SIPX VIDEOPHONE

Desarrollo de un cliente SIP con videollamada

Universidad de la República, Facultad de Ingeniería

Autores:
Mariana **Draper de Santiago**
Valeria **Meilán Morón**
Manuela **Ramos Malcuori**

Tutor:
José **Joskowicz**

Agosto **2007**

Resumen

En este proyecto se presenta el desarrollo de un cliente SIP con el cual es posible establecer llamadas de audio y video sobre una red LAN. Dicha aplicación se desarrolla a partir de un proyecto ya existente al cual se le incorporan nuevas facilidades multimedia; la transmisión de video y un nuevo codec de compresión de audio. El aplicativo resultante conforma el objetivo de este proyecto y se le denominó sipXvideoPhone.

Se presenta un estudio teórico de SIP, el principal protocolo en que se basó el desarrollo de sipXvideoPhone. Se realiza también un análisis de aquellos estándares que sirvieron de apoyo al momento de implementar cada una de las partes que conforman el desarrollo.

Finalmente, se presenta un análisis de los resultados obtenidos y las conclusiones asociadas con todo el trabajo realizado.

Agradecimientos

Durante el período en que se llevó a cabo el proyecto varias fueron las personas que nos ayudaron e hicieron posible su realización. En primer lugar nos gustaría agradecerle al tutor, José Joskowicz, por sugerirnos el proyecto y estar siempre dispuesto a evacuar nuestras dudas.

A Pablo Cancela por sus tantas horas de paciencia y valiosos aportes.

A los docentes del IIE Federico Lecumberry, Ignacio Ramírez, Víctor González y Rafael Sotelo quienes nos dieron una mano en algún momento.

A Florencia Meilán por diseñarnos el logo, la estética del teléfono y la tapa de la presente documentación.

A Federico Morales por prestarnos el servidor de respaldo que nos fue de mucha utilidad.

A nuestros compañeros de generación por acompañarnos en toda la carrera y brindarnos un apoyo constante.

Finalmente a nuestras familias y amigos que sin su apoyo no hubiese sido posible realizar este proyecto.

Índice general

Capítulo 1

Introducción

1.1. Motivación

Dado el auge de la telefonía sobre IP y el desarrollo previsto para los próximos años de la misma, resulta de gran interés el aprendizaje de esta tecnología y la continuación del desarrollo de aplicaciones existentes. Asimismo, el crecimiento del protocolo SIP para el establecimiento de sesiones multimedia sobre IP, y la tendencia de éste a reemplazar H.323 llevó a la elección del mismo como base para este proyecto.

El hecho de que cada vez haya más ancho de banda disponible, permite el uso de aplicaciones multimedia sobre redes IP. Por esta razón resulta interesante la utilización de este recurso en aplicaciones con requerimientos importantes de ancho de banda como lo es la transmisión de video.

Por otra parte, la optimización de los recursos es un objetivo que nunca debe perderse de vista en lo que refiere a aplicaciones de VoIP. A tales efectos resulta de gran utilidad la incorporación de procesos de codificación sobre los datos transmitidos a través de la red. Disponer de una variedad de codecs de compresión al momento de utilizar los aplicativos multimedia permite seleccionar el que resulte más adecuado en cada circunstancia.

Finalmente los proyectos de código abierto (*open source*) se encuentran hoy en día en una etapa de gran crecimiento, resultando muy atractiva su utilización para continuar de manera conjunta con el desarrollo de aplicaciones que resultan de interés en el área de las telecomunicaciones. La transmisión de video es una facilidad no disponible en la mayoría de los aplicativos de código abierto, haciendo atractivo el objetivo del presente proyecto.

1.2. Objetivos

El objetivo general del proyecto es el estudio del protocolo SIP, sus protocolos relacionados y finalmente el desarrollo de un cliente SIP con capacidad de transmisión de voz y video a

través de una LAN. El proyecto busca una comprensión extensiva del marco teórico para luego realizar una aplicación donde se reflejen todos los conceptos adquiridos.

Para desarrollar la aplicación SIP, se parte de un softphone de código abierto existente y se incorporan al mismo nuevas facilidades como incorporación de un nuevo codec de audio y la transmisión de un flujo de video.

¿Por qué partir de un softphone existente?

El proyecto pretende cubrir dos aspectos importantes: brindar a quienes lo realizan un conocimiento profundo de una nueva tecnología emergente y contribuir con el desarrollo de algún componente sobre la misma.

En base a estos criterios y luego de una investigación previa acerca de lo disponible, tanto en el mercado comercial, como en el mundo de código abierto, se decidió que lo mejor era partir de un proyecto existente. Se utiliza un desarrollo de código abierto al que se le puede agregar valor, aportando con esto a la comunidad de desarrolladores en telecomunicaciones, y permitiendo aplicar y afianzar los conocimientos teóricos que se obtendrán en las primeras etapas del proyecto.

El desarrollo del proyecto se puede dividir en tres grandes etapas:

- Estudiar en profundidad el protocolo SIP que será la base de todo el desarrollo posterior. A su vez, estudiar otros protocolos relacionados que también servirán de utilidad al momento de implementar la aplicación.
- Incorporar el nuevo codec de audio al cliente SIP que se tomó como punto de partida. Además de los codecs de audio soportados originalmente por el teléfono, se agregará soporte para uno de los siguientes: G.729, G.722, G.723, G.728, GSM.
- Agregar a la aplicación soporte para transmitir video y poder realizar llamadas con este flujo de medios. El codec de video a incorporar será uno de los siguientes: H.263, H.264, MPEG-1, MPEG-2.

Como resultado final se debe disponer de una plataforma con la cual sea posible establecer llamadas de audio y video entre dos softphones SIP conectados a través de una LAN. El aplicativo debe ser compatible con el sistema operativo *Windows*.

Se considerará que el proyecto fue exitoso si se lograron adquirir los conceptos teóricos y aspectos técnicos referentes a la temática del proyecto, si se logró una buena gestión del proyecto, y si se logró obtener un producto final que funcione y refleje los conocimientos previamente adquiridos.

Capítulo 2

Fundamento teórico

2.1. Introducción

En el presente capítulo se realiza un análisis teórico de los elementos que fundamentan el desarrollo de la aplicación sipXvideoPhone.

Se describen los conceptos que utiliza el protocolo de señalización SIP y en los cuales se basa el funcionamiento de nuestra aplicación. El estudio de este protocolo resulta de gran utilidad para comprender la arquitectura más comunmente utilizada hoy en día en aplicaciones de voz sobre IP. A su vez permitirá realizar las implementaciones necesarias para poder lograr el objetivo deseado.

Se presenta también un análisis del protcolo SDP (Session Description Protocol), y cómo este es utilizado en conjunto con el protocolo SIP para poder realizar de manera correcta la negociación de las llamdas.

Finalmente se describen los protocolos RTP (Real Time Protocol) y RTCP (Real Time Control Protocol), y cómo estos son utilizados en la transmisión de flujos de tiempo real.

2.2. Voz sobre IP

Voz sobre IP (VoIP) es una tecnología que permite la transmisión de señales de voz sobre una red de datos basada en el protocolo IP. En este tipo de redes la señal de voz se transmite en forma de paquetes, una metodología completamente distinta a la adoptada por las redes telefónicas tradicionales.

Las redes telefónicas tradicionales se basan en la conmutación de circuitos. En este tipo de redes diseñadas específicamente para transmitir señales de voz, los recursos quedan asignados aún cuando no hayan llamadas establecidas. Más aún, las redes funcionan de manera que para cada conversación establecida se asigna durante todo el tiempo un canal dedicado punta a pun-

ta. Esta técnica resulta en la subutilización de recursos dado que el canal se encuentra asignado en todo momento y que durante una llamada la señal de voz no se transmite constantemente en ambos sentidos.

En contraparte, existen las redes de conmutación de paquetes que originalmente fueron diseñadas con el objetivo de transmitir datos y no señales de audio. Este tipo de redes presenta un flujo de información no constante que es transmitido en forma de paquetes y que asigna los recursos de manera dinámica a medida que son requeridos.

Hoy en día Internet ha logrado proporcionar una red de conmutación de paquetes integradora de varios tipos de información como puede ser voz, datos o cualquier otra aplicación multimedia. El objetivo es poder brindar un conjunto de servicios a través de una misma red de manera de reducir costos y optimizar la utilización de recursos.

En el caso particular de VoIP, se trata de enviar información de voz a través de la red Internet y de esta forma aprovechar de mejor manera los recursos. Como se mencionó anteriormente, al realizar llamadas telefónicas el flujo de información no es constante y resultaría adecuado utilizar una red de transmisión que acompañe este comportamiento, como lo es la red de conmutación de paquetes sobre el protcolo Internet.

2.2.1. Ventajas y desventajas

Con VoIP es posible reducir los altos costos de la telefonía convencional ya que no es necesario disponer de un canal punta a punta especialmente dedicado para cada llamada. Más aún, al hacer uso de Internet se están aprovechando los recursos asignados a la red de datos para combinar varios servicios, especialmente cuando los usuarios no utilizan toda la capacidad de la red.

Otro punto a favor de utilizar VoIP es la posibilidad de comprimir los datos al momento de transmitirlos, logrando optimizar más el ancho de banda disponible.

Los distintos tipos de servicio brindados sobre IP, requieren disponer de la calidad de servicio adecuada. En aplicaciones de tiempo real como lo es la transmisión de voz y video los retardos juegan un papel muy importante y por ello se deben tomar acciones que permitan combatir este tipo de problemas inherentes al protocolo IP. Los problemas que más afectan la calidad de servicio son:

Latencia: este es el retardo extremo a extremo de los paquetes transmitidos, consecuencia de los tiempos consumidos en las etapas de codificación, paquetización, transmisión por la red, entre otros. Al tratarse de paquetes con información de voz se considera aceptable una latencia máxima de 150 ms, dado que al superar este valor la comunicación entre las puntas se torna molesta.

Jitter: es la variación de los retardos en el tiempo, provocando que los paquetes lleguen a destino con diferentes retardos. Esto puede ser consecuencia de los distinos caminos

seguidos por los paquetes, congestión, o por pérdida de sincronización en algún tramo del camino.

Para poder reproducir la voz de manera correcta en el receptor, es necesario que la señal arribe en bloques a una tasa constante. Debido a la presencia del *jitter*, en la práctica esto no sucede así. En consecuencia se utiliza un buffer receptor, comúnmente conocido como *Jitter Buffer*, que permite almacenar las muestras de audio que van llegando para luego retirarlas de manera adecuada al momento de procesarlas.

El tamaño del buffer es ajustable y marca un compromiso entre el *jitter* que se pretende evitar y el retardo adicional generado. Cuanto mayor es el buffer de recepción menor es el *jitter* percibido, pero se agrega un retardo adicional debido a la mayor cantidad de paquetes que son retrasados en el buffer.

Pérdida de paquetes: las aplicaciones de tiempo real comunmente utilizan redes de paquetes no orientadas a conexión, en las cuales frente a la pérdida de un paquete el mismo no es retransmitido. Este es el caso de VoIP que utiliza el protocolo UDP en la capa de transporte. Con esto se evita añadir retardos a la comunicación y retransmisión pero se generan pérdidas de información. A pesar de esto, este es el mecanismo utilizado dado que la transmisión de voz es claramente más sensible a los retardos que a la pérdida de alguna muestra de señal.

Las redes IP no utilizan ningún paliativo a la pérdida de paquetes y solamente ofrecen un servicio *best effort*.

2.2.2. Protocolos

La red de VoIP se conforma de varios dispositivos que interactúan para poder lograr de manera adecuada la transmisión de voz de una punta a otra. Actualmente existen dos arquitecturas principales que definen la manera en que se comunican los elementos de la red para lograr el objetivo deseado. Estos son el estándar H.323 desarrollado por la ITU-T, y el protocolo SIP (Session Initiation Protocol)[1].

H.323 es un estándar originalmente diseñado para comunicaciones con servicio multimedia, proporcionando la convergencia de voz, video y datos sobre una red de conmutación de paquetes.

Por otra parte se encuentra SIP, un protocolo desarrollado por el grupo MMUSIC (Multimedia Session Control) del IETF con el objetivo de definir una arquitectura de señalización para VoIP. Sus especificaciones se encuentran publicadas en la RFC 3261.

SIP es un protocolo de señalización a nivel de capa de aplicación para el establecimiento y manejo de sesiones multimedia con múltiples participantes. A tales efectos, utiliza mensajes de petición y respuesta que son transmitidos entre las puntas que desean establecer o finalizar una sesión. Se basa también en protocolos adicionales como lo son RTP/RTCP[2] y SDP (Session Description Protocol)[3], al mismo tiempo que reutiliza conceptos de HTTP[4] y SMTP[5].

H.323 es un protocolo que presenta mayor comlejidad a la hora de compararlo con SIP. En él se definen una gran cantidad de elementos y funciones que tornan al estándar poco flexible para adecuarse a cualquier tipo de aplicación. Esto genera además una mayor dificultad al momento de querer utilizarlo.

SIP, por su parte, es un protocolo extremo a extremo lo que implica que toda la lógica, a excepción del enrutamiento intermedio, es almacenada en los dispositivos finales. Esto permite independizar a las aplicaciones SIP desarrolladas del resto de los componentes de la red. Si bien la arquitecura de SIP es bastante menos definida que la de H.323, esto resulta una ventaja al momento de querer integrarla con otros aplicativos.

Al desarrollar una aplicación basada en **VoIP** es necesario elegir un protocolo que defina la arquitectura a utilizar y cómo se comunican los dispositivos a través de la red. Este es un tema de gran controversia hoy en día y la elección resulta de un análisis de los protocolos recién mencionados. Dado que no es posible afirmar que uno es mejor que otro, la decisión dependerá de cuál resulte más adecuado para la aplicación que se desea desarrollar.

2.3. Session Initiation Protocol: SIP

SIP, *Session Initiation Protocol*, es un protocolo de señalización que se utiliza para iniciar y manejar sesiones de medios.[1] Fue desarrollado por la IETF (*Internet Engineering Task Force*) y está descrito en la RFC 3261.

SIP le permite a usuarios finales de Internet descubrirse entre sí y acordar las características de la sesión. A los usuarios finales se les llama agentes de usuario o en inglés *user agents*. A su vez permite la utilización de servidores *proxy* donde los agentes de usuario se registran, y a través de los cuales envían invitaciones y otro tipo de solicitudes a otros usuarios.

Es un protocolo extremo a extremo perteneciente a la capa de aplicación que posibilita crear, modificar y terminar sesiones multimedia. Para ser usado no es necesario pasar por una red SIP o utilizar un servidor SIP, basta que dos extremos corran la pila (*satck*) para el protocolo y conozcan sus direcciones IP. Tiene una naturaleza cliente-servidor ya que un mismo dispositivo SIP actúa en una misma sesión como cliente y como servidor. Entre las principales aplicaciones de SIP se encuentra la telefonía IP aunque también puede ser usado en otros escenarios. En la figura 2.1 se puede ver la pila de protocolos utilizada en aplicaciones multimedia[6].

Para realizar el establecimiento y finalización de una sesión multimedia, SIP lleva a cabo las siguientes funciones:

- *Localización de usuarios.* Permite conocer dónde se encuentra el dispositivo remoto para poder establecer una comunicación.

[1] Una sesión se considera como el intercambio de datos entre un grupo de participantes.

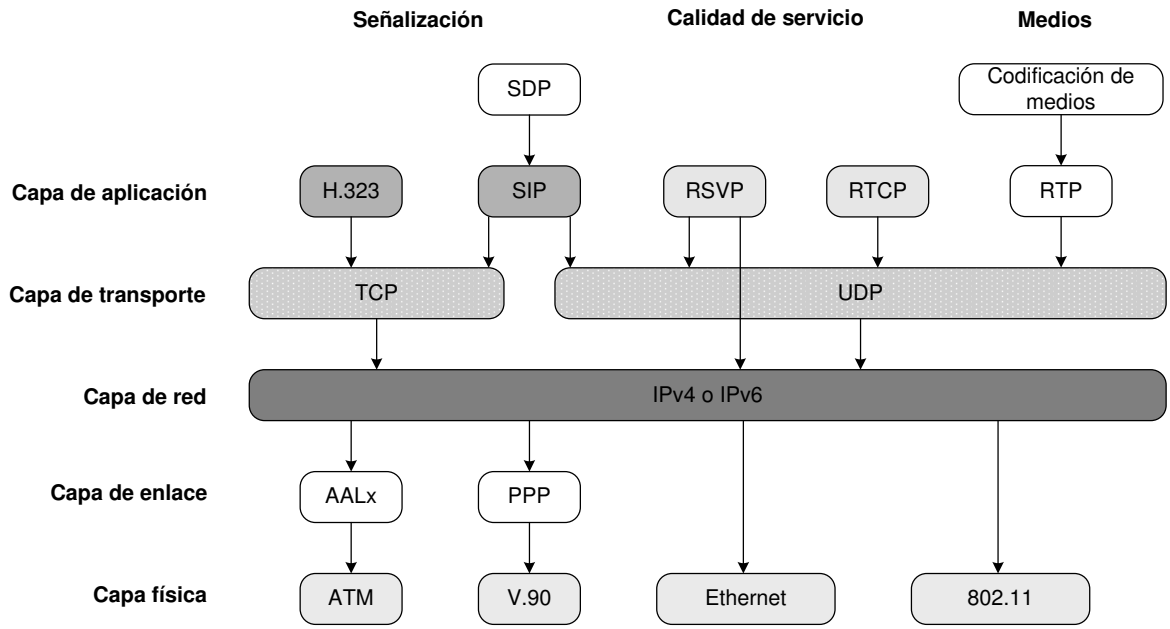


Figura 2.1: Pila de protocolos multimedia

- *Disponibilidad del usuario.* Se determina la voluntad del usuario llamado para contactarse.
- *Capacidades del usuario.* Se permite la negociación de medios para establecer la sesión.
- *Modificación de sesiones existentes.* Se incluye la transferencia, la terminación y modificación de parámetros de sesiones.
- *Configuración de sesiones.* Determinación de los parámetros de la sesión en ambos extremos.

SIP funciona con otros protocolos para construir la arquitectura necesaria para la comunicación multimedia interviniendo únicamente en la señalización. Como se puede ver en la figura 2.1, SIP encapsula el protocolo de descripción de sesiones SDP para la negociación de medios. Una vez establecida la comunicación utiliza el protocolo RTP (*Real-Time Transport Protocol*) para el transporte de los mismos. SIP puede correr sobre protocolos no seguros como UDP así como también sobre TCP o TLS. Para el primer caso provee mecanismos que le otorgan confiabilidad como temproizadores de retransmisión, incremento de un número de secuencia de comandos (CSeq) y reconocimiento positivo de recepción de mensajes.

El protocolo SIP se extendió para brindar las siguientes facilidades:

- Permitir la plublicación y solicitud de información de presencia.
- Notificación de eventos.

- Transporte de mensajeria instantánea.

En la RFC 3856 [7] se describe el uso de SIP para proporcionar servicios de presencia de un usuario por medio de suscripciones, notificaciones y publicaciones. La presencia se define como la disponibilidad y voluntad de un usuario para ser contactado por otros. En la RFC 3265 [8] se describen métodos para la notificación específica de eventos que hacen a SIP un protocolo extensible.

2.3.1. Clientes y servidores SIP

Los principales componentes de SIP son los agentes de usuario y los servidores. Ambos se comunican entre sí por medio de direcciones llamadas SIP URI (*Uniform Resource identificator*). Éstas se resuelven en direcciones IP a través de consultas a los servidores *proxy* SIP quienes realizan búsquedas en tablas DNS.

Agentes de usuario SIP

Los agentes de usuario (UA) son dispositivos SIP de uso final. Son entidades lógicas que dentro de una misma sesión actúan tanto como cliente como servidor. Cada UA representa a un usuario y actúa en su nombre para iniciar, configurar o terminar sesiones de medios con otros UA. [2]

Cuando un UA actúa como cliente se le llama UAC (*User Agent Client*) y cuando actúa como servidor se le llama UAS (*User Agent Server*). Un UAC es el que genera solicitudes mientras que un UAS es el que las responde. Cabe destacar que todos los mensajes SIP son o bien solicitudes o respuestas.

Un UA debe mantener el estado de las llamadas que inicia o en las que participa. Como mínimo debe guardar las etiquetas locales y remotas, `Call-ID`, `CSeq` y cualquier información de medios que necesite (estas etiquetas se describen en las siguientes secciones). Además debe soportar el uso del protocolo de descripción de medios SDP.

Servidores SIP

Los servidores SIP son aplicaciones que aceptan solicitudes SIP y las responden. Proveen servicios a los UA y no se deben confundir con los UAS ya que son otro tipo de entidad. Para poder interactuar correctamente con los UA, deben soportar los portocolos de transporte UDP, TCP o TLS que los mismos utilizan.

Hay tres tipos de servidores: servidores *proxy*, servidores de registro y servidores de redireccionamiento. Cada uno es una entidad lógica que puede estar implementada de forma independiente o se pueden implementar varias en un mismo servidor. En la figura 2.2 se puede apreciar la interacción entre los distintos servidores y los agentes de usuario.

[2] El usuario puede ser una persona o por ejemplo un *gateway*.

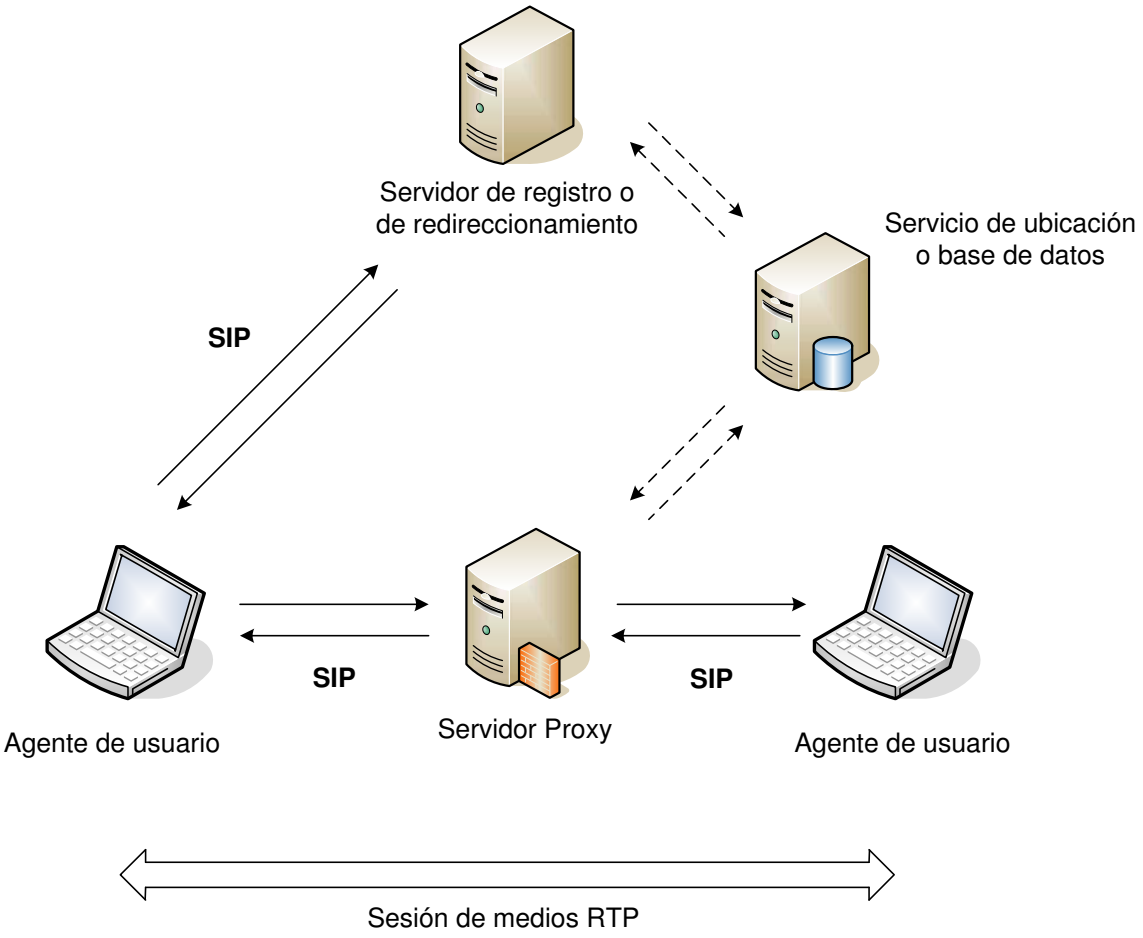


Figura 2.2: Interacción entre los UA y los servidores SIP [6]

1. **Servidor Proxy**

   Es una entidad intermedia que actúa tanto como cliente como servidor. Recibe solicitudes de un UA u otro servidor *proxy* y actúa en su nombre reenviando o respondiendo la solicitud. Su principal función es la de enrutar, es decir asegurar que la solicitud sea enviada a otra entidad cercana al usuario al que se le debe enviar. Un *proxy* puede reescribir partes de una solicitud en base a reglas muy estrictas que mantienen la transparencia extremo a extremo de SIP.

   Los servidores *proxy* tienen acceso a bases de datos o a servicios de ubicación (*location services*) para determinar a dónde entregar la solicitud. En esas bases de datos existe un registro que permite localizar al usuario.

   Hay dos tipos de servidores proxy, *stateless* y *stateful*.

***Stateless Proxy*:** reenvía las solicitudes y respuestas que recibe basándose en la información que contiene el mensaje. Una vez que el mensaje fue analizado, procesado y reenviando o respondido, no se guarda información del estado de la transacción. Nunca retransmite un mensaje, y no usa temporizadores.

***Stateful Proxy*:** es el tipo más común de *proxy*. Almacena copias de las solicitudes y respuestas de una transacción, y usa esa información para procesar futuras solicitudes y respuestas. A diferencia de los *stateless proxy* usa temporizadores.

2. **Servidor de redireccionamiento**

   Es una entidad que acepta solicitudes SIP, pero no las procesa. En respuesta genera una lista con todas las posibles ubicaciones alternativas de los usuarios destino de las solicitudes. Usa una base de datos creada por un servidor de registro para buscar a los usuarios y luego envía la información que encontró al usuario que generó la solicitud.

3. **Servidor de registro**

   Es una entidad SIP que acepta solicitudes de registro de clientes que indican las direcciones en las que se los puede encontrar. A cualquier otro tipo de solicitud que se le envíe responde con un mensaje indicando que no dispone de esa facilidad. Cuando el servidor de registro obtiene la información que el UA le envió, la almacena en una base de datos (*location service*) correspondiente con el dominio que maneja. Luego esta información podrá ser accedida por servidores *proxy* o de redireccionamiento.

Otras entidades SIP

***Agentes de presencia (PA)*:** es una nueva entidad lógica capaz de aceptar suscripciones, almacenar el estado de la suscripción y generar notificaciones cuando hay cambios en el estado de presencia [9].

***Agentes de usuario Back-to-Back (B2BUA)*:** es una entidad lógica que recibe solicitudes y las procesa actuando de UAS. Para generar las respuestas actúa como un UAC enviando solicitudes y reformulando las que le llegaron. Un uso común del B2BUA es dar un servicio para mantener en forma anónima la identidad de dos participantes permitiendo igualmente su comunicación. Los usuarios se pueden comunicar sin tener acceso a la identificación URI, dirección IP o cualquier otra información del agente remoto. Este tipo de agentes de usuario rompen el equema de comunicación extremo a extremo de SIP.

***Gateway SIP*:** es un tipo especial de agente de usuario que permite la conversión de un protocolo de señalización a otro. Esta entidad actúa en representación de otro protocolo y no en representación de una persona. Los gateways que convierten la señalización de SIP a H.323 permiten que un cliente SIP pueda comunicarse en forma directa con un temrinal H.323 sin que el flujo RTP atraviese el gateway. Por otra parte, un gateway de SIP a PSTN (*Public Switched Telephone Network*) interviene tanto en el camino de señalización como en el de los medios.

2.3.2. Diálogos y transacciones

Los diálogos y transacciones son dos conceptos importantes dentro de SIP.

***Diálogo*:** es una relación *peer-to-peer* SIP entre dos agentes de usuario que persiste por un cierto tiempo. El diálogo facilita la secuencia de mensajes entre los UA y el enrutamiento de los mismos. Un diálogo se identifica en cada UA mediante un identificador que consiste en el campo `Call-ID`, una etiqueta local (*tag*) y una etiqueta remota. El identificador de diálogo es diferente para cada agente de usuario.

***Transacción*:** SIP es un protocolo de transacciones, es decir que la comunicación se da en base al intercambio de mensajes independientes. Una transacción SIP consiste en una única solicitud y el conjunto de respuestas asociadas. Éste último puede componerse de una respuesta o varias intermedias (provisionales) y una respuesta final.

2.3.3. Direccionamiento

Si bien en las transacciones SIP pueden intervenir agentes de usuario y servidores, los únicos elementos que poseen direcciones SIP son los primeros. Los servidores se identifican mediante una dirección IP y un puerto TCP/UDP. El puerto por defecto es el 5060, y es el que se utiliza comúnmente.

Localización de UA

Para localizar a los agentes de usuario se hace uso de direcciones SIP, llamadas URI (*Unified Resource Identifier*). Este esquema de direccionamiento es similar al utilizado en las direcciones de correo electrónico (*mailto URL*) en cuanto a su sintaxis `usuario@servidor`. Una dirección URI está conformada por:

- Una parte que contiene información del usuario, donde se incluye el nombre o un número de teléfono, y opcionalmente una clave (*password*).
- Una parte con información del servidor, donde se incluye el puerto (opcional) y el nombre de dominio o la dirección de red.
- A su vez se pueden incluir una parámetros URI adicionales y encabezados.

La forma genérica de una dirección URI es:

```
sip:usuario:clave@servidor:puerto;parametros-uri?encabezados
```

`usuario`**:** identificador de un recurso en particular dentro de un mismo servidor. La información de usuario es opcional pero si el signo "@" está presente debe incluirse. A su vez si el servidor destino puede procesar números telefónicos, por ejemplo un *gateway* de telefonía, en este campo puede ir un número telefónico E.164.

`clave`**:** asociado al usuario. Se recomienda no utilizar este campo ya que por encontrarse en texto plano es débil ante ataques de seguridad.

`servidor:` es el servidor que contiene al recurso SIP. Puede ser o bien un nombre de dominio o una dirección numérica IPv4 o IPv6.

`puerto:` número de puerto al que se envía la solicitud.

`parametros-uri:` la notación para estos parámetros es `nombre-parametro = valor`. Dentro de los parámetros posibles se encuentra `transport` (para indicar el protocolo de transporte), `maddr` (indica la dirección del servidor para contactar al usuario), `ttl` (tiempo de vida, *time-to-live*, del paquete UDP *multicast*), `user` (define el uso de la dirección URI, puede ser `ip` o `phone`).

`encabezados:` entre los posibles encabezados se encuentran `subject` y `to`.

Como se mencionó anteriormente es posible que en el campo `usuario` de la dirección SIP vaya un número telefónico. Para esto el parámetro URI `user` debe estar definido como `phone`. El cuadro 2.1 muestra ejemplos de direcciones SIP.

`sip:bob@dominio.com`
`sip:alice:clave_alice@dominio.com;transport=tcp`
`sip:bob@192.168.1.100:5060;user=phone`

Cuadro 2.1: Ejemplos de SIP URI.

Localización de servidores

SIP provee mecanismos para el descubrimiento de otros usuarios. Si un usuario desea comunicarse con otro, primero debe conocer la ubicación en la que el usuario se encuentra disponible. Este descubrimiento se hace usualmente mediante los servidores *proxy* y los de redireccionamiento que son responsables de recibir las solicitudes y enviarlas a la ubicación correspondiente.

El problema está en cómo hacen los agentes de usuario para ubicar a los distintos servidores. En primer lugar los UA deben registrarse en los servidores de registro para que luego los servidores *proxy* y de redireccionamiento puedan acceder a los registros de los usuarios mediante una consulta.

En la RFC 3261 no se especifica cómo los UAC pueden descubrir la ubicación de un *proxy* local o de un servidor de redireccionamiento.

Una posibilidad para que un UAC descubra los servidores es realizar una consulta DNS. En la RFC 3263 se describen procedimientos de consulta DNS para permitirle a un UAC obtener la dirección IP de un servidor. También se describe una conversión de una dirección SIP URI en una dirección IP, un puerto y un protocolo de transporte y así permitirle al servidor contactar

al cliente [10].

Otro mecanismo para realizar esta tarea es que el UAC descargue un archivo de configuración que contenga la información de localización del servidor (DNS o dirección IP y puerto).

