Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Departamento de Ingeniería Telemática

ESTUDIO DE LA INFLUENCIA DE CÓDECS VOIP EN EL CONSUMO ENERGÉTICO EN SMARTPHONES CON SISTEMA OPERATIVO ANDROID

Autor: Esteban Castilla Forero
Tutor: Juan M. Vozmediano Torres

Proyecto Fin de Carrera

Ingeniería de Telecomunicación

Sevilla, 10 de noviembre de 2012

Prefacio

Este proyecto tiene como objetivo principal realizar un análisis de la influencia que diferentes códecs VoIP tienen sobre el consumo energético de un smartphone con S.O Android. Los códecs bajo estudio son G.711, G.729, G.723.1, iLBC y AMR, y los elementos del dispositivo que se analizarán con el fin de determinar el consumo son la CPU, encargada de tareas de codificación y decodificación de la voz, y la tarjeta de red inalámbrica, encargada del envío y la recepción de paquetes.

Los resultados muestran cómo la calidad de la voz o el ancho de banda no son los únicos criterios a la hora de elegir un determinado códec, siendo el consumo energético un aspecto a tener muy en cuenta, sobre todo en entornos en movilidad, donde el uso de un dispositivo portátil es obligado y cuya limitación más importante es la duración de la batería.

Índice

Índice de Figuras

Índice de Tablas

Capítulo 1

Introducción

1.1 Motivación y objetivos del proyecto

Desde que los ordenadores personales se implantaran de forma generalizada a mediados de la década de los ochenta la tecnología ha avanzado hasta la actualidad, donde los dispositivos portátiles (teléfonos móviles, lectores de libros electrónicos, tablets, smartphones, laptops, etc.) son cada vez más accesibles para la población en general, debido a su constante abaratamiento.

Sin embargo todos estos dispositivos portables presentan una importante limitación común: la duración de la batería. Es por ello que la rápida expansión del mercado multimedia y el desarrollo de aplicaciones demanda una optimización de la energía consumida por estos terminales.

Dentro del grupo de las aplicaciones multimedia destacan las destinadas a la comunicación VoIP tales como Skype, Viber, Google Talk, Tango, etc, las cuales son cada vez más utilizadas por los usuarios. Sin embargo, según los artículos de investigación [3] y [4], muestran un consumo energético mayor que el resto, debido principalmente al uso intensivo que realizan de la CPU y de la tarjeta de interfaz de red. Por todo ello, se hace necesario llevar a cabo un análisis de este tipo de aplicaciones a través de parámetros que normalmente son configurables por parte de desarrolladores o usuarios, como es caso de los códecs, cobrando una especial relevancia el estudio de la influencia de éstos sobre el consumo energético.

El objetivo del proyecto será por tanto realizar un estudio comparativo del consumo energético provocado por diferentes códecs usados en aplicaciones de VoIP: AMR, iLBC, g.711, g.729 y g.723.1. Este estudio se llevará a cabo analizando el consumo de los dos elementos de los dispositivos portátiles más determinantes en las aplicaciones VoIP, como son la CPU y la tarjeta de la interfaz WiFi.

De cualquier modo, de toda la variedad de dispositivos móviles existentes en el mercado, en este proyecto se analizarán los smartphones con sistema operativo Android, sistema cuya cuota de mercado está experimentando un mayor crecimiento. Actualmente Android, según la compañía IDC [5], cuenta con aproximadamente el 70 % de cuota de mercado a nivel mundial, tal y como se puede observar en la Figura 1.1. En Estados Unidos la cuota de mercado alcanza una cifra del 52 % [6] y [7] y en España un 84 % [8]. Además, en apenas 4 años de vida

de Android, se han activado un total de 400 millones de dispositivos [9]. Estos datos sobre la penetración de Android, hacen necesario que la influencia de los códecs VoIP sobre el consumo energético sea estudiada de forma separada en esta plataforma.

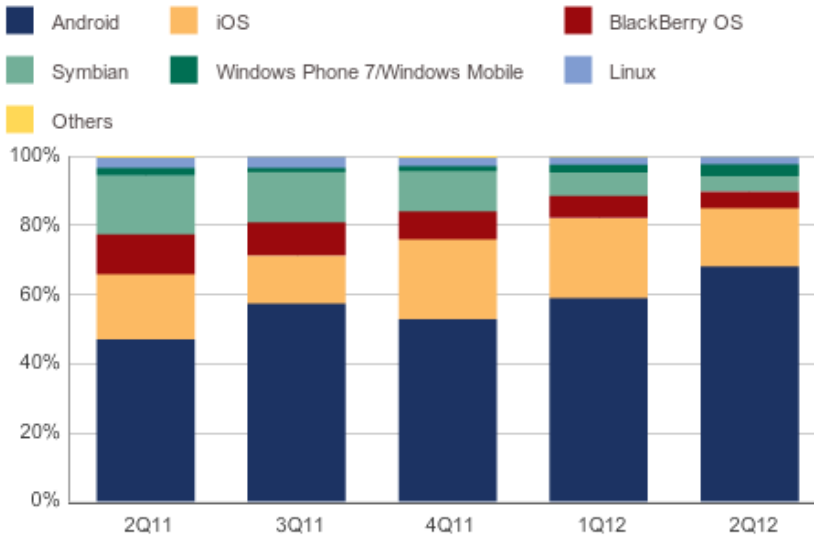


Figura 1.1: Cuota de mercado mundial S.O smartphones. 2º Cuatrimestre 2012

En cuanto a la versión del sistema operativo, se realizarán las medidas del consumo sobre la versión 2.3 (Gingerbread), ya que, tal y como se observa en la Figura 1.2, a pesar de existir varias versiones posteriores, sigue siendo la versión más extendida entre los usuarios [10].

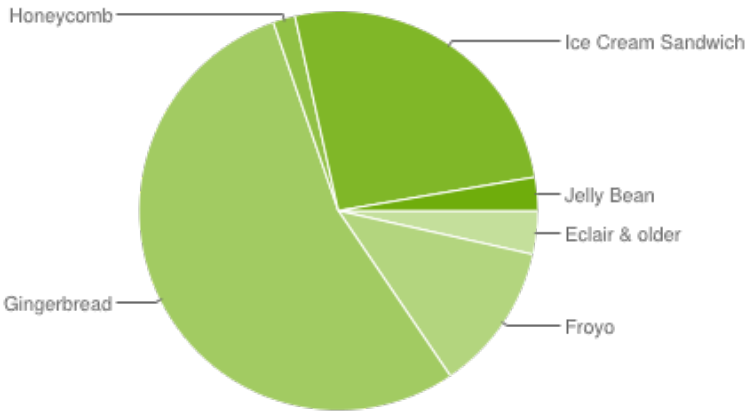


Figura 1.2: Distribución de versiones Android (Noviembre 2012)

1.2 Antecedentes

Este proyecto parte de un estudio similar realizado anteriormente [11] en el que en lugar de analizarse la influencia de los códecs VoIP sobre el consumo en smartphones Android, se realiza de forma general tomándose las medidas en un

ordenador portátil, con el fin de obtener el consumo para la CPU y la tarjeta de red inalámbrica.

Para la CPU, la metodología que sigue el estudio consiste en modificar el código fuente correspondiente a los codecs, con el fin de tomar medidas del tiempo empleado en codificar y decodificar conversaciones reales. A continuación se halla una curva de descarga de la batería del dispositivo con el fin de obtener la relación entre el tiempo y la energía consumida. Finalmete con este dato y las medidas de los tiempos de codificación-decodificación se obtienen los resultados de consumo para cada códec.

En cuanto al consumo de la interfaz Wifi, se estima de forma teórica, a partir de ciertos artículos de investigación, y teniendo en cuenta el funcionamiento de de la interfaz WiFi en aplicaciones VoIP, tanto en el modo con ahorro de energía (PSM) como sin él.

1.3 Aspectos teóricos de la codificación vocal

Un códec, palabra proveniente del inglés (Code-Decode), es un método capaz de comprimir y/o descomprimir una señal digital a través de un software o un circuito impreso. Por un lado, los códecs codifican flujos o señales que pueden ser posteriormente transmitidas, almacenadas o cifradas y por otro lado, decodifican estos flujos o señales para su posterior edición. Hay muchas maneras de transformar una señal de voz analógica, todas ellas gobernadas por varios estándares. El proceso de la conversión es complejo. Es suficiente decir que la mayoría de las conversiones se basan en la modulación codificada mediante pulsos (PCM) o variaciones.

El motivo de que existan diferentes algoritmos de compresión y descompresión radica en la necesidad de satisfacer necesidades relativas a distintos parámetros: calidad de la señal restituída, tiempos de compresión y descompresión, limitaciones relacionadas con el procesador o memoria, o en la tasa de bits a transmitir resultante de la compresión.

A continuación se explicarán brevemente los procesos previos a la codificación (muestreo y cuantificación) y se expondrá en qué consiste la codificación y las características principales de los códecs que serán analizados en el proyecto. Por último se detalla el funcionamiento de un algoritmo que incorporan algunos códecs, cuya función es detectar la actividad vocal y así ahorrar ancho de banda en los periodos en los que existan silencios.

1.3.1 Muestreo

El proceso de muestreo consiste en tomar valores instantáneos de una señal analógica, llamados muestras, a intervalos de tiempo iguales. El proceso se ilustra en la Figura 1.3.

La frecuencia de muestreo viene determinada por el teorema de Nyquist, según el cual la tasa de muestreo debe ser igual al doble de la mayor frecuencia de la

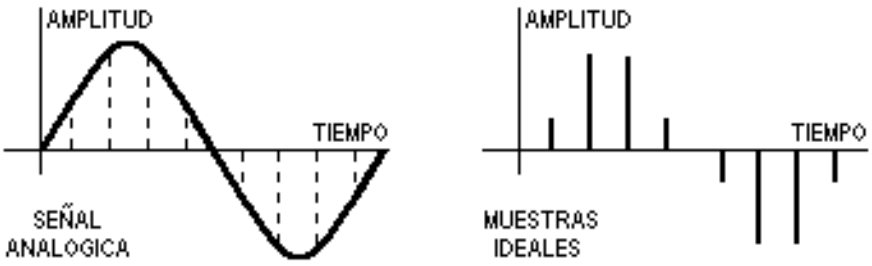


Figura 1.3: Proceso de muestreo.

fuente analógica.

De acuerdo con el teorema del muestreo, las señales telefónicas de frecuencia vocal (que ocupan la Banda de 300 Hz a 3400 Hz), se han de muestrear a una frecuencia igual o superior a 6800 Hz. En la practica, sin embargo, se suele tomar una frecuencia de muestreo de 8 KHz, es decir una muestra cada 125μs.

1.3.2 Cuantificación

El proceso de cuantificación consiste en la asignación de un valor discreto correspondiente a cada valor real de la muestra. Como la cuantificación siempre se lleva a cabo por aproximación, existe una diferencia entre la señal original y la cuantificada conocida como error o ruido de cuantificación, e interesa que sea lo más pequeño posible.

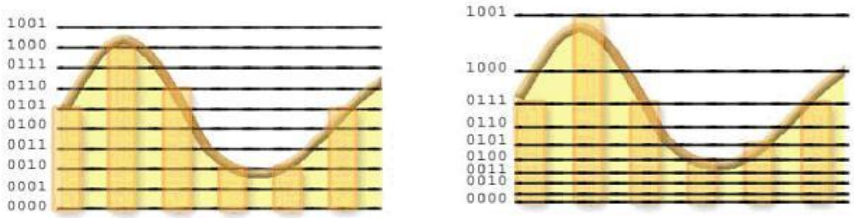
Cuantos más bits se utilicen para representar cada muestra, mayor será la fidelidad de la señal en destino y, por tanto, menor el ruido de cuantificación. El inconveniente es que el ancho de banda requerido es directamente proporcional al número de bits por muestra. Los tipos más destacados de cuantificadores son:

Cuantificador uniforme: el paso de cuantificación es constante. No hace suposición alguna acerca de la señal a cuantificar. Por ello, es el tipo más fácil y económico de implementar.

Cuantificador logarítmico: la resolución del cuantificador es mayor en las partes de la señal que presenta menor amplitud. Esto se traduce en un incremento de la distancia entre los niveles de reconstrucción conforme aumenta la amplitud de la señal. De este modo se consigue un error de cuantificación muy inferior al que se consiguiría usando un cuantificador uniforme. Este tipo de cuantificación es el más adecuado para la digitalización de señales vocales.

Cuantificador vectorial: las muestras se cuantifican en bloques de N muestras. Es el que menor error de cuantificación presenta. Sin embargo, es el más sensible a errores de transmisión y el más complejo de implementar.

En la Figura 1.4 se ilustran las cuantificaciones uniforme y logarítmica:



(a) Cuantificación uniforme (b) Cuantificación logarítmica

Figura 1.4: Cuantificación uniforme y logarítmica

1.3.3 Codificación

La codificación es la traducción al sistema binario de los valores analógicos ya cuantificados. Es posible distinguir tres grandes grupos de codificadores de voz, dependiendo del método de codificación utilizado:

Codificadores de forma de onda: realizan una aproximación no lineal de la la forma de onda de la señal original a una tasa de bits relativamente alta (valores típicos de 32 ó 64 kbps). Algunos algoritmos de codificación que pertenecen a este grupo son PCM (Pulse Code Modulation) o ADPCM (Adaptive Differential PCM).

Vocoders: intentan producir una señal que suene como la voz original, independientemente de si la forma de onda se parece o no. En el transmisor se analiza la voz y se extraen los parámetros del modelo y la excitación. Esta información se envía al receptor, donde se sintetiza la voz.Con este tipo de codificadores, se consiguen bajas tasas de bits, aunque a cambio de una mala calidad de la señal de voz producida. Un algoritmo muy usado de este grupo es el LPC (Linear Predictive Coder), utilizado frecuentemente para mitigar las pérdidas producidas entre la transmisión y la recepción de las muestras vocales.

