jefe de Proyecto
Fdo.: Ignacio Palacios Santos
Ingeniero Superior de Telecomunicación

PLIEGO DE CONDICIONES

Este documento contiene las condiciones legales que guiarán la realización, en este proyecto, de Codificación de Audio para Sonidos Cardiacos y Respiratorios Adquiridos con Estetoscopio Digital. En lo que sigue, se supondrá que el proyecto ha sido encargado por una empresa cliente a una empresa consultora con la finalidad de realizar dicho sistema. Dicha empresa ha debido desarrollar una línea de investigación con objeto de elaborar el proyecto. Esta línea de investigación, junto con el posterior desarrollo de los programas está amparada por las condiciones particulares del siguiente pliego.

Supuesto que la utilización industrial de los métodos recogidos en el presente proyecto ha sido decidida por parte de la empresa cliente o de otras, la obra a realizar se regulará por las siguientes:

Condiciones generales

1. La modalidad de contratación será el concurso. La adjudicación se hará, por tanto, a la proposición más favorable sin atender exclusivamente al valor económico, dependiendo de las mayores garantías ofrecidas. La empresa que somete el proyecto a concurso se reserva el derecho a declararlo desierto.

2. El montaje y mecanización completa de los equipos que intervengan será realizado totalmente por la empresa licitadora.

3. En la oferta, se hará constar el precio total por el que se compromete a realizar la obra y el tanto por ciento de baja que supone este precio en relación con un importe límite si este se hubiera fijado.

4. La obra se realizará bajo la dirección técnica de un Ingeniero Superior de Telecomunicación, auxiliado por el número de Ingenieros Técnicos y Programadores que se estime preciso para el desarrollo de la misma.

5. Aparte del Ingeniero Director, el contratista tendrá derecho a contratar al resto del personal, pudiendo ceder esta prerrogativa a favor del Ingeniero Director, quien no estará obligado a aceptarla.

6. El contratista tiene derecho a sacar copias a su costa de los planos, pliego de condiciones y presupuestos. El Ingeniero autor del proyecto autorizará con su firma las copias solicitadas por el contratista después de confrontarlas.

7. Se abonará al contratista la obra que realmente ejecute con sujeción al proyecto que sirvió de base para la contratación, a las modificaciones autorizadas por la superioridad o a las órdenes que con arreglo a sus facultades le hayan comunicado por escrito al Ingeniero Director de obras siempre que dicha obra se haya ajustado a los preceptos de los pliegos de condiciones, con arreglo a los cuales, se harán las modificaciones y la valoración de las diversas unidades sin que el importe total pueda exceder de los presupuestos aprobados. Por consiguiente, el número de unidades que se consignan en el proyecto o en el presupuesto, no podrá servirle de fundamento para entablar reclamaciones de ninguna clase, salvo en los casos de rescisión.

8. Tanto en las certificaciones de obras como en la liquidación final, se abonarán los trabajos realizados por el contratista a los precios de ejecución material que figuran en el presupuesto para cada unidad de la obra.

9. Si excepcionalmente se hubiera ejecutado algún trabajo que no se ajustase a las condiciones de la contrata pero que sin embargo es admisible a juicio del Ingeniero Director de obras, se dará conocimiento a la Dirección, proponiendo a la vez la rebaja de precios que el Ingeniero estime justa y si la Dirección resolviera aceptar la obra, quedará el contratista obligado a conformarse con la rebaja acordada.

10. Cuando se juzgue necesario emplear materiales o ejecutar obras que no figuren en el presupuesto de la contrata, se evaluará su importe a los precios asignados a otras obras o materiales análogos si los hubiere y cuando no, se discutirán entre el Ingeniero Director y el contratista, sometiéndolos a la aprobación de la Dirección. Los nuevos precios convenidos por uno u otro procedimiento, se sujetarán siempre al establecido en el punto anterior.

11. Cuando el contratista, con autorización del Ingeniero Director de obras, emplee materiales de calidad más elevada o de mayores dimensiones de lo estipulado en el proyecto, o sustituya una clase de fabricación por otra que tenga asignado mayor precio o ejecute con mayores dimensiones cualquier otra parte de las obras, o

en general, introduzca en ellas cualquier modificación que sea beneficiosa a juicio del Ingeniero Director de obras, no tendrá derecho sin embargo, sino a lo que le correspondería si hubiera realizado la obra con estricta sujeción a lo proyectado y contratado.

12. Las cantidades calculadas para obras accesorias, aunque figuren por partida alzada en el presupuesto final (general), no serán abonadas sino a los precios de la contrata, según las condiciones de la misma y los proyectos particulares que para ellas se formen, o en su defecto, por lo que resulte de su medición final.

13. El contratista queda obligado a abonar al Ingeniero autor del proyecto y director de obras así como a los Ingenieros Técnicos, el importe de sus respectivos honorarios facultativos por formación del proyecto, dirección técnica y administración en su caso, con arreglo a las tarifas y honorarios vigentes.

14. Concluida la ejecución de la obra, será reconocida por el Ingeniero Director que a tal efecto designe la empresa.

15. La garantía definitiva será del 4% del presupuesto y la provisional del 2%.

16. La forma de pago será por certificaciones mensuales de la obra ejecutada, de acuerdo con los precios del presupuesto, deducida la baja si la hubiera.