Para la ubicación de los servidores de registro, es posible configurar estáticamente a los UA o utilizar *multicast*.

2.3.4. Mensajes SIP

SIP es un protocolo basado en texto plano que usa el grupo de caracteres UTF-8 definido en la RFC 2279 [11].[3]

Los mensajes SIP están divididos en tres partes: una "línea de inicio" (*start-line*), uno o varios encabezados y el cuerpo del mensaje que puede ir vacío. Los encabezados y el cuerpo del mensaje van separados por una línea en blanco. El tipo de mensaje SIP se distingue a partir de la "línea de inicio".

La forma genérica del mensaje SIP es:

```
linea de inicio
encabezado del mensaje
CRLF
cuerpo del mensaje
```

Solicitudes SIP

El formato de la "línea de inicio" para las solicitudes es el siguiente:

```
<Metodo> <Solicitud-URI> <Version SIP>
```

- ***Método***: indica el tipo de solicitud.
- ***Solicitud-URI***: es una dirección URI que indica el destino de esta solicitud.
- ***Versión SIP***: indica la versión de SIP que se está usando. La versión actual de SIP es SIP/2.0 y es la misma que definida en la RFC 2543.

En en el estándar de SIP se definen seis tipos de métodos:

`INVITE`**:** este método es empleado para iniciar una sesión entre agentes de usuario. Este mensaje puede contener en su cuerpo la descripción SDP indicando la información de medios de quien lo envía.

[3]ISO/IEC 10646-1 define un grupo de caracteres de varios octetos universal llamado UCS por *Universal Character Set*. Al no ser compatibles con muchas aplicaciones se desarrolló UTF-8 (UCS Transformation Format).

`REGISTER`: este método es usado por un UA para informarle a una red SIP sobre su información de contacto actual, `Contact URI` (dirección IP) y la dirección URI a la que deben dirigirse las solicitudes para ese contacto. Estos mensajes son enviados a los servidores de registro para que almacenen su ubicación.

`BYE`: es usado para terminar una sesión de medios. Este mensaje solamente es enviado por alguno de los participantes de la sesión.

`CANCEL`: cancela las solicitudes pendientes. Puede ser enviado tanto por UA como por servidores *proxy*. Por ejemplo, los UA utilizan este método para cancelar un intento de llamada iniciado previamente y aún no respondido.

`ACK`: se utiliza para el reconocimiento final de solicitudes `INVITE`. Para el establecimiento de la sesión se usa *handshake* de tres vías. El usuario que es llamado retransmite periódicamente una respuesta final positiva hasta que recibe un `ACK`. Este mensaje puede contener en su cuerpo la descripción SDP si la misma no estaba contenida en el `INVITE`. Si esto no ocurre el cuerpo del mensaje `ACK` debe ir vacío.

`OPTIONS`: permite consultar y adquirir las capacidades de los UA y de los servidores. La respuesta lista los métodos SIP soportados. Un servidor *proxy* nunca genera una solicitud `OPTIONS`.

Además de estas solicitudes, se han agregado otras definidas en recomendaciones particulares. Entre ellas se encuentran las solicitudes de estado de presencia y de notificación de eventos `SUSCRIBE`, `NOTIFY` y de mensajería instantánea, `MESSAGE`.

Respuestas SIP

El formato de la "línea de inicio" para las respuestas es el siguiente:

```
<Version SIP> <Codigo de respuesta> <Frase razon>
```

- ***Versión SIP***: indica la versión de SIP que se está usando.
- ***Código de respuesta***: es un entero de tres dígitos que indica el resultado del procesamiento de la respuesta. Estos códigos son compatibles con algunas respuestas HTTP/1.1 e incluyen además extensiones.
- ***Frase razón***: es una descripción de texto del código de respuesta.

En el estándar de SIP se describen seies grandes categorías de códigos de respuesta:

`1xx` - **Provisional:** se conocen también como respuestas informativas. Se utilizan para indicar que la solicitud fue recibida y se está procesando. El servidor envía estas respuestas si considera que el tiempo procesamiento restante es mayor a 200 ms para obtener una respuesta final. El que envió la solicitud deja de enviarla una vez recibida esta respuesta. Típicamente los sevidores *proxy* envían una respuesta de código `100 Trying` (Intentando)

para indicar que están procesando el `INVITE`. Los UA envían respuestas con el código `180 Ringing` (Llamando) para indicar que el teléfono del usuario que se está llamando está timbrando.

`2xx` - **Éxito:** la solicitud fue recibida con éxito, comprendida y procesada. Típicamente se utiliza el mensaje `200 OK`.

`3xx` - **Redirección:** es necesario que el cliente realice otras tareas para completar el procesamiento de la solicitud. Generalmente este tipo de respuestas son enviadas por servidores de redirección tras un `INVITE`, indicando la nueva ubicación del contacto. Puede ser la ubicación de un *proxy* o directamente la del usuario.

`4xx` - **Error en el cliente:** la solicitud tiene algún error de sintáxis o el servidor en concreto no puede procesarla pudiendo ser válida para otro servidor.

`5xx` - **Error en el servidor:** a pesar de que la solicitud es en apariencia válida el servidor en cuestión no puede procesarla.

`6xx` - **Falla global:** la solicitud no se puede completar en ningún servidor.

Encabezados

Los encabezados especifican el usuario llamante, el usuario llamado, la ruta y el tipo de mensaje. Todos los encabezados tienen el siguiente formato:

```
<nombre del campo>:<valor>
```

Los encabezados se clasifican de acuerdo a si aparecen en solicitudes y/o respuestas.

- *Encabezados de solicitud y respuesta*: son comunes a ambos mensajes. Entre este tipo de encabezados se encuentra `Call-ID`, `Contact`, `CSeq`, `From`, `Record-Route`, `Subject`, `To` y `Via`.
- *Encabezados de solicitud*: son los campos que se encuentran únicamente en una solicitud. Entre estos encabezados se encuentra `Accept`, `Authorization`, `Max-Forwards`, `Refer-To` y `Route`.
- *Encabezados de respuesta*: se encuentran presentes solo en respuestas. Por ejemplo `Proxy-Authenticate` y `Warning`.
- *Encabezados de cuerpo de mensaje*: proveen información del cuerpo del mensaje. Por ejemplo `Allow`, `Content-Length`, `Content-Language` y `Content-Type`.

A continuación se describen algunos de los campos antes mencionados. Todos los nombres de los campos tienen una versión compacta, que es una abreviación del nombre, utilizada para mensajes que puedan superar el MTU.

`From:` indica el originador de la solicitud. Se puede incluir un nombre para ser desplegado. La forma compacta de este campo es `f`.
Por ejemplo: `‘‘Alice’’ <sip:alice@fing.edu.uy>; tag=a23s`

`To:` especifica el destinatario de la solicitud. Se puede incluir un nombre para ser desplegado. La etiqueta *tag* se usa para identificar el diálogo. La versión compacta de este campo es `t`. Ejemplo: `To: Bob <sip:bob@fing.edu.uy>; tag=550`.

`Call-ID:` identifica de forma única a una llamada. Está formado por un *string* aleatorio y el nombre o la dirección IP del llamante. La forma compacta de identificar a este campo es mediante `i`.
Por ejemplo: `Call-ID: 3kd45yt-i91qw60-jicskl@sitio.com`.

`CSeq:` contiene un número de secuencia de 32 bits entero sin signo y el tipo de solicitud. Sirve para ordenar transacciones dentro de un diálogo, provee medios para identificar en forma única una transacción y para diferenciar nuevas solicitudes de retransmisiones. Por ejemplo: `CSeq: 1503 INVITE`.

`Contact:` provee una URI cuyo significado depende del tipo de mensaje. Puede contener un nombre para desplegar. Para el caso de un `INVITE` especifica la ruta para contactar al usuario que envió la solicitud. La forma compacta es `m`.

`Content-Type:` indica el tipo de medios del cuerpo del mensaje. Este campo debe ir presente si el mensaje no tiene el cuerpo vacío. La forma compacta de este campo es `c`.

`Content-Length:` indica el tamaño del cuerpo del mensaje en bytes. Su forma compacta es `l`.

`Max-Forwards:` se usa para limitar la cantidad de veces que la solicitud puede ser reenviado por servidores *proxy* o *gateways*. Es un entero entre 0 y 255 que indica la cantidad restante de reenvíos permitidos y es decrementada por cada servidor que reenvía la solicitud. El valor inicial recomendado es 70. Cuando un elemento no es capaz de detectar bucles inserta este campo para limitar.

`User-Agent:` contiene información del UAC que originó la llamada.
Por ejemplo: `User-Agent: sipXvideoPhone`.

`Allow:` lista los métodos soportados por el UA que generó el mensaje. Ejemplo: `Allow: INVITE, ACK, OPTIONS, CANCEL, BYE`.

`Via:` indica el camino que ha tomado la solicitud y el camino que deben tomar las respuestas al mensaje. El parámetro `branch` sirve para identificar la transacción y es usado para la detección de bucles. Dentro de los parámetros del campo `Via` se pueden incluir el protocolo de transporte usado, el nombre del cliente o su dirección IP y el puerto en el que le gustaría recibir las respuestas. El parámetro `branch` debe comenzar con el valor “z9hG4bK”. La forma compacta de este campo es `v`.
Ejemplo: `Via: SIP/2.0/UDP 192.168.1.100:5060; branch=z9hG4bK87asdks7`.

2.3.5. Ejemplo de una llamada SIP

Se presenta a modo de ejemplo un intercambio típico de mensajes SIP para el establecimiento y fin de una llamada. Como se puede ver en la figura 2.3, Alice y Bob se desean comunicar entre sí. Se supone que ambos están conectados a una red IP y conocen la dirección del otro.

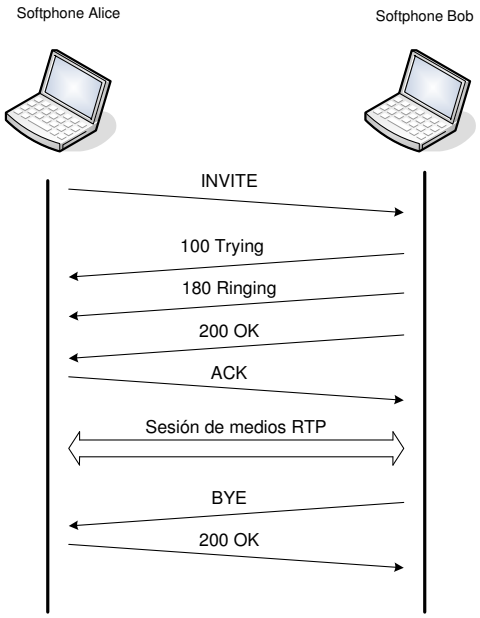


Figura 2.3: Intercambio de mensajes SIP en una llamada

Alice inicia la sesión enviando una solicitud `INVITE` a Bob, conteniendo los detalles del tipo de sesión que quiere establecer. En ese mensaje se incluyen los siguientes campos:

```
INVITE sip:bob@ieee.com SIP/2.0
Via: SIP/2.0/UDP iie.fing.edu.uy:5060;branch=z9hG4bKfw19b
Max-Forwards: 70
To: Bob <sip:bob@ieee.com>
From: Alice <sip:alice@fing.edu.uy>;tag=76341
Call-ID: 123456789@iie.fing.edu.uy
CSeq: 1 INVITE
Subject: About sipXvideoPhone...
Contact: <sip:alice@fing.edu.uy>
Content-Type: application/sdp
Content-Length: 158

v=0
o=Alice 2890844526 2890844526 IN IP4 iie.fing.edu.uy
```

```
s=Phone Call
c=IN IP4 100.101.102.103
t=0 0
m=audio 49170 RTP/AVP 0
a=rtpmap: 0 PCMU/8000
```

Como se puede observar, en la primera línea del mensaje `INVITE` se indica la dirección URI del destinatario (Bob), y la versión del protocolo SIP.

`Via` contiene la dirección URI donde Alice espera recibir las respuestas a su solicitud. También contiene el parámetro `branch` que identifica la transacción. Respuestas relacionadas con esta solicitud contienen el mismo identificador.

El campo `To` especifica el nombre y la SIP URI del destinatario de la solicitud. El identificador del nombre no es usado por el protocolo sino, por ejemplo, para ser desplegado mientras que el teléfono está timbrando en la punta remota.

El campo `From` es análogo al campo `To`. Contiene el nombre y la dirección SIP URI del originador de la solicitud. Este campo también contiene un parámetro `Tag` con un *string* aleatorio que se agrega a la URI y es utilizado con propósitos de identificación del diálogo.

El campo `Call-ID` contiene un identificador único para la llamada. Está formado por un *string* aleatorio y el nombre o la dirección IP del llamante.

La combinación de `To tag`, `From tag`, y `Call-ID` definen completamente una sesión *peer-to-peer* entre Alice y Bob, a la cual se le denomina diálogo como se explicó en 2.3.2.

`CSeq` como se explicó anteriormente, es un número entero que es incrementado con cada nueva solicitud dentro de un diálogo.

`Contact` contiene una SIP URI representando una ruta directa para contactar Alice. Generalmente está compuesta de un nombre de usuario en un dominio o una dirección IP. Cabe mencionar la diferencia entre la URI especificada en este campo y la especificada en el campo `Via`. La URI que aparece en el campo `Via` muestra la dirección a donde mandar la respuesta a la presente solicitud. Por otra parte, la URI especificada en el campo `Contact` muestra donde es posible enviar futuras solicitudes a Alice.

El campo `Max-Forwards` sirve para limitar el número de saltos que una solicitud puede tener en el camino a destino y vale 70 que es el valor incial recomendado en la RFC 3261.

`Content-Type` y `Content-Length` describen el contenido del cuerpo del mensaje. En este ejemplo, el cuerpo del mensaje contiene una descripción SDP con 158 bytes de datos.

Los detalles de la sesión como el medio a utilizar, el codec, la frecuencia de muestreo, no se describen usando SIP sino que se usa SDP contenido en SIP.

Luego de recibir el mensaje `INVITE` Bob envía un mensaje *100 Trying* para indicar que se está procesando la solicitud.

Seguido a este mensaje, envía un mensaje *180 Ringing* que indica que ya está en condiciones de ser establecida la sesión.

El mensaje `180 Ringing` contiene los siguientes campos:

```
SIP/2.0 180 Ringing
Via: SIP/2.0/UDP iie.fing.edu.uy:5060;branch=z9hG4bKfw19b;
     received=100.101.102.103
To: Bob <sip:bob@ieee.com>;tag=a53e42
From: Alice <sip:alice@fing.edu.uy>;tag=76341
Call-ID: 123456789@iie.fing.edu.uy
CSeq: 1 INVITE
Contact: <sip:bob@ieee.com>
Content-Length: 0
```

Este mensaje de respuesta se forma copiando muchos de los campos del `INVITE`. Tiene además la línea de comienzo con la versión de SIP y el número que identifica el tipo de respuesta.

El encabezado `Via` contiene el parámetro `branch` igual al mensaje `INVITE`, pero tiene un parámetro adicional `received` en el cual se especifica la dirección IP de origen de la solicitud. En general esta dirección coincide con la URI en el campo `Via`, luego de ser resuelta por DNS.

Cabe notar que los campos `To` y `From` se mantienen para la respuesta debido a que en SIP los campos son definidos para indicar la dirección de la solicitud y no de cada mensaje en particular.

El campo `To` ahora contiene un *tag* generado por Bob. Todas las solicitudes y respuestas futuras en este diálogo contendrán el *tag* generado por Alice y el *tag* generado por Bob.

Cuando Bob acepta la llamada se envía un mensaje de respuesta `200 OK`. Este mensaje contiene los siguientes campos:

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP iie.fing.edu.uy:5060;branch=z9hG4bKfw19b;
received 100.101.102.103
To: Bob <sip:bob@ieee.com>;tag=a53e42
From: Alice <sip:alice@fing.edu.uy>;tag=76341
```

```
Call-ID: 123456789@iie.fing.edu.uy
CSeq: 1 INVITE
Contact: <sip:bob@ieee.com>
Content-Type: application/sdp
Content-Length: 155

v=0
o=Bob 2890844528 2890844528 IN IP4 ieee.com
s=Phone Call
c=IN IP4 200.201.202.203
t=0 0
m=audio 5000 RTP/AVP 0
a=rtpmap: 0 PCMU/8000
```

Los medios para la sesión aceptados por Bob se envían mediante SDP en este mensaje. El mensaje SDP contiene la dirección IP de destino y el tipo de medios que para este caso es audio PCM leyμ a una frecuencia de 8000 Hz, por el puerto 5000.

Para que la sesión quede completamente establecida se debe confirmar la solicitud mediante un mensaje `ACK` enviado por Alice. El mensaje `ACK` contiene los siguientes campos:

```
ACK sip:bob@ieee.com SIP/2.0
Via: SIP/2.0/UDP iie.fing.edu.uy:5060;branch=z9hG4bK321g
Max-Forwards: 70
To: Bob <sip:bob@ieee.com>;tag=a53e42
From: Alice <sip:alice@fing.edu.uy>;tag=76341
Call-ID: 123456789@iie.fing.edu.uy
CSeq: 1 ACK
Content-Length: 0
```

El comando `CSeq` tiene el mismo número que el `INVITE` pero se cambia el tipo de método a `ACK`. Una vez que llega la confirmación `ACK` la sesión de medios queda establecida y se usa otro protocolo como por ejemplo RTP, para la transmisión de los mismos.

Cuando se quiere finalizar la sesión, Bob envía una solicitud de `BYE` a Alice.

```
BYE sip:alice@iie.fing.edu.uy SIP/2.0
Via: SIP/2.0/UDP ieee.com:5060;branch=z9hG4bK392kf
Max-Forwards: 70
To: Alice <sip:alice@iie.fing.edu.uy>;tag=76341
From: Bob <sip:bob@ieee.com>;tag=a53e42
Call-ID: 123456789@iie.fing.edu.uy
CSeq: 1 BYE
Content-Length: 0
```

Finalmente, Alice responde al mensaje `BYE` mediante un `200 OK`:

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP ieee.com:5060;branch= z9hG4bK392kf;received 200.201.202.203
To: Alice <sip:alice@fing.edu.uy>;tag=76341
From: Bob <sip:bob@ieee.com>;tag=a53e42
Call-ID: 123456789@iie.fing.edu.uy
CSeq: 1 BYE
Content-Length: 0
```

2.4. Session Description Protocol: SDP

SDP fue desarrollado por el grupo de trabajo MMUSIC de la IETF y está descrito en la RFC 4566. Es un protocolo utilizado para la descripción de sesiones multimedia. No se encarga de la negociación ni el transporte de la sesión sino que se remite únicamente a representar de forma estandarizada información descriptiva de la misma.

Es un protocolo de propósito general que puede usarse en combinación con protocolos de señalización y anuncio de sesiones, como *Session Initiation Protocol* SIP y *Session Announcement Protocol* SAP, y con protocolos de streaming en tiempo real como RTSP. La RFC 3264 [12] describe el uso de SDP para SIP.

2.4.1. Descripción SDP

El objetivo de SDP es comunicar la existencia de una sesión multimedia y brindar información de la misma entre sus participantes.

La descripción SDP incluye la siguiente información sobre la sesión de medios:

- Nombre y motivo de la sesión.
- Tiempo que lleva activa la sesión.
- Información de contacto de la sesión (direcciones, puertos, formatos).
- Protocolo de transporte.
- Dirección IP (IPv4 o nombre del host).
- Puerto utilizado (UDP o TCP).
- Tipo de medios (audio, video, etc.).
- Formato de los medios (G.729, MPEG video, etc.).

También se transmite información del ancho de banda necesario para la sesión e información de contacto del creador de la sesión.

2.4.2. Especificación de SDP

Los mensajes SDP están formados por líneas, denominadas campos, en las cuales se transmite la información asociada con la sesión de medios. Cada uno de estos campos se representa con una letra minúscula que resume su nombre, y deben enviarse en un orden específico. En el cuadro 2.2 se muestran los distintos campos de SDP, su descripción y su nivel de requerimiento, es decir si son obligatorios u opcionales.

El formato de cada campo SDP es `<tipo>=<valor>`, donde `<tipo>` es el identificador del campo (exactamente un caracter) y `<valor>` es texto estructurado. No se deben usar espacios en ninguno de los dos lados del `=`.

Para describir una sesión de medios, SDP utiliza dos niveles:

1. Por un lado está la información relativa a la sesión en sí misma donde se especifica por ejemplo el nombre, motivo, información de contacto. Esta información se transmite al principio de los mensajes SDP, comenzando por el campo `v=`.
2. Por otra parte se encuentra la información propia de cada tipo de medios que compone la sesión. Dado que en una misma sesión pueden coexistir varios medios, estos bloques de información pueden reiterarse en un mismo mensaje SDP cada uno de los cuales comienza con el campo `m=`.

A continuación se presentan los campos más utilizados que componen los mensajes SDP:

Versión del protocolo **(`v=`):** representa la versión utilizada del protocolo. Actualmente se encuentra en la versión 0 y no existen anteriores.

Origen **(`o=`):** especifica el originador de la sesión e identificadores que la describen de forma única. A tales efectos presenta los siguientes atributos:

```
o=<username> <sess-id> <sess-version> <nettype> <addrtype> <unicast-address>
```

- `<username>`: identificador del creador. Si el servidor no soporta el uso de `ID` se rellena con `-`.
- `<sess-id>`: es un número que identifica a la sesión de forma única. Una forma de implementarlo es usar una marca de tiempo NTP, *Network Time Protocol*.
- `<sess-version>`: es la versión de cada descripción y se incrementa cada vez que se realiza una modificación de la sesión. También se recomienda utilizar una marca de tiempo NTP.
- `<nettype>`: es un string que identifica el tipo de red utilizado. Un ejemplo `IN` es para Internet.
- `<addrtype>`: identifica el tipo de direcciones que utiliza la red. Ejemplos pueden ser `IP4` o `IP6`.
- `<unicast-address>`: es la dirección de la máquina que inició la sesión.

Campo	**Descripción**	**Nivel de requerimiento**
Descripción de la sesión		
v	Versión del protocolo	Obligatorio
o	Origen (creador) e identificador de la sesión	Obligatorio
s	Nombre de la sesión	Obligatorio
i	Información de la sesión	Opcional
u	URI de la descripción	Opcional
e	Dirección de e-mail	Opcional
p	Número de teléfono	Opcional
c	Información de la conexión	Opcional
b	Una o más líneas de información de ancho de banda	Opcional si se incluye en todos los medios
z	Ajustes de zona horaria	Opcional
k	Llave de encriptación	Opcional
a	Una o más líneas de atributos de la sesión	Opcional
Descripción de tiempo		
t	Tiempo de inicio y fin de la sesión	Obligatorio
r	Tiempos de repetición	Opcional
Descripción de medios		
m	Nombre de los medios y dirección de transporte	Obligatorio
i	Título de los medios	Opcional
c	Información de la conexión	Opcional si se incluye a nivel de sesión
b	Información de ancho de banda	Opcional
k	Llave de encripción	Opcional
a	Atributos de los medios	Opcional

Cuadro 2.2: Campos de SDP

Nombre de la sesión (`s=`)**:** es identificatorio de la sesión y debe haber un único nombre por sesión.

Información de la sesión (`i=`)**:** debe haber solamente uno por sesión y como máximo uno para describir cada tipo de medios.

URI (`u=`)**:** URI es la abreviación de *Uniform Resource Identifier* y es usado por clientes www.

Dirección de e-mail y teléfono (`e=`, `p=`)**:** contiene información de contacto de la persona responsable de la sesión.

Datos de la conexión (`c=`)**:** estos se transmiten de la forma que sigue:

```
c=<nettype> <addrtype> <connection-address>
```

- `<nettype>`: tipo de red, para Internet se usa `IN`.
- `<addrtype>`: IP4 o IP6.
- `<connection-address>`: es la dirección que envía los paquetes de medios y puede ser tanto *multicast* como *unicast*.

Ancho de banda (`b=`)**:** se transmite en el siguiente formato:

```
b=<bwtype>:<bandwidth>
```

- `<bwtype>`: es un modificador que puede tomar dos valores: CT que significa *conference total* y AS que significa *application specific*. CT se usa para sesiones *multicast* para especificar el ancho de banda total que pueden usar todos los participantes. AS se usa para especifiar el ancho de banda de un único sitio.
- `<bandwidth>`: especifica el valor del ancho de banda en kBytes por segundo.

Tiempo (`t=`)**:** Indica el principio y fin de la sesión expresado en unidades NTP. Si `<stop-time>` vale cero indica que la sesión sigue de forma indefinida. Si además `<start-time>` vale cero indica que la sesión es permanente.

```
t=<start-time> <stop-time>
```

Tiempo de repetición (`r=`)**:** puede expresarse en unidades NTP o en días, horas, minutos y segundos. El formato en que se transmite es el siguiente:

```
r=<repeat interval> <active duration> <offsets from start-time>
```

Zona horaria (`z=`)**:** Cuando se anuncia el tiempo de repetición también se deben anunciar los cambios que se pueden producir en el horario. Por ejemplo cuando se cambia el horario por ahorro de energía como sucede en verano. El formato en que se envía esta información es:

```
z=<adjustment time> <offset> <adjustment time> <offset> ....
```

Clave de encriptación **(`k=`):** Si la descripción de la sesión se transporta sobre un canal seguro, SDP puede transmitir las claves de encriptación.

```
k=<method>:<encryption key>
```

Atributos **(`a=`):** Este campo describe las características propias de cada tipo de medios disponible en la sesión. A nivel de medios, puede definirse cualquier cantidad de atributos. Si un usuario SDP no comprende alguno de los atributos, debe descartarlo.

```
a=<attribute>:<value>
```

Algunos atributos SDP:

- `a=cat:<category>`: sirve para filtrar sesiones en base a la categoría.
- `a=keywds:<keywords>`: le permite al receptor seleccionar sesiones interesantes en base a palabras clave.
- `a=rtpmap:<payload type> <encoding name>/<clock rate> [/<encoding parameters>]`: este atributo mapea el tipo de carga útil del paquete RTP en un número, un nombre del formato y una frecuencia de reloj.
- `a=fmtp`: describe parámetros específicos del tipo de medios.
- `a=type:<conference type>`: especifica el tipo de conferencia, el cual puede ser *broadcast*, *meeting*, *moderated*, o *test*.
- `a=framerate:<frame rate>`: especifica la tasa máxima de cuadros por segundo.

A modo de ejemplo, para RTP/AVP (Audio Video Profile) se utilizan tres atributos:

```
a=rtpmap:0 PCMU/8000
a=rtpmap:18 G729/8000
```

Descripción de medios **(`m=`):** Contiene información adicional de la sesión de medios.

```
m=<media> <port> <proto> <fmt> ...
```

- `<media>`: indica el tipo de medios. Hasta ahora están definidos: audio, video, aplicación, texto, mensajes.
- `<port>`: indica el puerto de transporte a donde se manda la información.
- `<proto>`: es el protocolo de transporte. Por ejemplo puede ser UDP, RTP/AVP (Audio Video Profile) corriendo sobre UDP, RTP/SAVP (Secure Audio Video Profile) corriendo sobre UDP.
- `<fmt>`: es una descripción del formato de los medios.

Continuando con el ejemplo, para audio en formato G729 y PCM se tiene la siguiente descripción:

```
m= audio 8000 RTP/AVP 0 8 18
```

donde el 0 se corresponde con PCMU, el 8 con PCMA y 18 con G729.

2.4.3. Utilización de SDP en SIP

Para la aplicación de SDP en SIP se usa la RFC 3264 en donde se presenta un modelo de SDP para comunicaciones del tipo Petición/Respuesta. En este modelo un participante le ofrece a otro la descripción de la sesión deseada desde su perspectiva y el otro le responde con la descripción de la sesión desde su perspectiva. Se usa en sesiones *unicast* donde la información de ambos participantes es necesaria para poder establecer la sesión.

El modelo Petición/Respuesta opera de la siguiente forma: un participante de la sesión envía un mensaje SDP, llamado oferta, donde lista el tipo de medios y codecs que desea utilizar. A su vez, el mensaje contiene la dirección IP y los puertos que destinará para la recepción de los datos. El otro participante recibe ese mensaje y a partir del mismo envía una respuesta SDP explicitando para cada uno de los medios propuestos si lo acepta o no. En este mensaje se detalla también la dirección IP y los puertos por los que desea recibir los datos una vez establecida la sesión.

Este intercambio de peticiones y respuestas asume que existe un protocolo de capa superior, como SIP, que se encarga del intercambio de los mensajes para llevar a cabo la negociación SDP. En este caso, el agente de usuario originador de la llamada envía las capacidades soportadas en un mensaje `INVITE` o `ACK`. El receptor por su parte genera un mensaje `200 OK` en respuesta al `INVITE`, donde pone de manifiesto sus capacidades soportadas.

En el caso de que el agente de usuario receptor posea distintas capacidades, esto se refleja en el campo `o` del mensaje `200 OK`. El valor `<sess-version>` de este campo es incrementado en una unidad indicando que la sesión aún no puede ser establecida.

Al utilizar SDP en conjunto con SIP, muchos de los campos de SDP no son necesarios pero para mantener compatibilidad deben incluirse. Un ejemplo típico se da al momento de incluir los campos versión, origen, nombre, tiempo, conexión, y los atributos de los medios. Algunos de estos campos no resultan necesarios pero igualmente se incluyen: origen, nombre y tiempo. Para el nombre o motivo de la sesión se rellena con `s=-`. El campo del tiempo se suele setear en `t=0`, indicando que el tiempo de duración de la sesión es ilimitado.

Se recomienda que el atributo `a=rtpmap` se utilice para cada tipo de medios. Este campo como se mencionó anteriormente indica el codec empleado mediante un número según se define en el perfil RTP/AVP de la RFC 3551.

Otro uso de SDP en SIP se da cuando se desea poner una llamada en espera (*hold*). En esta situación se envía un mensaje `INVITE` donde solo se encuentra presente el atributo `a=sendonly`. La llamada se vuelve activa nuevamente enviando un `INVITE` con la presencia solamente del atributo `a=sendrecv`.

2.5. Real-Time Transport Protocol: RTP

Este protocolo está descrito en la RFC 3550. Fue diseñado para permitir el transporte de datos en tiempo real extremo a extremo como audio y video sobre protocolos de transporte no seguros (asume la existencia de pérdidas, llegada en desorden y con retardo variable de paquetes). RTP no hace reserva de recursos ni provee calidad de servicio. Como complemento se usa el protocolo *Real-Time Transport Control Protocol*, RTCP, que brinda información de calidad.

Para situar al protocolo RTP en un modelo de capas se debería hacer referencia al modelo híbrido entre OSI y TCP/IP. Ubicarlo dentro del modelo OSI resulta difícil ya que el protocolo lleva a cabo varias tareas de la capa de transporte (sin ser completamente de transporte), realiza tareas de la capa de sesión (maneja la identificación de participantes) y de la de presentación (define representaciones estándar de medios). Dentro del modelo híbrido se lo puede ubicar al nivel de capa de aplicación [13].

Para transportarse sobre IP, RTP generalmente utiliza el protocolo UDP. También es posible usarlo sobre redes que no son IP como ATM (*Asynchronous Transfer Mode*). La figura 2.4 muestra RTP sobre IP/UDP.

RTP permite detectar:

- Pérdida de paquetes.
- Retardos variables en el transporte.
- Llegada de paquetes fuera de secuencia.

Entre los servicios que provee RTP se incluye un número de secuencia para la detección de pérdida de paquetes, identificación de la fuente y del tipo de datos, una marca de tiempo que permite reconstruir el tiempo de la fuente.

La especificación de RTP es muy general e intenta describir los aspectos comunes a las distintas aplicaciones que lo utilizan. Por esta razón además de esta, existen especificación para aplicaciones particulares. Hay perfiles que describen un conjunto de codecs para distintos medios y se los mapea a un tipo de formato de los que lleva RTP. Por ejemplo, para el caso de audio y video el perfil es RTP/AVP (*Audio Video Profile*) y está descrito en la RFC 3551.

2.5.1. Conceptos preliminares

A continuación se presentan conceptos que será de utilidad a lo largo de esta sección.