Codificadores híbridos: se combinan las técnicas de los codificadores de forma de onda con las de los vocoders, con el propósito de obtener una alta calidad de voz a tasas bajas. En estos codificadores, las muestras de la señal de entrada se dividen en bloques (vectores) que son procesados como si fueran uno solo. Llevan a cabo una representación paramétrica de la señal de voz para tratar que la señal sintetizada sea lo más parecida a la original. Algoritmos de codificación de este tipo son, entre otros, CELP (Code-Excited Linear Prediction), ACELP (Algebraic CELP) y CS-ACELP (Conjugated Structure ACELP).

1.3.4 Transmisión discontinua

Con el fin de reducir el consumo de energía en los terminales y aliviar la congestión de las redes, los códecs AMR, G.723.1 y G.729 ofrecen la posibilidad de interrumpir la transmisión de datos durante los periodos de silencio.

Dos módulos se encargan de administrar esta funcionalidad:

- El módulo de detección de actividad vocal, VAD (Voice Activity Detection), que se encarga de detectar la ausencia o presencia de actividad vocal en la señal de entrada.
- El módulo de transmisión discontinua, DTX (Discontinuous Transmission), que gestiona el envío o no de tramas en función de la información proporcionada por el VAD.

Durante los periodos de inactividad o no transmisión, en el extremo del decodificador, las tramas inactivas se reconstruyen mediante la generación de ruido de confort (CNG, Comfort Noise Generator), un ruido artificial de fondo que evita que resulte desagradable la ausencia total de sonido en una conversación. Las características de este ruido de fondo son enviadas por el extremo del codificador a través de tramas SID (Silence Insertion Descriptor). Este tipo de tramas son transmitidas al principio de los periodos de inactividad vocal y posteriormente, dependiendo del códec, a intervalos regulares o irregulares de tiempo, durante toda la duración del silencio.

1.3.5 Códecs VoIP

Las distintas técnicas de codificación existentes han ido dando lugar con el paso del tiempo a diferentes recomendaciones creadas por diversos organismos de estandarización, tales como la ITU-T (Unión Internacional de Telecomunicaciones), el 3GPP (3rd Generation Partnership Project) y el IETF (Internet Engineering Task Force). A continuación se detallan las características principales de los códecs VoIP más utlizados.

G.711

Fue publicado en 1972 por la ITU-T [12] y además de ser usado en VoIP, es el estándar de codificación de la voz usado principalmente en las redes de telefonía conmutadas.

El G.711 representa una modulación por impulsos codificados (PCM, Pulse Code Modulation) logarítmica de muestras de voz a una frecuencia de 8 kHz. Se han definido dos algoritmos de compresión para este estándar:

- Ley-μ, utilizado en Norte América y Japón.
- Ley-A, utilizado en Europa

Los algoritmos de la Ley-μ y de la Ley-A codifican las muestras de 14 y 13 bits respectivamente en otras de 8 bits. La tasa de bits del códec G.711 es de 64 kbps. Ambos algoritmos están basados en una cuantificación logarítmica, lo que implica que la densidad de niveles de cuantificación es mayor alrededor del nivel cero de amplitud. La principal diferencia entre ellos radica en que los bits del 2 al 8 están invertidos de uno con respecto al otro. El bit 1, el que está más a la izquierda, es en ambos el que representa el signo, y es 1 para valores positivos y 0 para negativos.

Debido al elevado consumo de ancho de banda, G.711 se reserva a aplicaciones de VoIP que requieren la máxima calidad posible.

G.729

Este códec hace uso de un algoritmo de compresión mediante predicción lineal con excitación por código algebraico de estructura conjugada (CS-ACELP) a 8 kbps como esquema de codificación.

Ha sido normalizado por la ITU-T [13], y ofrece una alta calidad y un rendimiento robusto a cambio de una alta complejidad. Opera con paquetes de 10 ms y genera tramas de 80 bits de longitud con un retardo de 15 ms, de los cuales 5 ms corresponden a retardo de anticipación.

Al basarse en un codificador CELP, cada segmento de 80 bits producido contiene los coeficientes de predicción linear, los índices de excitación y los parámetros de ganancia que serán usados en el extremo del decodificador para reproducir la señal de voz recibida. Las entradas y salidas de estre algortimo son muestras PCM de 16 bits que se convertirán a/de muestras de 8 bits.

Existen diferentes extensiones de la norma, cuyas características más significativas se expoenen a continuación:

- G.729A: Es una variante que requiere una potencia de cálculo menor a cambio de una disminución en la calidad de la voz. Las modificaciones que presenta este anexo conllevan simplificaciones en las rutinas de búqueda de los codebook y en el postfiltrado del decodificador.
- G.729B: En este anexo se describe un esquema de compresión de silencios para G.729. Se basa en el uso transmisión discontinua (DTX), detección de actividad vocal (VAD) y generación de ruido de confort (CNG), lo que permite reducir el ancho de banda utilizado al disminuir el número de tramas enviadas en periodos de silencios.
- G.729AB: Se trata del G.729 simplificado (G.729A) aunque añadiendo un esquema de compresión de silencios.

G.723.1

G.723.1 es un códec de la ITU-T [14] que asegura la compresión de la voz en tramas de 30 ms. Es muy utilizado en VoIP debido a su bajo consumo de ancho de banda. Por contra presenta un retardo de codificación de 37,5 ms, repartidos en 30 ms de retardo de empaquetado y 7,5 ms de anticipación.

Incorpora dos algoritmos de codificación de tramas que utilizan dos tasas de bits diferentes:

- 6,3 kbps con tramas de 24 bytes utilizando el algoritmo MPC-MLQ.
- 5,3 kbps con tramas de 20 bytes utilizando el algoritmo de codificación ACELP.

Posee una variante en la que se lleva a cabo compresión de silencios.

iLBC

iLBC (Internet Low Britrate Codec) fue estandarizado por el IETF [15]. Se basa en el algoritmo LPC (Linear-Predective Coding) y puede trabajar con dos tamaños de trama diferentes: 20 ms a 15,2 kbps y 30 ms, a 13,33 kbps. Es uno de los códecs que proporcionan mayor calidad, comparable con la que se obtiene con G.711 aunque con mucho menos consumo de ancho de banda.

Aunque el iLBC carece de un mecanismo específico que mitigue los efectos de la pérdida de paquetes, sí que permite una fácil implementación del mismo, por ejemplo, interpolando los paquetes inmediatamente anterior y posterior al perdido para suministrar un paquete sustituto. Por este motivo, es uno de los códecs más robustos frente a pérdidas y retrasos: si se perdieran tramas vocales durante el transcurso de una conversación, el impacto sobre ésta sería poco notable, bastante menor en comparación con los demás códecs descritos anteriormente.

AMR

AMR, del inglés Adaptative Multi-Rate, es un códec estandarizado por el 3GPP [17] y ampliamente utilizado en comunicaciones móviles.

Como su nombre indica, es un codificador multitasa que permite funcionar a 12,2, 10,2, 7,95, 7,40, 4,70, 5,90, 5,15 y 4,75 kbps, consiguiendo bloques codificados de 244, 204, 159, 148, 134, 118, 103 y 95 bits respectivamente, con una duración de trama constante de 20 ms.

AMR utiliza un codificador de tipo de híbrido con un algoritmo de predicción lineal con excitación por código algebraico de estructura conjugada (ACELP). El retraso de empaquetado es de 20 ms, al que hay que sumar un retardo de anticipación de 5 ms para todos los modos, salvo para la tasa de 12,2 kbps.

AMR, al igual que G.723.1 o G.729B, utiliza también un esquema de compresión de silencios, lo que permite reducir el ancho de banda durante los periodos inactivos.

1.4 El sistema operativo Android

1.4.1 Introducción e Historia

Android es un sistema operativo basado en Linux especialmente diseñado para dispositivos móviles. Es muy común en smartphones y tablets, aunque también es posible encontrarlo en set-top-boxes para TV, en los propios televisores, cámaras digitales, etc.

El sistema operativo Android nace con la compañía Android Inc, fundada en Octubre de 2003 por Andy Rubin entre otros, y que en verano de 2005 es adquirida por Google. El anuncio del sistema Android se realizó el 5 de noviembre de 2007 junto con la creación de la Open Handset Alliance, un consorcio de casi 100 compañías de hardware, software y telecomunicaciones dedicadas al desarrollo de

estándares abiertos para dispositivos móviles.

Android se distribuye bajo dos licencias, una que abarca todo el código del kernel, GNU GPLv2, y otra para el resto de componentes del sistema, que es Apache v2. Cuenta con una comundiad de desarrolladores muy amplia, lo que ha permitido que en su tienda oficial de aplicaciones, Google Play, se haya alcanzado el número de 675000 disponibles.

Actualmente el sistema cuenta con diez versiones, siendo la última la 4.2. Las diferentes versiones toman nombres de dulces o postres, empezando por una letra distinta en orden alfabético en cada cambio de la misma [10]. Este gran número de versiones distintas añadido a la gran variedad de dispositivos diferentes y a que la actualización de versión depende de tres agentes principalmente que son Google, fabricante y compañía operadora, provocan uno de los mayores problemas del sistema Android: la fragmentación. Como consecuencia se puede observar en la Figura 1.2 y Figura 1.5, por un lado cómo la versión más extendida (y sobre la que se realizarán las medidas) es la versión 2.3 Gingerbread, que se lanzó en Diciembre de 2010 y por otro, la lenta expansión de las versiones posteriores, por ejemplo de las 4.0 y 4.1 que fueron lanzadas en Octubre de 2011 y Junio de 2012 respectivamente [18].

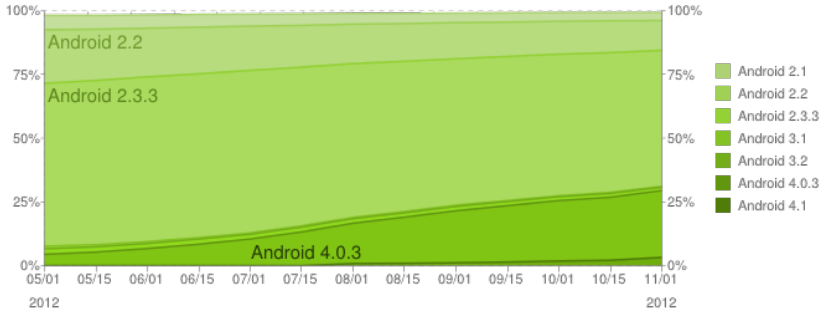


Figura 1.5: Distribución histórica de versiones Android (1 año)

1.4.2 Arquitectura

Como se muestra en la Figura 1.6, los componentes que forman Android se agrupan en capas. Cada una de estas capas utiliza elementos de la capa inferior para realizar sus funciones.

A continuación se expondrá con más detalle en qué consiste cada capa.

Kernel de Linux

El núcleo del sistema operativo Android es un kernel Linux versión 2.6, similar al que puede incluir cualquier distribución de Linux, solo que adaptado a las características del hardware en el que se ejecutará Android (normalmente, un smartphone).

Proporciona una capa de abstracción para los elementos hardware a los que tienen que acceder las aplicaciones. Esto permite que se pueda acceder a esos

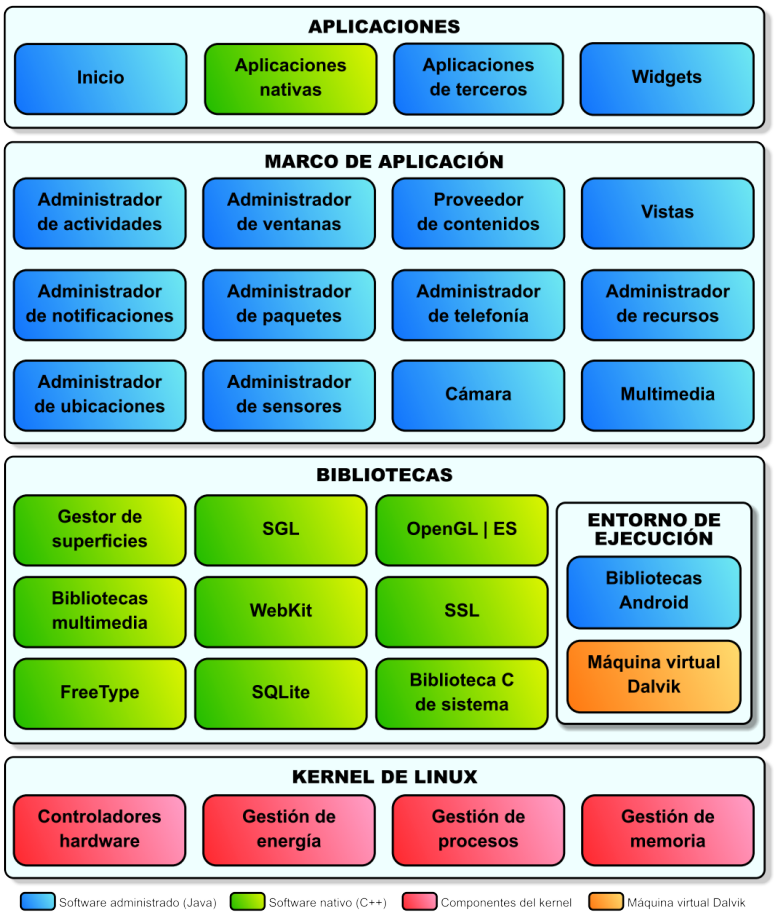


Figura 1.6: Arquitectura del S.O Android

componentes sin necesidad de conocer el modelo o características precisas de los que están instalados en cada teléfono. Para cada elemento hardware del teléfono existe un controlador (o driver) dentro del kernel que permite utilizarlo desde el software. Además de proporcionar controladores hardware, el kernel se encarga de gestionar los diferentes recursos del teléfono (energía, memoria, ...) y del sistema operativo en sí: procesos, comunicaciones, etc.