17. La fecha de comienzo de las obras será a partir de los 15 días naturales del replanteo oficial de las mismas y la definitiva, al año de haber ejecutado la provisional, procediéndose si no existe reclamación alguna, a la reclamación de la fianza.

18. Si el contratista al efectuar el replanteo, observase algún error en el proyecto, deberá comunicarlo en el plazo de quince días al Ingeniero Director de obras, pues transcurrido ese plazo será responsable de la exactitud del proyecto.

19. El contratista está obligado a designar una persona responsable que se entenderá con el Ingeniero Director de obras, o con el delegado que éste designe, para todo relacionado con ella. Al ser el Ingeniero Director de obras el que interpreta el proyecto, el contratista deberá consultarle cualquier duda que surja en su realización.

20. Durante la realización de la obra, se girarán visitas de inspección por personal facultativo de la empresa cliente, para hacer las comprobaciones que se crean oportunas. Es obligación del contratista, la conservación de la obra ya ejecutada hasta la recepción de la misma, por lo que el deterioro parcial o total de ella, aunque sea por agentes atmosféricos u otras causas, deberá ser reparado o reconstruido por su cuenta.

21. El contratista, deberá realizar la obra en el plazo mencionado a partir de la fecha del contrato, incurriendo en multa, por retraso de la ejecución siempre que éste no sea debido a causas de fuerza mayor. A la terminación de la obra, se hará una recepción provisional previo reconocimiento y examen por la dirección técnica, el depositario de efectos, el interventor y el jefe de servicio o un representante, estampando su conformidad el contratista.

22. Hecha la recepción provisional, se certificará al contratista el resto de la obra, reservándose la administración el importe de los gastos de conservación de la misma hasta su recepción definitiva y la fianza durante el tiempo señalado como plazo de garantía. La recepción definitiva se hará en las mismas condiciones que la provisional, extendiéndose el acta correspondiente. El Director Técnico propondrá a la Junta Económica la devolución de la fianza al contratista de acuerdo con las condiciones económicas legales establecidas.

23. Las tarifas para la determinación de honorarios, reguladas por orden de la Presidencia del Gobierno el 19 de Octubre de 1961, se aplicarán sobre el denominado en la actualidad “Presupuesto de Ejecución de Contrata” y anteriormente llamado ”Presupuesto de Ejecución Material” que hoy designa otro concepto.

Condiciones particulares

La empresa consultora, que ha desarrollado el presente proyecto, lo entregará a la empresa cliente bajo las condiciones generales ya formuladas, debiendo añadirse las siguientes condiciones particulares:

1. La propiedad intelectual de los procesos descritos y analizados en el presente trabajo, pertenece por entero a la empresa consultora representada por el Ingeniero Director del Proyecto.

2. La empresa consultora se reserva el derecho a la utilización total o parcial de los resultados de la investigación realizada para desarrollar el siguiente proyecto, bien para su publicación o bien para su uso en trabajos o proyectos posteriores, para la misma empresa cliente o para otra.

3. Cualquier tipo de reproducción aparte de las reseñadas en las condiciones generales, bien sea para uso particular de la empresa cliente, o para cualquier otra aplicación, contará con autorización expresa y por escrito del Ingeniero Director del Proyecto, que actuará en representación de la empresa consultora.

4. En la autorización se ha de hacer constar la aplicación a que se destinan sus reproducciones así como su cantidad.

5. En todas las reproducciones se indicará su procedencia, explicitando el nombre del proyecto, nombre del Ingeniero Director y de la empresa consultora.

6. Si el proyecto pasa la etapa de desarrollo, cualquier modificación que se realice sobre él, deberá ser notificada al Ingeniero Director del Proyecto y a criterio de éste, la empresa consultora decidirá aceptar o no la modificación propuesta.

7. Si la modificación se acepta, la empresa consultora se hará responsable al mismo nivel que el proyecto inicial del que resulta el añadirla.

8. Si la modificación no es aceptada, por el contrario, la empresa consultora declinará toda responsabilidad que se derive de la aplicación o influencia de la misma.

9. Si la empresa cliente decide desarrollar industrialmente uno o varios productos en los que resulte parcial o totalmente aplicable el estudio de este proyecto, deberá comunicarlo a la empresa consultora.

10. La empresa consultora no se responsabiliza de los efectos laterales que se puedan producir en el momento en que se utilice la herramienta objeto del presente proyecto para la realización de otras aplicaciones.

11. La empresa consultora tendrá prioridad respecto a otras en la elaboración de los proyectos auxiliares que fuese necesario desarrollar para dicha aplicación industrial, siempre que no haga explícita renuncia a este hecho. En este caso, deberá autorizar expresamente los proyectos presentados por otros.

12. El Ingeniero Director del presente proyecto, será el responsable de la dirección de la aplicación industrial siempre que la empresa consultora lo estime oportuno. En caso contrario, la persona designada deberá contar con la autorizacion del mismo, quien delegara en el las responsabilidades que ostente.