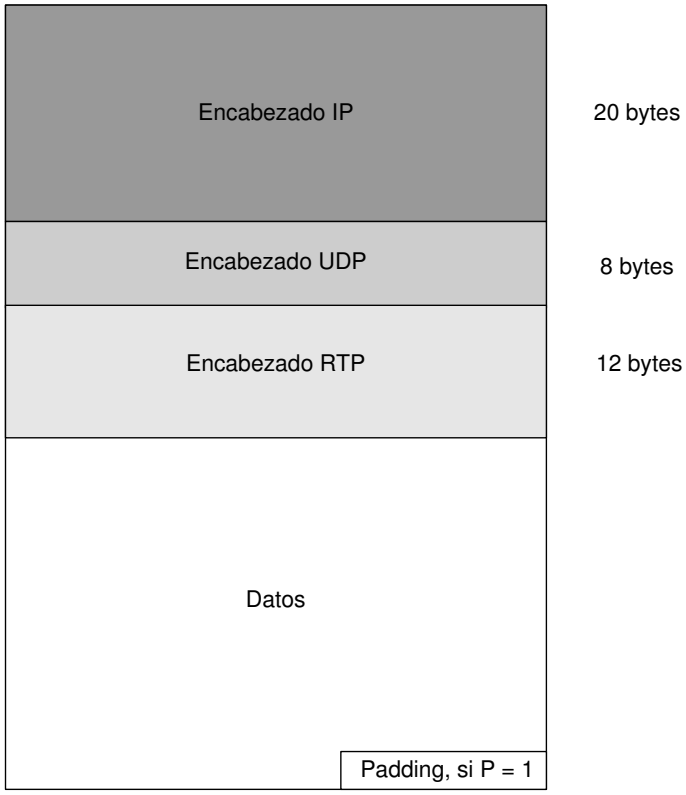


Figura 2.4: RTP sobre IP/UDP

***Paquete RTP*:** está formado por un encabezado fijo, una lista de fuentes que contribuyen y la carga útil. El encabezado permite reconstruir la información en el receptor.

***Sesión RTP*:** es un grupo de participantes que se comunican utilizando RTP, donde un participante puede pertenecer a varias sesiones. Cada tipo de medios debe ir en una sesión RTP separada. Cada sesión queda definida por un par de direcciones de transporte. Una dirección de transporte es una dirección de red más dos puertos, uno para RTP y otro para RTCP.

***Fuente de sincronización (SSRC)*:** representa a una fuente de un stream RTP que se identifica por 32 bits. Es un valor elegido aleatoriamente que debe identificar de forma globalmente única a una fuente dentro de una sesión RTP. Los paquetes que provienen de una misma fuente de sincronización tienen marcas de tiempo y números de secuencia comparables. Un participante no usa el mismo identificador para distintas sesiones RTP dentro de una sesión multimedia. La conexión entre sesiones RTP de una misma fuentes se hace mediante RTCP (con el nombre canónico).

***Fuente contribuyente (CSRC)*:** una fuente de un *stream* RTP que contribuye en uno combinado a la salida de un mezclador. El mezclador inserta una lista de identificadores SSRC

de las fuentes contribuyentes. Por ejemplo en una conferencia de audio el mezclador indica a todos los participantes de la conversación la lista con los SSRC que contribuyen.

***Mezclador*:** es un sistema intermedio que recibe paquetes RTP de una o más fuentes, puede cambiar el formato de los datos, combina los paquetes y los envía como un nuevo paquete RTP. Como las marcas de tiempo entre fuentes no están sincronizadas, el mezclador hace ciertos ajustes entre los *streams* y genera su propia marca de tiempo. Por lo tanto todos los paquetes provenientes de un mezclador tienen al mezclador como fuente de sincronización (SSRC).

***Traductor*:** es un sistema intermedio que reenvía paquetes RTP con su identificador de fuente de sincronización sin cambiar. Son dispositivos que convierten tipos de codificación sin mezclar y actúan en caso de que haya firewalls de por medio.

2.5.2. Aplicaciones

Conferencia de audio y video

Para conferencias de audio y video cada medio es transportado en una sesión RTP diferente. El único vínculo existente entre los medios de un mismo participante es el nombre canónico en los paquetes RTCP. A su vez para cada uno de los dos medios se utilizan dos puertos; uno para la sesión RTP y otro para la RTCP como se describirá más delante.

Uno de los motivos por los cuales se separan los medios en distintas sesiones RTP es para permitirle a cada participante recibir solo audio o video. De esta forma si un participante no quiere/puede recibir video, igualmente se le puede enviar el audio. Esto posibilita la elección de medios que se desean recibir. A pesar de esto es posible sincronizar ambos medios utilizando RTCP. El mecanismo empleado se describe en la sección 6.4.

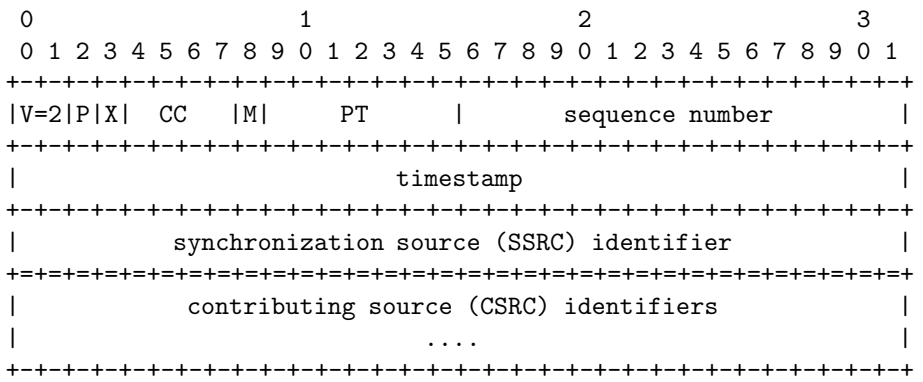
Traductores y mezcladores

En las conferencias puede pasar que no todos los participantes tengan el mismo tipo de conexión. Para el caso en que algunos participantes tengan una conexión de baja velocidad en vez de degradar la calidad de la conferencia para todos los participantes, lo que se hace es usar un mezclador cerca de los participantes con menor ancho de banda. Este mezclador resincroniza los paquetes que llegan, traduciéndolos a una codificación de menor ancho de banda y entregándolos a los usuarios de conexión lenta.

Para el caso en el que haya un firewall que no permita la entrada de paquetes IP no se utilizans mezcladores sino traductores, colocándose uno a cada lado del firewall.

2.5.3. Encabezado RTP

El encabezado RTP tiene el formato que se muestra a continuación . Los primeros 12 bytes se encuentran en todos los paquetes mientras que la lista de identificadores CSRC solo está presente si hay mezcladores.



Los campos que forman esta estructura son:

- **Versión** (`V`): 2 bits. Especifica la versión del protocolo RTP. La versión actual es 2.
- **Padding** (`P`): 1 bit. Si este bit está en uno, el paquete contiene uno o más bytes de padding al final. El último byte del padding indica la cantidad de estos presentes en el paquete incluyéndose a sí mismo.
- **Extensión** (`X`): 1 bit. Si está en uno el encabezado RTP debe estar seguido de un encabezado de extensión.
- **Cuenta CSRC** (`CC`): 4 bits. Cantidad de identificadores CSRC que hay después del encabezado fijo.
- **Marcador** (`M`): 1 bit. La interpretación de este bit está definida en cada perfil.
- **Tipo de carga** (`PT`): 7 bits. Este campo identifica el tipo de medios del paquete RTP. En la RFC 3551 se asigna un número que mapea los codecs de audio y video con el tipo de carga.
- **Número de secuencia** (`sequence number`): 16 bits. Este número se incrementa en uno por cada paquete RTP enviado y el receptor lo usa para detectar pérdidas de paquetes y para recuperar el orden. El valor inicial debe ser aleatorio para ser más robusto ante ataques sobre el texto plano. En la RFC 3550 se describen métodos para elegir este valor inicial.
- **Marca de tiempo** (`timestamp`): 32 bits. Representa el instante de muestreo del primer byte del paquete RTP. El reloj usado debe incrementarse monótonamente y linealmente para permitir sincronización y cálculo del *jitter*. La frecuencia del reloj depende del tipo de datos de la carga útil y se especifica en cada perfil.

  Al igual que para el número de secuencia el valor inicial de la marca de tiempo debe ser aleatorio. Varios paquetes RTP consecutivos pueden tener la misma marca de tiempo si fueron generados al mismo tiempo como puede ser varios paquetes pertenecientes a un mismo cuadro de video.

No es posible comparar marcas de tiempo de *streams* diferentes ya que éstas puede que no se incrementen de igual forma. Lo que se usa para la sincronización es el mapeo de las marcas de tiempo con un reloj de referencia llamado reloj de pared (*wallclock*) que viene dado por los paquetes RTCP *Sender Report* que se describen más adelante.

- **Identificador de fuente de sincronización** (SSRC): 32 bits. Identifica de forma única a la fuente que envía los paquetes RTP dentro de la misma sesión. Puede producirse una colisión entre el identificador de dos participantes. A pesar de que esta probabilidad es baja todas las implementaciones de RTP deben tener mecansimos para resolver este problema.

- **Lista de fuentes contribuyentes** (CSRC): de 0 a 15 ítems de 32 bits cada uno. Identifica las fuentes que contribuyen en la carga útil del paquete. La cantidad de identificadores se indica en el campo CC. Si hay más de 15 fuentes que contribuyen solamente se incluyen 15. Estas fuentes son insertadas por los mezcladores.

RTP permite detección de pérdida de paquetes y llegada en orden incorrecto mediante el número de secuencia. No trabaja sobre la corrección de estos errores. Las variaciones en los retardos (*jitter*) se puede detectar mediante la marca de tiempo.

2.5.4. RTP sobre UDP

En la RFC 3550 se describe cómo se usa RTP sobre distintos protocolos de transporte y de red. Cuando se usa sobre UDP, RTP debe asociarse a un puerto par y el protocolo de control RTCP debe asociarse al siguiente puerto mayor impar (puertos consecutivos). En las aplicaciones en las que se da un único puerto como parámetro se pueden deducir ambos puertos RTP y RTCP. Si el parámetro es impar, entonces para RTP se usa el siguiente puerto menor par como base para el par de puertos. También es posible usar dos puertos cualesquiera aunque se recomienda hacer la elección de acuerdo con el procedimiento anterior.

2.6. Real-Time Transport Control Protocol: RTCP

Al igual que RTP se describe en la RFC 3550. Este protocolo se basa en la transmisión periódica de paquetes de control a todos los participantes de la sesión. Desempeña las siguientes funciones:

- Provee información de la calidad de la distribución de datos.

- Lleva un identificador persistente a nivel de transporte para una fuente RTP llamado CNAME (*canonical name*) de gran utilidad ya que el identificador SSRC puede cambiar si se descubre un problema o se reinicia un programa. El CNAME se usa para asociar varios *streams* de datos de un participante en un grupo de sesiones RTP relacionadas, como por ejemplo para sincronizar audio y video.

- La tasa de envío de paquetes RTCP debe ser controlada para que sea escalable a muchos participantes. Como cada participante envía paquetes de control a los otros, es posible estimar el número de participantes y así calcular la tasa de envío deseada.

- Dar la mínima información de control de sesión. Esta función es opcional.

2.6.1. Tipos de paquetes RTCP

Hay cinco tipos de mensajes RTCP:

- **Sender Report** (`SR`): para estadísticas de transmisión y recepción de los participantes activos (aquellos que generan y reciben tráfico). Son utilizados para sincronización.

- **Receiver Report** (`RR`): para recepción de estadísticas de participantes pasivos (sólo reciben tráfico).

- **Source Description Items** (`SDES`): mensajes de descripción de fuentes que permiten asociar el identificador con un nombre, un teléfono, un e-mail.

- **BYE**: Fin de la participación (no significa el fin de la sesión a menos que haya solamente dos participantes).

- **APP**: Intercambio de información entre aplicaciones que usan RTP.

Cada tipo de paquete tiene asociado un número que lo identifica y se muestran en el cuadro 2.3.

Tipo de paquete	**Identificador**
SR	200
RR	201
SDES	202
BYE	203
APP	204

Cuadro 2.3: Tipos de paquetes RTCP

Todos los paquetes RTCP tienen campos en común y contienen datos de largo variable pero deben ser múltiplo de 32 bits. Se pueden agrupar varios paquetes RTCP en un paquete compuesto y enviarse sobre un mismo paquete UDP (figura 2.5). Hay ciertas reglas sobre el orden en el que se deben mandar:

- Las estadísitcas de recepción, es decir los paquetes `SR` o `RR`, deben mandarse lo más frecuentemente posible dadas las restricciones de ancho de banda. Por lo tanto, cada paquete RTCP compuesto debe incluir al menos un paquete de reporte.

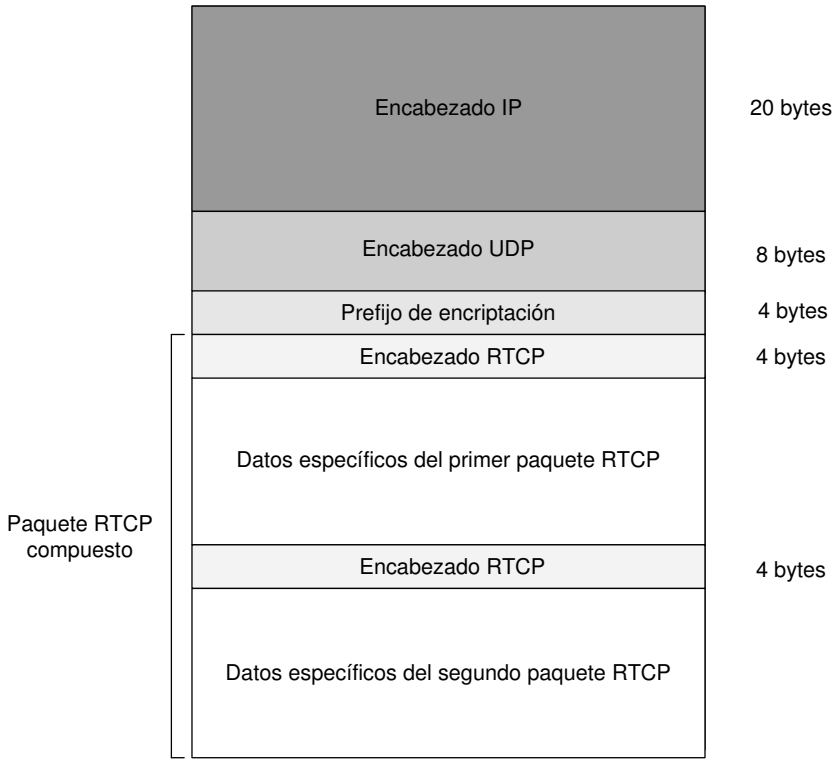


Figura 2.5: Paquete RTCP compuesto sobre IP/UDP

- Los nuevos participantes deben recibir el `CNAME` de una fuente lo antes posible para identificarla y asociar sus medios para poder sincronizarlos (*lip synchronization*) por lo que cada RTCP compuesto debe contener un paquete del tipo `SDES CNAME`.

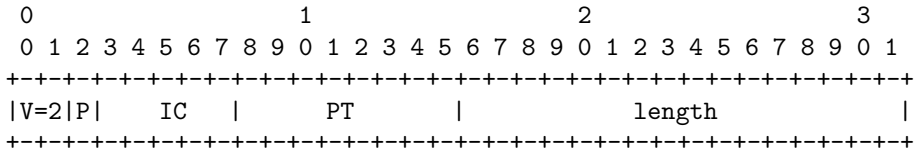
Teniendo en cuenta esto se desprende que un paquete RTCP compuesto debe tener al menos dos paquetes con el siguiente formato:

- Prefijo de encriptación: solo si el paquete compuesto está encriptado debe estar precedido por un número aleatorio de 32 bits.
- El primer paquete debe ser siempre un `SR` o un `RR`.
- Un paquete `SDES CNAME` debe ser incluido en todos los paquetes compuestos.
- El último paquete debe ser un `BYE` o `APP`.

Cada participante RTP envía únicamente un paquete compuesto RTCP por intervalo de reporte para que de esta forma sea posible estimar el ancho de banda por participante. Si hay más fuentes de las que entran en el MTU de la red, solo se envía en el reporte los que entran.

2.6.2. Encabezado RTCP

Se usa el nombre encabezado RTCP para denominar a la parte común a los distintos tipos de paquetes RTCP. Está formado por cuatro bytes como sigue:



- **Versión** (`V`): 2 bits. Representa la versión del protocolo y al igual que para RTP la actual es la 2.
- **Padding** (`P`): 1 bit. Si vale uno el paquete contiene bits de relleno. El último byte del padding indica la cantidad de bytes de relleno.
- **Cuenta de ítems** (`IC`): 5 bits. Indica la cantidad de ítems incluidos en el paquete. Dependiendo del tipo de paquete RTCP tiene otro nombre e indica ítems distintos.
- **Tipo de paquete** (`PT`): 8 bits. Indica el tipo de paquete de acuerdo con los cinco tipos descritos en la especificación.
- **Tamaño** (`length`): 16 bits. Indica el tamaño del paquete en unidades de 32 bits ya que todos los paquetes RTCP tienen un largo múltiplo de este valor.

2.6.3. Intervalo de transmisión RTCP

RTP le permite a una aplicación escalarse automáticamente desde unos pocos participantes a miles. Si todos los participantes enviaran reportes de recepción a una tasa constante, el tráfico de la sesión incrementaría linealmente con el número de participantes. Por este motivo resulta necesario calcular de manera dinámica un intervalo de tiempo en el cual sea apropiado transmitir los paquetes RTCP.

Para cada sesión hay un límite llamado *ancho de banda de la sesión* que se divide entre todos los participantes. El máximo ancho de banda puede estar limitado por la red o por un valor que se considere razonable. Todos los participantes deben disponer del mismo ancho de banda para que el intervalo RTCP pueda ser calculado.

Se recomienda que el ancho de banda agregado por RTCP sea del 5 por ciento del ancho de banda de la sesión. Sin embargo. algunos perfiles definen el ancho de banda de RTCP independiente al de la sesión. El intervalo de transmisión de paquetes RTCP también debe tener un límite inferior y para el caso en el que hay pocos participantes el valor recomendado es de 5 segundos.

El cálculo del intervalo de paquetes RTCP depende de una estimación del número de sitios que participan en la sesión. Cuando se reciben paquetes de nuevos sitios, se debería agregar una entrada en una tabla indexada por el identificador `SSRC` o `CSRC` para llevar el registro. Entradas nuevas se podrían considerar no válidas hasta que múltiples paquetes que lleven un nuevo `SSRC` hayan sido recibidos o hasta que un paquete `SDES` RTCP conteniendo el `CNAME` para ese `SSRC` se haya recibido.

Una vez recibido un paquete RTCP `BYE` para un cierto `SSRC`, se marcará la entrada como que se recibió un `BYE` de la misma y se borrará luego de un cierto tiempo. Esto se hace por si llega algún paquete válido luego del `BYE`. Un participante puede marcar a otro sitio inactivo o borrarlo si no se ha recibido ningún paquete valido o si ningún paquete RTP o RTCP se recibió para un número pequeño de intervalos RTCP (se recomiendan 5 intervalos). Para sesiones con un gran número de participantes, puede ser poco práctico mantener una tabla almacenando identificadores SSRC e información de estado de todos ellos. Es recomendable que una implementación use muestreo de `SSRC` para reducir los requerimientos de almacenamiento.

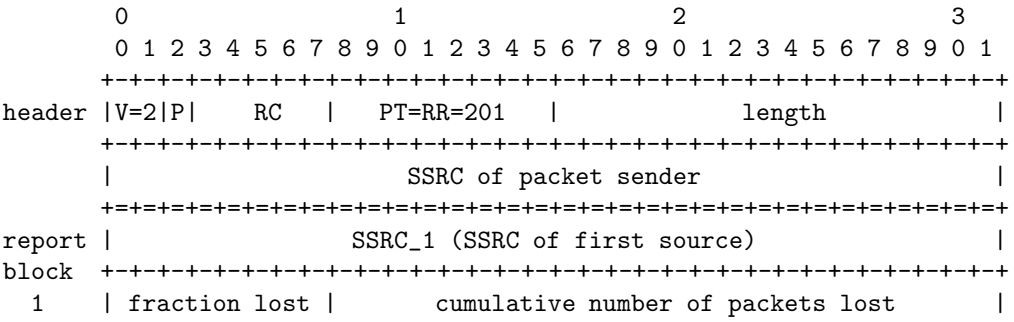
2.6.4. Reportes de envío y de transmisión

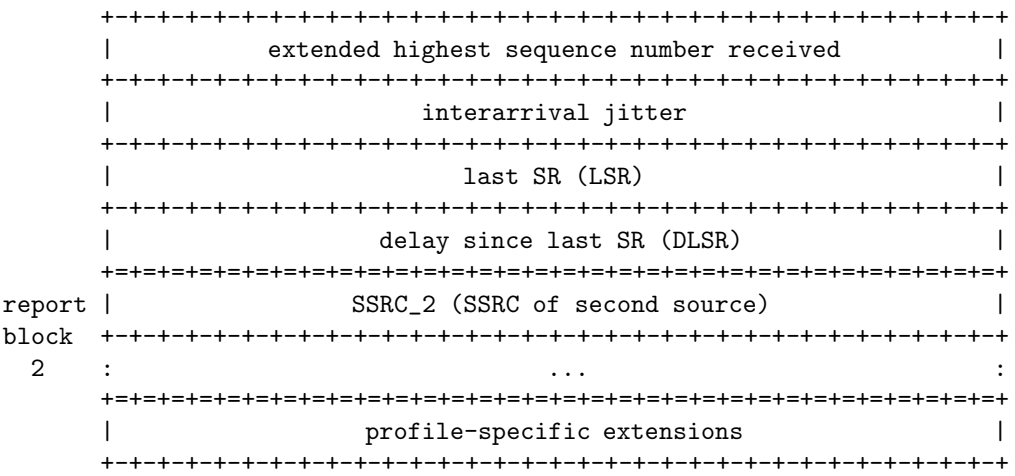
Los receptores RTP proveen información de la calidad de recepción usando reportes RTCP que pueden tomar dos formas dependiendo si el receptor también es transmisor. La única diferencia entre el `SR` (*sender report*) y el `RR` (*receiver report*), es que el primero incluye una sección de información del transmisor de 20 bytes para ser usada por transmisores activos.

Cada reporte recibido proporciona estadísticas de los datos recibidos por una fuente en particular. Como el número máximo de reportes que entran en un `SR` o `RR` es 31, se envían paquetes adicionales luego del inicial que contienen reportes de todas las fuentes escuchadas durante el intervalo a partir del último reporte. Si hay muchas fuentes de forma que se excede el MTU del camino de la red, se incluye únicamente un subconjunto que entre en cada intervalo.

RR: Receiver Report

Los reportes `RR` son enviados por los participantes que han recibido datos recientemente y se identifican con el número 201 (cuadro 2.3). En este paquete se indica la fuente que envía el reporte y la fuente sobre la cual está reportando. El campo `IC` del encabezado RTCP (2.6.2) se llama `RC` y representa la cantidad de reportes de recepción.





- **Indicador de fuente que reporta** (`SSRC of packet sender`): 32 bits. Es el identificador de fuente del originador del paquete.

- **Identificador de fuente reportada** (`SSRC_n`): 32 bits. Es el identificador de la fuente sobre la cual se está reportando.

- **Fracción de pérdida** (`fraction loss`): 8 bits. Número de paquetes perdidos en este intervalo de reporte, es decir desde el último `SR` o `RR` enviado.

- **Cantidad de paquetes perdidos** (`cumulative number of packet loss`): 24 bits. Cantidad de paquetes RTP perdidos desde el comienzo de la recepción. Se calcula como el número de paquetes esperados menos el número de paquetes recibidos. El número de paquetes esperados es la diferencia entre el número de secuencia actual y el número de secuencia inicial.

- **Mayor número de secuencia recibido** (`extended highest sequence number received`): 32 bits. Es el mayor número de secuencia de un paquete RTP recibido de `SSRC_n`.

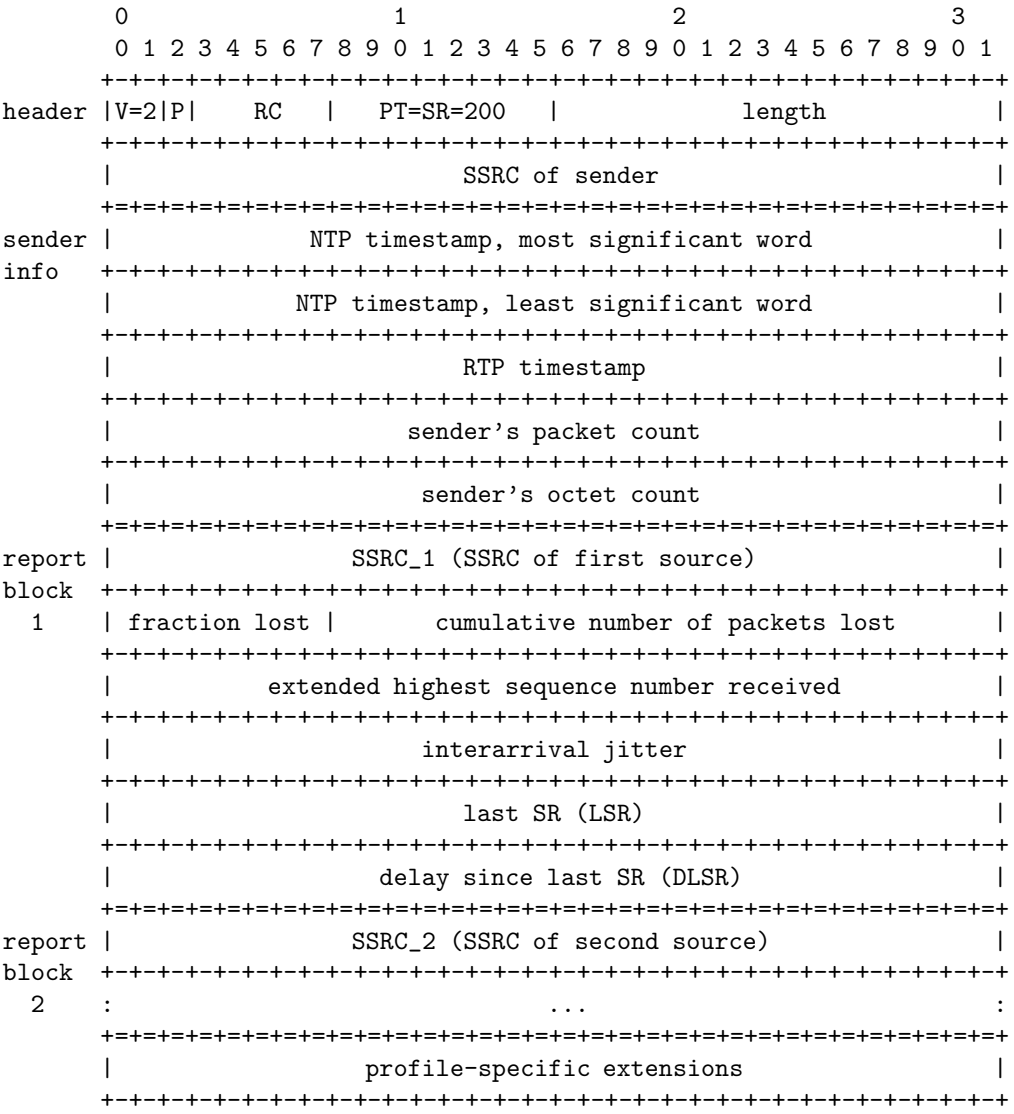
- **Fluctuaciones en el retardo** (`interarrival jitter`): 32 bits. Es una estimación de la variación en el tiempo de la llegada de paquetes RTP. Como unidad de medida se usa la marca de tiempo RTP.

- **Último SR** (`LSR`): 32 bits. Son los 32 bits del medio de la etiqueta *NTP Timestamp* del último paquete RTCP `SR` recibido.

- **Retardo desde el último SR** (`DLSR`): 32 bits. Es el retardo en unidades de 1/65536 segundos desde que se recibió el último `SR` y se envió este bloque `RR`.

SR: Sender Report

Los reportes SR son enviados por los participantes que han enviado datos recientemente y se identifica con el número 200 (cuadro 2.3). Está compuesto por 24 bytes de información del transmisor seguidos de cero o más bloques del receptor cuya cantidad se indica en el campo RC (en el encabezado de RTCP se le llama IC).

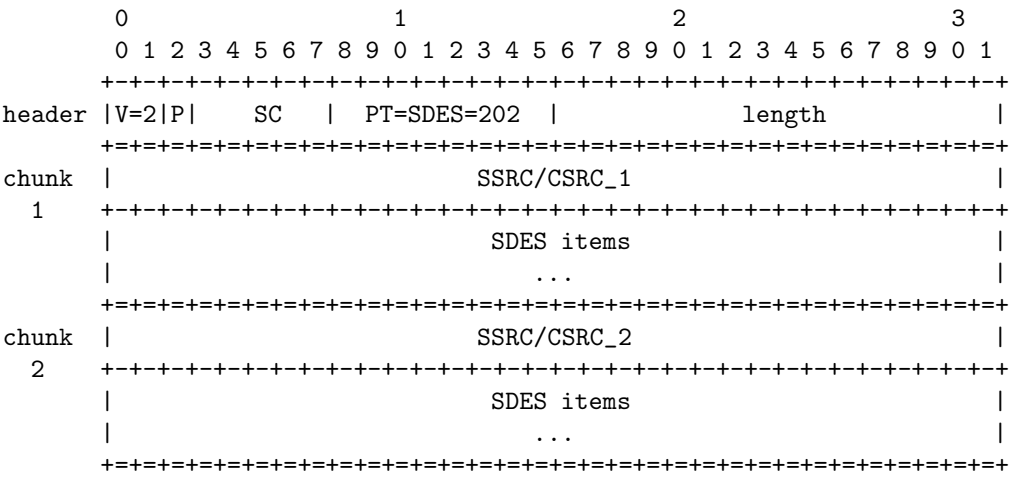


Los primeros 4 bytes pertenecen al encabezado RTCP descrito en 2.6.2. El campo RC indica la cantidad de bloques de recepción incluidos en el paquete. Estos 4 bytes más los siguientes 4 forman el encabezado SR. Los 20 bytes que siguen contienen información del transmisor y por último pueden haber bloques de recepción de todas las fuentes escuchadas por esta fuente.

- **Indicador de fuente** (`SSRC`): 32 bits. Es el identificador de fuente del originador del paquete.
- **Marca de tiempo NTP** (`NTP Timestamp`): 64 bits. Indica el tiempo de envío del paquete SR expresado en el formato NTP que cuenta los segundos transcurridos desde el 1 de enero de 1900. En los primeros 32 bits se indican los segundos y en los segundos 32 bits las fracciones de segundo. Este tiempo es la referencia del reloj de pared. Este reloj es el mismo para todos los medios enviados.
- **Marca de tiempo RTP** (`RTP Timestamp`): 32 bits. Se corresponde con la marca de tiempo NTP pero está expresada en unidades del reloj RTP de los medios.
- **Cuenta de paquetes del transmisor** (`Sender's packet count`): 32 bits. Cantidad de paquetes RTP enviados por esta fuente SSRC desde el comienzo de la sesión. Si el transmisor cambia su identificador SSRC la cuenta se resetea.
- **Cuenta de bytes del transmisor** (`Sender's octect count`): 32 bits. Cantidad de bytes enviados en los paquetes RTP sin contar los encabezados ni el *padding*.

2.6.5. SDES: Source Description

Las descripciones de fuentes proveen identificación a los participantes y detalles como ubicación, dirección de correo electrónico, número de teléfono. Este tipo de paquete RTCP tiene asociado el número 202 (cuadro 2.3). El campo IC indicado en el encabezado RTCP (2.6.2) se llama SC e indica la cantidad de SSRC/CSRC del paquete.

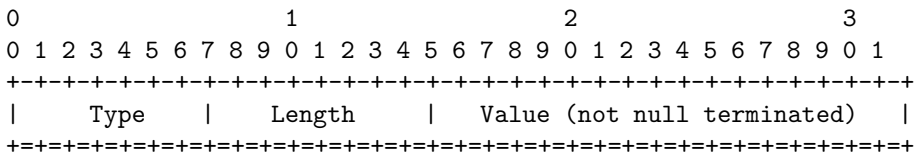


Cada *chunk* consite en un identificador SSRC/CSRC y una lista de ítems a continuación.

- `SSRC/CSRC`: 32 bits.
- `SDES items`: 64 bits. Lista de ítems SDES.

Formato de los ítems SDES

Todos los ítems SDES tienen el mismo formato.