Bibliotecas

La capa que se sitúa justo sobre el kernel la componen las bibliotecas nativas de Android. Estas bibliotecas están escritas en C o C++ y compiladas para la arquitectura hardware específica del dispositivo. Su cometido es proporcionar funcionalidad a las aplicaciones, para tareas que se repiten con frecuencia, evitando tener que codificarlas cada vez y garantizando que se llevan a cabo de la forma más eficiente.

Estas son algunas de las bibliotecas que se incluyen habitualmente:

- Gestor de superficies (Surface Manager): se encarga de componer las imágenes que se muestran en la pantalla a partir de capas gráficas 2D y 3D. Cada vez que la aplicación pretende mostrar algo en pantalla, la biblioteca realiza

los cambios en imágenes (mapas de bits) que almacena en memoria y que después combina para formar la imagen final que se envía a pantalla. Esto permite realizar con facilidad diversos efectos: superposición de elementos, transparencias, transiciones, animaciones, etc.

- SGL (Scalable Graphics Library): se encarga de representar elementos en dos dimensiones. Es el motor gráfico 2D de Android.
- OpenGL ES (OpenGL for Embedded Systems): motor gráfico 3D. Utiliza aceleración hardware (si el dispositivo la proporciona) o un motor software altamente optimizado.
- Bibliotecas multimedia: basadas en OpenCORE, permiten visualizar, reproducir y grabar numerosos formatos de imagen, vídeo y audio como JPG, GIF, PNG, MPEG4, AVC (H.264), MP3, AAC o AMR.
- WebKit: motor web utilizado por el navegador (tanto como aplicación independiente como embebido en otras aplicaciones).
- SSL (Secure Sockets Layer): proporciona seguridad al acceder a Internet.
- FreeType: permite mostrar fuentes tipográficas, tanto basadas en mapas de bits como vectoriales.
- SQLite: motor de bases de datos relacionales, disponible para todas las aplicaciones.
- Biblioteca C de sistema (libc): está basada en la implementación de Berkeley Software Distribution (BSD), pero optimizada para sistemas Linux embebidos. Proporciona funcionalidad básica para la ejecución de las aplicaciones.

Entorno de ejecución

El entorno de ejecución de Android, aunque se apoya en las bibliotecas enumeradas anteriormente, no se considera una capa en sí mismo, dado que también está formado por bibliotecas. En concreto, las bibliotecas esenciales de Android, que incluyen la mayoría de la funcionalidad de las bibliotecas habituales de Java así como otras específicas de Android.

El componente principal del entorno de ejecución de Android es la máquina virtual Dalvik, componente que ejecuta todas y cada una de las aplicaciones no nativas de Android. Las aplicaciones se codifican normalmente en Java y son compiladas, pero no para generar un ejecutable binario compatible con la arquitectura hardware específica del dispositivo Android. En lugar de eso, se compilan en un formato específico para la máquina virtual Dalvik, que es la que las ejecuta. Esto permite compilar una única vez las aplicaciones y distribuirlas ya compiladas teniendo la total garantía de que podrán ejecutarse en cualquier dispositivo Android que disponga de la versión mínima del sistema operativo que requiera cada aplicación.

Aunque las aplicaciones se escriben en Java, Dalvik no es realmente una máquina virtual Java. Java se usa únicamente como lenguaje de programación, pero

los ejecutables que se generan con el SDK de Android no son ejecutables Java convencionales. Durante el proceso de compilación de los programas Java (normalmente archivos .java) sí que se genera, de forma intermedia, el bytecode habitual (archivos .class). Pero esos archivos son convertidos al formato específico de Dalvik en el proceso final (.dex, de Dalvik executable).

Los archivos .dex son mucho más compactos que los .class equivalentes (hasta un 50 % menos de tamaño), lo que permite ahorrar espacio en el teléfono y acelerar el proceso de carga. Además, a diferencia de las máquinas virtuales tradicionales, Dalvik se basa en registros en lugar de una pila para almacenar los datos, lo que requiere menos instrucciones. Esto permite ejecuciones más rápidas en un entorno con menos recursos.

Las aplicaciones Android se ejecutan cada una en su propia instancia de la máquina virtual Dalvik, evitando así interferencias entre ellas, y tienen acceso a todas las bibliotecas mencionadas antes y, a través de ellas, al hardware y al resto de recursos gestionados por el kernel.

Marco de aplicación

Esta capa la forman todas las clases y servicios que utilizan directamente las aplicaciones para realizar sus funciones, y que se apoyan en las bibliotecas y en el entorno de ejecución previamente mencionados. La mayoría de los componentes de esta capa son bibliotecas Java que acceden a los recursos a través de la máquina virtual Dalvik. Entre las más importantes se encuentran las siguientes:

- Administrador de actividades (Activity Manager): se encarga de controlar el ciclo de vida de las actividades y la propia pila de actividades. Las actividades se pueden definir como las ventanas que se muestran, una sobre otra, en la pantalla del dispositivo Android.

- Administrador de ventanas (Windows Manager): se encarga de organizar lo que se muestra en pantalla, creando superficies que pueden ser rellenadas por las actividades.

- Proveedor de contenidos (Content Provider): permite encapsular un conjunto de datos que va a ser compartido entre aplicaciones creando una capa de abstracción que hace accesible dichos datos sin perder el control sobre cómo se accede a la información.

- Vistas (Views): las vistas se pueden definir como los controles que se suelen incluir dentro de las actividades (ventanas). Android proporciona numerosas vistas con las que construir las interfaces de usuario: botones, cuadros de texto, listas, etc.

- Administrador de notificaciones (Notification Manager): proporciona servicios para notificar al usuario cuando algo requiera su atención.

- Administrador de paquetes (Package Manager): las aplicaciones Android se distribuyen en paquetes (archivos .apk) que contienen tanto los archivos .dex como todos los recursos y archivos adicionales. Esta biblioteca permite

obtener información sobre los paquetes instalados en el dispositivo, además de gestionar la instalación de nuevos paquetes.

- Administrador de telefonía (Telephony Manager): proporciona acceso a la pila hardware de telefonía del dispositivo Android, si la tiene. Permite realizar llamadas o enviar y recibir SMS/MMS.
- Administrador de recursos (Resource Manager): proporciona acceso a todos los elementos propios de una aplicación que se incluyen directamente en el código: cadenas de texto traducidas a diferentes idiomas, imágenes, sonidos y disposiciones de las vistas dentro de una actividad (layouts). Permite gestionar esos elementos fuera del código de la aplicación y proporcionar diferentes versiones en función del idioma del dispositivo o la resolución de pantalla.
- Administrador de ubicaciones (Location Manager): permite determinar la posición geográfica del dispositivo Android (usando el GPS o las redes disponibles) y trabajar con mapas.
- Administrador de sensores (Sensor Manager): permite gestionar todos los sensores hardware disponibles en el dispositivo Android.
- Cámara: proporciona acceso a las cámaras del dispositivo Android.
- Multimedia: conjunto de bibliotecas que permiten reproducir y visualizar audio, vídeo e imágenes en el dispositivo.

Aplicaciones

La capa superior de esta pila software la forman, como no podría ser de otra forma, las aplicaciones. Estas pueden ser nativas (escritas en C o C++) o administradas (escritas en Java), con interfaz gráfica o sin ella.

Aquí está también la aplicación principal del sistema: Inicio (Home), también llamada a veces lanzador (launcher), porque es la que permite ejecutar otras aplicaciones proporcionando la lista de aplicaciones instaladas y mostrando diferentes escritorios donde se pueden colocar accesos directos a aplicaciones o incluso pequeñas aplicaciones incrustadas o widgets, que son también aplicaciones de esta capa.

Capítulo 2

Metodología

El presente capítulo explica la metodología utilizada en el desarrollo del proyecto. Como se ha explicado anteriormente el análisis de la influencia en el consumo energético de los códecs VoIP se realiza teniendo en cuenta los dos componentes del dispositivo portátil que más utilizan las aplicaciones VoIP: la CPU para la codificación-decodificación y la tarjeta de la interfaz de red, en este caso la tarjeta WiFi, para la transmisión de los paquetes de voz.

Con el objetivo de medir el consumo energético de la CPU, existen estudios que proponen modelos para ello, como [20] y [21]. Sin embargo resultan complejos, por lo que se utilizará el modelo propuesto en [11], el cual propone que el consumo se puede expresar como el producto del tiempo en que la CPU tarda en desempeñar una tarea por una constate que dependerá de cada dispositivo.

La metodología seguida en el análisis del consumo se basará a grandes rasgos en la seguida en [11]. Para el de la CPU será prácticamente idéntica, mientras que para la interfaz Wifi se seguirá una metodología distinta. A diferencia del citado artículo en que el análisis del consumo se realiza de forma teórica, en el proyecto se seguirá la misma metodología que el consumo de la CPU, realizándose medidas reales durante el proceso de envío de paquetes codificados de voz.

El smartphone con S.O. Android sobre el que se realizarán las medidas se trata de un Samsung Galaxy S II. Sus especificaciones se detallan a continuación:

- Procesador Exynos DualCore 1.2 GHz.
- 1 GB de memoria RAM
- 16 GB de almacenamiento interno.
- Versión de Android 2.3 Gingerbread.

Con el objetivo de poder recoger ciertos datos del dispositivo, es necesario disponer de acceso root en el mismo. Afortunadamente en la gran mayoría de dispositivos Android no supone un problema y se ha conseguido instalando un kernel modificado. El kernel instalado incluye un conjunto de herramientas llamada *busybox*, gracias a la cual se extienden las posibilidades de usar comandos Linux que el kernel que trae instalado el smartphone de serie no incopora.

Además del smartphone para la recogida de medidas se ha utilizado un ordenador, para realizar tanto las tareas de programación de scripts como las de manipulación de los datos recogidos para su posterior representación gráfica. Las principales características del mismo son las siguientes:

- Procesador Intel Core 2 Duo 2.4 GHz.
- 2 GB de memoria RAM
- 30 GB de disco duro.
- Sistema Operativo Linux Ubuntu 12.04 de 32 bits.

2.1 Análisis del consumo de CPU

Siguiendo el modelo propuesto en [11] y que ha sido introducido anteriormente, la metodología seguida para analizar el consumo energético de la CPU ha consistido a grandes rasgos en tomar medidas de los tiempos de codificación y decodificación para cada códec analizado. Acto seguido se halla una curva de descarga de la batería, con el fin de obtener la constante de potencia y finalmente calcular el consumo en unidades de energía a partir de las medidas temporales tomadas anteriormente.

A continuación se detallan los pasos seguidos.

2.1.1 Instalación del entorno de trabajo

En primer lugar será necesario instalar en el ordenador una serie de software necesario para llevar una serie de tareas necesarias para trabajar con Android. A continuación se expondrá el software instalado.

Android SDK

El SDK (Software Development Kit) es el conjunto de herramientas y librerías que permiten crear aplicaciones para el sistema operativo Android. Como ya se expondrá más adelante, no se desarrollarán los programas de medida con esta herramienta, sin embargo sí que será de gran utilidad el ADB (Android Debug Bridge) que incorpora. El ADB [22] es una herramienta de línea de comandos muy versátil que permite la comunicación con una dispositivo Android conectado a través del cable USB. Se trata de un programa cliente-servidor que permite, entre otras las siguientes operaciones:

- Administrar el estado del dispositivo.
- Ejecutar comandos de shell en el dispositivo.
- Copiar archivos a/desde el dispositivo.

Android NDK

El NDK [23] es un conjunto de herramientas que permite incorporar los componentes que hacen uso de código nativo en aplicaciones de Android. Las aplicaciones de Android se ejecutan en la máquina virtual Dalvik. El NDK permite implementar partes de aplicaciones utilizando código nativo en lenguajes como C y C++. Esto puede proporcionar beneficios para ciertas clases de aplicaciones, como por ejemplo en reutilización de código existente y hasta un aumento de velocidad en algunos casos.

2.1.2 Descarga y modificación del código fuente de los códecs

Con el objeto de obtener las medidas de consumo de CPU, se realizarán modificaciones en el código fuente de cada códec, descargado previamente de la página web de los organismos oficiales que los estandarizan.

El lenguaje de programación con el que se escriben la mayoría de las aplicaciones en Android es Java. Sin embargo, Android es un sistema operativo muy versátil, permitiendo trabajar también con código nativo, esto es C y C++. La ejecución de código nativo puede ser tanto a través de aplicaciones escritas enteramente en estos lenguajes como a través de aplicaciones que integran ambos tipos de lenguajes. Esto es posible gracias al NDK y al JNI (Java Native Interface), que consiste en un conjunto de herramientas que permite a una aplicación Java que está ejecutándose en una máquina virtual poder interactuar con programas escritos en C o C++.

El código fuente de los diferentes códecs están escritos en el lenguaje de programación C, por tanto existen a priori dos opciones a la hora de abordar la toma de medidas:

- Hacer uso de una aplicación Java para la toma de medidas, integrando con ella el código C de los códecs y sus modificaciones.
- Escribir la aplicación directamente en código nativo y compilarla para ejecutarla en el dispositivo.