Tipos de ítems SDES

Item SDES	Tipo	Descripción
END	0	Fin de la lista SDES.
CNAME	1	Nombre de usuario y dominio. Persistente para un participante.
NAME	2	Nombre para ser desplegado en una lista de participantes.
EMAIL	3	Dirección de correo electrónico de una fuente.
PHONE	4	Número telefónico para una fuente.
LOC	5	Ubicación geográfica.
TOOL	6	Nombre de una aplicación de una fuente.
NOTE	7	Notas del usuario. Por ejemplo, Almorzando.
PRIV	8	Para extensiones.

Cuadro 2.4: Tipos de ítems SDES

El más importante de los ítems de la cuadro 2.6.5 es el nombre canónico, `CNAME`. Este nombre es único, persistente y estable para cada fuente. No es posible usar únicamente el identificador `SSRC` debido a que puede cambiar durante una sesión. Su principal función es la de permitir asociar distintos medios provenientes de un mismo participante. De esta forma es posible sincronizar por ejemplo, el audio y video de una misma fuente o el video de dos cámaras distintas.

Capítulo 3

Solución de Partida

3.1. Introducción

En base al objetivo del proyecto, se debió buscar una aplicación existente y desarrollada por terceros que tuviera disponible el código fuente para poder incorporarle nuevas facilidades. La búsqueda se focalizó en aplicaciones de teléfonos SIP a los cuales fuese posible agregar soporte para un nuevo codec de audio y la transmisión de un flujo de video.

Al momento de elegir la solución de partida se buscó que la misma fuese para la plataforma Windows y que cumpliera con ciertos requisitos que facilitaran la comprensión y desarrollo del mismo. Uno de los aspectos más importantes considerados fue que el proyecto tuviese documentación, tanto una descripción general como documentación más detallada sobre la arquitectura, las clases de cada librería utilizada, y la interrelación entre el conjunto total de clases.

Otro punto deseable era que el proyecto tuviese foros de desarrolladores activos a los cuales poder consultar y de este modo evacuar posibles dudas u obtener soluciones a distintos problemas.

Por último se consideró que el lenguaje de desarrollo fuese adecuado para implementar aplicaciones de tiempo real, como sería en este caso el video teléfono SIP.

Esta búsqueda nos derivó en aplicaciones desarrolladas por la organización SIPfoundry[14]. SIPfoundry es una organización sin fines de lucro que pertenece a la comunidad de desarrolladores de programas de código abierto, la cual se centra en promover el desarrollo de aplicaciones relacionadas con el protocolo SIP. La misma fue fundada en Marzo de 2004 y mantiene estrecha relación con SIP forum[15] e IETF [16]. SIPfoundry promueve la estandarización de SIP y la interoperabilidad entre productos y soluciones SIP a través del SIP Forum Test Framework [17]. La tecnología original utilizada por SIPfoundry fue donada por la empresa Pingtel Corp[18] cuando decidió adoptar un modelo de negocio de código abierto.

Entre los aplicativos de SIPfoundry se encontraron dos teléfonos SIP que podrían servir como punto de partida en nuestro desarrollo. Estos son conocidos como sipXphone y sipXezPhone. Dado que el primero contaba con mayor documentación se consideró conveniente tomarlo como primera opción. Se procedió entonces a instalar el código fuente de esta aplicación, compilarlo y comenzar a comprender su arquitectura y funcionamiento.

3.1.1. sipXphone

Este proyecto es un teléfono SIP realizado en software para plataformas Windows y Linux, que formaba parte de la familia de proyectos sipX, de SIPfoundry. La arquitectura del programa está compuesta de varias bibliotecas que implementan distintas funcionalidades e interactúan para formar en conjunto la pila de protocolos en los que se basa la aplicación.

Todas las bibliotecas están desarrolladas en el lenguaje de programación C/C++, mientras que la interfaz gráfica está desarrollada en el lenguaje Java.

Las funcionalidades básicas que posee son:

- Llamadas múltiples simultáneas
- Interfaz gráfica con posibilidades de realizar:
  - Hold
  - Transfer
  - Reenvío de llamadas
  - Registro de llamadas
  - Rediscado
  - Identificador de llamadas
  - Manos libres
- Codec G.711 ley A y ley μ

Al momento de la compilación del programa se encontró cierta dificultad para generar la aplicación, en parte por la falta de conocimiento del lenguaje de programación y la herramienta de desarrollo, y en parte debido a que el teléfono contenía ciertos *bugs*.

Finalmente luego de haber eliminado los *bugs*, se logró una compilación exitosa del programa obteniendo el aplicativo sipXphone buscado. Al momento que se dispuso de la solución instalada y compilada, SIPfoundry le dio de baja.

Frente a esto surgió la disyuntiva de si contiunuar trabajando con el teléfono elegido o cambiar al otro teléfono SIP previamente considerado. Continuar trabajando con un proyecto que la organización había dado por obsoleto podría dificultarnos el trabajo dado que ya no sería posible disponer de ayuda por parte de otros desarrolladores. Por este motivo, se consideró que la mejor alternativa sería abandonar el teléfono sipXphone y trabajar en la solución sipXezPhone.

Figura 3.1: Interfaz gráfica de sipXphone

3.1.2. sipXezPhone

SipXezPhone es un software que implementa un teléfono para realizar llamadas de voz sobre IP utilizando SIP como protocolo de señalización. Es un aplicativo perteneciente a SIPfoundry que se basa en el desarrollo de otros proyectos realizados en el área.

SIPfoundry dispone del desarrollo de un **stack** de protocolos SIP, denominado sipX SIP stack, formado por un conjunto de librerías y herramientas con el fin de utilizarlo en distintas aplicaciones SIP. Estas herramientas fueron utilizadas en el desarrollo de los teléfonos SIP, en particular en sipXezPhone donde se hace uso de la librería **SDK sipXtapi** que a su vez interactúa con otras librerías del proyecto sipX.

La aplicación sipXezPhone se desarrolló usando C/C++ como lenguaje de desarrollo, y wxWidgets para la interfaz de usuario y el API de sipXtapi. Funciona tanto para Windows como para Linux.

En cuanto a la documentación del proyecto, la misma no es buena. Existe una descripción general de la función que cumple cada biblioteca en el programa, pero no existe documentación a nivel de clases indicando la herencia, relación entre clases de una librería y relación entre clases de distintas librerías. Dado esto y que el teléfono original está compuesto por más de 500 clases hubo gran dificultad en la comprensión del funcionamiento del teléfono al nivel necesario para poder agregar las nuevas funcionalidades.

Se tomó el código de sipXezPhone como solución de partida para desarrollar las nuevas funcionalidades y convertir al teléfono en una plataforma con soporte de video. El código del teléfono se encuentra en un repositorio SVN [19] en el cual se mantienen versiones estables y versiones en continuo desarrollo. La versión del repositorio de la cual se partió es la número

7725, en la fecha 24 de agosto de 2007 encontrándose en la revisión número 9964.

Se procedió a instalar y compilar la solución hasta lograr generar la aplicación que de aquí en más sería el punto de partida del proyecto sipXvideoPhone. Se consideró adecuado denominar a nuestro aplicativo sipXvideoPhone teniendo en cuenta que sería desarrollado a partir de librerías del proyecto sipX, y que su objetivo final sería lograr que la plataforma soporte llamadas de video.

Figura 3.2: Interfaz gráfica de sipXezPhone

3.2. Funcionalidades de la solución de partida

Las funcionalidades con las que contaba el teléfono sipXezPhone del cual partimos son:

- Identificador de llamadas
- Manejo de direcciones SIP
- Lista de números más usados
- Ajuste de volumen de parlantes y micrófono
- Hold y mute

- Transferencia de llamadas [1]
- Codec G.711 ley A y ley μ
- Cancelación automática de eco **(para G.711)**
- Soporte de STUN (Simple Traversal of UDP (User Datagram Protocol) bajo la precencia de NATs (Network Address Translators))

3.3. Arquitectura de la solución de partida

La solución de sipXezPhone se compone de una variedad de librerías desarrolladas en C/C++. Cada una de estas librerías fue diseñada con un objetivo particular y pretende resolver una de las etapas necesarias para disponer de un agente de usuario SIP.

Seguidamente se listan las librerías más importantes que componen la solución que se tomó como punto de partida, pasando luego a detallar el rol que cada una cumple en lo que refiere a la aplicación. Como se mencionó desde un principio, no se cuenta con suficiente documentación como para presentar exhaustivamente el funcionamiento de las mismas.

- `sipXtapi`
- `sipXcallLib`
- `sipXmediaAdapterLib`
- `sipXmediaLib`
- `sipXtackLib`
- `sipXportLib`

La figura 3.3 muestra la relación de dependencia existente entre las distintas librerías y servirá de ayuda para comprender la dinámica de las mismas.

3.3.1. sipXtapi

`sipXtapi` es la librería que permite desarrollar una aplicación de agente de usuario SIP. Si bien esta librería hace uso de algunas otras, es quien proporciona los lineamientos básicos para desarrollar el agente de usuario.

Para cumplir con el objetivo, es necesario desarrollar una interfaz gráfica que interactúe con `sipXtapi` y que utilice los métodos que la misma provee. El usuario desarrollador de la aplicación únicamente debe conocer qué funcionalidades son posibles de lograr con la librería

[1] Esta función no se realiza correctamente en la solución de partida.

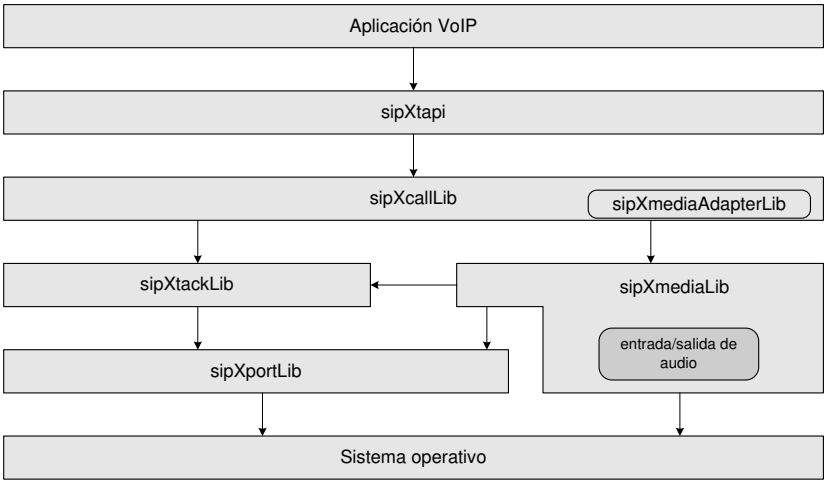


Figura 3.3: Arquitectura de Sipxtapi

y cómo manejar sus herramientas.

`sipXtapi` provee varias funcionalidades, algunas de las cuales se basan en estándares y protocolos según como se describe a continuación:

- SIP. Definido en la RFC 3261.
- Tonos DTMF fuera de banda, definido en la RFC 2833
- Transferencia de llamada no transparente, definido en el draft draft-ieft-sip-cc-transfer-05
- Negociación de codecs (SDP), definido en la RFC 3264
- Notificación de eventos, definido en la RFC 3265
- Hold y off-Hold
- Mute y Unmute
- Ubicación de servidores SIP, definido en la RFC 3263
- Puerto SIP configurable
- Puertos RTP/RTCP configurables
- Transporte sobre UDP

Estas funcionalidades mencionadas no son implementadas directamente por `SipXtapi` sino que son las librerías que están por debajo de ella quienes en conjunto permiten lograr esto. El objetivo de `SipXtapi` es proporcionar un API a desarrolladores de telecomunicaciones y usuarios estándar, evitando que los mismos no entren en detalle de como están implementadas las cosas ni necesiten conocer el funcionamiento de los protocolos utilizados como base.

3.3.2. sipXcallLib

Esta librería provee una capa abstracta de procesamiento de llamadas, que se encuentra sobre `sipXmediaAdapterLib`, `sipXmediaLib` y `sipXtackLib`. Aquí se construye un modelo para las llamadas telefónicas haciendo uso de los recursos y elementos básicos de SIP.

`sipXcallLib` es el corazón del procesamiento de llamadas e incluye clases que implementan un administrador de llamadas, clases que implementan conexiones, y clases que actúan de interfaz con el medio de transporte RTP y con el stack de protocolos SIP.

Para modelar las llamadas se hace uso de un diagrama de estados y es el administrador de llamadas quien controla el estado de la llamada en cada momento. Para cada posible estado el administrador se encarga de darle a la llamada el funcionamiento adecuado. En la figura 3.4 se observa un diagrama simplificado de la transición de estados para el caso de una llamada entrante.

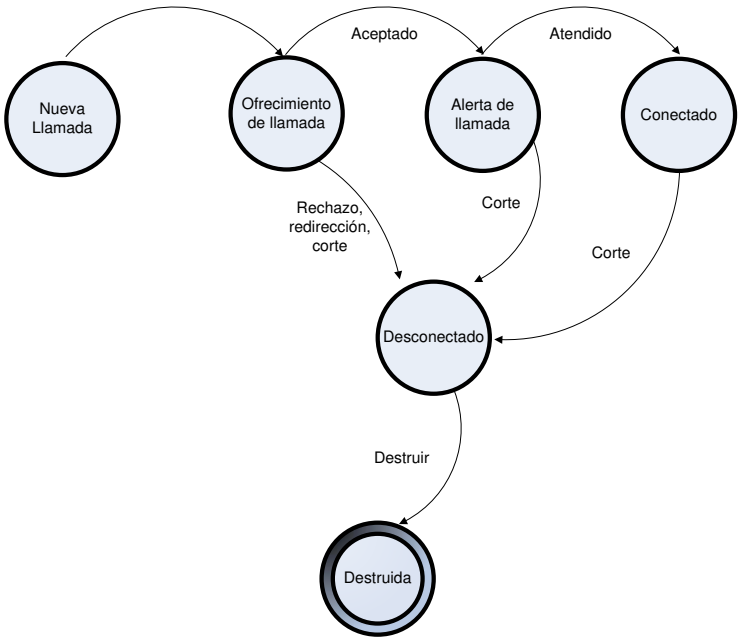


Figura 3.4: Diagrama de estados de una llamada simplificado

3.3.3. sipXmediaLib

Esta es la librería encargada del procesamiento de medios de audio durante a una llamada. En la figura 3.5 puede observarse un esquema del camino que siguen los datos durante una llamada. A cada bloque se le denomina *recurso de procesamiento de medios*, y representa una función bien determinada para procesar la información de audio.

Existen dos posibles caminos dentro del diagrama:

1. Un camino representa la transmisión de datos desde que son generados en el agente de usuario local hasta que son enviados por la red hacia la aplicación remota. En la figura 3.5 se puede apreciar este camino en trazo punteado. Los datos son capturados por el micrófono y luego, en el caso que se utilice algún tipo de compresión de voz, son entregados a la red para enviarse al agente de usuario remoto.

2. El otro posible camino representa la recepción de los datos de audio que el agente remoto envió hacia el agente de usuario local. Aquí se toman los datos de la red, se descomprimen si es necesario y se envían al parlante local para que se reproduzca el audio recibido. Este camino se aprecia en la figura 3.5 en trazo continuo.

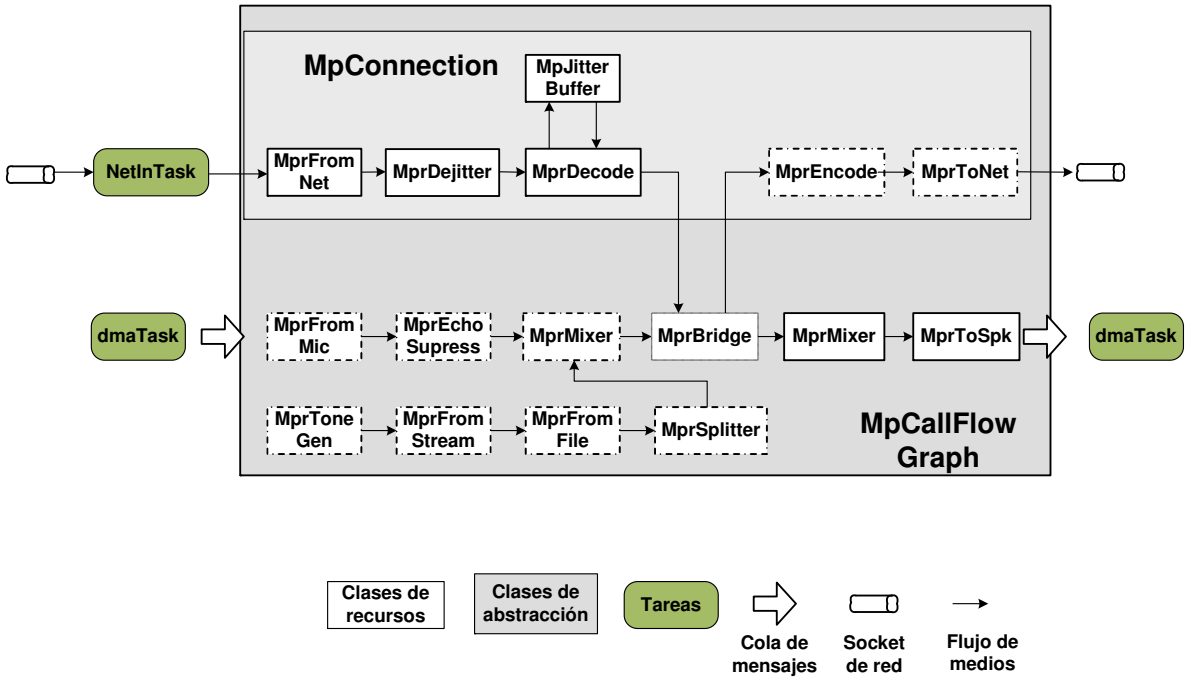


Figura 3.5: AudioFlowGraph

3.3.4. sipXmediaAdapterLib

La librería `sipXmediaAdapterLib` es la encargada de establecer una comunicación punta a punta a nivel de flujo de medios. Más aún, es quien se encarga de crear y mantener actualizada la lista de los tipos de medios soportados por la aplicación, y con ellos las características propias asociadas.

`sipXmediaAdapterLib` tiene estrecha relación con dos de las librerías: `sipXcallLib` y `sipXmediaLib`.

Por una parte se recurre a la librería `sipXcallLib` quien se encarga del manejo de las llamadas. Dado que el procesamiento de medios comienza a realizarse una vez establecida la llamada, será necesario que `sipXmediaAdapterLib` sea notificada del estado de la misma.

Por otra parte, `sipXmediaAdapterLib` se vincula con `sipXmediaLib` cuando esta última requiere información aceca de los medios que fueron negociados durante el establecimiento de la llamada. De esta manera será posible brindar a los datos el procesamiento que corresponda.

3.3.5. sipXtackLib

El proyecto `sipXtackLib` implementa una pila de protocolos SIP (*stack*) totalmente basado en las RFCs 3261 y 3265. La pila fue creada con el objetivo de poder ser usada en puntos terminales como lo es la aplicación que en este caso nos interesa; el softphone sipXezPhone.

`sipXtackLib` provee las herramientas necesarias para modelar la señalización SIP de una llamada. A tales efectos dispone de la implementación de todos los mensajes SIP que son intercambiados durante el establecimiento y finalización de las llamadas. El manejo de estos mensajes conciernen específicamente a esta librería, transformándola en imprescindible al momento de querer establecer sesiones de medios entre agentes de usuarios SIP.

Las características SIP más destacadas que posee `sipXtackLib` son:

- Compatibilidad con la RFC 3261 donde se define el protocolo SIP
- Balanceo de carga y confiabilidad utilizando la RFC 3263 (DNS SRC)
- Negociación de codecs, RFC 3264
- Transferencia, REFER y REPLACES, RFCs 3420, 3515, 3891
- Servidor y cliente de presencia, RFCs 3856 y 3863
- Eventos SIP (suscripción y notificación), RFC 3265
- Mensajería instantánea, RFC 3428
- Indicación de mensaje en espera, RFC 3842
- Traducción NAT vía rport, STUN y TURN, RFC 3581

La biblioteca puede ser utilizada en las siguientes plataformas:

- Windows (MSVC 6, .NET, MSVS 2003, MSVS 2005)
- Linux (GNU)

- Mac OS X
- VxWorks (proyecto en camino)
- WinCE/Pocket PC (proyecto en camino)
- Symbian (proyecto en camino)

3.3.6. sipXportLib

Esta biblioteca proporciona un conjunto de clases que logran abstraer la aplicación del sistema operativo. Los proyectos de SIPfoundry utilizan esta librería para asegurar un fácil montaje sobre cualquier sistema operativo.

`sipXportLib` dispone de clases que implementan funciones y operaciones para manejar los siguientes aspectos:

- Hilos (comúnmente conocidos como Threads)
- *Locks* y *Mutexes*
- Semáforos
- Mensajes y colas
- Temporizadores
- Fecha y hora
- *Sockets*
- Archivos y directorios
- Procesos a nivel de sistema operativo
- *Dynamic loading of shared libraries and symbols*

La biblioteca es compatible con las siguientes plataformas:

- Windows/win32
- Windows CE
- Linux
- Solaris

3.4. Inicialización del teléfono

El teléfono sipXezPhone cuenta con una interfaz gráfica a través de la cual se inicializa la aplicación. Esta interfaz fue desarrollada en el lenguaje C++ y hace uso de una librería externa denominada wxWidgets [20].

La interfaz gráfica del teléfono presenta la posibilidad de configurar ciertos parámetros relativos al usuario y a la llamada que se desea establecer. Esto se realiza a través de la ventana de configuración ilustrada en la figura 3.6, la cual se hace presente cada vez que se ejecuta la aplicación previo a la apertura del teléfono.

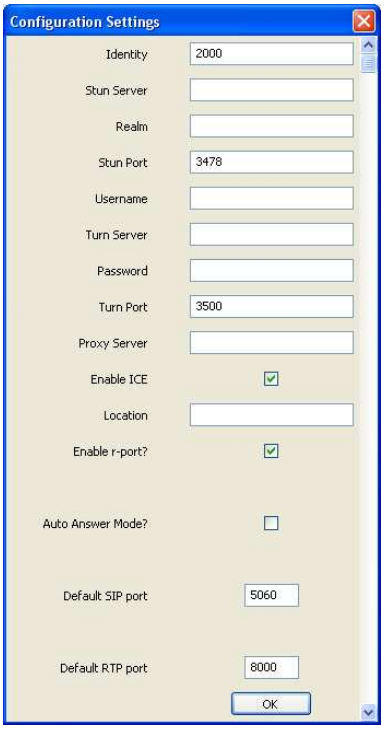


Figura 3.6: Pantalla de configuración inicial de sipXezPhone

La ventana de configuración tiene varios campos a completar por el usuario que permiten caracterizar cómo va a ser la llamada. Los que resultan de interés para la aplicación son:

- ***Identity***: permite establecer la SIP URI con que el usuario va a ser contactado en el teléfono inicializado, por ejemplo: sip: 1000@192.168.1.100.
- ***Auto answer mode***: habilita que el teléfono se auto-atienda cada vez que recibe una llamada.
- ***Default SIP port***: permite al usuario determinar el puerto a utilizar para la negociación

SIP de la llamada. El puerto recomendado por el protocolo SIP para TCP y UDP es el 5060, y es el que se utilizó en nuestra aplicación.

- ***Default RTP port***: permite determinar el primer puerto a partir del cual se van a establecer todas las sesiones RTP.

El resto de los parámetros que se observan en la figura 3.6 se utilizan cuando se requiere utilizar algún servidor proxy SIP. En nuestro caso esto no es necesario ya que el ambiente de trabajo es una LAN, dejando sin efecto la utilización de servidores SIP.

Luego de haber configurado los parámetros antes descritos y presionado el botón `OK`, aparece la interfaz gráfica que representa el teléfono en sí mismo. Esta interfaz se puede observar en la figura 3.2.

El teléfono presenta una barra de menús entre los que se encuentra el menú *Settings*. Aquí se despliegan varias opciones de configuración donde es posible setear parámetros relativos a la sesión de audio, entre otros. Esto se encuentra bajo la opción *Audio Settings* y es el menú que nos resulta de interés en el desarrollo de nuestra aplicación. Éste pérmite seleccionar el codec de compresión a utilizar durante la llamada de audio.

sipXezPhone es muy fácil de utilizar, y su manejo se detalla más adelante en el manual de usuario.

3.5. Testeo del teléfono de partida

Luego de definir la solución utilizada como punto de partida y previo a la integración de las nuevas funcionalidades sobre la misma, se procedió a verificar su correcto funcionamiento.

Se realizaron distintas pruebas sobre el aplicativo sipXezPhone que permitieron reconocer cuáles de las facilidades ofrecidas por el teléfono funcionaban correctamente.

Las funcionalidades probadas fueron:

- Llamada entre dos agentes de usuario sipXezPhone utilizando el codec de audio G.711 ley A
- Llamada entre dos agentes de usuario sipXezPhone utilizando el codec de audio G.711 ley μ
- Hold
- Mute
- Transferencia

Los resultados de estas pruebas fueron casi satisfactorios. Se encontró que las llamadas de audio se realizaban correctamente y el flujo RTP correspondía en ambos caso con la sesión de medios establecida (G.711 ley A ó G.711 ley μ). En cuanto a las facilidades adicionales, se observó que *Hold* y *Mute* funcionaban correctamente mientras que la transferencia no se podía realizar de manera adecuada. Esto no resultó un problema dado que la transferencia de llamadas no repercute en el desarrollo de nuestro aplicativo sipXvideoPhone. La funcionalidad de transferencia podría dejarse como un posible trabajo a futuro, y en caso de disponer del tiempo necesario se procedería a corregir el código para que la plataforma cuente con esta facilidad.

En lo que refiere al funcionamiento básico del teléfono, se verifcó que sipXezPhone cumplía con los requerimientos y efectivamente era posible realizar llamadas de audio entre dos agentes de usuario. Se capturó el tráfico de paquetes intercambiados entre los dos agentes de usuario y con ello fue posible comprobar que la señalización SIP de la llamada se realizaba acorde con las especificaciones del protcolo. A continuación se muestra el flujo de paquetes resultante.

```
| 192.168.1.104              192.168.1.103 |
|         INVITE SDP ( G.711mu )           |SIP From: sip:2000 To:sip:p3
|(5060)   ------------------>  (5060)      |
|             100 Trying                   |SIP Status
|(5060)   <------------------  (5060)      |
|             180 Ringing                  |SIP Status
|(5060)   <------------------  (5060)      |
|         200 OK SDP ( G.711mu )           |SIP Status
|(5060)   <------------------  (5060)      |
|                ACK                       |SIP Request
|(5060)   ------------------>  (5060)      |
|           RTP (G.711mu)                  |RTP
|(8000)   <------------------  (8000)      |
|           RTP (G.711mu)                  |RTP
|(8000)   ------------------>  (8000)      |
|                BYE                       |SIP Request
|(5060)   ------------------>  (5060)      |
|               200 OK                     |SIP Status
|(5060)   <------------------  (5060)      |
```

Teniendo en cuenta nuestros objetivos de integrar al teléfono otros codecs de compresión, resultó interesante observar el intercambio de información en el cuerpo SDP de los mensajes SIP. Aquí se transporta información descriptiva de las sesiones de medios que se pueden establecer y a modo de ejemplo se muestra el cuerpo del mensaje INVITE, donde se describe la sesión de medios utilizando el codec G.711μ.

```
Message body
     Session Description Protocol
         v = 0
         o = sipX 5 5 IN IP4 192.168.1.101
         s = call
         c = IN IP4 192.168.1.101
         t = 0 0
         m = audio 8000 RTP/AVP 0 8 96
         a = rtpmap:0 pcmu/8000/1
         a = rtpmap:8 pcma/8000/1
         a = rtpmap:96 telephone-event/8000/1
```

Estas pruebas permitieron observar el funcionamiento del protocolo SIP y los protocolos relacionados plasmando así el conocimiento teórico adquirido.

Finalmente, fue posible verificar el correcto funcionamiento del teléfono SIP que se tomó como punto de partida. A partir de sipXezPhone se procede entonces con la incorporación de las nuevas funcionalidades con el objetivo final de desarrollar la aplicación sipXvideoPhone.

Capítulo 4

Agregado de un codec de audio

4.1. Introducción

El desarrollo de sipXvideoPhone comenzó por la incorporación de un codec de compresión de audio. El punto de partida fue la plataforma diseñada para sipXezPhone, la cual presenta la posibilidad de transmitir sesiones de audio. El único codec soportado por sipXezPhone es el estándar G.711 y en esta etapa del trabajo se incorpora soporte para flujo de audio codificado en G.729.

Para incorporar el nuevo codec de compresión se utilizó una librería ya desarrollada y que proporcionó las herramientas necesarias para poder codificar y decodificar señales en G.729. Se consideró que desarrollar el codec de compresión de cero escaparía al alcance del proyecto y por tanto se creyó adecuado alcanzar los objetivos integrando una implementación existente del codec G.729.

Para ello, fue necesario realizar un estudio de la recomendación del codec en cuestión, así como también de las librerías existentes que pudieran servir de apoyo en nuestro desarrollo.

En este capítulo se comienza por realizar un análisis teórico del codec G.729, luego se introduce la librería elegida y finalmente se detallan las implementaciones realizadas en sipXvideoPhone que permitieron lograr el objetivo deseado.

4.2. Introducción a G.729

El codec G.729 es un estándar de codificación para señales de audio desarrollado por la ITU (International Telecommunications Union). G.729 codifica las señales de voz a 8 kbit/s utilizando CS-ACELP (*Conjugate-Structure Algebraic-Code-Excited Linear-Prediction*) [21].

Este codec está diseñado para recibir en su entrada una señal digital obtenida luego de filtrar la voz mediante el filtro pasa-banda usado en telefonía, muestrearla a una frecuencia de

8000 Hz y finalmente convertirla a PCM de 16 bits (figura 4.1).[1] A la salida del codificador se obtiene un *stream* de 8 kbit/s. En el decodificador la señal se convierte nuevamente a analógica.

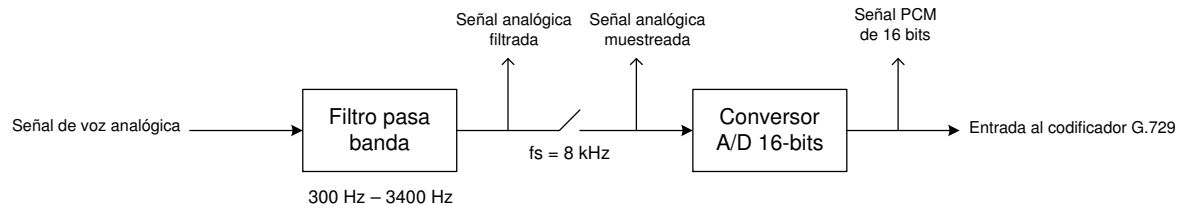


Figura 4.1: Pre-filtrado de la señal de voz

El objetivo de la codificación de voz es representar en forma digital la señal analógica utilizando la menor cantidad de bits posibles manteniendo un cierto nivel de calidad. La mayoría de los codificadores de voz se basan en predicciones lineales (LPC, *Linear Prediction Coding*) debido a que son fáciles de implementar [23].

La voz se produce por la interacción entre el tracto vocal y las cuerdas vocales. Los codificadores basados en predicciones lineales tienen en cuenta la redundancia de la señal de voz modelando el tracto vocal como un filtro lineal auto regresivo (AR, *AutoRegresive*). Los codificadores lineales predictivos usan cuadros de audio de entre 1 y 100 ms para obtener los coeficientes LP y la señal residual. La duración del cuadro de audio establece un límite inferior al retardo de codificación ya que al menos se debe esperar que transcurra ese tiempo para codificar las muestras.

Como se mencionó G.729 utiliza ACELP que se basa en el modelo CELP operando con cuadros de audio de 10 ms correspondientes a una cantidad de 80 muestras, ya que la frecuencia de muestreo es de 8000 muestras por segundo. En ACELP los códigos son algebraicos. El algoritmo CELP se basa en cuatro ideas principales:

- Usa un filtro fuente para modelar la voz que se basa en LP (*Linear Prediction*).
- Usa códigos adaptativos y fijos como entrada (excitación) del modelo LP.
- Hace una búsqueda en lazo cerrado.
- Aplica cuantización de vectores (VQ, *Vector Quantization*).

CELP no conserva la forma de onda, codifica en base a características del oído y de la voz humana. Dado que el oído no detecta la forma de onda, se genera una nueva forma de onda que reproduce la voz en base a instrucciones. Así se tiene una tabla de posibles tonos generados por las cuerdas vocales (fonemas) y otra con filtros que modelan la posición de la boca, faringe,

[1] Las frecuencias de corte del filtro pasa-banda están definidas en la recomendación de la ITU, G.712 que describe la performance de canales de transmisión para PCM [22].

laringe y lengua, que filtran los tonos generados por las cuerdas vocales. Las señales enviadas se llaman descriptores y son una combinación de fonemas y filtros.