De las dos opciones anteriormente mencionadas se ha optado por aplicar en el proyecto la segunda. Para justificarlo, habría que tener en cuenta la arquitectura de Android. Durante la ejecución de la aplicación en un dispositivo, interesa que la toma de medidas se realice lo más directamente posible, de forma que las mismas no resulten falseadas. Así, resulta mas conveniente escribir la aplicación enteramente en código nativo, en este caso C, ya que de tratarse de una aplicación Java, ésta haría uso de una multitud de elementos que afectarían al consumo energético y a la toma de medidas que no merecen interés, tales como la pantalla por ejemplo. Además, al tener que hacer uso de una máquina virtual, su ejecución se vería afectada por procesos intermedios que no se pueden controlar y que no permitirían tomar las medidas de una forma tan directa.

En definitiva, al estar el código fuente de los códecs escritos en C, se utilizarán aplicaciones en este mismo lenguaje para aprovechar el código ya implementado y además, al tratarse de una ejecución de código enteramente nativo se procurará

de que usen sólamente las bibliotecas que resulten estrictamente necesarias, con el fin de que la toma de medidas se realice únicamente sobre los procesos que realmente son de interés, la codificación y la decodificación.

Una vez que se ha decidido sobre qué tipo de aplicación se va a utilizar, habría que proceder a modificar el código fuente de los códecs, con el fin de hacer posible la toma de medidas. Las posibilidades son cuantiosas, y tanto C como Java, ofrecen una multitud de opciones a la hora de realizar esta tarea.

Para la obtención de medidas de la CPU y la memoria se ha optado por el uso de la función `getrusage`, perteneciente a la librería GNU C. La función de la librería C utilizada proporciona datos exactos del consumo de CPU del sistema y de usuario cada vez que se ejecuta. Previamente se han descartado otras metodologías como por ejemplo la toma de marcas temporales mediante la función `gettimeofday` y similares, menos adecuadas, ya que `getrusage` sólo mide el tiempo de CPU, mientras que las otras miden el tiempo total transcurrido.

La función GETRUSAGE

La función `getrusage`, junto con el tipo de datos `struct rusage`, se pueden utilizar para monitorizar los recursos consumidos por un proceso. Ambos están declarados en el archivo `sys/resource.h`. A continuación se explican brevemente:

- `int getrusage (int processes, struct rusage *rusage)`

  Esta función informa sobre el uso de recursos totales del proceso especificado por el parámetro `processes` y almacena la información en `*rusage`. En la mayoría de los sistemas, `processes` tiene sólo dos valores válidos:

  `RUSAGE SELF`: Hace referencia al proceso en curso.

  `RUSAGE CHILDREN`: Todos los procesos hijos (directos o indirectos) que ya han terminado.

- `struct rusage`

  Este tipo de datos almacena estadísticas del uso de recursos. Es una estructura que está compuesta por los siguientes elementos:

  - `struct timeval ru_utime`: Tiempo consumido al ejecutar instrucciones de usuario.
  - `struct timeval ru_stime`: Tiempo de sistema consumido.
  - `long int ru_maxrss`: Tamaño máximo de la parte establecida como residente, en kilobytes. Esto es, el número máximo de kilobytes de memoria física que un proceso usa simultáneamente.
  - `long int ru_ixrss`: Tamaño total de la memoria compartida con otros procesos.
  - `long int ru_idrss`: Tamaño total de la memoria no compartida usada para datos.
  - `long int ru_isrss`: Tamaño total de la memoria no compartida usada como pila.

- `long int ru_minflt`: Número total de fallos de página que han sido proporcionadas sin requerir ninguna entrada/salida.
- `long int ru_majflt`: Número total de fallos de página proporcionados a través de entrada/salida.
- `long int ru_nswap`: Número de veces que los procesos han utilizado intercambios de memoria.
- `long int ru_inblock`: Número de veces que el sistema de archivos ha leído del disco.
- `long int ru_oublock`: Número de veces que el sistema de archivo ha escrito en el disco.
- `long int ru_msgsnd`: Número de mensajes IPC (Inter-Process Communication) enviados.
- `long int ru_msgrcv`: Número de mensajes IPC recibidos.
- `long int ru_nsignals`: Número de señales recibidas.
- `long int ru_nvcsw`: Número de veces que los procesos invocan un cambio de contexto voluntariamente.
- `long int ru_nivcswl`: Número de veces que ha tenido lugar un cambio de contexto involuntario (porque se haya desbloqueado, por ejemplo, un proceso con mayor prioridad).

Modificaciones del código fuente

Antes de explicar las modificaciones realizadas en el código fuente de cada códec conviene exponer cómo es su funcionamiento de manera resumida. La forma de trabajar de todos los programan que implementan los códecs es similar, de forma que mediante un bucle se va procesando trama a trama del fichero de audio de entrada de manera individual, tanto en al codificación como en la decodificación.

En cuanto a las modificaciones en el código fuente de cada códec, serán básicamente dos:

- Cada vez que una trama es procesada, se llama a la función `getrusage`, guardándose su contenido en una tabla de tipo `struct rusage`. De todos los elementos de los que se compone la estructura `rusage`, se tomarán dos: `ru_utime` y `ru_stime`. La suma de estos valores indica el tiempo total de CPU acumulado que ha consumido el proceso desde que se inició. Se ha definido una función auxiliar, común para todos los códecs, que se encarga de escribir en un fichero externo el contenido de la tabla de tipo `struct rusage`.

- Los códecs que implementan un esquema de detección de la actividad vocal y compresión de silencios, generan por tanto tramas SID y NOTX cuando codifican la ausencia de la voz. El tipo de la trama generada se guarda en una variable de tipo entero y toma un valor distinto según el tipo que se haya generado. Con el objetivo de diferenciar en los resultados los periodos de silencios y los periodos de voz y conocer el número de tramas generadas de cada tipo, se ha escrito en un fichero, llamado `ftype`, el valor de la variable anterior cada vez que se genera una trama. La variable toma el nombre de

`FTYP` para los códecs G.723, G.729B Y G.729AB, y `TXTYPE` en el caso de AMR. En la Tabla 2.1 se recogen los valores que pueden tomar según la decisión del algoritmo DTX.

Valor	G.723.1, G.729AB, G.729B	AMR
0	Trama no transmisión NOTXN	Trama con actividad vocal ACT
1	Trama con actividad vocal ACT	Primera trama SID
2	Trama SID	Trama SID de actualización
3		Trama no transmisión NOTXN

Tabla 2.1: Tipos de trama según módulo DTX

Se han reutilizado del estudio del consumo en laptops las modificaciones realizadas en el código fuente de los códecs, ya que tras comprobar que en el teléfono Android los programas funcionan correctamente, carece de sentido realizar desde el principio todas las modificaciones.

2.1.3 Compilación, ejecución y toma de medidas

Para ejecutar los programas que implementan los códecs es necesario disponer de ficheros de audio que servirán como entrada. Para el desarrollo del proyecto se tomará un único fichero de tres minutos en formato RAW. Estos ficheros se han obtenido a partir de un banco de conversaciones procedentes del Linguistic Data Consortium [24], más concretamente de las 120 conversaciones incluídas, se ha creado un único fichero de 3 minutos de duración para la toma de medidas.

Una vez se han realizado los cambios oportunos en el código fuente de los códecs es necesario compilarlos para que se puedan ejecutar. En primer lugar se han compilado para ser probados en el ordenador, con el fin de comprobar de que el comportamiento de los programas es el adecuado.

Se han modificado los ficheros `makefile`, sustituyendo el compilador `gcc` por una variable que, con ayuda de un script ejecutado con anterioridad, podrá tomar los valores `gcc` o `arm-linux-androideabi-gcc`, el compilador cruzado incluído con el NDK de Android, preparado para compilar los programas para la arquitectura de Android, que es ARM.

A partir de los `makefile` modificados se obtienen los binarios que implementan los códecs.

A continuación se expondrá la manera de ejecutar los programas que implementan cada códec, donde `exec` se referirá al ejecutable del programa, `fichentrada` al archivo de audio a ser procesado y `fichsalida` al archivo de salida codificado o decodificado:

- Para el **G.711** la ejecución es:

```
$exec Law Transf fichentrada fichsalida
```

donde `Law` es la ley de compresión (A o μ) y `Transf` debe ser *lilo* para comprimir y *loli* para descomprimir.

- Para el **G.723.1** la ejecución es:

```
$exec [options] fichentrada fichsalida
```

donde `[options]` son las distintas opciones posibles: *-c* para codificar, *-d* para decodificar, *-r63* o *-r53* según cual sea la tasa de bits elegida y *-v* si se elige compresión con detección de la actividad vocal.

- Para el **G.729** y sus variantes **G.729B** (con esquema de compresión de silencios), **G.729A** (anexo de complejidad reducida), y **G.729AB** (anexo de complejidad reducida con esquema de compresión de silencios), la ejecución es:

```
$encoder_exec fichentrada fichsalida
```

para codificar. Y para decodificar:

```
$decoder_exec fichentrada fichsalida
```

- Para el **AMR** la ejecución es:

```
$encoder_exec [-dtx] mode fichentrada fichsalida
```

para codificar. Para decodificar la ejecución es de la siguiente manera:

```
$decoder_exec [-dtx] mode fichentrada fichsalida
```

donde la opción `[-dtx]` sirve para usar esquema de detección de actividad vocal y `mode` indica el modo utilizado, que en el proyecto serán *MR122*, *MR74* o *MR475* para tasas de 12,2, 7,4 y 4,75 kbps respectivamente.

- Para el códec **iLBC** la ejecución es:

```
$exec mode fichentrada encoded decoded
```

donde `mode` puede tomar los valores *20* o *30* según el tamaño de trama elegido en ms, `encoded` el archivo de salida codificado y `decoded` el archivo decodificado obtenido a partir de `encoded`.

Una vez se ha comprobado que el funcionamiento de los programas es el deseado, se procede a compilarlos y ejecutarlos en el smartphone. La ejecución se realizará de forma automatizada a partir de un script, cuyo contenido se detallará en el siguiente apartado.

Sin embargo antes de la toma de medidas, es necesario realizar una serie de operaciones en el dispositivo, de forma que los valores obtenidos tras la medida sean lo más reales posible:

- Por defecto la CPU trabaja bajo el perfil *ondemand*, que consiste en que va variando su frecuencia según la exigencia de los procesos que están ejecutándose. Esto haría que las medidas tomadas no fueran del todo fieles a la realidad. Por ello es necesario, obligar a la CPU a que trabaje al máximo, a través del perfil *performance*. Se realizará mediante un script, cuyo contenido se explicará en el siguiente apartado.

- Con el fin de evitar que se ejecuten procesos innecesarios durante la toma de medidas y como consecuencia el gasto energético no sea debido a lo que se desea medir, es necesario que el smartphone permanezca en modo avión, esto es, con todas las interfaces inalámbricas desconectadas. Asimismo, se matarán todos los procesos del sistema que sean innecesarios.

- Las baterías de medidas deben ser ejecutadas mediante la línea de comandos. Para ello se han contemplado dos opciones posibles:

  - Utilizar una aplicación para Android cuya función sea emular una consola de comandos. En este caso los programas que implementan los códecs se ejecutarían sobre otra aplicación, que además corre sobre una máquina virtual. Esto supondría la existencia de un procesado extra además del de los códecs, por lo que se falsearían las medidas de consumo.
  - Utilizar el ADB (Android Debug Bridge). Con esta herramienta es posible acceder a una consola de comandos del dispositivo desde el ordenador, además la ejecución de los programas de medida sería mucho más directa que en la opción anterior, por lo que se ha escogido esta alternativa.

- La CPU debe procesar los programas de codificación y decodificación con la máxima prioridad posible. Con ayuda del comando de linux *nice* se logrará dar la máxima prioridad a la ejecución de estos programas.

Ya se ha explicado que la ejecución se realizará a través de scripts, cuyo funcionamiento se expondrá más detenidamente en el siguiente apartado. De la misma forma, en el apéndice correspondiente se mostrarán las instrucciones para ejecutar las pruebas. A continuación se procederá a la toma de medidas.

Tras conectar el dispositivo al ordenador a través del cable USB, el siguiente paso es copiar los programas previamente compilados y los scripts al dispositivo, tras lo cual habrá que poner el teléfono en modo avión, cerrar todos los procesos innecesarios y ejecutar el script para que la CPU trabaje a máxima frecuencia. Por último, se ejecuta el script que arranca toda la batería de medidas. En este script, a su vez, se ejecutarán dos más: uno para obtener los ficheros `ftype` y otro que se encarga de la toma de medidas de consumo.

Una vez haya finalizado, para su posterior procesado, se han copiado en el ordenador los ficheros de datos que se han creado. Estos serán de dos tipos: por un lado los `ftype`, cuyo contenido ya se ha explicado con anterioridad y por otro, los ficheros generados durante el proceso de medida propiamente dicho.

Por cada trama procesada se va imprimiendo en estos ficheros, uno por códec y variante, la suma de los dos parámetros que son de interés de todos los datos que genera la función `getrusage`, tal y como se expuso en el apartado anterior.

El siguiente paso será por tanto tomar el contenido de estos ficheros, manipularlos y representar gráficamente los resultados obtenidos.

2.1.4 Creación de scripts

En esta sección se va a proceder a explicar el cometido de los diversos scripts que se han programado durante el desarrollo del proyecto. En su mayoría se han tomado del estudio del consumo que se realizó en laptops y sobre ellos se han realizado las modificaciones necesarias.