El modelado de la boca y la garganta, como se mencionó anteriormente, se hace por medio de filtros lineales y la voz se genera a partir de una vibración periódica de aire que los excita. El proceso por el cual se extraen los parámetros del filtro y de la excitación se llama *Analysis-by-Synthesis*, AbS. El codificador busca los parámetros y realiza una decodificación interna (señal sintetizada) para comparar con la señal original. Se eligen los parámetros que concuerdan mejor y se los envía al decodificador [25].

La calidad de voz de un codec se mide de acuerdo al MOS, *Mean Opinion Score*, que es una medida subjetiva del oyente. Para calcular el MOS, una amplia variedad de oyentes califica la calidad de una muestra de voz del 1 al 5 haciéndose luego un promedio (mayor calificación mejor calidad). G.729 tiene un MOS de 4.

Codificación

Cada 10 ms se extraen los parámetros del modelo CELP (coeficientes del filtro lineal predictivo, códigos adaptativos y fijos, gananacias). A partir de éstos se computan los coeficientes LP, se convierten a LSP (*Line Spectrum Pairs*) y se cuantizan usando vectores predictivos de dos etapas (VQ) con 18 bits.

LSP se usa para representar a los coeficientes LP para su transmisión sobre el canal [26]. Tiene la característica de proveer gran inmunidad al ruido por lo que se utiliza para la codificación de la voz. Se usa una representación algebraica para la *glotis* abierta y otra para cuando está cerrada.[2] La señal de excitación a la salida se elige en base a un procedimiento de búsqueda en donde el error entre la señal original y la reconstruida es minimizado de acuerdo a una medida ponderada porcentualmente de la distorsión. Los códigos adaptativos y fijos se extraen cada medio cuadro, es decir cada 5 ms (40 muestras). Los códigos son los que contienen los vectores de excitación.

Dado que a la salida del codificador la tasa de bits es de 8 kbit/s y se toman cuadros de 10 ms, se deben usar exactamente 80 bits para representar a cada cuadro de audio. Las etapas por las que atraviesa el audio hasta estar pronto para ser enviado al decodificador se describen en lo que sigue:

1. Preprocesamiento: Antes de codificar la señal PCM de 16 bits se escala la señal dividiéndola entre dos para evitar desbordes en la implementación con punto flotante y se hace un filtrado pasa alto. El filtro pasa alto tiene una frecuencia de corte de 140 Hz y es de segundo orden. Se usa como precaución para eliminar componentes de baja frecuencia.

2. Análisis de predicción lineal y cuantización: Para la obtención de los coeficientes LP se usan predicciones a corto y a largo plazo. Las predicciones a corto plazo miran el

[2] La glotis es el espacio entre las cuerdas vocales [27].

formato de la voz, mientras que las de largo plazo estudian la correlación de la voz y su periodicidad.

Los filtros a corto plazo se basan en filtros de predicción lineal de décimo orden. Dado que la autocorrelación varía con el tiempo, se estima sobre intervalos de 30 ms que incluyen 15 ms del pasado, 10 ms del cuadro actual y 5 ms del futuro. Esto implica que el retardo total de codificación sea de 15 ms (10 ms de cuadro más 5 ms de predicción a futuro). Para extraer los coeficientes LP se toma una ventana asimétrica.

Una vez que se tienen los coeficientes LP se convierten a LSP para cuantización.

3. Cálculo de códigos: El siguiente paso es el cálculo de los códigos para extraer la excitación. Esto se hace cada sub-cuadro de 5 ms. En cada uno se calcula la exitación como la suma de dos componentes: la excitación que se está usando retrasada (contribución de código adaptativo) más una señal compuesta por cuatro impulsos en distintas posiciones (contribución de código fijo).

4. Finalmente los parámetros obtenidos son cuantizados y la señal queda pronta para ser enviada al decodificador.

Decodificación

Cuando se recibe una señal desde el codificador, de la misma se extraen los parámetros correspondientes a un cuadro de 10ms. Algunos parámetros son los coeficientes LSP, los dos códigos adaptativos y los dos códigos fijos. Una vez obtenidos los códigos, para cada cuadro de 5ms los coeficientes LSP son convertidos a coeficientes para el filtro LP.

Luego, para cada uno de los dos cuadros de 5ms se efectúan tres pasos. La dinámica de estos pasos se puede observar en la figura 4.2 y se describen a continuación:

1. En el primero los códigos fijos y adaptativos son ponderados por las ganancias correspondientes y a continuación sumados, obteniendo la exitación.

2. En el segundo paso la exitación es filtrada por el filtro LP, obteniendo así la señal de voz.

3. Por último, luego que el decodificador reconstruye la señal esta es post-procesada. El post-procesamiento consiste en el pasaje de la señal por tres filtros, más un factor de compensación de ganancia. Los filtros son un filtro a largo plazo, uno a corto plazo y un filtro de compensación de tilt.

Completado este la señal es recuperada, pareciéndose fuertemente a la señal de voz original.

G.729A

En el anexo A de la recomendación G.729 se define una variante de este codec logrando así un codec de menor complejidad llamado G.729A. Éste es interoperable con G.729, pudiéndose utilizar un codificador G.729A con un decodificador G.729 y viceversa.

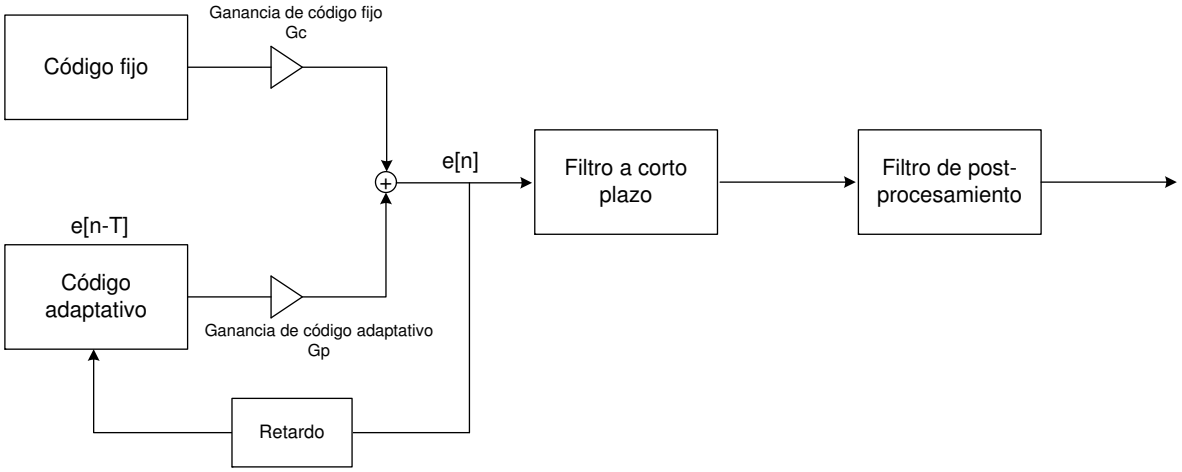


Figura 4.2: Principio del decodificador CS-ACELP

Los cambios respecto a la versión completa se deben a simplificaciones en los algoritmos empleados. La reducción de la complejidad incluye la sustitución de algunos bloques de procesamiento por otros más sencillos. También incluye el mantener fijos parámetros que en la versión completa del codec varían dependiendo del audio a codificar o decodificar.

Debido al compromiso entre la calidad y la complejidad, esta versión de codec sufre una degradación en la calidad de la voz. Mientras G.729 obtiene un MOS de 4, la versión G.729A presenta un MOS de 3,8.

4.3. Elección de la librería

Durante la búsqueda de un módulo que codifique y decodifique señales en G.729 se encontró una implementación realizada por Voice Age [28], una corporación de desarrollo de soluciones tecnológicas para voz y audio.

La implementación de Voice Age está formada por una librería que se integra a la aplicación deseada y permite, mediante la invocación de funciones específicas, codificar y decodificar información de audio según la recomendación del codec G.729, anexo A. Dado que los codificadores G.729 y G.729A son totalmente interoperables, de aquí en más no se hará distinción entre ellos.

Como usuarios del desarrollo aplicativo de Voice Age solamente se tuvo acceso a un conjunto de archivos que permitieron el uso de la aplicación sin poder acceder al código fuente del desarrollo ya que este no está disponible a terceros.

El detalle de los documentos a los que se accedió se describen a continuación:

`va_G.729a.lib`: es una librería de vínculo estático para ser utilizada en Win32.

`va_G.729a.h`: es el API de la aplicación y en ella se describen los métodos disponibles y cómo los mismos deben ser utilizados.

`va_G.729a_encoder.c`: es un código de ejemplo que muestra cómo se utiliza la aplicación para codificar.

`va_G.729a_decoder.c`: es un código de ejemplo de que muestra cómo se utiliza la aplicación para decodificar.

`va_G.729a_encoder.exe`: es un programa ejecutable que recibe como entrada un archivo de audio y tiene como salida un archivo codificado en G.729.

`va_G.729a_decoder.exe`: es un ejecutable que recibe como entrada un archivo de audio codificado en G.729 y retorna un archivo de audio decodificado.

Las especificaciones del codec proporcionado por Voice Age utilizado se encuentran en el cuadro 4.1

Bit rate	8 kbps
Frecuencia de muestreo de la voz	8000 Hz
Tamaño del cuadro de voz	10 ms
Retardo por la codificación	5 ms
Canales soportados	1

Cuadro 4.1: Especificaciones del codec G.729 Voice Age

4.3.1. Formato de entrada y salida del codificador

El formato de entrada que requiere el método codificador es una señal de audio mono PCM, codificada con 16 bits, muestreada a una frecuendia de 8000 Hz y sin nignún tipo de encabezado. Cada cuadro (*frame*) de voz a la entrada está formado por una cantidad de 160 bytes ($\frac{10ms \cdot 16bits}{8000Hz}$), que a la salida se traducen en un paquete de 10 bytes (10*8 bits). Este paquete emitido a la salida contiene una lista de parámetros que se corresponden con los definidos en la recomendación de G.729.

La lista de parámetros obtenida a la salida del codificador se transmite como un continuado de datos comenzando por el parámetro de mayor orden y finalizando en el de menor orden. La transmisión de los bits se envía desde el bit más significativo al menos significativo.

4.3.2. Formato de entrada y salida del decodificador

El decodificador de Voice Age requiere que la señal de audio le ingrese en el mismo formato en que es emitida por el codificador G.729. Este es un paquete de un tamaño igual a 10 bytes como se describió anteriormente.

En lo que refiere a la salida, los datos son retornados en cuadros de audio mono PCM, codificado con 16 bits y muestreado a una frecuencia de 8000Hz.

4.3.3. Funciones disponibles en la librería

La librería proporcionada por Voice Age dispone de cuatro funciones que hacen posible la codificación y decodificación según el estándar G.729. Dos de estas funciones permiten llevar a cabo la codificación y las dos restantes la decodificación. A continuación se describe brevemente cada una de ellas.

Las funciones que refieren al codificador son:

`va_G.729_init_encoder(void)`: este método se utiliza para inicializar la memoria estática requerida durante el proceso de codificación. Debe ser invocado previo a cualquier utilización del canal de audio en la aplicación.

`va_G.729_encoder(short *speech, unsigned char *bitstream)`: cada vez que este método es invocado, codifica un único cuadro de voz en el formato especificado anteriormente. Cada cuadro de voz a la entrada está formado por 80 muestras de audio (`speech`) y a la salida se genera un flujo de datos codificado en G.729 de tamaño igual a 10 bytes (`bitstream`).

Las funciones relacionadas con el decodificador son:

`va_G.729_init_decoder(void)`: este método es utilizado para inicializar la memoria estática requerida durante el proceso de decodificación. Es obligatorio invocar este método previo a recibir datos de audio en el decodificador.

`va_G.729_decoder(unsigned char *bitstream, short *synth, int bfi)`: este método decodifica un paquete de audio de 10 bytes (`bitstream`), en un cuadro de voz mono PCM 16 bits formado por 80 muestras (`synth`). El parámetro `bfi` es utilizado para indicarle al decodificador cuando el cuadro que se va a decodificar presenta algún error. Para el caso, este parámetro no es utilizado ya que no se posee un detector de errores externo a la solución de Voice Age.

4.4. Integración de G.729 a la aplicación

Luego de estudiar la recomendación de G.729 y analizar la librería elegida (mirar apéndice A), se procedió a incorporar el codec de compresión a la aplicación.

Aprovechando que la aplicación original fue diseñada en un lenguaje de programación orientado a objetos y pensando en la posibilidad de incluir nuevas funcionalidades, el agregado de variantes no debe resultar en la modificación de una cantidad importante de clases. Esto resultaría de gran utilidad en esta etapa del desarrollo dado que la aplicación ya dispone de una plataforma base que tiene implementada la transmisión de un flujo de audio.

Fue necesario comprender el desarrollo de sipXezPhone que se tomó como punto de partida. Se estudió la arquitectura del programa para poder determinar las bibliotecas y clases que deben ser modificadas al agregar el codec G.729. Dado que el código de SIPfoundry está formado por una gran cantidad de clases, en esta etapa solamente se buscó adquirir una idea general de cómo funciona cada una de las librerías que conforman la aplicación.

Como apoyo se utilizó un programa que genera documentación a partir de código fuente, denominado Doxygen[29]. El programa se aplicó a cada librería, obteniendo un diagrama de herencia y los diagramas de clases correspondientes. Si bien el diagrama de herencia resultó de gran utilidad, los diagramas de clases no tanto dado que se genera uno para cada clase perdiendo de vista el polimorfismo, herramienta muy utilizada en todo el código. En los diagramas de clases obtenidos no se pudo incluir la relación entre clases pertenecientes a distintas librerías, perdiendo información muy importante a la hora de comprender el funcionamiento completo del programa.

En base a esto, se optó por dividir el trabajo en varias etapas de manera de tener objetivos más pequeños y alcanzables. Las metas planteadas fueron:

1. Negociación de una llamada utilizando el nuevo codec de audio
2. Implementación del codificador y decodificador

4.4.1. Negociación de una llamada con G.729

El objetivo de esta primera etapa del desarrollo fue lograr la negociación de una llamada con el codec G.729. Hasta el momento la aplicación contaba únicamente con soporte para el codec de audio G.711 y aquí se intenta agregar una segunda opción para poder seleccionar al momento de iniciar una sesión de medios con otro agente de usuario.

Para poder establecer una llamada entre dos agentes de usuarios SIP (Session Initiation Protocol), es necesario que primero se lleve a cabo la negociación de la misma. A tales efectos existe el protocolo de señalización SIP, mediante el cual los agentes de usuario acuerdan los tipos de medios que van a transmitirse durante la llamada y las características de los mismos.

Haciendo referencia al desarrollo teórico realizado para el protocolo SIP, se desprende que la negociación de una sesión de medios se lleva a cabo mediante el intercambio de mensajes SIP. Cuando un agente de usuario desea establecer una sesión, envía un mensaje del tipo `INVITE` y en él transmite la información correspondiente con el tipo de medios, el codec de compresión utilizado, y los puertos asociados. Esta información se transmite en el cuerpo del mensaje `INVITE`, denominado cuerpo SDP (Session Descritpion Protocol). Si el agente de usuario receptor soporta una sesión de medios con esas características entonces acepta la invitación con un mensaje del tipo `200 OK` en el cual nuevamente se envía a través del cuerpo SDP, una descripción del medio negociado.

Aplicando estos conceptos para poder incorporar la negociación de una sesión de audio con G.729, se debió lograr que los mensajes SDP intercambiados entre los agentes de usuario contuvieran la información correspondiente con dicha sesión. Esto incluye básicamente la descripción del codec de compresión G.729, y la especificación de los puertos a través de los cuales se establece la sesión.

Para poder utilizar el codec G.729, se lo debió incorporar previamente a la aplicación. Por otra parte, no fue necesario determinar los puertos asociados al canal dado que el el teléfono original ya contaba con la posibilidad de establecer sesiones de audio. El hecho de incorporar otro codec a la aplicación solamente otorga una nueva posibilidad al momento de negociar el tipo de medios, pero el canal determinado por los puertos no se ve afectado.

Agregado del nuevo codec

Para comenzar con la integración del nuevo codec, se procedió primero a estudiar cómo se hace la implementación para el codec de audio ya soportado por sipXezPhone (G.711). Se hizo un seguimiento del programa en busca de aquellas clases que contribuyen a la definición de G.711, al listado del codec en el menú de configuración del teléfono, entre otros. La idea fue intentar comprender cómo el programa original resolvía estos puntos para tener un punto de partida en nuestro desarrollo.

Luego de analizar el desarrollo de sipXezPhone, se encontró que incorporar un nuevo codec de compresión implica crear un objeto codec con todas las características propias que lo definen. A tales efectos, la arquitectura del programa cuenta con una estructura de datos `SdpCodec` donde es posible especificar los parámetros relativos a cada codec en particular. La definición de este nuevo codec se realiza en la clase `SdpCodecFactory`, la cual contiene un listado del grupo de todos los codecs soportados por el teléfono.

Para poder crear el codec representativo de G.729 fue necesario definir previamente en el programa una serie de etiquetas que lo identifiquen según su nombre, su tipo de carga útil, tipo de medios (audio), entre otros. A modo de ejemplo se listan algunas de las utilizadas:

```
SIPX_CODEC_ID_G729 "G729"
MIME_SUBTYPE_G729 "G729"
SDP_CODEC_G729 = 18
CODEC_TYPE_G729 18
```

En la estructura `SdpCodec` se definieron las características propias de G.729 que determinan cómo es la sesión de medios una vez establecida la llamada. La información contenida en esta estructura de datos es la que se utiliza para crear los mensajes SDP enviados durante la negociación SIP. A continuación se muestra un ejemplo de cómo se usó dicha estructura al momento de incorporar G.729 a la fábrica de codecs, `SdpCodecFactory`:

```
case SdpCodec::SDP_CODEC_G729:
        {
           SdpCodec aCodec(SdpCodec::SDP_CODEC_G729,
                           SdpCodec::SDP_CODEC_G729,
                           MIME_TYPE_AUDIO,
                           MIME_SUBTYPE_G729,
                           8000,
                           10000,
                           1,
                           "annexb=no",
                           SdpCodec::SDP_CODEC_CPU_HIGH,
                           SDP_CODEC_BANDWIDTH_LOW);
           addCodec(aCodec);
           aCodec.getMediaType(codecMediaType);
           aCodec.getEncodingName(codecEncodingName);
        }
```

A continuación se describe brevemente cada uno de los parámetros definidos:

Tipo de carga útil: es el valor que representa el campo *payload type* en el encabezado de los paquetes RTP. Para cada tipo de medios este valor se especifica en la RFC 3551[30], y en el caso de G.729 se corresponde con el número 18. En nuestra aplicación, este valor quedó referenciado por la etiqueta `SDP_CODEC_G729`.

Mime Type: con este parámetro se asigna al codec de compresión el medio asociado (audio o video), el cual en este caso corresponde a audio.

Mime Subtype: es el formato que describe el tipo de medios utilizado. En este caso es G.729A.

Frecuencia de muestreo: es la frecuencia de muestro de la señal de entrada al codificador. Es un valor propio del codec G.729 y según la especificación del mismo se corresponde con 8000 Hz.

Cantidad de canales: aquí se especifica la cantidad de canales soportados por el codec utilizado, que queda determinado por la implementación. En este caso ese valor lo determina la aplicación de Voice Age la cual soporta solamente un canal.

Parámetros específicos del codec: en este caso se especifica que el codec G.729 utilizado es sin anexo b.

CPU que se consume: este es un parámetro utilizado a nivel de capa de aplicación para que el agente de usuario sepa de qué recursos debe disponer al querer establecer esta sesión de medios. En el caso de G.729 se consideró conveniente asignarle la categoría `SDP_CODEC_CPU_HIGH` correspondiente con un uso alto de recursos de CPU.

Ancho de banda que consume: con este parámetro es posible categorizar al codec según el ancho de banda consumido. Para esto el programa define tres categorías, y en base a ellas el codec G.729 califica como de bajo ancho de banda. Los tres rangos considerados son:

- `VIDEO_CODEC_BW_LOW`, para codecs que utilizan una tasa de bit de hasta 8 Kbps
- `VIDEO_CODEC_BW_NORMAL`, para codecs que utilizan una tasa de bit de hasta 70 kbps
- `VIDEO_CODEC_BW_HIGH`, para codecs que utilizan tasas de bits hasta 400 Kbps

Para completar los datos correspondientes con estos parámtros se recurrió a las especificaciones del aplicativo de Voice Age, al RFC 3551, y al propio desarrollo de sipXezPhone. Definida la estructura `SdpCodec` para el codec G.729 de Voice Age, la aplicación dispone de toda la información necesaria para formar los mensajes SDP intercambiados durante la negociación de la llamada.

Para completar la integración del codec al desarrollo, también fue necesario modificar adecuadamente la clase `sipXmediaFactoryImpl`. Esta clase contiene la implementación de todo lo que refiere a información de medios como por ejemplo el acceso a algún elemento codec o a características del mismo.

Resultados obtenidos

Realizados los primeros agregados en el código de SIPfoundry, se procedió a probar el comportamiento del aplicativo resultante. Se realizó una llamada utilizando el codec G.729 para observar cómo se llevaba a cabo la señalización SIP.

El primer efecto de las recientes modificaciones, se vio reflejado en la interfaz de usuario a través del menú de configuración que la misma presenta. El teléfono presenta un menú principal de configuración donde es posible setear distintos parámetros que definen cómo es la llamada que se desea realizar. Este presenta seis ítems, entre ellos: *Configuration*, *Audio Settings*, *Video Settings*, *Security Settings*. Al seleccionar cada uno de los ítems, se accede a una ventana específica donde es posible configurar los parámetros relativos al ítem seleccionado. En esta ocasión, el ítem de interés fue el de *Audio Settings* donde se determinan las carcterísticas propias de la sesión de audio.

Al abrir la aplicación, se encontró en el menú de configuración la posibilidad de seleccionar el codec de audio recién integrado, G.729. En la figura 4.3 se ilustra una imagen de la ventana de *Audio Settings* donde se lista el codec G.729. Se observa también que el codec está categorizado como de bajo ancho de banda, como se definió en la estructura `SdpCodec`.

Figura 4.3: Audio Settings

Se verificó el funcionamiento capturando los paquetes SIP que se intercambian entre los agentes de usuario:

```
| 192.168.1.104              192.168.1.103 |
|          INVITE SDP (G.729)              |SIP From: sip:2000 To:sip:p3
|(5060)   ------------------>  (5060)      |
|             100 Trying                   |SIP Status
|(5060)   <------------------  (5060)      |
|             180 Ringing                  |SIP Status
|(5060)   <------------------  (5060)      |
|          200 OK SDP (G.729)              |SIP Status
|(5060)   <------------------  (5060)      |
|                ACK                       |SIP Request
|(5060)   ------------------>  (5060)      |
|                ACK                       |RTP
|(8000)   <------------------  (8000)      |
|                ACK                       |RTP
|(8000)   ------------------>  (8000)      |
|                BYE                       |SIP Request
|(5060)   ------------------>  (5060)      |
|               200 OK                     |SIP Status
|(5060)   <------------------  (5060)      |
```

En el flujo de paquetes descrito se observan por una parte, los cuatro mensajes SIP que permitieron lograr el establecimiento de una sesión de audio con el codec G.729 y por otra los dos paquetes mediante los cuales se dio por finalizada la sesión.

En lo que refiere al establecimiento de la llamada, se envió el mensaje `INVITE` donde se propuso utilizar G.729 como tipo de medios. Por su parte, el agente de usuario remoto actuó según lo esperado enviando los mensajes `100 Trying`, `180 Ringing`, y finalmente el `200 OK` para dar por aceptada la sesión con el codec G.729. Para terminar de concretarse la llamada, el agente de usuario originador envió el mensaje `ACK`.

Se observa que tanto en el mensaje `INVITE` como en el `200 OK` se envió la información del codec negociado. Esto se puede apreciar mejor en el cuerpo SDP de los mensajes, como se ilustra a continuación:

```
Message body
     Session Description Protocol
         v = 0
         o = sipX 5 5 IN IP4 192.168.1.104
         s = call
         c = IN IP4 192.168.1.104
         t = 0 0
         m = audio 8000 RTP/AVP 18 96
         a = rtpmap:18 G.729/8000/1
         a = fmtp:18 annexb=no
         a = rtpmap:96 telephone-event/8000/1
```

Al finalizar la llamada uno de los agentes de usuarios envió el mensaje `BYE` que luego fue aceptado por la otra punta a través del `200 OK`.

Este es el procedimiento que según el protocolo SIP debe seguirse al negociar una sesión de medios. Por tal motivo se consideró que a partir de este momento sipXvideoPhone cuenta con la capacidad de establecer una llamada utilizando el codec de audio G.729.

Cabe señalar que una vez que la sesión quedó establecida, por el canal RTP no se enviaron paquetes. Esto se debió a que si bien la aplicación logró negociar el codec G.729 correctamente, aún no contaba con las funcionalidades para poder transmitir audio en este formato. Esto forma parte de nuestro próximo objetivo en el desarrollo de sipXvideoPhone.

4.4.2. Implementación del codificador y decodificador

Una vez implementada la negociación de la llamada, el próximo paso a seguir es el desarrollo de lo necesario para que la aplicación sipXvideoPhone disponga de transmisión de paquetes RTP con el codec de audio recién incorporado. Para que dos agentes de usuario puedan establecer una conversación de voz de manera adecuada, es necesario que los datos de audio sean paquetizados correctamente en el origen y luego correctamente reconstruidos en destino.

El codec de compresión disponible en la aplicación original para la transmisión de audio es G.711 y en esta oportunidad se agrega soporte para G.729. Si bien los codecs de compresión son diferentes, el flujo de datos podría recibir el mismo procesamiento en gran parte del camino.

Se analizó cómo se lleva a cabo el procesamiento de los datos de audio cuando son G.711 para así localizar aquellos puntos del programa en los que resulta necesario diferenciar a qué codec de audio corresponden. Se identificaron dos etapas de procesamiento en las que los datos de audio deben recorrer caminos paralelos según si se trata de una sesión de medios con G.711 o con G.729. Estas etapas corresponden, como era previsible, a la codificación y decodificación. Con esto el camino de procesamiento queda dividido en tramos como se observa en las figuras 4.5 y 4.4.

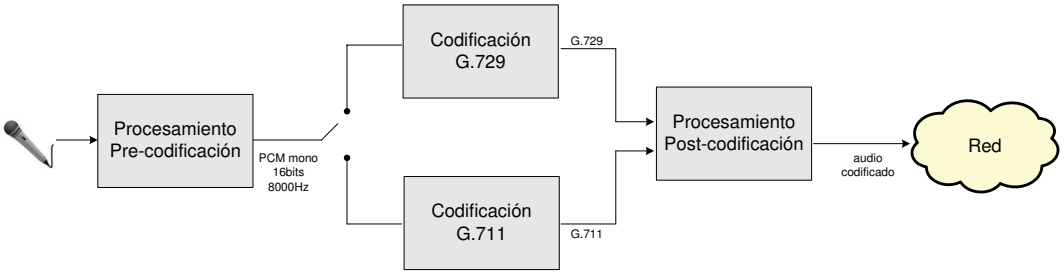


Figura 4.4: Etapas en la transmisión de datos

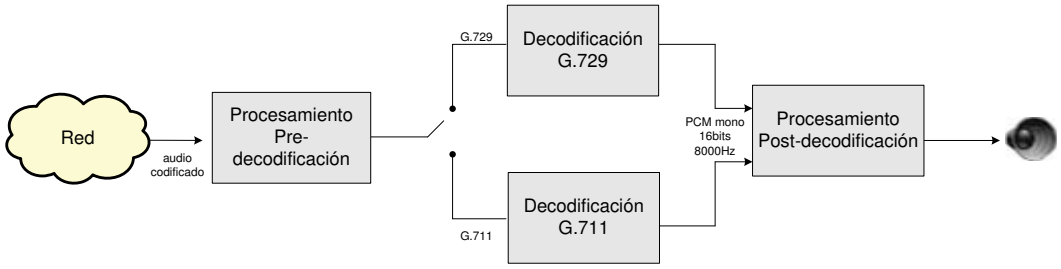


Figura 4.5: Etapas en la recepción de datos

En la transmisión de los datos:

1. Desde que el micrófono recibe la señal de voz hasta que los datos de audio están prontos para ser codificados.

   En el programa original de SIPfoundry, para todo este tramo de procesamiento el flujo de audio se encuentra en formato PCM 16 bits mono, muestreado a una frecuencia de 8000Hz. Dadas las características del aplicativo de Voice Age este es el formato requerido a la entrada del codificador. Por este motivo resulta conveniente que la información de audio siga este camino hasta el momento justo antes de ingresar al codificador, independientemente del codec utilizado.

   En el caso que se utilice un codificador que requiera los datos de entrada en otro formato, podría igualmente utilizarse el mismo camino y realizar la conversión al formato requerido directamente dentro del codificador.

2. Durante la codificación de la señal de audio.
   Esta segunda etapa de procesamiento resulta específica para cada codec de compresión, lo que resulta en la necesidad de crear un objeto codificador que permita modelar el codec de compresión negociado en la llamada. Una vez que los datos se encuentran en esta etapa del procesamiento, se codifican según el codec correspondiente.

3. Luego que se obtienen datos codificados hasta que son enviados por la red.
   En este tercer tramo de procesamiento se dispone de paquetes con información de audio codificado que deben ser enviados por la red. En esta etapa del camino los datos de audio ya se encuentran paquetizados de alguna forma. Esto permite que independientemente del codec utilizado todos los datos puedan seguir el mismo camino hasta llegar a la red. Aquí se elimina la necesidad de implementar procesamientos alternativos según el codec de compresión utilizado en la sesión.

En la recepción de los datos:

1. Desde que se reciben los datos provenientes de la red hasta el momento previo a la decodificación.
   Aquí se reciben los paquetes RTP de la red y su contenido debe ser procesado hasta el momento en que se invoca al decodificador. Dado que la decodificación aún no se ha llevado a cabo, el procesamietno de los datos se puede llevar a cabo sin tener conocimiento del tipo de información que cada paquete RTP contiene. Esto resulta en que todos los paquetes puedan seguir el mismo camino de procesamiento sin considerar a qué codec de compresión corresponden.

2. Durante la decodificación de la señal de audio.
   Al momento de decodificar los paquetes de audio, se hace necesario que el flujo de información tome distintos rumbos correspondientes con el codec de compresión que se está utilizando en la sesión. Si los datos recibidos se encuentran codificados en G.729, estos deben ingresar a un objeto decodificador del tipo G.729. Análogamente para el caso del codec G.711.

3. Desde que los datos son decodificados hasta que son reproducidos en el parlante.
   En la aplicación original de SIPfoundry, el procesamiento post-decodificación se lleva a cabo para señales de audio en formato PCM 16 bits mono, muestreada a una frecuencia de 8000Hz. Este es el tipo de señal obtenido a la salida del decodificador G.729 de Voice Age. De esta manera se conlcuye que para ambos codecs, el camino de procesamiento seguido por la señal de audio decodificada puede ser el mismo.

A modo resumen, las únicas etapas de procesamiento que necesariamente deben diferenciarse según el codec de compresión utilizado son la codificación y la decodificación. Por el momento sipXvideoPhone presenta disponibilidad de realizar estas etapas de procesamiento solamente para el caso en que el codec de compresión utilizado sea G.711. Con la ayuda del aplicativo de Voice Age se procede a desarrollar las etapas de codificación y decodificación para el caso en que la sesión de medios se corresponda con el nuevo codec incorporado G.729.

Se desarrollaron clases que permitierib hacer uso de las herramientas proporcionadas por la librería de Voice Age. Para ello se implementaron las clases *codificadorG729* y *decodificadorG729* que modelan los bloques ilustrados en las figuras 4.4 y 4.5 respectivamente.

En estas clases se invocan los métodos disponibles en el módulo de Voice Age de manera adecuada y sin perder de vista la compatibilidad con el resto de la aplicación sipXvideoPhone. El objetivo de estas clases es que permitan modelar al nuevo codec de compresión G.729 al momento de requerir el procesamiento de codificación ó decodificación de una señal de audio.

Siguiendo el lineamiento del código original de SIPfoundry, los nombres elegidos para el codificador y decodificador son `MpeG729` y `MpdG729` respectivamente. A continuación se explica el funcionamiento de cada una de estas clases y luego se profundiza en su interacción con el resto de la aplicación.

MpeG729

En base al análisis realizado anteriormente, se implementó la clase `MpeG729` que modela el comportamiento del codificador G.729. Esta clase hace uso de las herramientas codificadoras de Voice Age, adaptándolas a la aplicación sipXvideoPhone de manera adecuada.

Para incorporar en sipXvideoPhone una nueva clase codificador, fue necesario utilizar correctamente la arquitectura ya existente en la aplicación. Como se mencionó, la arquitectura de diseño del código original hace gran uso polimorfismos. En el caso particular de los objetos codificadores, la aplicación de SIPfoundry cuenta con una clase padre denominada `MpEncoderBase` y de la cual deben heredar todos los codificadores de audio que presentes el programa.

En consecuencia, la nueva clase `MpeG729` debe heredar de `MpEncoderBase` y con ella todos sus métodos.

Esta clase recibe paquetes de audio sin codificar de 10ms de duración. Junto con la utilización de las distintas funciones disponibles en la API de Voice Age codifica estas muestras y obtiene un paquete de datos codificados de 10 bytes de tamaño.

Aquí se crea también una estructura estática a través de la cual se definen las características propias del codec de compresión. Estas se observan el el cuadro 4.2

MpdG729

Análogamente se procedió a implementar la clase `MpdG729` que modela el decodifcador G.729. Esta clase actúa de interfaz entre la librería de Voice Age y la aplicación sipXvideoPhone.

Nuevamente se debió seguir el lineamiento del código de SIPfoundry y respetar los polimorfismos. En el caso del decodificador, la aplicación cuenta con una clase padre denominada

Parámetro	Valor
versión del codec	G.729
frecuencia de muestreo de los datos PCM esperados	8000Hz
cantidad de canales soportados por el codec	1
tasa de bits del codec	8 kbps
cantidad de muestras PCM en el paquete entrante	80
cantidad de bits en un paquete codificado	80 bits

Cuadro 4.2: Características del codec

`MpDecoderBase` de la cual deben heredan todos los elementos decodificadores de audio presentes en el programa.

Esta clase es quien realiza la decodificación del audio recibido en formato G.729. Actúa de interfaz entre los métodos proporcionados por la librería de Voice Age, y el resto de la aplicación. Aquí se recibe un paquete con datos de audio (80 bytes) codificado según la recomendación G.729 y luego de procesarlo se obtiene un paquete de audio en formato PCM de 10ms de duración.

También se inicializa la instancia del *jitter buffer* [3]utilizado en la recepción. Este buffer permite almacenar los paquetes recibidos por la red para luego enviarlos a decodificar cada intervalos fijos de tiempo. De esta manera se logra reproducir el audio a una cadencia fija.

Al igual que en la clase codificadora, aquí se define también una estructura de datos que permite caracterizar los parámetros de interés en el codec G.729.

Interacción de MpeG729 y MpdG729 con la aplicación

Para comprender cómo las clases `MpeG729` y `MpdG729` se integran con el resto de la aplicación sipXvideoPhone es necesario dar una descripción del conjunto de clases que interactúa con ellas. Estas son:

`MprEncode`: es una clase que recibe los datos de audio previamente capturados por el micrófono y los envía al codificador `MpeG729` para que los procese ejecutando el método `encode()`. Una vez obtenidos los datos codificados son entregados a la clase `MprToNet`.

`MprToNet`: esta clase es la encargada de armar los paquetes RTP para enviar por la red. Utilizando los datos de audio codificados como carga útil, se agrega el encabezado RTP correspondiente y se envía el paquete a la red.

[3]Se dice que existe *jitter* cuando hay variaciones en los tiempos entre arribos de paquetes, lo que puede ocurrir por motivos como la congestión de la red.

Si bien esta clase no tiene una relación directa con `MpeG729`, igualmente se encuentra vinculada a través de la clase `MprEncode` mencionada en el punto anterior.

`MpJitterBuffer`: esta clase tiene una relación directa con el objeto decodificador ya que es quien se encarga de invocar al método `decode()` presente en `MpdG729`. A través de este método se le entrega el paquete RTP recibido al decodificador para que obtenga las muestras decodificadas. Estas muestras de audio quedan guardadas en un arreglo específico que oficia de *jitter buffer*.

`MpCodecFactory`: aquí se crean las instancias de todos los codificadores y decodificadores soportados por la aplicación. Estos objetos quedan almacenados en un arreglo con el objetivo de poder ser utilizados en otras partes de la aplicación cuando así se requiera.

En las figuras 4.6 y 4.7 se ilustra la relación que tienen el codificador y decodificador con las clases recién mencionadas.

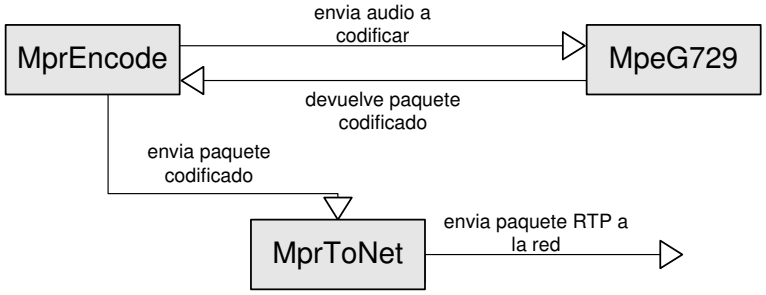


Figura 4.6: Codificador de Audio

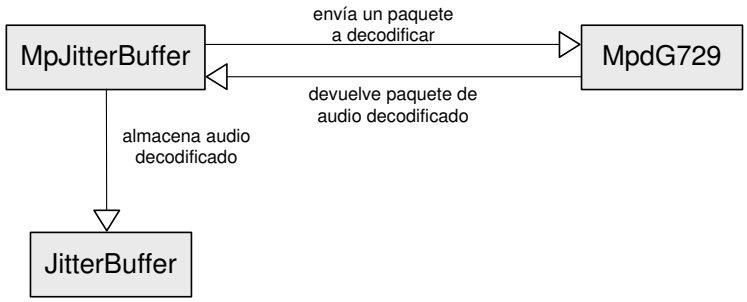


Figura 4.7: Decodificador de Audio

4.4.3. Encapsulado de G.729 en RTP

Al incorporar el codec de audio a la aplicación, fue necesario implementar no solamente las clases que modelen el codec de compresión, sino también todo aquello que permite lograr la correcta transmisión de este nuevo flujo de audio en RTP.

La transmisión en RTP de un flujo de audio codificado en G.729 se describe en la RFC 3551 del IETF. En el se definen los tipos de carga útil del encabezado RTP según el tipo de medios, ya sea audio o video. A su vez se especifica para algunos tipos de medios, la manera en que los datos deben ser transportados sobre RTP. En particular se especifica para el codec de compresión G.729.

En base a lo especificado en la RFC 3551, se debe encapsular el flujo de audio G.729 en paquetes RTP con un tamaño de carga útil de 20ms. También es aceptado encapsular de a 10 ms, si así resulta más conveniente. En el caso de sipXvideoPhone se optó por trabajar con paquetes RTP de carga útil 10ms por motivos de implementación. Si luego al probar el programa se encontraba algún tipo de problema con este encapsulado, se procedería a utilizar un mayor tamaño de carga útil por paquete RTP (20ms).

4.5. Pruebas y compatibilidad con otro teléfono

Para finalizar con el agregado del codec G.729 a sipXvideoPhone se quiso probar su comportamiento con otros teléfonos proporcionados por terceros. Para esto fue necesario encontrar una aplicación SIP con la cual fuera posible establecer sesiones de audio G.729, y que además pudiera obtenerse gratuitamente.

Se logró acceder a una única aplicación con estas características. Este es el teléfono WWTelcoPhone que se ilustra en la figura 4.8

Se realizó una llamada de audio entre la aplicación sipXvideoPhone y WWTelcoPhone, en la que se utilizó G.729 como tipo de medios. Esta prueba no resultó del todo satisfactoria, pues se encontró que el audio se lograba escuchar correctamente sólo en un sentido.

La información de audio transmitida por el agente de usuario sipXvideoPhone no resultaba audible en el agente de usuario WWTelcoPhone. En contraprte, el audio transmitido desde el aplicativo WWTelcoPhone llegaba correctamente a la punta remota, siendo reproducida por los parlantes asociados al teléfono sipXvideoPhone.

Frente a este comportamiento se procedió a estudiar el contenido de los paquetes RTP transmitidos en ambos sentidos encontrando diferencias en el tamaño de la carga útil enviada por cada una de las puntas.

Figura 4.8: Interfaz gráfica de WWTelcoPhone

Los paquetes enviados por el agente de usuario sipXvideoPhone presentaban un tamaño de 66 bytes (incluyendo los encabezados IP, UDP y RTP), mientras que los enviados por WWTelcoPhoneX presentaban 64 bytes. Esta diferencia de dos bytes era consecuencia de los bytes de *padding* que sipXvideoPhone estaba agregando al final de cada paquete RTP. Estos bytes únicamente actúan como relleno y tienen valor cero, salvo el último byte que indica la cantidad de bytes que se agregaron de relleno.

Se intentó localizar en el código de SIPfoundry el punto en que estos bytes de *padding* se estaban agregando a la información enviada, encontrando que esto se realiza en el método `writeRtp` de la clase `MprToNet`. Como se explicó anteriormente, este método es el encargado de armar los paquetes RTP que luego son enviados a la red.

La presencia de bytes de *padding* en los paquetes RTP no es obligatoria. Generalmente se utiliza para lograr un largo de paquete RTP múltiplo de algún número. En el caso de la aplicación se estaba agregando bytes de relleno para generar paquetes RTP que tuvieran un largo múltiplo de cuatro bytes. En caso que se desee mayor seguridad en la transmisión de datos una opción es encriptar los mismos. Para hacerlo correctamente es que se utiliza la estrategia de bytes de relleno descrita.

La aplicación actualmente no envía los datos encriptados ni permite al usuario optar por ello. Dado esto se decidió eliminar el *padding* al momento de armar el paquete RTP. Con esto se volvió a probar la aplicación en una llamada hacia un agente de usuario WWTelcoPhone, obteniendo que los teléfonos se comunicaban sin ningún problema.

SipXvideoPhone logró comunicarse de manera adecuada con un teléfono proporcioando por terceros, reflejando que:

- La codificación G.729 se realiza de manera correcta según la recomendación del codec.
- El desarrollo para integrar el nuevo codec de audio fue satisfactorio.

4.6. Resultados y conclusiones

A modo resumen, se considera que se logró integrar adecuadamante el codec de audio G.729 a la aplicación sipXvideoPhone.

Durante la búsqueda de una librería externa que prorcione las herramientas de codificación y decodificación G.729 se encotró solamente una opción, el aplicativo de Voice Age. Dado que no fue posible probarlo en su totalidad, fue necesario confiar en la correcta implementación del estándar G.729. Con esta premisa, se procedió a integrar el módulo a la aplicación obteniendo resultados exitosos por parte del mismo.

Para integrar el codec de compresión se consideró conveniente comenzar por implementar la señalización SIP de la llamada. Aquí se hizo uso de los conceptos teóricos del protocolo SIP así como también de las características propias que definen al codec de audio que se deseaba integrar.

Una vez lograda la negociación de la llamada, se integró la librería de Voice Age al desarrollo. Se buscó disponer de los objetos codificador y decodificador para realizar las tareas correspondientes con el codec de audio G.729. Para esto se analizó el desarrollo del programa original de SIPfoundry, y en base a ello se implementaron aquellas clases que resultaron necesarias para disponer de un flujo RTP de audio G.729. Estas clases a su vez actúan de interfaz entre la aplicación sipXvideoPhone y las herramientas proporcionadas por Voice Age.

Luego de incorporar el codec G.729 a la plataforma, se realizaron las pruebas necesarias que permitieron validar su comportamiento. Estas consistieron en realizar llamadas entre dos agentes de usuario y verificar que el flujo de audio originado en una punta fuera reproducido de manera correcta en la punta remota. Las pruebas se hicieron no solamente entre agentes de usuario sipXvideoPhone, sino también entre uno sipXvideoPhone y uno proporcionado por terceros. En ambos casos se obtuvieron resultados satisfactorios con lo cual se dio por finalizada la incorporación del nuevo codec de audio a la aplicación.

Con el desarrollo realizado en esta etapa del trabajo, sipXvideoPhone presenta la posibilidad de establecer una llamada de audio utilizando G.729 como tipo de medios. El aplicativo no solamente es capaz de comunicarse con un agente de usuario igual sino también con otros agentes de usuario SIP que soporten sesiones de medios con G.729.

Capítulo 5

Agregado de soporte de video

5.1. Introducción

El trabajo realizado hasta el momento permitió una familiarización con la solución que se tomó como punto de partida, lo que sirvió como base para continuar con el desarrollo del proyecto y alcanzar el siguiente objetivo: la incorporación de una sesión de video sobre sipXvideoPhone.

El trabajo se dividió en etapas de manera de ir alcanzando pequeñas metas que finalmente condujeran al objetivo planteado. En este capítulo se presenta el desarrollo que permitió incorporar la transmisión de un flujo de video a la aplicación.

En primer lugar se detallan los conceptos teóricos del codec elegido para la transmisión de video, MPEG-1, y se explica cómo se realiza el encapsulado en RTP para poder transportar flujo de video de por la red. Se presenta también una breve descripción de otros codecs de compresión analizados: Theora y H.263.

Seguidamente se presenta el criterio de elección para la plataforma desarrollada. Se detalla el modelo de arquitectura elegido así como también las características del flujo de video soportado. Se realiza un análisis de los codecs de compresión analizados y los motivos que llevaron a elegir MPEG-1 para incorporar a la aplicación.

Luego se describe cómo se implementó la negociación SIP de una llamada de video entre dos agentes de usuario sipXvideoPhone.

Posteriormente se detalla la arquitectura base elegida para incorporar video a la aplicación, mencionando los componentes fundamentales presentes en dicha arquitectura. Se explica el desarrollo que permitió disponer a la aplicación de ventanas de visualización y previsualización de las imágenes de video.

Una vez expuesta la arquitectura básica de sipXvideoPhone y las interfaces para visualizar

las imágenes de video, se describe cómo se procedió para incorporar el codec de compresión a la plataforma básica. Aquí se incluye la integración de una librería adicional que proporciona herramientas de codificación y decodificación MPEG-1. Se describen las clases codificador y decodificador que modelan el codec de video y finalmente se realiza un detalle de las etapas de procesamiento que recibe el flujo de video cuando éste es MPEG-1.

Para concluir se presentan los resultados obtenidos luego de haber incorporado la transmisión de video al teléfono sipXvideoPhone.

5.2. Conceptos preliminares sobre codecs de video

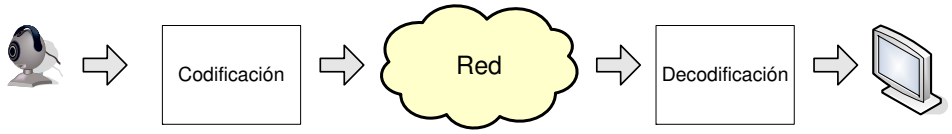


Figura 5.1: Sistema de codificación

Para llevar a cabo la compresión de video, existen estándares que definen la forma de realizarse. Actualmente se dispone de varios codecs de compresión, cada uno de los cuales fue diseñado pensando en un tipo de aplicación particular. De esta manera es posible que un codec que resulta muy eficiente en algunas situaciones, no lo sea en otras.

Los codecs de compresión utilizan ciertas técnicas que permiten reducir el ancho de banda consumido al momento de transportar señales de video. A continuación se hará una breve descripción de los codecs de video analizados en esta oportunidad, y qué técnicas de compresión utiliza cada uno. Para lograr una mejor comprensión de las técnicas en las que se basan los codecs de compresión, se presentan algunos conceptos que serán utilizados luego:

Algoritmo de compresión con pérdidas: es una técnica de compresión mediante la cual se hace imposible reconstruir la información original. Cuando se utiliza un codec de compresión con pérdidas, la información reconstruida en el decodficador es siempre una aproximación de la información que originalmente fue codificada. Existen dos mecanismos para realizar este tipo de compresión:

- por codecs de transformación, donde los datos se transforman simplificando la información.
- por codecs predictivos, donde se tiene en cuenta la naturaleza de la señal para predecir su comportamiento y de esta manera enviar únicamente las diferencias con otras muestras. Para audio una muestra es lo que se denomina cuadro y para video es lo que se denomina imagen.

Escaneo progresivo (Progressive scann): es una metodología utilizada en la transmisión de video en la que cada línea de cada imagen es desplegada a continuación de la otra. Las imágenes son recorridas de izquierda a derecha desde la línea superior a la línea inferior. En contraparte, existe la transmisión de video entrelazado.

Escaneo entrelazado (Interleaved scann): aquí la transmisión de cada imagen se lleva a cabo en dos tandas. Primero se transmiten las líneas impares desde el comienzo hasta el final, y luego se recorre la imagen nuevamente pero esta vez transmitiendo las líneas pares. Esta técnica se utiliza comúnmente en aplicaciones de televisión analógica.

Codificación entrópica: este tipo de codificación tiene en cuenta la frecuencia de aparición de cada componente de frecuencia al momento de asignar los códigos. Muchas veces esto es utilizado para asignar códigos más largos a las componentes de frecuencia que aparecen menos, y códigos más cortos a aquellos componentes que tienen más tendencia a aparecer.

Compensación de movimiento: esta técnica tiene como objetivo eliminar la redundancia temporal entre imágenes dinámicas de una misma secuencia de video, logrando mayor nivel de compresión. Para esto se utilizan vectores de movimiento que estiman el movimiento que sufre cada píxel entre una imagen y la siguiente. De esta manera, al decodificador debe entregársele el vector calculado y la diferencia entre los valores de los píxeles.

Representación del color: cada píxel de una imagen puede ser representado de distintas formas según el espacio de colores elegido. Los espacios de colores representan un modelo con el cual es posible describir un color utilizando un conjunto de parámetros independientes. Dentro de los espacios más conocidos se encuentran RGB, HUV, CMYK y YC_bC_r. En lo que refiere a compresión de video digital es YC_bC_r el más utilizado.

5.3. MPEG-1

5.3.1. Conceptos teóricos de MPEG-1

MPEG es un acrónimo de *Moving Pictures Expert Group* formado por ISO (International Standard Organization). El estándar MPEG-1 está descrito en la norma ISO/IEC 11172 [31]. Fue desarrollado para la compresión y transmisión de audio y video y se compone de varias partes; la parte 2 de la norma es la que describe acerca del video.

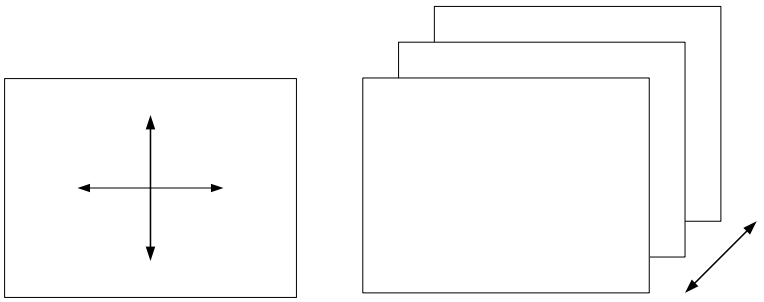
MPEG es un sistema asimétrico ya que el codificador es mucho más complejo que el decodificador [32]. El codificador utiliza algoritmos complicados mientras que el decodificador simplemente realiza funciones fijas. Esto es muy útil al trabajar con *broadcasting* ya que se usan unos pocos codificadores caros y muchos decodificadores simples. Para aplicaciones como las del video teléfono no se aprovecha esta ventaja de MPEG.

En el estándar de ISO se definen características del codificador pero el mismo no está completamente especificado. Lo que sí se encuentra estandarizado es el decodificador.

La técnica de codificación que utiliza MPEG lo categoriza como un codec de compresión con pérdidas. Luego de decodificar se obtienen datos que no son idénticos bit a bit a los originales antes de codificar. Este tipo de codecs se basa en la percepción de la vista para el caso del video y en la percepción acústica en el caso del audio. Por este motivo reciben el nombre de códigos perceptivos.

Generalidades del algoritmo de compresión de video

La compresión de video se aplica a las dos dimensiones de cada imagen y a su vez se hace una compresión temporal entre imágenes. La compresión *Intra* utiliza redundancia dentro de una imagen y la compresión *Inter* utiliza redundancia entre imágenes.



(a) Compresión Intra (b) Compresión Inter

Figura 5.2: Redundancia espacial y temporal

Este codec se caracteriza por:

- Gran compresión preservando la calidad de la imagen.
- Es con pérdidas, ya que el valor exacto de cada píxel no se conserva en la codificación.
- Codificación *Intra* e *Inter* cuadros.
- Compensación de movimiento causal y no causal (interpolación).
- Vectores de movimiento para cada región de 16 x 16 de la imagen.
- La señal diferencia (error de predicción) se comprime usando la transformada discreta del coseno (DCT) para eliminar la correlación espacial.
- Los vectores de movimiento se combinan con la información residual DCT y se transmiten usando códigos de largo variable (VLC) que aprovechan la redundancia estadística.
- Ofrece una tasa de transferencia máxima de 1.856 Mbps.

En la figura 5.3 se muestra la clasificación de las distintas técnicas empleadas en MPEG-1 en lo que refiere a la introducción de pérdidas en el algoritmo.

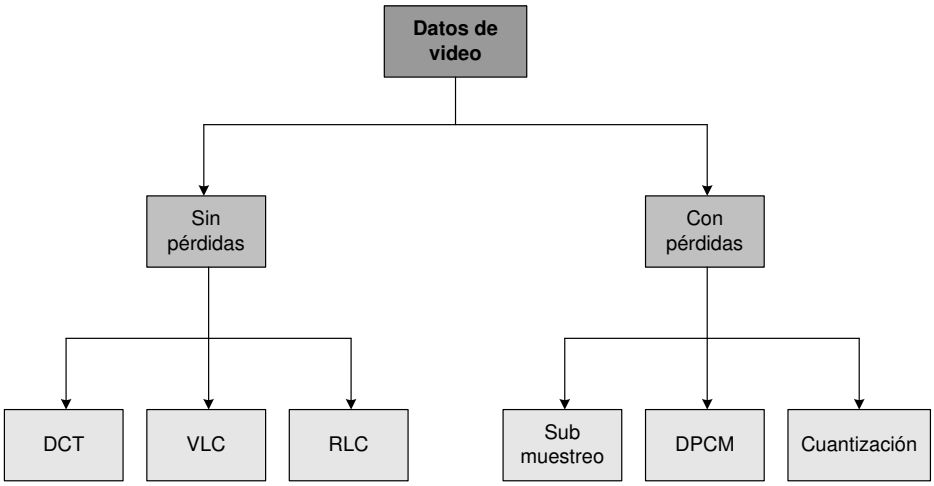


Figura 5.3: Técnicas

Como uno de los usos de MPEG es el almacenamiento en disco, es necesario tener puntos de acceso aleatorios a los datos. Estos puntos vienen dados por las imágenes *Intra* ya que no necesitan de otras imágenes para ser decodificadas. Existe un compromiso entre el acceso aleatorio a los datos y el nivel de compresión. Debido a que las imágenes *Intra* usan únicamente la redundancia espacial, requieren más bits para su codificación que las imágenes *Inter*. Entonces para las tasas de bit deseadas si se desea tener una buena calidad de video se deben comprimir mucho las imágenes por lo que se deben usar imágenes *Intra*.

Tipos de imágenes

Las imágenes se clasifican en tres tipos: imágenes Intra (Imágenes I), imágenes predecidas (Imágenes P) e imágenes predecidas bidireccionalmente (Imágenes B).

Las imágenes Intra (I) se codifican sin usar otras imágenes, se usa solamente la redundancia espacial. Proveen puntos de acceso en la secuencia codificada. Las imágenes predecidas (P) son codificadas más eficientemente usando compensación de movimiento prediciendo con una imagen Intra anterior o con otra imagen P. Se manda la diferencia entre la imagen de referencia y la actual. Se usan como referencia para predicción. Las imágenes predecidas bidireccionalmente (B) son codificadas usando como referencia imágenes pasadas y futuras. Nunca se usan como referencia para predicción.

Los distintos tipos de imágenes se repiten en secuencias llamadas grupos de imágenes (GOP). El orden de visualización suele ser distinto al orden en el tren de bits codificados.

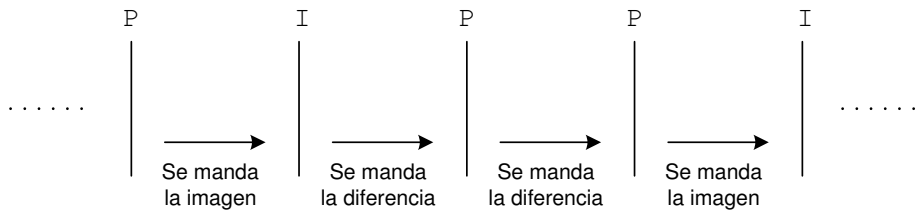


Figura 5.4: Envío de datos en una secuencia

Si el orden de visualización es: B1 B2 I3 B4 B5 P6, el orden de transmisión es: I3 B1 B2 P6 B4 B5. Es necesario que se manden antes todas las imágenes I o P utilizadas para deducir las B.

Un grupo de imágenes se puede describir por dos parámetros: N que es la cantidad de imágenes del GOP y M que es el espaciado entre imágenes P. Para el ejemplo anterior, $N = 6$ y $M = 3$.

Existe una relación entre la compresión lograda por cada tipo de imagen. Las imágenes I son aproximadamente 50 por ciento más grandes que las P que a su vez son 50 por ciento más grandes que las B.

Compensación de movimiento y redundancia espacial

Para la compensación de movimiento se toman áreas de la imagen de 16 x 16 píxeles llamadas macrobloques. Cada macrobloque es de un cierto tipo, puede ser I, P ó B.

Tanto las imágenes originales como las señales de error tienen gran redundancia temporal. El estándar usa DCT y una ponderación en la cuantificación. Luego de la predicción o interpolación, la imagen residual se divide en bloques de 8 x 8 píxeles. Éstos se transforman con la DCT donde se pesan y cuantizan. Debido a la cuantización muchos de los coeficientes resultan cero. Por este motivo se emplea la codificación *Run-Length Codign* o RLC y *Variable Length Coding* o VLC para codificar los coeficientes que se detallan más adelante.

Codificación

El estándar no especifica un proceso de codificación sino que especifica la sintaxis y semántica del tren de bits y el procesamiento de la señal en el codificador.

La señal de entrada al codificador viene dada por la señal de luminancia y dos señales diferencia de color (Y, Cr, Cb). El estándar requiere que Cr y Cb sean submuestreados respecto a la luminancia en una relación 2 a 1, tanto vertical como horizontalmente. Se usa el muestreo 4:2:0, cuatro muestras de luminancia, una de croma rojo y otra de croma azul. Este submuestreo de las componentes de color introduce pérdidas en los datos.

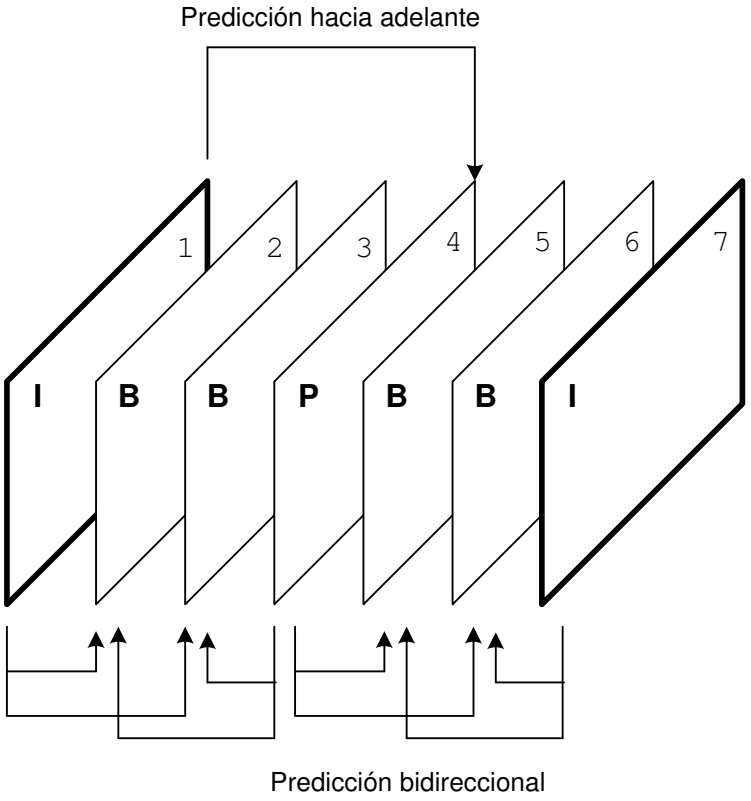


Figura 5.5: Grupo de imágenes

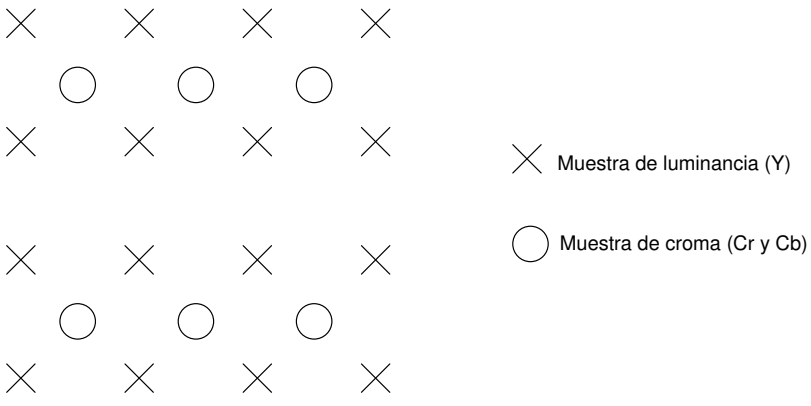


Figura 5.6: Muestreo 4:2:0

Una forma de tomar las muestras de croma es hacer el promedio de los cuatro píxeles que representa dicha muestra.

Si se usan imágenes imágenes B, es necesario reordenar la salida de las imágenes del codificador. Como se vio en la sección anterior se deben mandar primero las imágenes de referencia necesarias para decodificar las predecidas.

La unidad básica de codificación es el macrobloque. Éstos se codifican en secuencia, de izquierda a derecha y de abajo hacia arriba. Cada macrobloque consiste en seis bloques de 8x8; cuatro de luminancia, uno de la componente de croma azul y otro de la componente de croma roja. El área cubierta por cada bloque es la misma.

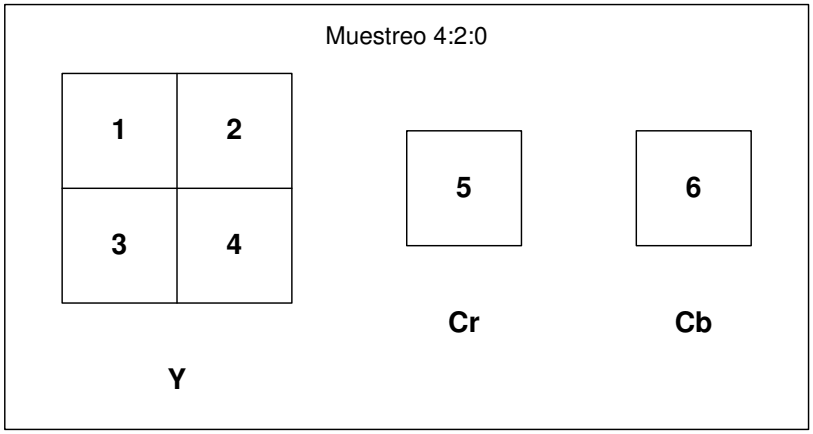


Figura 5.7: Bloques de un macrobloque

El primer paso del proceso de codificación es elegir para cada macrobloque el modo de codificación que depende del tipo de imagen y de otros factores. Segundo dependiendo del modo de codificación se forma una predicción de la compensación de movimiento del bloque basada en imágenes pasadas y/o futuras. La predicción se le resta al macrobloque actual. A los seis bloques de 8 x 8 que forman el macrobloque (4 de luminancia y 2 de croma) se les aplica la transformada discreta del coseno. Los bloques 8x8 con los coeficientes DCT se convierten en vectores de una dimensión (se recorren en zig-zag, figura 5.9). Estos vectores son codificados mediante códigos de ráfagas (Run-Length Coding, RLC) y por último para reducir aún más la información a enviar se codifican usando tablas de códigos de largo variable (Variable Length Coding - VLC).