Scripts para la ejecución de códecs y toma de medidas

Estos son los scripts que se han copiado en el teléfono. Se exponen a continuación:

- `maxcpu.sh`: se encarga de que la CPU trabaje con el perfil *performance*, para ello imprime esta misma palabra en el fichero `scaling_governor`.
- `start.sh`: se encarga de arrancar el proceso de ejecución. Llama a su vez a `get_ftype.sh` y `bucleN_get_consumo_encdec.sh`.
- `get_ftype.sh`: su función es la de ir ejecutando los programas que han sido modificados para obtener los ficheros `ftype`.
- `bucleN_get_consumo_encdec.sh`: consiste en dos bucles que que se ejecutan tantas veces como el parámetro que es es necesario pasarle. Un bucle es para el proceso de codificación en el que se llama a `get_consumo_encoding.sh` y el otro para el proceso de decodificación, en cuyo caso se llama al script `get_consumo_decoding.sh`. En cada iteración del bucle y tras ejecutarse el script correspondiente se renombran los ficheros generados para distinguirlos entre una iteración y otra después al ser manipulados.
- `get_consumo_encoding.sh`: su función es ir ejecutando los programas para tomar medidas en el proceso de codificación.
- `get_consumo_decoding.sh`: su función es ir ejecutando los programas para tomar medidas en el proceso de decodificación.

Scripts para la manipulación de datos obtenidos

Una vez haya finalizado todo el proceso de toma de medida, los ficheros con los datos recogidos se han copiado en el ordenador para su posterior procesado. Los scripts encargados de ello son los siguientes:

- `mediasvar.sh`: este script se encarga de generar tanto para codificación como para decodificación sendos ficheros. El contenido de los mismos consite en el nombre de cada códec junto con los tiempos medios de procesado e intervalos de confianza del 95 % por cada tipo de trama así como del total. Estos ficheros se obtienen a partir de los que recogen las medidas

temporales y de los `ftype`. En este script se utilizarán también un filtro `awk` y un ejecutable `.exe` para calcular las medias y los intervalos de confianza respectivamente.

- `grafvar.sh`: este script se encarga de ejecutar `mediasvar.sh`, y con los ficheros generados por el mismo, unifica todos los datos en un único archivo. A partir de éste, se representa gráficamente el tiempo normalizado de codificación y decodificación, esto es el tiempo transcurrido en el proceso dividido por el tiempo de conversación (180 segundos).

2.1.5 Obtención de curva de descarga de la batería

El último paso tras obtener las medidas temporales es calcular el consumo energético a partir de esos tiempos. La forma que se ha escogido para hallar estos datos ha sido similar a la metodología seguida a partir del modelo propuesto en el estudio del consumo para laptops. Se basa en que la energía consumida durante la descarga de la batería es lineal con el tiempo.

El objetivo, por tanto, es hallar la constante de descarga con el fin de calcular la energía consumida a partir de ella y de las medidas temporales recogidas con anterioridad.

La forma de calcular la constante ha consistido en ir recogiendo datos de la batería mientras se descarga en dos situaciones:

- Descarga de la batería durante la codificación. Por simplicidad se utilizará el códec G.723.1 a una tasa de 6,3 kbps, ya que se trata del códec en cuyo procesado la CPU necesita más tiempo. De la misma forma, se utilizará un fichero de audio de mayor duración que en la toma de medidas, concretamente de 30 minutos. Se utilizará el script `bateriacon.sh`.
- Descarga de la batería con el teléfono ocioso, mediante la ejecución del script `bateriasin.sh`.

El valor buscado de la constante de descarga de la batería será la resta entre las dos constantes correspondiente a las dos situaciones anteriores, función que realiza el script `cte.sh`.

Al igual que en el caso de la toma de medidas temporales, el dispositivo debe permanecer en modo avión y con la CPU trabajando al máximo rendimiento. Además, si al igual que en el caso de las medidas temporales se ejecutan los programas a través del ADB, al estar el teléfono conectado al PC a través del cable USB, también estaría cargando la batería, por lo que no es posible la toma de datos de la descarga en este caso.

Este problema se intentó solucionar utilizando un cable alargador USB en el que se cortó la alimentación. Sin embargo el dispositivo, al no existir alimentación, no detecta la conexión por lo que es imposible la comunicación de este modo. Finalemente, para solventar esto, se ha utilizado para la ejecución del programa una aplicación que emula una consola de comandos.

Los dispositivos portátiles en general proporcionan datos, como son la carga [Ah] y el voltaje [V], que permiten obtener valores de energía. En este caso, multiplicando ambas magnitudes.

Sin embargo existe un problema con el dispositivo bajo el que se están tomando las medidas y es que no es posible obtener directamente los valores de carga, ya que el kernel no lo permite. Tan sólo es posible obtener el voltaje y el porcentaje de carga de batería restante, en unidades enteras. Con este valor y sabiendo que la carga máxima de la batería del smartphonees de 1650 mAh, ha sido posible estimar los valores de carga en cada instante.

Sabiendo esto, el último paso es tomar medidas del estado de la batería del dispositivo mediante un script. Los valores se tomarán del fichero `uevent`, dentro de la ruta `/sys/class/power_supply/battery`. El contenido del fichero `uevent` se muestra a continuación:

```
POWER_SUPPLY_NAME=battery
POWER_SUPPLY_STATUS=Discharging
POWER_SUPPLY_HEALTH=Good
POWER_SUPPLY_PRESENT=1
POWER_SUPPLY_TEMP=310
POWER_SUPPLY_ONLINE=0
POWER_SUPPLY_VOLTAGE_NOW=3948750
POWER_SUPPLY_CAPACITY=84
POWER_SUPPLY_TECHNOLOGY=Li-ion
```

El fichero `uevent` proporciona toda la información necesaria para el cálculo de la pendiente de la recta de descarga de la batería. De todos los elementos mostrados anteriormente interesan `POWER_SUPPLY_VOLTAGE_NOW`, dado en microvoltios y `POWER_SUPPLY_CAPACITY`, dado en %.

Así, se va tomando el contenido del fichero `uevent` cada 3 minutos y finalmente se calcula la pendiente de la recta, estando el eje de abcisas en horas y el de ordenadas en Wh, resultado de multiplicar el porcentaje de la capacidad, la carga máxima de la batería y el voltaje. Como resultado, la unidad de la pendiente, será el Watio (J/s).

El valor que resulta de la resta entre las dos pendientes se multiplica por los tiempos obtenidos en las medidas de los códecs, y así es posible hallar energía consumida (J) por cada códec.

2.2 Análisis del consumo de la tarjeta WiFi

La metodología seguida para realizar el análisis del consumo de la interfaz WiFi consiste, al igual que en el análisis del consumo de codificación-decodificación, en realizar medidas temporales del proceso de envío de paquetes de voz ya codificados, siendo distinta de la metodología seguida en el artículo [11]. De la misma forma que en el análisis anterior se obtendrá la curva de descarga de la batería, con el fin de hallar valores de energía consumida a partir de las medidas temporales.

2.2.1 Análisis de la solución adoptada

Con el objetivo de tomar las medidas del consumo de la interfaz WiFi, se ha adoptado una solución basada en un modelo cliente-servidor. Se crearán dos programas en C, un cliente y un servidor, cuyo funcionamiento se detallará más adelante. Las razones por las que se ha escogido el C como lenguaje de programación son las mismas que las que se han justificado en la medida de consumo de los códecs.

En este caso, a diferencia del caso del consumo de los códecs, las medidas temporales no se tomarán haciendo uso de la función `getrusage`, ya que esta función solamente da información de la CPU y de la memoria. Al resultar de interés la tarjeta WiFi el uso de la función `getrusage` aquí carece de sentido, por lo que es necesaria otra función que tome medidas temporales.

La función que se encargará de tomar las medidas será `gettimeofday`. Cada vez que esta función es llamada, almacena en una estructura que se le pasa como parámetro la hora con una precisión de microsegundos. Se llamará a la función en el inicio del proceso y cuando este finalice. La diferencia entre estos valores será el tiempo transcurrido en el proceso de envío de los paquetes, que se corresponderá con el tiempo en que la tarjeta WiFi está en funcionamiento.

La idea general de la solución consite en que el programa cliente vaya enviando tramas al programa servidor según el códec que corresponda, a partir del fichero `numtramas.dat`, creado a partir de un script que recopila los parámetros de los códecs y a partir del cual se obtendrá el tamaño del paquete a enviar. En caso de que el códec utilice detección de la actividad vocal, se irá leyendo su fichero `ftype`, con el objeto de decidir el tipo y tamaño de la trama que debe ser enviada al programa servidor.

2.2.2 Creación de programas cliente y servidor

En esta sección se va a proceder a explicar el funcionamiento de los programas cliente y servidor, a partir de los cuales se obtendrán las medidas temporales. Se han probado varias metodologías diferentes para medir el tiempo de envío de paquetes. Se expondrá en primer lugar la que se utilizó finalmente. En segundo lugar se desarrollarán las metodologías restantes, así como las razones por las cuales se ha decidido su descarte.

En todas las propuestas se han utilizado sockets UDP, con el fin de que el escenario sea lo más parecido a una comunicación VoIP real. La explicación del funcionamiento de los programas se centrará en lo relativo a la medida del consumo de la tarjeta WiFi, obviando todo aquello necesario realizar en programas de este tipo, como puede ser por ejemplo la creación de sockets.

Programa Servidor

El programa servidor se encargará de ir recibiendo los paquetes desde el programa cliente. Para su ejecución será necesario especificar el puerto que debe

utilizar para establecer la comunicación con el cliente.

El servidor será el encargado de medir el tiempo que emplea el cliente en el envío de las tramas codificadas. Podrá recibir tres tipos de paquetes: de inicio, de fin y de códec. El sistema de medida del tiempo consiste en tomar una marca de tiempo cuando reciba un paquete de inicio y otra cuando reciba el paquete de fin, y restarlas. Irá guardando los resultados en ficheros distintos para cada códec. Para distinguir el fichero, recibirá también el nombre del códec por parte del cliente junto al paquete de inicio. Los paquetes correspondientes a tramas codificadas, se descartarán simplemente.

Cuando transcurra un tiempo determinado sin que reciba ningún paquete terminará el programa.

Programa Cliente

El programa cliente es el encargado de enviar los paquetes al servidor. Para su ejecución habrá que pasarle como argumentos la dirección IP del servidor, el puerto, el nombre del códec, el número de tramas de voz, el tamaño de las tramas de voz y el de las de silencio y por último el intervalo de tiempo entre tramas.

En primer lugar el programa toma de los argumentos los valores correspondientes a los tamaños de las tramas de voz y silencio, dados en bits, los pasa a bytes y le suma el valor de la cabecera RTP, con el fin de que el modelo sea lo más parecido a la realidad.

Seguidamente se distinguen dos opciones, que el códec utilice detección de actividad vocal o no. El programa intentará abrir el fichero `ftype`, si tiene éxito es que usa detección de actividad vocal, en el caso contrario se tratará de un códec que no hace uso de ella.

En cualquier caso se envía un paquete de inicio junto con el nombre del códec que el servidor reconocerá para comenzar a medir el tiempo transcurido, y tras el envío de todos los paquetes de la conversación codificada, se transmitirá otro paquete de fin para que el servidor detenga la cuenta.

Si el códec no utiliza detección de actividad vocal, se irán enviando tantos paquetes de voz como marque el parámetro que se pasó como argumento. Entre paquete y paquete el programa se detendrá el periodo entre tramas que se le pase como argumento, con el objetivo de ofrecer la mayor realidad posible al modelo, ya que los paquetes no se transmiten todos seguidos, sino que existe un periodo de tiempo de generación de la trama.

En cambio, si el códec utiliza el esquema de detección de actividad vocal la forma de saber si el paquete es de voz o silencio o no hay que transmitir nada, es leyendo el fichero `ftype`. Si el valor leído es un 1 se enviará una trama de voz, si es un 2, se enviará una trama de silencio y si es un 0 no se enviará nada. Tras enviar el tipo de paquete correspondiente se detendrá el programa al igual que se expuso anteriormente mediante la función `usleep`, a la cual se le pasa como

argumento los microsegundos que es preciso detener el programa.

Otras metodologías

En esta sección se expondrán otras metodologías que se probaron a la hora de abordar la creación de los programas servidor y cliente, y que fueron descartadas por distintas razones.

Una de ellas fue similar a la utilizada finalmente. La base era la misma, con la diferencia de que en este caso la medida la realizaba el cliente, por lo que el servidor tenía que volver a responder al cliente tras recibir los paquetes de inicio y fin para que éste llamara a la función `gettimeofday`. Este método fue descartado por el innecesario envío de paquetes a la hora de tomar las medidas.

El primer método utilizado y con el que se llegó a tomar medidas, no estaba basado en enviar paquetes a modo de bandera entre el cliente y el servidor. En este caso, el servidor únicamente recibía los paquetes con las tramas codificadas y todo el trabajo de medida recaía en el programa cliente.

La toma de medidas se basa en un sistema a dos pasadas. Se mide el tiempo de ejecución del programa enviando los paquetes y sin enviarlos, y por último se restan estos tiempos. Así se obtendría el tiempo de uso de la tarjeta WiFi. La forma de transmitir las tramas codificadas es la misma que en la solución elegida, mediante un bucle en el que se va decidiendo qué tipo de trama hay que transmitir. Las llamadas a la función `gettimeofday` se realiza justo antes de entrar en el bucle y nada más salir del mismo.

Esta solución fue descartada a causa de que las medidas obtenidas no eran congruentes con el resultado esperado, hasta el punto de que tras realizar el cálculo de la medida del tiempo de transmisión de las tramas, restando los valores de las dos pasadas, en algunos casos, resultaban valores negativos. Este hecho se debe a que cuando se realiza la segunda llamada a la función `gettimeofday` todavía no se habían transmitido la totalidad de las tramas, por lo que el tiempo medido resultaba ser menor que el tiempo real que estaba la tarjeta WiFi en uso.

2.2.3 Creación de scripts y ejecución

Con el fin de tomar las medidas temporales oportunas y establecer la comunicación entre los programas cliente y servidor, éstos se ejecutarán en el teléfono y el ordenador portátil respectivamente.