Tanto la codificación de largo variable como la codificación de ráfaga y la transformada discreta del coseno son técnicas sin pérdidas. Las pérdidas ocasionadas en la codificación se deben al submuestreo y a la cuantización.

La cuantización se hace de acuerdo a tablas con coeficientes de 8x8. Las tablas que se usan por defecto son las de JPEG que son distintas para la luminancia y la croma.

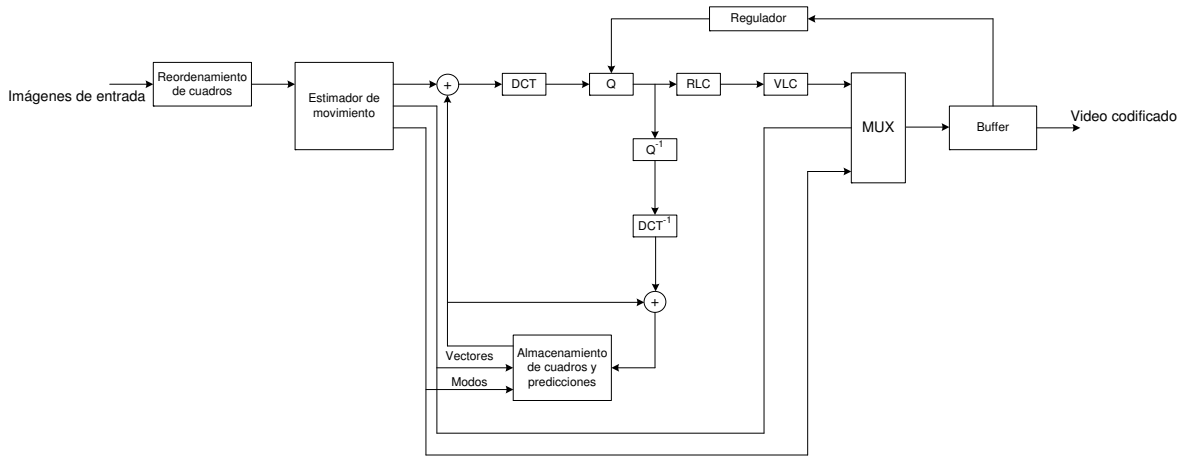


Figura 5.8: Modelo simplificado del codificador MPEG-1

La codificación intra se hace sobre los datos de la imagen en las imágenes I y sobre los errores de predicción en las imágenes P y B.

La transformada discreta del coseno usa características de la percepción humana a las distintas frecuencias espaciales para asignarles pesos. De esta forma se puede lograr una codificación eficiente. La componente dominante es la componente de continua y tiene un valor positivo. El incremento de la componente de continua hace que se eleve el valor de luminancia de cada píxel.

En sí la transformada discreta del coseno no brinda compresión sino que logra, luego de aplicarla, que la mayoría de los coeficientes quedan en cero por lo que combinándola con códigos de ráfagas se reducen en gran medida los datos a enviar.

Los códigos de ráfagas (RLC) son útiles cuando se quiere codificar una secuencia de datos que presenta un valor con gran probabilidad de ocurrencia [33]. La forma de codificar es guardar el valor y una cuenta que indica el tamaño de la racha (cantidad de veces que se repitió ese valor). En este caso se tendrá gran cantidad de ceros por lo que se cuenta la cantidad de ceros entre valores distintos de cero.

Si luego de recorrer la matriz de coeficientes DCT en zig-zag quedan los siugientes valores: 25 11 9 0 0 0 0 0 0 0 3 0 0 0 0 0 0 -1. Se codifican como (25);(0,11);(0,9);(7,3);(5,-1); EOB. El primer número representa la cantidad de ceros que hay entre el valor anterior distinto de cero y el segundo es el valor que viene después de la racha de ceros. El *End Of Block* indica el final de los valores distintos de cero en el bloque 8x8.

Los códigos de largo variable utilizan la probabilidad de aparición de determinadas secuencias para codificar con palabras más cortas a los valores más probables [34]. Para esto se usan

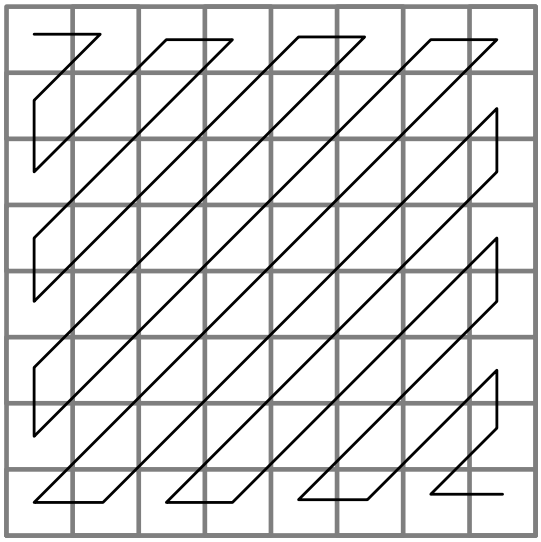


Figura 5.9: Recorrido en zig-zag de bloques para aplicar la DCT

las tablas de Huffman.

Decodificación

Esta norma define el proceso de decodificación, el cual resulta mucho más simple que el de codificación. Se puede resumir en los siguientes pasos:

- Para tasas constantes el decodificador llena una cola FIFO con los datos codificados.
- El decodificador lee el tren de bits codificado del buffer e identifica el comienzo y tipo de imagen.
- Lee cada macrobloque identificando el tipo y los vectores de movimiento (si los hay).
- Utiliza los macrobloques leídos para predecir el macrobloque actual.
- Se aplica el decodificador VLC, la inversa de cuantización y la transformada inversa del coseno a cada macrobloque.
- Luego de que se tienen todos los macrobloques de una imagen, si ésta es I o P es una referencia para futuras imágenes por lo que es almacenada remplazando a la referencia anterior.
- Por último se reordenan las imágenes para su presentación.

En la figura 5.10 se muestra el modelo simplificado de un decodificador. Pueden haber otros decodificadores con distintas arquitecturas.

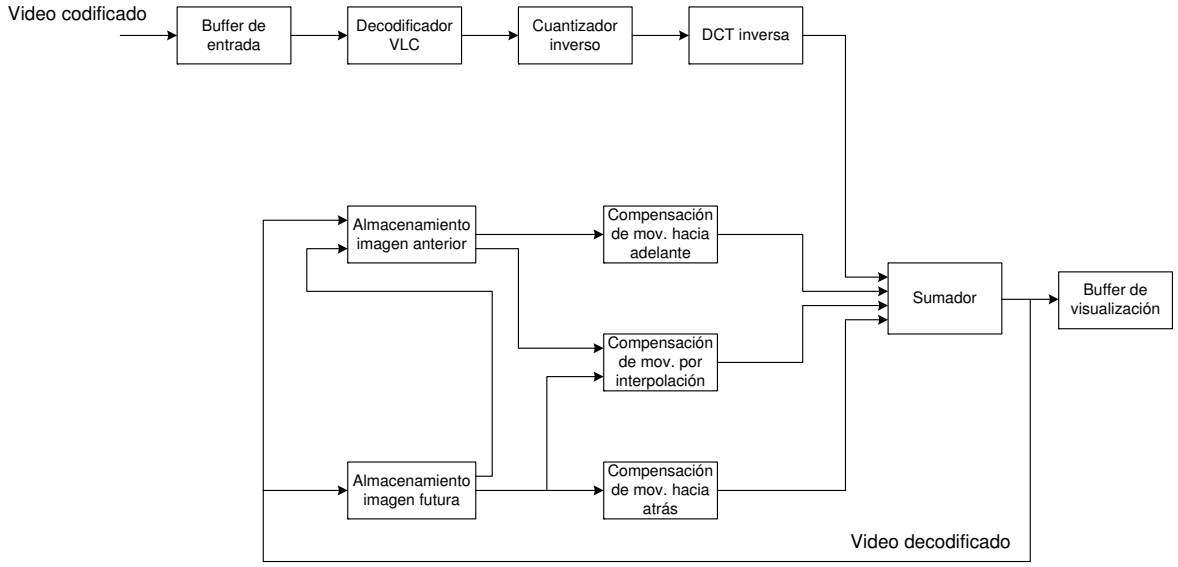


Figura 5.10: Modelo simplificado del decodificador MPEG-1

Jerarquía del tren de bits

El estándar de MPEG-1 establece jerarquías para procesar los datosde video. A tales efectos define un modelo de capas que presentan distintas funcionalidades. En la figrua 5.11 se puede observar la jerarquía de las capas MPEG-1 [35].

***Capa de secuencia (Video Sequence Layer)*:** esta es la capa superior e incluye un encabezado y un grupo de imágenes (GOP - *Group Of Pictures*). El encabezado de secuencia inicializa el estado del decodificador, así se puede decodificar sin importar el pasado.

Capa de grupo de imágenes (Group Of Pictures Layer, GOP): originalmente un GOP fue pensado como la mínima unidad independiente para decodificación. Como MPEG-1 fue diseñado principalmente para el almacenamiento de datos, es importante tener puntos de acceso bastante seguido en el stream, por tal motivo éste es el punto de acceso aleatorio. Es la unidad más pequeña dentro de la secuencia que es independiente para la decodificación. Consiste en un encabezado y algunas imágenes. El encabezado GOP contiene tiempo e información de edición.

Capa de imagen (Picture Layer): representa un cuadro de video. Hay cuatro tipo de imágenes: I, P, B y DC. Típicamente una imagen I ocurre cada medio segundo para dar un acceso aleatorio bueno. La información correspondiente con esta capa consiste en un encabezado y rebanadas. El encabezado contiene tiempo, tipo de imagen e información propia de la codificación.

Capa de rebanada (Slice Layer): esta es la unidad de codificación primaria. Provee cierta

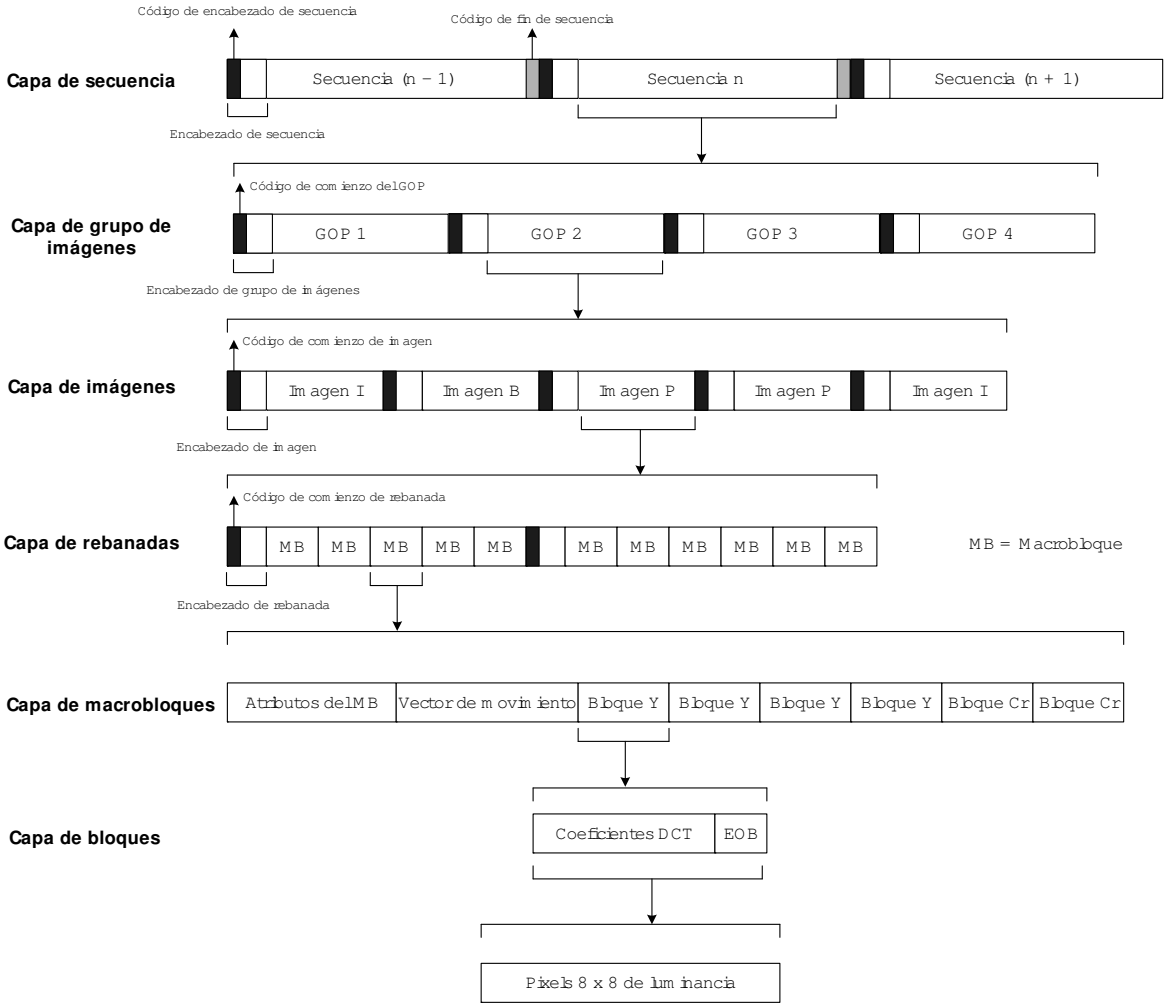


Figura 5.11: Capas MPEG-1

inmunidad a la corrupción de datos. Consiste en un encabezado y macrobloques. El encabezado contiene la posición de la rebanada e información de la escala de cuantificación.

Capa de macrobloque (Macroblock Layer): es la unidad básica para la compensación de movimiento y cambios en la escala de cuantización. Cada macrobloque está compuesto de seis bloques: cuatro de luminancia y dos de croma.

Capa de bloque (Block Layer): es la unidad básica de codificación y se le aplica la transformada discreta del coseno (DCT). Cada bloque es de tamaño 8x8, es decir de 64 píxeles. Cada píxel de luminancia corresponde a un píxel de la imagen.

Sintáxis del tren de bits

A continuación se presenta una descripción más detallada de la información correspondiente con cada capa de las mencionadas anteriormente. Se presenta cómo es el tren de bits en cada nivel de jerarquía y cómo se agrupan para formar el contenido de video codificado.

1. **SECUENCIA**

   Comienza con un encabezado de secuencia y termina con un código de finalización (*sequence end code* = 0000 0000 0000 0000 0000 0001 1011 0111). El encabezado puede repetirse dentro de la secuencia para sincronización. A continuación se detallan los campos de información en una secuencia:

   Código de encabezado de secuencia: este código está al comienzo del encabezado de la secuencia. Tiene alineación de byte, es decir que debe comenzar en un byte. Si esto no ocurre el codificador hace rellenado de bits con ceros (*bit stuffing*). Es un tren de bits único de largo 32 bits y su valor es: 0000 0000 0000 0000 0000 0001 1011 0011.

   Tamaño vertical: es la altura de la imagen en píxeles. Son 12 bits donde el bit más significativo está al comienzo. El rango de valores va de 1 a 4095. En la práctica los valores son múltiplos de 16 (para que quede un número entero de macrobloques).

   Tamaño horizontal: es el ancho de la imagen en píxeles. Al igual que para el tamaño vertical se usan 12 bits donde el rango de valores va de 1 a 4095 y es un múltiplo de 16.

   Relación de aspecto de píxel: es la forma del píxel en pantalla. Son 4 bits y hay una tabla con códigos para representar la relación de aspecto. Para píxeles cuadrados el código es 0001.

   Tasa de cuadros: para indicar el *frame rate* se usan 4 bits y una tabla con códigos. Para 25 cuadros por segundo el código es 0011 y para 30 cuadros por segundo es 0101.

   Tasa de bits: se usan 18 bits y la unidad es 400 bps. La tasa de bits se asume constante en toda la secuencia. Se redondea el *bit rate* actual a múltiplos de 400 bps. Si los 18 bits valen uno el tren de bits opera con una tasa de bits variable.

   Bit marcador: un bit siempre seteado en uno.

   VBV Buffer Size: son 10 bits que representan el mínimo tamaño del buffer de entrada en el modelo de decodificador en unidades de 16384 bits (2048 bytes). VBV - *Video Buffering Verifier*.

   Bandera de parámetros restringidos: es un bit que se setea en uno si los parámetros están restringidos de la siguiente forma:

   - Tasa de bits: menor igual a 1.856 Mbps
   - Tasa de imágenes: menor igual a 30 cuadros por segundo

- Tamaño horizontal: menor igual a 720 píxeles
- Tamaño vertical: menor igual a 576 líneas
- Macrobloques: menor igual a 396
- Tasa de píxeles: menor igual a 2534400 píxeles por segundo
- Tamaño del buffer: menor igual a 20

Descargar la matriz de cuantización intra: es un bit que si vale uno siguen a continuación 64 enteros de 8 bits que representan la matriz de cuantización que es un grupo de 8x8 pesos para cuantizar los coeficientes DCT. Se transmiten en zig-zag. Si este bit vale cero la matriz se resetea al valor por defecto que es la matriz de cuantización de JPEG.

Descargar la matriz de cuantización no-intra: es un bit que si vale uno siguen a continuación 64 enteros de 8 bits que representan la matriz de cuantización 8x8 no-intra recorrida en zig-zag. Si es cero se usa la matriz por defecto donde todos los coeficientes valen 16.

Extensión de datos: son 32 bits destinados a futuras extensiones. Es un código con alineación de byte: 0000 0000 0000 0000 0000 0001 1011 0101.

Datos del usuario: son 32 bits que se dejan para ser usados por el codificador como quiera. Código con alineación de byte: 0000 0000 0000 0000 0000 0001 1011 0110.

2. **GOP**

Los grupos de imágenes se rigen por las siguientes reglas:

- Un GOP en el orden del tren de bits debe comenzar con una imagen I.
- Un GOP en el orden de visualización debe comenzar con una imagen I o B y terminar con una I o P. El GOP más chico es una imagen I.
- Un GOP comienza con un encabezado y puede terminar con el siguiente encabezado GOP o el siguiente encabezado de secuencia o con el código de fin de secuencia.

A continuación se detallan los campos de información en un GOP:

Código de comienzo del grupo de imágenes: son 32 bits con alineación de byte: 0000 0000 0000 0000 0000 0001 1011 1000.

Código de tiempo: representado por 25 bits y se refiere a la primera imagen del grupo en orden de visualización. Está compuesto por los siguientes campos:

- Bandera de cuadro descartado: 1 bit
- Horas: 5 bits. De 0 a 23
- Minutos: 6 bits. De 0 a 59
- Fijo: 1 bit
- Segundos: 6 bits. De 0 a 59
- Número de imagen dentro del GOP: 6 bits. De 0 a 60

GOP cerrado: es un bit que si es cero indica que el GOP es abierto. Los GOP cerrados se decodifican sin usar imágenes decodificadas de grupos de imágenes anteriores para la compensación de movimiento.

Enlace roto: es un bit que indica si el GOP anterior se puede usar para decodificar el actual. Si es cero se puede usar.

Datos de extensión: son 32 bits con alineación de byte: 0000 0000 0000 0000 0000 0001 1011 0101.

Datos del usuario: son 32 bits para que use el decodificador.

3. **IMAGEN**

A continuación se detallan los campos de información en la capa de imagen:

Código de comienzo de imagen: la imagen comienza con un encabezado que tiene un código de 32 bits con alineación de byte: 0000 0000 0000 0000 0000 0001 0000 0000.

Referencia temporal: son 10 bits que definen el orden de presentación de las imágenes. La primer imagen en orden de visualización debe tener la referencia temporal en cero. Se incrementa en uno para cada imagen del GOP.

Tipo de imagen: son 3 bits: donde 001 es una imagen I, 010 es una imagen P y 011 es una imagen B.

Retardo VBV: son 16 bits que se usan para sincronización entre el codificador y decodificador. Se aplica cuando hay una tasa de bits constante. Está expresado en unidades de 1/90000 segundos.

Vector de píxel entero hacia adelante: un bit que si vale uno la precisión de los vectores es de un píxel, sino es de medio píxel. Solo está presente si la imagen es P o B.

Código F hacia adelante: presente únicamente para imágenes P o B. Son 3 bits que contienen información del tamaño máximo de los vectores hacia adelante que pueden ser codificados. El tamaño se representa mediante un código donde 000 es 1 y 110 es 64.

Vector de píxel entero hacia atrás: es un bit que está únicamente presente para imágenes B.

Información extra de la imagen: puede tener cualquier cantidad de bytes. La información está precedida por un uno y termina con un cero de la siguiente forma 1 E E E E E E E 0.

Extensión de datos: son 32 bits destinados a futuras extensiones. Es un código con alineación de byte: 0000 0000 0000 0000 0000 0001 1011 0101.

Datos del usuario: son 32 bits que se dejan para ser usados por el codificador como quiera. Código con alineación de byte: 0000 0000 0000 0000 0000 0001 1011 0110.

4. **REBANADA**

Las imágenes se dividen en rebanadas, cada una conteniendo un número entero de macrobloques. Dentro de una misma imagen pueden encontrarse rebanadas de distintos tamaños. Como mínimo puede haber una rebanada y como máximo una cantidad igual a la cantidad de macrobloques de la imagen.

Una división de la imagen en rebanadas muy común es que cada fila de macrobloques sea una rebanada. Las rebanadas dentro de una imagen no tienen porqué tener la misma forma.

A continuación se detallan los campos de información en la capa de rebanada:

Código de comienzo del encabezado de la rebanada: es un código de 32 bits que tiene alineación de byte. Los últimos 8 bits pueden estar dentro de un rango de valores que definen la posición vertical de la rebanada dentro de la imagen. Es decir que los primeros 28 bits son fijos pero los últimos 8 no. El código va desde: 0000 0000 0000 0000 0000 0001 0000 0001 hasta: 0000 0000 0000 0000 0000 0001 1010 1111. La posición vertical de la rebanada es igual a la fila del macrobloque más uno.

Escala de cuantización: son 5 bits que indican la escala usada por el decodificador para calcular los coeficientes DCT a partir de los coeficientes transmitidos.

Información extra de la rebanada: puede tener cualquier cantidad de bytes: 1 E E E E E 0. Su uso no está definido as que los codificadores no deben generarlos y los decodificadores deben ser capaces de descartarlos.

5. **MACROBLOQUE**

Los macrobloques son estructuras de 16 x 16 píxeles que forman una rebanada. Cada macrobloque tiene información de su dirección, su tipo y la escala de cuantización. Está formado por un encabezado, los seis bloques de 8 x 8 píxeles que lo conforman y un código de fin del macrobloque.

A continuación se detallan los campos de información en un macrobloque:

Relleno del macrobloque (Macroblock Stuffing): este es el primer campo del encabezado del macrobloque y es opcional. Su función es prevenir *underflow*. El codificador evalúa si puede llegar a haber *underflow* y en base a esto inserta códigos de relleno que son palabras de 11 bits: "'0000 0001 111"'. El decodificador descarta este código.

Dirección del macrobloque: la dirección del macrobloque es su lugar de aparición en el escaneo de la imagen. El primer macrobloque es el de la esquina izquierda superior y tiene la dirección "'0"'.n En realidad lo que se manda es el incremento de la ubicación del macrobloque que es la diferencia entre la dirección del macrobloque anterior y la actual. Esta diferencia se codifica usando VLC.

Fin del macrobloque: se usa solo para imágenes D.

6. **BLOQUE**

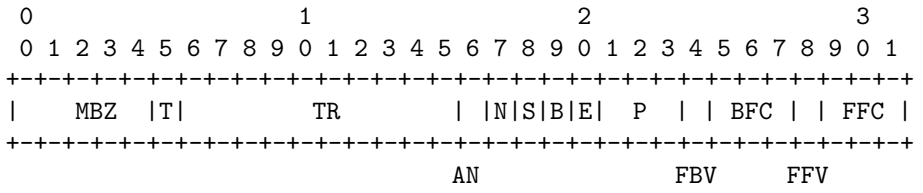
   Son bloques de 8 x 8 píxeles usados para la aplicación de la transformada discreta del coseno en dos dimensiones. Los coeficientes DCT se transforman en un arreglo.

5.3.2. MPEG-1 sobre RTP. RFC 2250

La codificación de imágenes en formato MPEG-1 resulta en un stream de video, que en nuestro caso será enviado por la red. Dado el stack de protocolos utilizado, el video debe ser encapsulado en RTP. Cómo debe hacerse la encapsulación está definido en un Request for Comments, la RFC 2250 [36], que describe como encapsular MPEG-1 y MPEG-2 sobre RTP.

Las imágenes MPEG-1 generalmente tienen un tamaño mayor que la MTU de la red LAN, siendo imprescindible definir un criterio para partir las imágenes y enviarlas en distintos paquetes RTP. Este RFC recomienda como armar la carga útil del paquete RTP, separando el paquete en dos partes bien diferenciadas. La primera es un encabezado que describe la porcion de el video que está siendo transportado por el paquete, llamado encabezado específico de video, y la segunda es directamente la carga útil de video.

El encabezado específico de video mencionado se describe a continiuación:



`MBZ:` (3 bits). Actualmente estos bits no se usan y deben ser seteados en cero.

`T:` (1 bit). Estos bits deben ser seteados en cero para MPEG-1. Se usan en MPEG-2 para indicar cuando está presente la extensión de encabezado.

`TR:` Referencia Temporal (10 bits). Representa una referencia dentro del GOP de la imagen que se está enviando. Su valor varía entre 0 y 1023, siendo constante en todos los paquetes RTP que transportan la misma imagen.

`AN:` Bit activo para recuperación de errores (1 bit). En MPEG-1 debe ser seteado en cero. En el caso de MPEG-2, si está seteado en cero indica que el bit (N) no se usa. En caso de setearse en uno, el siguiente bit (N) indica que hubieron cambios en el encabezado de la imagen.

`N:` Nuevo encabezado de imagen (1 bit). Para cargas útiles correspondientes a MPEG-1 este bit se debe setear en cero. En caso de transportarse MPEG-2 se usa solamente cuando el

bit previo (AN) esta en uno. Se setea en uno en caso de que la información de encabezado de imagen de la imagen anterior no sirve para reconstruir un encabezado para la imagen actual. Esto sucede cuando la imagen anterior fue codificada con parámetros distintos de la imagen actual. Para una misma imagen este bit debe manteners en el mismo valor.

`S:` Encabezado de secuencia (1 bit). Usualmente se setea en cero y se cambia el valor a uno cuando en la carga útil del video está presente un encabezado de secuencia.

`B:` Comienzo de rebanada (1 bit). Seteado en uno cuando en el comienzo de la carga útil de video está presente el código de comienzo de rebanada, o está precedido por un encabezado de secuencia y/o un encabezado de imagen.

`E:` Fin de rebanada (1 bit). Seteado cuando el último byte de cárga útil es el fin de una rebanada.

`P:` Tipo de imagen (3 bits). Estos bits indican el tipo de imagen que el paquete transporta. Para todos los paquetes RTP que transmiten la misma imagen este valor debe ser constante. Los posibles valores son: 001 (imágenes I), 010 (imágenes P), 011 (imágenes B), 100 (imágenes D). El valor 000 no es válido y los valores restantes están reservados para uso futuro.

`FBV:` Vector de pixel entero hacia atrás.

`BFC:` Código F hacia atrás.

`FVV:` Vector de pixel entero hacia adelante.

`FFC:` Código F hacia adelante

Para imágenes I los cuatro valores anteriores no se utilizan. Para imágenes P solamente se utilizan los últimos dos valores, y para imágenes B se utilizan los cuatro valores.

Respecto a la la carga útil de video la recomendación describe como esta debe partirse para poder ser enviada en paquetes RTP. La forma de dividir la carga útil tiene el conjunto de reglas que se listan:

1. El encabezado de sequencia de video, en caso de estar presente, debe ir al principio de la carga útil del paquete RTP (luego del encabezado específico de video antes descrito).
2. El encabezado de un GOP, cuando está presente debe ir al comienzo de la carga útil del paquete RTP (luego del encabezado específico de video), o precediendo al encabezado de secuencia de video.
3. El encabezado de una imagen cuando está presente debe ir al principio de la carga útil del paquete RTP (luego del encabezado específico de video antes descrito), o precediendo al encabezado del GOP.

4. El comienzo de una rebanada debe ir ser el primer dato de la carga útil del RTP (luego de algún encabezado MPEG) o debe ir en el paquete RTP después de un número entero de rebanadas. Generalmente las rebanadas ocupan más de un paquete RTP

5. Dado que usualmente las rebanadas ocupan un tamaño mayor a la MTU está permitido partirla y enviarla en distintos paquetes siempre que se respeten las reglas anteriores.

Estrategias de recuperación frente a errores y resincronización

Las redes de paquetes que existen hoy en día tienen congestionamiento y trafícan una cantidad muy grande datos, resultando algunas veces en la perdida de paquetes. En caso de haber pérdida, el receptór debe tomar alguna acción para que esto lo perjudique lo menos posible. Este RFC no recomienda una estrategia de recuperación frente a pérdida de paquetes, pero sí da una guía de como reaccionar frente a esto.

Lo primero que se dice es que al comienzo de la recepción se deben descartar todos los paquetes RTP hasta que se encuentre un paquete con el encabezado de comienzo de secuencia presente. Además de esto se da una idea de como detectar pérdidas de encabezados de GOP y de imágenes, dejando que el lector haga su propia implementación.

Tomando esto como guia y definiendo algunos criterios nuestra aplicación tiene su propia estrategia de recuperación frente a pérdida de paquetes, la cual se explicará más adelante.

5.4. Otros codecs de compresión de video analizados

Durante la búsqueda del codec de video más adecuado para integrar a la aplicación se realizó un estudio de otros codecs, Theora y H.263. En esta sección se realiza una breve descripción teórica de estos codecs.

Theora

Theora es un codec de video de código abierto desarrollado por la fundación *Xiph.org*[37] como parte de otro proyecto llamado *Ogg*. Su implementación está ampliamente basada en el codec VP3, donado por On2 Technologies.

La compresión que realiza es según un algoritmo con pérdidas, siendo un codec de transformación y predictivo. El tipo de datos que soporta es "progressive video" de cualquier tamaño a una tasa constante. El único espacio de colores que admite es YC_bC_r con un máximo de 8 bits para representar cada componente del espacio y se puede utilizar cualquiera de los siguientes tipos de submuestreo: 4:2:0, 4:2:2, 4:4:4.

Cada imagen se divide en bloques que a su vez se dividen en bloques, obteniendo una jerarquía de capas necesarias para la compresión. Dos características fundamentales que describen

a la compresión es el uso de la Transformada de Coseno Discreta Tipo II y la utilización de compensación de movimiento.

Al codificar imágenes es posible obtener cuadros codificados únicamente teniendo en cuenta la redundancia espacial (intra) y cuadros que además tienen en cuenta la redundancia temporal (inter) para predicción hacia atrás. Este codec no genera cuadros que utilizan predicción bidireccional, es decir predicción hacia atrás y hacia adelante.

Cuando se desea enviar por la red información de video comprimida, es necesario analizar cómo debe ser transportada. Esto se refiere a posibles fragmentaciones, encapsulados, entre otros.

Realizando un estudio del codec Theora, se enocntró un draft RFC [38] que especifica cómo debe ser el transporte de dicho codec sobre el protocolo RTP. Este draft propone metodologías para fragmentar los *frames* de video Theora y luego para realizar su encapsulamiento en paquetes RTP. A tales efectos, se definen encabezados que permiten transportar información referente a la tira de datos de video enviada en cada paquete.

Por otra parte, se encontró que muchas veces el video codificado en Theora se transporta en una estructura de datos denominada *Ogg*. Esta estructura es un contenedor de información que presenta además de los datos Theora, un determinado encabezado. Al analizar detalladamente la librería se descubrió que cuando un cuadro de video ingresa al codificador, el mismo retorna ya encapsulado en este contenedor *Ogg*.

H.263

H.263 es un codec de compresión publicado por la ITU-T que surgió con el objetivo de mejorar la eficiencia del codec ya existente H.261 en lo que refiere a performance y recuperación de errores. Las técnicas de codificación utilizadas en H.263 se basan ampliamente en el desarrollo realizado para H.261 pero se le agregan algunas características que lo tornan en un codec sumamente utilizado hoy en día en transferencia de datos por internet.