Las medidas han sido tomadas bajo las siguientes condiciones:

- La CPU de ambos dispositivos, smartphone y laptop, trabaje a máxima frecuencia, por las razones expuestas en la sección relativa al consumo de CPU.
- Los experimentos se realizarán bajo un entorno limpio. Los dispositivos serán los únicos conectados en una red local y ésta se creará en un área y canal más o menos libres, es decir, con un bajo número de otras redes inalámbricas.

- Los intervalos de tiempo entre tramas es distinto entre unos códecs y otros, por lo que también será distinto el número de tramas que deberían transmitirse. Por ello, para igualar este número y no falsear medidas en este sentido, se han modificado los archivos `ftype` de los códecs involucrados y el archivo en el que se recogen los parámetros de los mismos. Se igualarán todos los intervalos a 30 ms.

- Debido al excesivo tiempo que lleva realizar las medidas, se ha codificado un fragmento de conversación de 30 segundos, con un factor de actividad de aproximadamente del 50 %.

El contenido del fichero `numtramas.dat` modificado se muestra a continuación:

```
g.711            1000    0       0       256     0       30
g.723_1_53dtx    542     39      419     160     32      30
g.723_1_63dtx    542     39      419     192     32      30
g.729ab          490     112     398     80      10      30
g.729b           490     112     398     80      10      30
amr475dtx        500     78      422     95      39      30
amr74dtx         500     78      422     148     39      30
amr122dtx        500     78      422     244     39      30
g.729a           1000    0       0       80      0       30
g.729            1000    0       0       80      0       30
iLBC15.2         1000    0       0       303     0       30
iLBC13.3         1000    0       0       399     0       30
amr4.75          1000    0       0       95      0       30
amr7.4           1000    0       0       148     0       30
amr12.2          1000    0       0       244     0       30
g.723_1_53       1000    0       0       160     0       30
g.723_1_63       1000    0       0       192     0       30
```

Los datos recogidos en el fichero son por orden de izquierda a derecha: Nombre de códec, número de tramas de voz, silencio y de no transmisión, tamaño de trama de voz y silencio en bits y periodo entre tramas.

A continuación se detallarán el cometido de los scripts y filtros `awk` creados para la toma de medidas del consumo de la tarjeta WiFi y su posterior manipulación y representación gráfica.

- `amr.awk`: Este filtro `awk` sirve para cambiar el número que corresponde a cada tipo de trama del códec AMR, para que tengan el mismo valor que los demás, ya que como se explicó en el apartado del consumo de CPU, el programa que implementa el códec AMR etiquetaba de distinta forma las tramas.

- `maxcpu.sh`: Es el mismo script utilizado en la sección del consumo de la CPU.

- `run.sh`: Se encarga de realizar el número de ejecuciones que se le pase como parámetro mediante un bucle.

- `grafwifi.sh`: Se encarga de tomar los valores recogidos en los ficheros que genera el programa servidor, les calcula la media y el intervalo de confianza

y los representa gráficamente. En este caso, al igual que con el consumo de CPU, se representa el tiempo normalizado, es decir el tiempo de transmisión empleado dividido por el tiempo de conversación.

Por último, para calcular el consumo energético a partir de las medidas temporales, se ha procedido de la misma forma que para el cálculo del consumo de la CPU. Para hallar el valor de la pendiente de descarga se ha realizado transmitiendo paquetes con las tramas del códec G.723.1 a 6,3 kbps.

Capítulo 3

Resultados

En este capítulo se presentan los resultados obtenidos durante la realización del proyecto. En primer lugar se mostrarán y discutirán las gráficas para el consumo de CPU, estableciendo una comparativa con el consumo en laptops y en segundo lugar se hará lo propio con el consumo de la interfaz WiFi.

3.1 Consumo CPU

Como ya se ha explicado anteriormente la CPU es la encargada de las tareas de codificación y decodificación. Los resultados mostrados se han obtenido tras realizar una batería de 5 medidas con un audio de entrada correspondiente a una conversación de tres minutos.

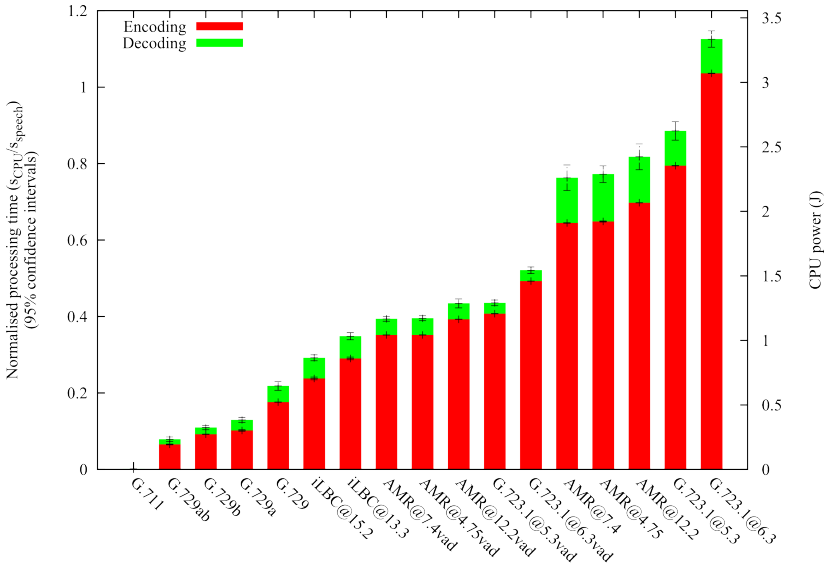


Figura 3.1: Consumo de CPU en smartphone

En la Figura 3.1 se muestra una gráfica con el tiempo de procesamiento de CPU que requiere cada códec para las labores de codificación y decodificación. Este tiempo se da normalizado, es decir, el tiempo empleado en el proceso dividido por el tiempo que dura la conversación. De la misma forma, se puede observar

en la gráfica la energía consumida durante el proceso, obtenida a partir del valor de la constante de descarga de la batería del dispositivo, que en la medida ha resultado ser de 2.963 J/s.

El primer hecho significativo que se puede observar en la misma es que el consumo en el proceso de decodificación es muy inferior al de codificación. Por otro lado, la gráfica muestra que existen códecs que no son aptos para la codificación en tiempo real en el dispositivo, como es el caso del G.723.1 a 6.3 kbps, ya que su tiempo normalizado de codificación es superior a 1, es decir, el dispositivo necesita más tiempo del que dura la conversación para codificarla con este códec.

Según la figura los códecs que demandan más tiempo de CPU y que por lo tanto consumen más son el G.723.1 y el AMR. Por el contrario los códecs que permiten ahorrar más batería son el G.711, los G.729 (como era esperado el G.729a necesita menos procesamiento de CPU, ya que se trata de una versión simplificada) y el iLBC.

Además, tal y como se esperaba, el uso del algoritmo de detección de actividad vocal reduce el consumo energético.

La Figura 3.2 muestra el tiempo medio que emplea la CPU en codificar y decodificar una trama, ya sea de voz o también de silencio en el caso de los códecs que hagan uso de un esquema de detección de actividad vocal. El resultado que se puede observar en la gráfica concuerda con lo esperado: las tramas de SID generadas por el algoritmo DTX que implementan algunos códecs requieren menos tiempo de procesamiento que las tramas de voz.

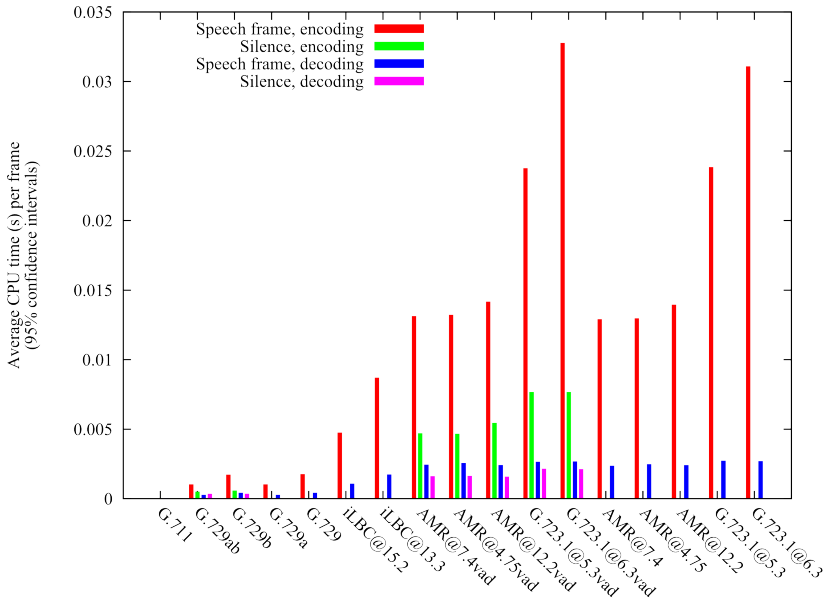


Figura 3.2: Tiempo de CPU por códec y tipo de trama

En la Figura 3.3 se muestra el consumo de CPU de cada códec, correspondiente al estudio que se ha realizado anteriormente para laptops. El objetivo de esta gráfica es establecer una comparativa con el resultado obtenido en el smartphone.

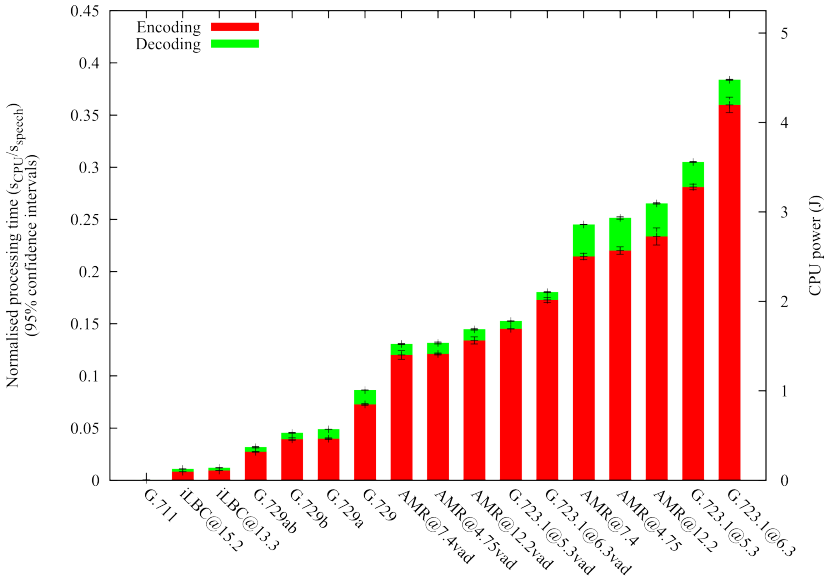


Figura 3.3: Consumo de CPU en laptop

Comparando las dos gráficas se pueden observar varias diferencias significativas.

La primera es relativa al tiempo de procesamiento y el consumo energético. La CPU del smartphone necesita más tiempo que el ordenador portátil para codificar y decodificar, lo cual es lógico, ya que la CPU del smartphone está enfocada hacia un bajo consumo con el fin de priorizar la duración de la batería. Este hecho también se puede observar en los datos de energía consumida en el proceso, evidentemente menor en el smartphone.

En las gráficas los códecs aparecen ordenados según el tiempo total empleado en el proceso de codificación/decodificación. El orden de los códecs coincide entre ambas a grandes rasgos, excepto el códec iLBC, siendo su consumo en el smartphone proporcionalmente muy superior al consumo que presenta en el laptop.

Debido a esta anomalía, se han repetido las medidas en el smartphone exclusivamente con el códec iLBC, modificando de forma alternativa su código. Tras no observar diferencias, se ha llegado a la conclusión de que el códec iLBC presenta en el smartphone un consumo sensiblemente superior que en el laptop debido a que no se encuentra optimizado para la plataforma o arquitectura del dispositivo.

3.2 Consumo Tarjeta WiFi

Los resultados para el consumo de la interfaz WiFi que se exponen en este capítulo han sido hallados tras una batería de 6 ejecuciones tomando como referencia una conversación de 30 segundos. La razón por la que se ha utilizado una conversación con menor duración que en el análisis del consumo de CPU se debe a que así se pueden tomar más medidas en menos tiempo, ya que la duración de

una ejecución es muy elevada. El estudio se ha realizado sobre un entorno limpio y la red inalámbrica es de tipo N.

En la Figura 3.4 se muestra la gráfica con los resultados de la toma de medidas de consumo. En ella se puede apreciar cómo los códecs que utilizan detección de actividad vocal consumen menos que los que no lo hacen, lo cual es lógico ya que se transmiten menos tramas.

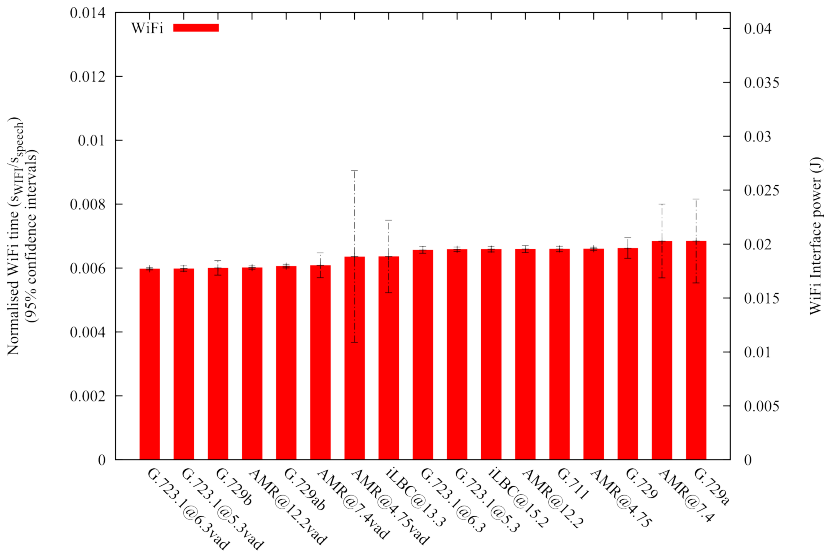


Figura 3.4: Consumo de interfaz WiFi

Por otro lado, la transmisión es casi instantánea, debido al escaso tráfico producido, el entorno limpio y al alto régimen binario que permite la red inalámbrica.

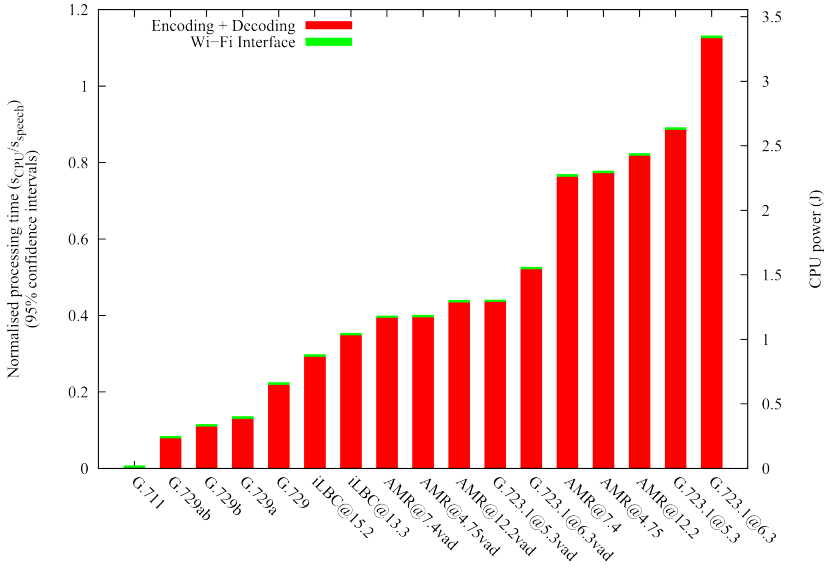


Figura 3.5: Consumo combinado CPU y WiFi

En conclusión, se puede afirmar que el consumo de la interfaz WiFi es despre-

ciable comparándolo con el de CPU, tal y como se puede observar en la Figura 3.5, en la que aparcen los consumos de CPU y de interfaz WiFi combinados.

Capítulo 4

Conclusiones

En este proyecto se ha realizado un análisis del rendimiento energético de los distintos códecs que se utilizan actualmente en aplicaciones de VoIP en un dispositivo con sistema operativo Android. Para ello se ha seguido una metodología sencilla ya diseñada en un estudio similar realizado en laptops, y cuya finalidad es tomar medidas en los dos elementos que presentan un consumo más significativo, CPU e interfaz WiFi.

Los resultados en el análisis de la CPU muestran que no todos los códecs son válidos para las tareas de codificación y que los que menos consumen son G.711, G.729 e iLBC, mientras que los que más lo hacen son AMR y G.723.1. De este modo es posible escoger un códec u otro teniendo en cuenta no sólo la calidad que ofrece sino también el ahorro energético.

En cuanto al análisis del consumo de la interfaz WiFi, los resultados muestran por un lado que los códecs que aportan detección de la actividad vocal consumen ligeramente menos energía que los que no detectan silencios en la conversación. Por otro lado, el consumo de la interfaz WiFi puede considerarse despreciable frente al de CPU.

Finalmente, como posibles líneas de continuación, se proponen las siguientes:

- Extender el estudio a las siguientes versiones de Android, con el fin de comprobar si el consumo cambia de forma significativa al hacerlo de versión. También sería deseable extender el estudio a los dispositivos que actualmente están más en auge, como las tablets.
- Incluir en el estudio factores que afecten a la calidad de servicio, como la pérdida de paquetes, número de tramas por paquete, etc.
- Realizar el estudio del consumo de la interfaz WiFi en un entorno sucio, con más dispositivos interactuando en la red y con tráfico de fondo, con el fin de analizar el impacto que supondrían las posibles colisiones generadas en este escenario.

Bibliografía

[1] J.A.C. Falcón: *VoIP: La telefonía de Internet.* Thomson, 2007, ISBN 9788428329521.

[2] D Collins: *Carrier Grade Voice Over Ip.* McGraw-Hill Education (India) Pvt Limited, 2005, ISBN 9780070603233.

[3] Trevor Pering, Yuvraj Agarwal, Rajesh Gupta, and Roy Want: *Coolspots: reducing the power consumption of wireless mobile devices with multiple radio interfaces.* In *Proceedings of the 4th international conference on Mobile systems, applications and services*, MobiSys '06, pages 220–232, New York, NY, USA, 2006. ACM, ISBN 1-59593-195-3.

[4] Vijay Raghunathan, Trevor Pering, Roy Want, Alex Nguyen, and Peter Jensen: *Experience with a low power wireless mobile computing platform.* In *Proceedings of the 2004 international symposium on Low power electronics and design*, ISLPED '04, pages 363–368, New York, NY, USA, 2004. ACM, ISBN 1-58113-929-2.

[5] *Worldwide quarterly mobile phone tracker.* Technical report, IDC, August 2012. `http://www.idc.com/getdoc.jsp?containerId=prUS23638712`.

[6] *U.s. mobile subscriber market share.* Technical report, comScore Inc., June 2012. `http://www.comscore.com/esl/Press_Events/Press_Releases/2012/8/comScore_Reports_June_2012_U.S._Mobile_Subscriber_Market_Share`.

[7] *Smartphone operating system share.* Technical report, Nielsen, july 2012. `http://blog.nielsen.com/nielsenwire/?p=32494`.

[8] *Worldpanel comtech.* Technical report, Kantar inc., June 2012. `http://www.kantarworldpanel.com/dwl.php?sn=publications&id=45`.

[9] `http://www.android.com/`.

[10] `http://developer.android.com/intl/es/about/dashboards/index.html`.

[11] Antonio J. Estepa, Juan M. Vozmediano, Jorge López, and Rafael M. Estepa: *Impact of voip codecs on the energy consumption of portable devices.* In *Proceedings of the 6th ACM workshop on Performance monitoring and measurement of heterogeneous wireless and wired networks*, PM2HW2N '11, pages 123–130, New York, NY, USA, 2011. ACM, ISBN 978-1-4503-0902-8.

[12] ITU-T: *Pulse code modulation (pcm) of voice frequencies,*. In *Recommendation G.711*, November 1988. http://www.itu.int/rec/T-REC-G.711-198811-I/en.

[13] ITU-T: *Coding of speech at 8 kbit/s using conjugate-structure algebraic-code-excited linear prediction (cs-acelp).* In *Recommendation G.729*, January 2007. http://www.itu.int/rec/T-REC-G.729-200701-I/en.

[14] ITU-T: *Dual rate speech coder for multimedia communications transmitting at 5.3 and 6.3 kbit/s.* In *Recommendation G.723.1*, May 2006. http://www.itu.int/rec/T-REC-G.723.1-200605-I/en.

[15] S. Andersen, A. Duric, H. Astrom, R. Hagen, W. Kleijn, and J. Linden: *Internet low bit rate codec (ilbc).* In *IETF RFC3951*, December 2004. http://www.ietf.org/rfc/rfc3951.txt.

[16] http://www.ilbcfreeware.org/.

[17] ETSI: *Adaptative multi-rate speech codec.* In *Recommendation TS 126 073*, December 2008. http://www.3gpp.org/ftp/Specs/html-info/26073.htm.

[18] http://www.xcubelabs.com/the-android-story.php.

[19] http://www.gnu.org/software/libc/manual/.

[20] Ramkumar Jayaseelan, Tulika Mitra, and Xianfeng Li: *Estimating the worst-case energy consumption of embedded software.* In *Proceedings of the 12th IEEE Real-Time and Embedded Technology and Applications Symposium*, RTAS '06, pages 81–90, Washington, DC, USA, 2006. IEEE Computer Society, ISBN 0-7695-2516-4. http://dx.doi.org/10.1109/RTAS.2006.17.

[21] Aqeel Mahesri and Vibhore Vardhan: *Power consumption breakdown on a modern laptop.* In *Proceedings of the 4th international conference on Power-Aware Computer Systems*, PACS'04, pages 165–180, Berlin, Heidelberg, 2005. Springer-Verlag, ISBN 3-540-29790-1, 978-3-540-29790-1. http://dx.doi.org/10.1007/11574859_12.

[22] http://developer.android.com/intl/es/tools/help/adb.html.

[23] http://developer.android.com/intl/es/tools/sdk/ndk/index.html.

[24] A. Canavan, D. Graff, and G. Zipperlen: *Callhome american english speech.* In *LDC97S42*, Linguistic Data Consortium, 1997.

Apéndices

Apéndice A

Manual de usuario

En este apéndice se explica como instalar todo lo necesario para la realización del proyecto, así como la compilación de los diferentes programas y su ejecución.

A.1 Instalación

En esta sección se detallará todo lo relacionado con la instalación de los programas necesarios para el desarrollo del proyecto.

Lo primero de todo es proceder a la instalación del SDK, a partir del cual se podrá utilizar el ADB, y el NDK, con el fin de compilar los programas para el dispositivo Android.

La descarga del Android SDK se puede hacer desde la web:
http://developer.android.com/intl/es/sdk/index.html.
De esta web se descargará un archivo en formato `tgz` que tan sólo habrá que descomprimir en el directorio deseado.

Una vez ha quedado instalado el SDK, es necesario hacer lo propio con el NDK. Como ya se ha explicado en capítulos anteriores, el NDK permitirá compilar el código fuente para que pueda ser ejecutado en el dispositivo Android. El NDK se descargará de:
http://developer.android.com/intl/es/tools/sdk/ndk/index.html.
De la misma forma que con el SDK se descargará un archivo en formato `tgz`, que habrá que descomprimir también en el directorio que se desee.

Tras descomprimir los ficheros, es conveniente añadir los directorios al `PATH`, con el fin de tener la posibilidad de acceder a los programas de forma sencilla desde cualquier ruta.

A.2 Estructura de directorios y listado de ficheros

En esta sección se desarrollará, a modo de resumen, la estructura de los directorios y los ficheros contenidos en cada uno, con el fin de que durante la ejecución de los scripts no se produzcan errores. Se ha creado la misma estructura en el laptop y en el dispositivo, por sencillez. Sin embargo habrá ficheros que se ejecutarán

únicamente en el smartphone o en el laptop.

1. Consumo interfaz WiFi.

   Todos los ficheros se encuentran dentro del directorio `wifi`. A continuación se enumeran los ficheros utilizados:

   - Los ficheros `ftype` modificados de los códecs.
   - `numtramas.dat`: es el fichero que contiene la información de todos los códecs. También se ha modificado para intentar igualar lo más posible el intervalo entre tramas.
   - `cliente`
   - `servidor`
   - `grafwifi.sh`
   - `run.sh`
   - `compila.sh`
   - `maxcpu.sh`
   - `meanvar.awk`
   - `tstudent.exe`

2. Consumo de CPU.

   - `androscripts`: en esta carpeta estarán presentes los scripts para llevar a cabo las medidas de consumo de CPU. Los ficheros contenidos en ella son los siguientes:
     - `maxcpu.sh`
     - `start.sh`
     - `get_ftype.sh`
     - `bucleN_get_consumo_encdec.sh`
     - `get_consumo_encoding.sh`
     - `get_consumo_decoding.sh`
     - `mediasvar.sh`
     - `bateriacon.sh`
     - `bateriasin.sh`
     - `cte.sh`
     - `grafvar.sh`
     - `meanvar.awk`
     - `tstudent.exe`
     - `compila.sh`
   - `execs`: es este directorio se encuentran los ejecutables de los programas que implementan los códecs.
     - `cons`: en esta carpeta se han copiado los ejecutables de los códecs para medir el consumo.
     - `ftype`: en esta carpeta se han copiado los ejecutables de los códecs para generar los ficheros `ftype`.

- `results`: este es el directorio donde se almacenan los ficheros generados por los programas.
  - `audio_decoded`: en esta carpeta se guardan los ficheros de audio decodificados.
  - `audio_encoded`: en esta carpeta se guardan los ficheros de audio codificados.
  - `ftype`: en esta carpeta se guardan los ficheros `ftype`.
  - `dat`: en este directorio se guardan los ficheros con los datos de consumo.
- `src`: es este directorio se almacena todo el código fuente de los códecs.

A.3 Compilación y Ejecución

En esta sección se muestran las instrucciones para compilar el código y ejecutar la batería de medidas.

Para compilar todos los programas tan sólo es necesario ejecutar el script `compila.sh`, que se encargará además de crear los binarios correspondientes, de copiar los mismos en la carpeta `exec`.

A continuación se exponen los comandos que hay que introducir en la consola, tras conectar el dispositivo a través del cable USB, para proceder a realizar las baterías de medidas.

El primer paso es comprobar si el ordenador reconoce correctamente al dispositivo. Si tras introducir el siguiente comando aparece el número de serie del smartphone la comunicación podrá desarrollarse satisfactoriamente:

```
$adb devices
```

Tras comprobar que el teléfono está correctamente conectado al PC, se deben copiar en él los ficheros para poder llevar a cabo las medidas. Para ello, basta con situarse en el directorio del PC donde se encuentren los archivos del proyecto e insertar el siguiente comando:

```
$adb push ./* /data/pfc
```

Una vez se han copiado los ficheros al dispositivo, se deben introducir los siguientes comandos para comenzar con la toma de medidas:

```
$adb shell
#cd /data/pfc
#./start.sh
```

Cuando haya finalizado todo el proceso, el siguiente paso es copiar los ficheros generados durante la toma de medidas al ordenador, mediante el siguiente comando:

```
$cd carpeta_android/results
$adb pull /data/pfc/results/* .
```

El último paso será por tanto ejecutar los scripts para el cálculo de la constante de descarga de la batería, `descargacon.sh`, `descargasin.sh` y `cte.sh` y el script para generar la gráfica con los datos recogidos, `grafvar.sh`.

En el caso del estudio del consumo de la interfaz Wifi, el proceso es análogo, sólo cambia el script que manipula los datos recogidos y genera la gráfica, que en este caso es `grafwifi.sh`.

Apéndice B

Código fuente

En esta sección se incluirán el código fuente de los programas cliente y servidor usados para medir el consumo de la interfaz WiFi, así como los scripts más importantes que han sido ejecutados en los análisis de CPU e interfaz WiFi. Sin embargo, no se incluirá el código tomado del estudio en laptops ni los scripts del mismo que se han utilizado en este proyecto.

B.1 Scripts

B.1.1 `maxcpu.sh`

```
echo performance > /sys/devices/system/cpu/cpu0/cpufreq/
   scaling_governor
echo 1200000 > /sys/devices/system/cpu/cpu0/cpufreq/
   scaling_max_freq
echo 1200000 > /sys/devices/system/cpu/cpu0/cpufreq/
   scaling_min_freq
```

B.1.2 `compila.sh` (Análisis CPU)

```
export CC=arm-linux-androideabi-gcc

for i in amr g711 g723.1 g729ab g729b ilbc
do
        (cd ../src/cons/$i; make clean; make)
done

cd ../execs/cons
cp  ../../src/cons/amr/encoder amrencoder
cp  ../../src/cons/amr/decoder amrdecoder
cp  ../../src/cons/g711/g711demo  g711demo
cp  ../../src/cons/g729ab/coder g729abencoder
cp  ../../src/cons/g729ab/decoder g729abdecoder
cp  ../../src/cons/g729b/decoder g729bdecoder
cp  ../../src/cons/g729b/coder g729bencoder
cp  ../../src/cons/g723.1/lbccodec g723.1
cp  ../../src/cons/ilbc/ilbctest_enc ilbcencoder
```

```
cp  ../../src/cons/ilbc/ilbctest_dec ilbcdecoder

for i in amr g723.1 g729ab g729b
do
        (cd ../../src/ftype/$i; make clean; make)
done

cd ../../execs/ftype
cp ../../src/ftype/amr/encoder amrencoder
cp ../../src/ftype/amr/decoder amrdecoder
cp ../../src/ftype/g729ab/coder g729abencoder
cp ../../src/ftype/g729b/coder g729bencoder
cp ../../src/ftype/g723.1/lbccodec g723.1
```

B.1.3 compila.sh (Análisis interfaz WiFi)

```
export CC=arm-linux-androideabi-gcc
make cliente

export CC=gcc
make servidor
make clean
```

B.1.4 descargacon.sh

```
#CPU maximo rendimiento
echo performance > /sys/devices/system/cpu/cpu0/cpufreq/
   scaling_governor
echo 1200000 > /sys/devices/system/cpu/cpu0/cpufreq/
   scaling_max_freq
echo 1200000 > /sys/devices/system/cpu/cpu0/cpufreq/
   scaling_min_freq
rm descargacon.log

function trazadescarga () {
touch $1
while [ 1 ]
do
echo TIME=`date +%s` >> $1
cat  /sys/class/power_supply/battery/uevent >> $1
sleep 180
done
}
trazadescarga descargacon.log &

while [ 1 ]
do
  nice -n -20 ./lbccodec -c -r53 -v ../inputlong.raw ./cod.
     g723153
  rm ./cod.g723153
done
```

B.1.5 descargasin.sh

```
#CPU maximo rendimiento
echo performance > /sys/devices/system/cpu/cpu0/cpufreq/
   scaling_governor
echo 1200000 > /sys/devices/system/cpu/cpu0/cpufreq/
   scaling_max_freq
echo 1200000 > /sys/devices/system/cpu/cpu0/cpufreq/
   scaling_min_freq

rm descargasin.log
#Toma de datos de bateria
while [ 1 ]
do
echo TIME=`date +%s` >> descargasin.log
cat  /sys/class/power_supply/battery/uevent >> descargasin.
   log
sleep 180
done
```

B.1.6 cte.sh

```
ctesin=`cat descargasin.log | awk -F= 'BEGIN{Ti=0;T=0;Ah=0;
   volts=0;Ei=0;Ef=0}\
/TIME=/{if(Ti==0){Ti=$2}else{T=$2-Ti}}\
/POWER_SUPPLY_VOLTAGE_NOW=/{volts=$2/1000000}\
/POWER_SUPPLY_CAPACITY=/{Ah=$2*1650/100000; if (Ei==0){Ei=
   volts*Ah*3600}else{Ef=volts*Ah*3600}}\
END{print (Ei-Ef)/T}'`

ctecon=`cat descargacon.log | awk -F= 'BEGIN{Ti=0;T=0;Ah=0;
   volts=0;Ei=0;Ef=0}\
/TIME=/{if(Ti==0){Ti=$2}else{T=$2-Ti}}\
/POWER_SUPPLY_VOLTAGE_NOW=/{volts=$2/1000000}\
/POWER_SUPPLY_CAPACITY=/{Ah=$2*1650/100000; if (Ei==0){Ei=
   volts*Ah*3600}else{Ef=volts*Ah*3600}}\
END{print (Ei-Ef)/T}'`

echo $ctecon $ctesin | awk '{print($1 - $2)}'
```

B.2 Análisis Consumo WiFi

B.2.1 Programa Servidor

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>
#include <unistd.h>
```

```
#include <netdb.h>
#include <arpa/inet.h>
#include <sys/types.h>
#include <time.h>
#include <sys/resource.h>
#include <sys/time.h>

#define BUFFER_LEN 1024
#define NUMARGS 2
#define PLAZO 20

int recibido_en_plazo (int descriptor, int segundos);
int main(int argc, char *argv[])
{
  int bytes=0;
  if(argc!=NUMARGS)
    {
      fprintf(stderr,"Uso: servidor puerto\n");
      exit(1);
    }
  int sockfd;
  struct timeval ti, tf;
  struct sockaddr_in my_addr;
  struct sockaddr_in their_addr;
  int addr_len, numbytes;
  double tiempo;
  char buf[BUFFER_LEN];
  char fichero_datos[24];
  FILE *fichero;

  if ((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) == -1) {
    perror("socket");
    exit(1);
  }

  my_addr.sin_family = AF_INET;
  my_addr.sin_port = htons(atoi(argv[1]));
  my_addr.sin_addr.s_addr = INADDR_ANY;
  bzero(&(my_addr.sin_zero), 8);

  if (bind(sockfd, (struct sockaddr *)&my_addr, sizeof(
     struct sockaddr)) == -1) {
    perror("bind");
    exit(1);
  }

  addr_len = sizeof(struct sockaddr);
  printf("Esperando datos ....\n");
  while (1)
    {
      if (recibido_en_plazo(sockfd, PLAZO))
        {
          if ((numbytes=recvfrom(sockfd, buf, BUFFER_LEN, 0,
               (struct sockaddr *)&their_addr, (socklen_t *)&
              addr_len)) == -1) {
```

```
            perror("recvfrom");
            exit(1);
          }

          if ((char)buf[0]=='V')
            {
              printf("Paquete con codec\n");
              if ((numbytes=recvfrom(sockfd, buf , 24, 0, (
                  struct sockaddr *)&their_addr, (socklen_t
                  *)&addr_len)) == -1) {
                     perror("recvfrom");
                     exit(1);
              }
              printf("%s\n",buf);
              sprintf(fichero_datos,"%s.temp",buf);
            }
          if ((char)buf[0]=='i')
            {
              printf("Paquete inicial\n");
              gettimeofday(&ti, NULL);
            }
          if ((char)buf[0]=='f')
            {
              gettimeofday(&tf, NULL);
              printf("Paquete final\n");
              tiempo=tf.tv_sec + (double)tf.tv_usec/1000000
                  -ti.tv_sec - (double)ti.tv_usec/1000000;
              if ((fichero=fopen(fichero_datos,"a"))){
                fprintf(fichero,"%f\n",tiempo - (double)30);
                fclose(fichero);
                printf("%f\n",tiempo);
              }
            }
          bytes+=numbytes;
        }
      else
        {
          fprintf(stdout,"Han pasado 20 segundos sin recibir
              peticiones\n");
          printf("Se han recibido %d bytes\n",bytes);
          close(sockfd);
          exit(0);
        }
    }
}

int recibido_en_plazo (int descriptor, int segundos)
{
  struct timeval plazo;
  fd_set readfds;
  plazo.tv_sec = segundos;
  plazo.tv_usec = 0;
  FD_ZERO(&readfds);
  FD_SET(descriptor, &readfds);
  return
```

```
    select(descriptor+1, &readfds,
           (fd_set *)NULL,
           (fd_set *)NULL,
           &plazo) != 0;
}
```

B.2.2 Programa cliente

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <string.h>
#include <arpa/inet.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
#include <fcntl.h>
#include <unistd.h>
#include <time.h>
#include <sys/resource.h>
#include <sys/time.h>

#define NUMARGS 10

double consendto(char* ip, int puerto, char* cadena, int
   framesvoz,
                 int framessid, int framesntx, int tamvoz,
                    int tamsil, int ms);
int main(int argc, char* argv[])
{
  if(argc!=NUMARGS)
    {
      printf("Uso: cliente IPservidor puerto fichcodec
         tramasvoz tramassil tramasnotx tamvoz tamsil ms");
      exit(1);
    }
  consendto(argv[1],atoi(argv[2]),argv[3],atoi(argv[4]),atoi
     (argv[5]),atoi(argv[6]),atoi(argv[7]),atoi(argv[8]),
     atoi(argv[9]));
  return 0;
  }

double consendto(char* ip, int puerto, char* cadena, int
   framesvoz, int framessid, int framesntx, int tamvoz, int
   tamsil, int ms)
{
  int soc;
  struct sockaddr_in addr;
  int numbytes;
  FILE *idfich;
  char fin = 'f';
  char inicio ='i';
```

```
int ftype;
char p = 'V';

if ((tamvoz%8) == 0)
  tamvoz = tamvoz/8;
else
  tamvoz = (tamvoz/8)+1;

if ((tamsil%8) == 0)
  tamsil = tamsil/8;
else
  tamsil = (tamsil/8)+1;

char voz[16+tamvoz];
char sil[16+tamsil];

if ((soc = socket(AF_INET, SOCK_DGRAM, 0)) == -1) {
  perror("socket");
  exit(1);
}
addr.sin_family = AF_INET;
addr.sin_port = htons(puerto);
addr.sin_addr.s_addr = inet_addr(ip);
bzero(&(addr.sin_zero), 8);
if((idfich = fopen(cadena,"r")) != NULL)
  {
    numbytes=sendto(soc,(char *)&p,sizeof(char),0,(struct
       sockaddr *)&addr, sizeof(struct sockaddr));
    sleep(1);
    numbytes=sendto(soc,cadena,strlen(cadena)+1,0,(struct
       sockaddr *)&addr, sizeof(struct sockaddr));
    sleep(1);
    numbytes=sendto(soc,(char *)&inicio,sizeof(char),0,(
       struct sockaddr *)&addr, sizeof(struct sockaddr));
    printf("%s\n",cadena);
    while((ftype=fgetc(idfich)) != EOF)
      {
        switch(ftype) {
        case '1':
          usleep(ms*1000);
          if((numbytes=sendto(soc,(char *)&voz,sizeof(voz)
             ,0,(struct sockaddr *)&addr, sizeof(struct
             sockaddr))) == -1) {
            perror("sendto");
            close(soc);
            fclose(idfich);
          }
          break;

        case '2':
          usleep(ms*1000);
          if((numbytes=sendto(soc,(char *)&sil,sizeof(sil)
             ,0,(struct sockaddr *)&addr, sizeof(struct
             sockaddr))) == -1) {
            perror("sendto");
```

```
              close(soc);
              fclose(idfich);
            }
            break;

          case '0':
            usleep(ms*1000);
            break;
          }
        }
      numbytes=sendto(soc,(char *)&fin,sizeof(char),0,(
          struct sockaddr *)&addr, sizeof(struct sockaddr));
      fclose(idfich);
    }
  else
    {
      numbytes=sendto(soc,(char *)&p,sizeof(char),0,(struct
          sockaddr *)&addr, sizeof(struct sockaddr));
      sleep(1);
      numbytes=sendto(soc,cadena,strlen(cadena)+1,0,(struct
          sockaddr *)&addr, sizeof(struct sockaddr));
      sleep(1);
      printf("%s\n",cadena);
      numbytes=sendto(soc,(char *)&inicio,sizeof(char),0,(
          struct sockaddr *)&addr, sizeof(struct sockaddr));
      while (framesvoz>0)
        {
          usleep(ms*1000);
          if((numbytes=sendto(soc,(char *)&voz,sizeof(voz)
              ,0,(struct sockaddr *)&addr, sizeof(struct
              sockaddr))) == -1) {
            perror("sendto");
            close(soc);
            fclose(idfich);
          }
          framesvoz--;
        }
      numbytes=sendto(soc,(char *)&fin,sizeof(char),0,(
          struct sockaddr *)&addr, sizeof(struct sockaddr));
    }
  close(soc);
  return 0;
}
```