H.263 es usualmente utilizado en aplicaciones de videoconferencia, y permite lograr muy bajas tasas de bits. Para ello, implementa algoritmos que eliminan la redundancia temporal y espacial. Para evitar todo tipo de redundancia entre imágenes de una misma secuencia de video, se utiliza predicción temporal. A su vez, para comprimir la información correspondiente con cada imagen se utiliza la transformada de la señal de la cual solamente se consideran significantes algunos de los coeficientes.

El codec H.263 presenta opciones de codificación que pueden ser negociables a la hora de comprimir. Con estas opciones es posible lograr una mejor performance en la compresión alcanzada así como también agregar capacidades extras al codificador o decodificador. Luego de la versión original de H.263, surgieron dos versiones más que incorporan anexos con más funcionalidades opcionales. La última versión existente hoy en día es la de H.263v3, la cual

presenta aproximadamente veinte anexos que permiten disponer de beneficios adicionales frente a la versión original.

Al hablar de video en H.263 se define una estructura jerárquica modelada en cuatro capas, cada una de las cuales representa un conjunto de datos de las imágenes que conforman el video. El objetivo es particionar las imágenes en bloques e ir incorporando encabezados que permitan facilitar la reconstrucción del video al decodificar. A continuación se describen brevemente cada una de las capas mencionadas, tomando como ejemplo una tira de imágenes en formato CIF (352*288):

Capa de imagen: la información correspondiente a la capa de imagen está formada por un encabezado de imagen y a continuación datos correspondientes con los grupos de bloques. Al final de esta estructura de datos puede aparecer un códgo indicador de fin de secuencia, pero la presencia de éste es opcional.

Capa de grupo de bloques: esta capa contiene información de encabezado para cada grupo de bloques y a continuación se encuentran los datos correspondientes a uno o más macrobloques. En el encabezado de los GOB es posible determiar a qué imagen pertenece el presente GOB así como también el número de GOB dentro de dicha imagen.

Cada GOB se forma con líneas de imagen y puede contener como máximo k*16 líneas, donde k depende de la resolución de la imagen. En el caso de una imagen en formato CIF, k es la unidad y los GOB quedan formados por 16 líneas de imagen. La numeración de los GOB se realiza de forma vertical, comenzando por el GOB 0 en el extremo superior de la imagen, hasta el GOB 17 en el caso de una imagen del tipo CIF.

Capa de macrobloque: esta capa está formada por un encabezado de macrobloque seguida de los datos de bloque.

Cuando no se dispone de las características opcionales, cada macrobloque se compone de cuatro bloques con información de luminancia (Y) y dos bloques con información de croma (Cb y Cr). Los bloques de luminancia son de 16 píxeles por 16 líneas, mientras que los de croma son de 8 píxeles por 8 líneas. La numeración de los macrobloques se lleva en forma horizontal a través de las líneas de la imagen, de izquierda a derecha.

Capa de bloque: este es el último nivel de partición de una imagen y corresponde, como su nombre lo indica, a bloques de información. Estos bloques son de tamaño 8 píxeles por 8 líneas y pueden contener información de luminancia (Y) o de croma (Cb, Cr).

A continuación se describen las características más destacadas de la codificación y decodificación:

1. **Codificación**

   En lo que refiere al codificador H.263 puede funcionar tanto con formatos estándares de resolución de imagen, como con formatos personalizados. En cuanto a los formatos estándares, el codec soporta SCIF, QCIF, CIF, 4CIF y 16CIF. Al disponer de soporte

para las resoluciones 4CIF y 16CIF el codec se torna competente con otros estándares de compresión como por ejemplo los MPEG. En cuanto a los formatos personalizados, en caso de querer utilizarlos, es necesario que el codificador y el decodificador negocien mediante algún protocolo adicional el tipo de formato a utilizar.

Al codificador deben ingresar imágenes progresivas en alguno de los formatos soportados por el codec, que queda determinado por la resolución de la imagen (CIF, QCIF, etc.) y la frecuencia en que las imágenes son generadas (*frame rate*). Una vez disponibles las imágenes de entrada se codifican en el formato YCbCr formado por una componente de luminancia (Y) y dos componentes de croma (Cb, Cr). El muestreo para obtener los valores de luminancia se efectúa para la cantidad de píxels por línea, y líneas por imagen correspondientes con la resolución de la imagen. En lo que refiere a los componentes de croma, para estos se realiza un muestreo a la mitad de píxeles por línea y la mitad de líneas por imagen en referencia con la componente de luminancia (muestreo 4:2:0). Finalmente, al momento de codificar la información, se utilizan códigos de largo variable transformando al codec H.263 en un código entrópico.

2. **Decodificación**

   El decodificador, por su parte, dispone de compensación de movimiento para predecir el valor de las imágenes de una misma secuencia. A tales efectos el decodificador utiliza precisión de medio píxel resultando en una mayor compresión que la disponible en H.261, donde se utiliza un píxel completo.

   Para la predicción del movimiento se calculan los vectores de movimiento necesarios. En el caso más simple se utiliza un mismo vector para todos los píxeles de los cuatro bloques de luminacia pertenecientes al macrobloque, que se predice a partir de los tres macrobloques adyacentes. El valor para el vector de crominancia se obtiene dividiendo entre dos el valor correspondiente a los bloques de luminancia.

5.5. Elección de diseño

Dentro de los temas análizados en cuanto al diseño de la aplicación incluye la arquitectura del software, las características del video y la elección del codec de compresión a utilizar. En lo que sigue se detalla las desiciones tomadas respecto a cada uno de estos temas.

Luego de haber elegido los puntos básicos para el desarrollo, se procedió a implementar una gran cantidad de clases que le permiten al teléfono incorporar la transmisión de video sobre la llamada. En la medida que resultó necesario, también se fueron realizando modificaciones sobre las clases ya existentes de modo de no alterar la transmisión del flujo de audio.

5.5.1. Arquitectura del software

Previo a integrar la transmisión de video a la aplicación, fue necesario analizar las distintas maneras en que se podía desarrollar y qué opción resultaría más conveniente. Dado que

sipXvideoPhone no contaba con nada de implementación de video, se debió crear de cero todo aquello relativo a este flujo de medios.

En primer lugar se debió elegir una arquitectura de código que permitiera desarrollar una plataforma con transmisión de video. A tales efectos se consideró que la mejor opción sería tomar como base la arquitectura que presenta sipXezPhone para la transmisión de audio e implementar una similar.

En sipXezPhone el procesamiento de la información de audio se lleva a cabo en etapas, a través de los denominados recursos de procesamiento de medios. La idea es que cada recurso cumple una función específica y la interconexión entre ellos permite lograr el objetivo deseado. En el caso de sipXezPhone se trata de establecer una sesión de audio entre dos clientes SIP, y en el caso de la nueva aplicación se trata de transmitir además un flujo de video.

Los recursos se pueden agrupar en dos categorías que representan las dos partes fundamentales del procesamiento de medios. Por una parte se encuentra lo relativo a la conexión entre los puertos que conforman el camino para el flujo de tráfico a través de la red, y por otra parte se encuentra el procesamiento de la información a nivel local de cada cliente SIP.

El procesamiento a nivel de la conexión incluye la codificación de los datos, transmisión de los paquetes RTP, recepción de los mismos, y finalmente la decodificación de la información recibida.

En cuanto al procesamiento de la información a nivel local, esto se refiere a la captura de los datos para luego enviarlos por la red y la reproducción de los mismos en destino. En el caso de audio esto se refleja en el manejo del micrófono y los parlantes, mientras que en el caso de video se trata del procesamiento de los datos cuando son capturados por la cámara web y luego reproducidos en las pantallas de previsualización y visualización.

Se le llama pantalla de previsualización a la ventana en donde se pueden observar los datos capturados por la cámara local. La pantalla de visualización se refiere a la ventana en donde se muestran los datos originados en el agente de usuario remoto y recibidos por la red.

Tomando como base la arquitectura recién descrita, se creyó conveniente proceder de manera similar para el caso de video. Esto implica crear un nuevo conjunto de recursos que tengan las características necesarias para poder modelar el procesamiento de un flujo de video transmitido entre dos agentes de usuario. De esta manera se logra que cada sesión de medios se maneje de manera independiente, a través de sus recursos de procesamiento asociados. Cada grupo de recursos se encarga de realizar sobre los datos, el procesamiento en base a los protocolos y estándares correspondientes a cada flujo de medios (audio o video).

5.5.2. Características del video

Otro de los aspectos de diseño fue la elección de características asociadas a las imágenes de video que se transmiten. Se debió elegir el tamaño que se utiliza para las pantallas de visualización y previsualización, la velocidad a la que se mandan las imágenes, y el formato en que se representan las mismas. A tales efectos se decidió trabajar con:

- Un tamaño de imagen correspondiente con la resolución QCIF (176 x 144 píxeles).
- Imágenes en formato BGR24.
- Tasa de cuadros de video de 25 cuadros/s.

5.5.3. Elección del codec de compresión de video

Una opción válida para visualizar video durante la comunicación sería transmitir las imágenes sin comprimir, pero luego de evaluarlo se consideró que no era la mejor opción. A modo de ejemplo, para poder transmitir una imagen sin comprimir en formato RGB 24 bits y de tamaño QCIF, se necesitan aproximadamente cincuenta y ocho paquetes RTP. Teniendo en cuenta esto y la cantidad de imágenes por segundo que es necesario transmitir para obtener una comunicación aceptable, se puede ver que transmitir imágenes en formato sin comprimir enlentece demasiado la aplicación. Principalmente porque el programa debe recibir los paquetes RTP correspondientes a cada imagen, procesarlos uno a uno e irlos guardando para luego retirarlos y procesarlos de forma adecuada. Esto resulta en tiempos de procesamiento muy grandes, deteriorando en gran medida la performance de la aplicación.

Dado que transmitir video de la forma antes descrita no es la mejor opción, se decidió utilizar un codec de video, el cual sea capaz de comprimir las imágenes capturadas por la cámara de video. El objetivo de la compresión en este caso es poder usar más eficientemente el ancho de banda disponible en la red y no enlentecer demasiado tiempos de procesamiento de la aplicación.

La elección del codec se basó no solamente en un estudio teórico de los mismos, sino también en el hecho de poder encontrar una librería que proporcione las funcionalidades de codificación y decodificación. Cabe aclarar que el objetivo del proyecto no es desarrollar ningún codec de compresión sino implementar la transmisión de video en la aplciación sipXezPhone para lo cual se consideró válido poder utilizar librerías adicionales que, integrándolas al programa, provean las funcionalidades necesarias.

Se encontraron básicamente dos librerías de código abierto en las cuales se profundizó el estudio para luego decidir si resultaban adecuadas o no. Por una parte se trabajó con el proyecto Theora, el cual proporciona un codec de compresión no basado en estándares pero muy utilizado en desarrollos gratuitos y de código abierto. Por otra parte se estudió la librería Libavcodec, una librería que dispone de varios codecs de compresión, estos sí basados en estándares.

Comparación entre codecs

Comenzando con Theora, vimos que el mismo es encapsulado en un tipo de contenedor llamado *Ogg*. Dado que nuestra aplicación fue desarrollada para utilizar el protocolo RTP como medio de transporte, surge como necesidad que se sea capaz de encapsular la estrucutra de datos *Ogg* en paquetes RTP. Sin embargo, luego de investigar más acerca del tema se concluyó que los paquetes *Ogg* no se transportan por RTP si no que Ogg ya es por sí mismo un mecanismo para transportar la información codificada. Más aún, la especificación del borrador (*draft*) no determina cómo encapsular paquetes *Ogg* sobre RTP sino que hace referencia al encapsulado de Theora directamente.

Continuando con el análisis del codec Theora, cabe destacar que éste no es un codec estándar e incorporarlo a nuestra aplicación tornaría a la misma solamente compatible con teléfonos que tengan soporte para Theora. Sería interesante poder desarrollar una aplicación que soporte codecs estándar como lo son los desarrollados por la ITU-T o la ISO.

Por los motivos citados, se consideró que utilizar Theora como codec de compresión de video no sería la opción más adecuada. Se procedió entonces a buscar otra librería que proporcione las herramientas para poder comprimir y descomprimir utilizando un codec estándar y compatible con la arquitectura ya disponible en la aplicación sipXvideoPhone.

Luego de descartar Theora, se consideró la librería Libavcodec, quien posee distintos codecs de compresión de video desarrollados por la ITU-T o la ISO.

Pasando a los codecs desarrollados por la ITU-T nos encontramos con H.263, que en estos tiempos es un codec muy utilizado para comprimir video. H.263 es comunmente utilizado en videoconferencias por internet y en otras aplicaciones de tiempo real. Estas aplicaciones multimedia resultan similares al objetivo buscado con sipXvideoPhone, y por tanto podría resultar adecuado incorporar H.263 a nuestra aplicación.

H.263 es un codec de alta compresión desarrollado para proporcionar una buena calidad de imagen a tasas de bit muy pequeñas. Esto lo categoriza como un codec de compresión sumamente eficaz ya que logra reducir el ancho de banda consumido a valores en el entorno de 24 kbps. Si bien esto resulta una característica a favor del codec, puede llegar a tener un efecto negativo cuando los tiempos de procesamiento juegan un papel importante. En el caso de sipXvideoPhone, la arquitectura original del programa es extremadamente complicada y rebuscada provocando que el procesamiento de los datos no se realice de manera eficiente, y los tiempos de ejecución no sean los mejores. Si sumado a esto, se incorpora un codec de compresión que consuma elevados recursos de procesamiento entonces podríamos disminuir la eficiencia de la aplicación. Por tales motivos, se optó por dejar de lado el codec H.263 e intentar seleccionar otro que permita una mejor performance de sipXvideoPhone.

Finalmente, se analizó la posible incorporación del codec MPEG-1 a nuestra aplicación. Este codec, también publicado por la ISO (ISO/IEC 11172), no presenta tanta eficiencia a la

hora de comprimir como sí lo hacen los codecs mencionados anteriormente. MPEG-1 es un codec de compresión de video que permite alcanzar como máximo una tasa de 1,856 Mbps. Esto refleja una baja performance en lo que refiere a compresión propiamente dicha, pero puede ser una alternativa adecuada para integrar en nuestra aplicación de tiempo real donde los tiempos de procesamiento son sumamente importantes. Cabe señalar que el hecho de que MPEG-1 no sea el codec más eficaz para comprimir datos de video, no implica que no sea bueno.

El ancho de banda del que se dispone en la LAN es suficiente para transportar video de las características elegidas, por lo que el hecho de que MPEG-1 no logre reducir el ancho de banda como lo hace H.263 no lo excluye de esta elección. Luego de testear la librería Libavcodec resultó que ésta no soporta como entrada del codec H.263 las características del video elegido, por tanto se procedió a elegir MPEG-1 como el codec de video a incluir en la aplicación.

5.6. Negociación SIP de la llamada con video

En esta etapa del trabajo, se procedió a desarrollar la negociación SIP de una sesión de video sobre la plataforma sipXvideoPhone.

En etapas previas del desarrollo de sipXvideoPhone se logró implementar la negociación de una sesión de audio utilizando el codec G.729. Para incorporar a la aplicación el establecimiento de una sesión de video, se consideró conveniente proceder de manera similar.

En base al análisis realizado al momento de incorporar el codec G.729 de audio, se busca que los mensajes SDP intercambiados entre los agentes de usuario contengan la información correspondiente con la sesión de video que se desea establecer. Estos mensajes deben especificar el codec de video utilizado, sus características asociadas, y los puertos que se designan para establecer la sesión. A tales efectos, se procedió a realizar las siguientes tareas:

- Agregar al teléfono el soporte de un codec de compresión de video.
- Determinar qué puertos se utilizan para establecer la sesión de video

5.6.1. Incorporación de un codec de video genérico

En esta instancia del desarrollo, se procedió a agregar un codec de video a la plataforma de manera que éste pase a formar parte de la lista de codecs actualmente soportados por el teléfono. Por el momento no se implementa nada de la transmisión ni recepción de video sino que solamente se definen las etiquetas y estructuras necesarias para incorporar un nuevo codec de compresión a la aplicación.

Al comienzo de esta etapa de incorporación de video, el codec de compresión aún no había sido elegido, lo que llevó a la necesidad de tomar un codec genérico. Este codec genérico se denominó `CODECVIDEO` y resultó de gran utilidad para llevar adelante gran parte del desarrollo

de sipXvideoPhone. `CODECVIDEO` no exsite como codec en sí mismo, solamente se utilizó a modo representativo.

Una vez eligido este codec genérico, se procedió de manera análoga a la incorporación de G.729. Se creó un nuevo objeto *codec* donde se definen las características propias del codec que se está incorporando, que en este caso corresponde a `CODECVIDEO`. Para ello, al igual que en la creación del codec de audio, se realizaron las siguientes modificaciones en el programa:

- Se definieron las etiquetas correspondientes a `CODECVIDEO`.
- Se recurrió a la estructura `SdpCodec` y a la clase `SdpCodecFactory` donde se especificaron los parámetros correspondintes a `CODECVIDEO`.

```
CODEC_TYPE_CODECVIDEO = 99
SIPX_CODEC_ID_CODECVIDEO = CODECVIDEO
MIME_SUBTYPE_CODECVIDEO = CODECVIDEO
SDP_CODEC_CODECVIDEO = 99
```

```
 case SdpCodec::SDP_CODEC_CODECVIDEO:
        {
           SdpCodec aCodec(SdpCodec::SDP_CODEC_CODECVIDEO,
                           SdpCodec::SDP_CODEC_CODECVIDEO,
                           MIME_TYPE_VIDEO,
                           MIME_SUBTYPE_CODECVIDEO,
                           90000,
                           20000,
                           1,
                           "",
                           SdpCodec::SDP_CODEC_CPU_LOW,
                           SDP_CODEC_BANDWIDTH_HIGH,
                           SDP_VIDEO_FORMAT_QCIF);
           addCodec(aCodec);
           aCodec.getMediaType(codecMediaType);
           aCodec.getEncodingName(codecEncodingName);
        }
```

Dado que `CODECVIDEO` no es en sí mismo un codec de compresión, al momento de definir las etiquetas y la estructura `SdpCodec` fue necesario improvisar el valor de algunos parámetros. Esto se ve reflejado en el ejemplo anterior.

El valor asociado al tipo de carga útil fue seleccionado de acuerdo a lo establecido en la RFC 3551. Aquí se reservan una serie de números que no se corresponden con ningún codec de compresión en particular. Para representar al codec genérico se eligió el número 99.

Con estas modificaciones, el nuevo codec de video queda incorporado a la lista de codecs soportados por sipXvideoPhone. La aplicación ya cuenta con la información correspondiente a `CODECVIDEO` para su utilización durante la negociación SIP de una llamada entre dos agentes de usuario.

5.6.2. Selección de los puertos

El establecimiento de una sesión de video requiere que se designe un puerto a través del cual se transmita el flujo de medios. En sipXezPhone, la asignación de puertos ya se encuentra resuelta para una única sesión de medios y por tanto fue necesario comprender la metodología empleada para poder incorporar nuevas sesiones.

Para crear los *sockets*, sipXvideoPhone cuenta con una clase que modela los mismos denominada `OsNatDatagramSocket`. Dado que un *socket* se define en base a una dirección IP y un puerto, es necesario determinar el puerto que se utliza para crear el *socket* correspondiente a la sesión de video. Aquí surgen vairas interrogantes en cuanto a la implementación:

- ¿Cómo se seleccionan los puertos?
- ¿Quién los elige?
- ¿Deben setearse de forma forzada?

El procedimiento utilizado por sipXezPhone para designar puertos a las diferentes sesiones que se desean establecer, se basa en llevar un registro de todos los puertos que son utilizados durante una llamada. Cada sesión de medios requiere de dos puertos: uno para el canal RTP y otro para el canal RTCP. La asignación de puertos para estos dos canales se realiza de manera tal que a la sesión RTP le corresponde un número par, a su respectiva sesión RTCP le corresponde el siguiente número (impar). Esto se rige por la RFC 3550 donde se establece que a las sesiones RTP les corresponden los puertos pares y a sus respectivas sesiones RTCP los números impares.

El teléfono cuenta con un parámetro configurable por el usuario donde se especifica el primer puerto a partir del cual se puede establecer una sesión de medios. Este parámetro se setea al abrir la aplicación, en el menú de configuración, y por defecto se encuentra seteado en el puerto 8000.

5.6.3. Resultados obtenidos

Luego de incorporar el codec genérico se observaron los resultados obtenidos. En una primera instancia se verificaron aquellas modificaciones visibles para el usuario del teléfono, para así percibir fácilmente si los cambios efectuados quedaron incorporados.

Al igual que en audio, el hecho de haber incorporado un nuevo codec de compresión a la aplicación se traduce en cambios en la interfaz de usuario. El menú de configuración del teléfono debe ser capaz de presentar la opción de seleccionar `CODECVIDEO` al momento de querer establecer una llamada.

En esta oportunidad el menú de interés fue *Video Settings* para el cual sipXezPhone ya disponía de la implmentación de la ventana, pero se encontraba originalmente sin contenido. Con la incorporación de `CODECVIDEO` a la aplicación se obtuvo una ventana de configuración con

las características del codec de video recién incorporado. La aplicación presentó la posibilidad de seleccionar `CODECVIDEO` como codec de compresión. Esto se ilustra en la figura 5.12.

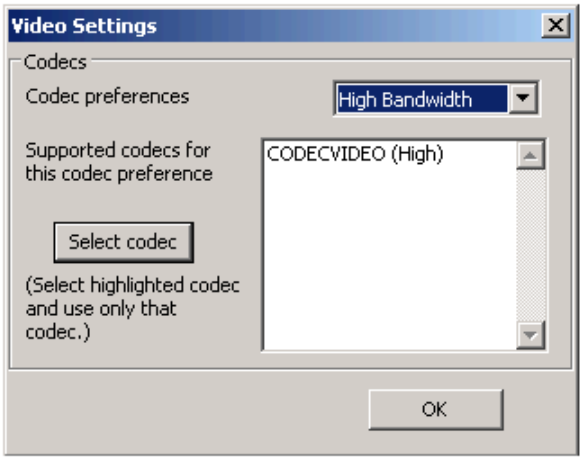


Figura 5.12: Video Settings

Una vez disponible el codec de video, se realizó una llamada entre dos agentes de usuario sipXvideoPhone. A continuación se muestra el intercambio de mensajes SIP que se llevó a cabo durante el establecimiento y finalización de la llamada:

```
| 192.168.1.104              192.168.1.103 |
|         INVITE SDP ( G.729 CODECVIDEO )  |SIP From: sip:2000 To:sip:p3
|(5060)   ------------------>  (5060)      |
|             100 Trying                   |SIP Status
|(5060)   <------------------  (5060)      |
|             180 Ringing                  |SIP Status
|(5060)   <------------------  (5060)      |
|         200 OK SDP ( G.729 CODECVIDEO )  |SIP Status
|(5060)   <------------------  (5060)      |
|                ACK                       |SIP Request
|(5060)   ------------------>  (5060)      |
|                RTP                       |RTP
|(8000)   <------------------  (8000)      |
|                RTP                       |RTP
|(8000)   ------------------>  (8000)      |
|                BYE                       |SIP Request
|(5060)   ------------------>  (5060)      |
|               200 OK                     |SIP Status
|(5060)   <------------------  (5060)      |
```

Para un mejor análisis de la información transmitida en los mensajes SIP, se muestra un trozo del mensaje `200 OK` donde se puede apreciar el contenido del cuerpo SDP con la información característica de las sesiones de medios que se establecieron.

```
Message body
     Session Description Protocol
         v = 0
         o = sipX 5 5 IN IP4 192.168.1.104
         s = call
         c = IN IP4 192.168.1.104
         t = 0 0
         m = audio 8000 RTP/AVP 18
         a = rtpmap:18 G.729/8000/1
         a = fmtp:18 annexb=no
         m = video 8002 RTP/AVP 99
         a = rtpmap:99 CODECVIDEO/90000/1
         a = fmtp:99 size:QCIF
```

Como se observa en el mensaje SDP, se lograron establecer dos sesiones de medios: una de audio y otra de video. En lo que refiere a la sesión de audio, se establece por el puerto 8000, con el codec G.729.

Para el establecimiento de la sesión de video, se observa que el codec de compresión que se negoció fue el `CODECVIDEO` con tipo de carga útil 99. Este es el codec de video que se acaba de incorporar a la aplicación y en el mensaje SDP se pueden ver reflejados todos los agregados que se realizaron previamente al programa. En cuanto a los puertos utilizados se observa que el elegido para el flujo de video fue el 8002 como se previó en el análisis realizado anteriormente.

5.7. Arquitectura base para la transmisión de video e integración con audio

En esta etapa del trabajo, se implementó la arquitectura base para disponer de transmisión de un flujo de video entre dos agentes de usuario sipXvideoPhone que establecen una llamada.

Cuando dos agentes de usuario establecen una sesión RTP, se genera un flujo de tráfico que requiere cierto procesamiento de medios para poder transmitirse correctamente desde un extremo hacia el otro. Este procesamiento implica varias etapas como pueden ser la captura de los datos en el origen, codificación, encapsulado en RTP y decodificación, entre otros.

Esto conduce a la necesidad de desarrollar una plataforma que sea capaz de procesar un flujo RTP de video, y con esto lograr que sipXvideoPhone pueda transmitir de manera adecuada la infromación correspondiente con la sesión de medios establecida.

En esta sección se detalla la base de la arquitectura elegida que nos permitió llevar adelante la incorporación de video a sipXvideoPhone. Se muestra también la interacción entre las clases más importantes del desarrollo y finalmente, se explica con un ejemplo el camino de procesamiento que siguen los datos de video mientras se mantiene establecida la llamada entre dos agentes de usuario sipXvideoPhone.

5.7.1. Descripción de los recursos de procesamiento de medios

Para manejar los recursos de procesamiento de video, sipXvideoPhone dispone de clases que los representan y que tienen implementadas las tareas que cada uno debe desempeñar.

La arquitectura del código que modela los recursos de procesamiento de medios, se ilustra en la figura 5.13 y se descibe a continuación.

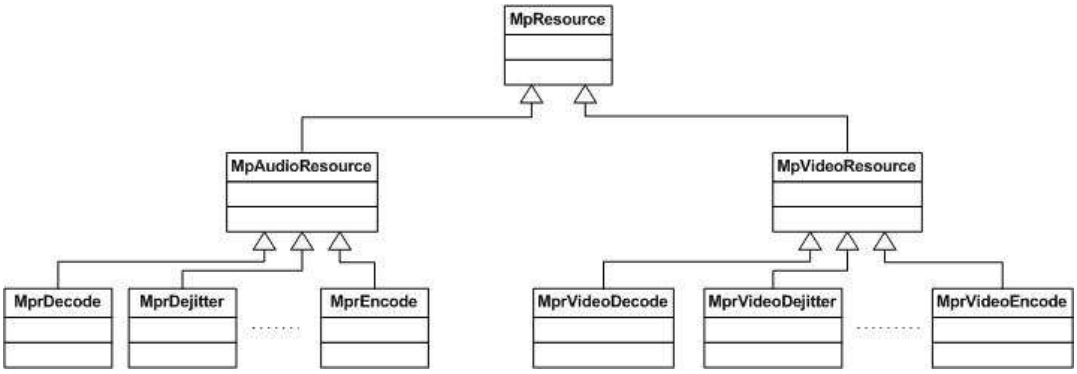


Figura 5.13: Diagrama de herencia de los recursos

Por una parte, se dispone de una clase base denominada `MpResource`, a partir de la cual heredan todos los recursos de procesamiento que se utilizan a lo largo de todo el desarrollo. Esto incluye tanto los recursos utilizados para el modelado de la llamada de audio, como para el modelado de la llamada de video.

A partir de `MpResource` surgen dos clases que marcan la diferencia entre los recursos de audio y los de video. En el caso de audio se trata de la clase `MpAudioResource`, la cual ya se encontraba implementada en la aplicación sipXezPhone y representa la clase padre para todos los recursos específicamente de audio. Sobre esta clase fue necesario realizar algunos agregados y algunas modificaciones de modo de lograr una arquitectura compatible con el resto del desarrollo.

Siguiendo con el lineamiento anterior, la integración de transmisión de video al código de SIPfoundry hizo necesaria la creación de una clase `MpVideoResource` que también heredara de `MpResource` y que tuviera como hijos a todos los recursos que fueran necesarios para modelar una sesión de video. Esta clase se implementó teniendo en cuenta las características propias de una llamada de video comunes a todos los recursos de procesamiento de video.

Finalmente, para implementar los recursos de video, se tomaron como referencia los utilizados para la transmisión de audio y a partir de ellos se eligieron los que resultaran de utilidad al momento de modelar una sesión de video.

Para modelar la llamada de audio en sipXezPhone se utilizan los recursos de procesamiento: `MprFormNet`, `MprDejitter`, `MprFromFile`, `MprDecode`, `MprBridge`, `MprMixer`, `MprToSpkr` `MprFromMic`, `MprEchoSupress`, `MprEncode`, `MprToNet`, `MprRecord` , `MprFromFile`, `MprSplitter`, `MprToneGen`, `MprFromStream`.

Se consideró que para lograr una correcta transmisión de video sería necesario disponer de los siguientes recursos de procesamiento:

`MprVideoFromNet:` este recurso recibe los paquetes UDP que arriban a la aplicación por los puertos seleccionados para las sesiones RTP y RTCP de video, extrae el contenido correspondiente a los paquetes RTP y lo envía al próximo recurso.

`MprVideoDejitter:` la función de este recurso es almacenar los paquetes RTP recibidos. Dado que el protocolo de capa de transporte utilizado es UDP, empleado en aplicaciones de tiempo real, no se puede garantizar que los paquetes arriben en orden. Por esta razón es necesario almacenar los paquetes en la recepción, y así poder entregarlos en el orden correcto al recurso que se encarga de decodificar el video recibido.

`MprVideoDecode:` este recurso recibe los paquetes RTP ordenados por su número de secuencia y se encarga de extraer de los mismos la carga úitl de video. Cuando logra reunir la información correspondiente a una imagen completa entonces entrega esta información al decodificador para obtener las imágenes decodificadas. Estas se van guardando en un buffer diseñado especialmente con este propóstio (`MpVideoJitterBuffer`) de donde luego se retiran para ser entregadas de forma adecuada al siguiente recurso.

`MprVideoBridge:` este recurso cumple la función de interconectar los recursos de modo que todos los posibles caminos de procesamiento que pueden seguir los datos queden conectados. Es decir, conecta el procesamiento local del flujo de video, con el procesamiento a nivel de la conexión.

`MprToScreen:` este recurso actúa de interfaz entre los datos que se recibieron de la red y la pantalla del agente de usuario. Su tarea es recibir las imágenes de video previamente decodificadas y colocarlas en un buffer propio del recurso (`mpScreenQ`) para que el hilo `ScreenThreadWnt` que maneja la visualización de las imágenes en pantalla, los vaya retirando cuando lo crea conveniente.

`MprFromCamera:` este recurso se encarga de recibir los datos correspondientes con las imágenes capturadas por la cámara web para enviarlas al siguiente recurso. Para obtener esta información, el recurso iteractúa con el hilo `CameraThreadWnt` quien va colocando en un buffer del recurso (`mpCameraQ`) cada imagen capturada para que `MprFromCamera` las retire al comenzar con su procesamiento.

`MprVideoEncode:` una de la funciones de este recurso es recibir la información de las imágenes capturadas por la cámara y enviarla al codificador. Éste codifica cada cuadro de video recibido y lo fragmenta para poder encapsularlo en varios paquetes RTP. A cada fragmento se le agrega el encabezado de video según se define en la RFC 2250. Una vez codificados y fragmentados los cuadros de video, son enviados al siguiente recurso.

`MprVideoToNet:` este recurso recibe los cuadros de video codificados y ya fragmentados, y los encapsula en paquetes RTP para enviarlos por la red. El encapsulado en RTP se realiza según la RFC 3550.

El objetivo de modelar la transmisión de un flujo de medios a través de los mencionados recursos es separar el procesamiento de los datos en distintas etapas de modo de poder trabajar con la información de medios de manera precisa y ordenada. Para esto se establece un orden de ejecución entre los recursos, y cuando cada uno finaliza su tarea le entrega los datos procesados al siguiente para que continúe con el resto de las tareas.

En general, las tareas de procesamiento propias de cada recurso se definen en un método común a todos ellos denominado `doProcessFrame`. Este es un método de gran importancia en el desarrollo de la aplicación a través del cual se vinculan los recursos sucesivos para pasarse los datos procesados. En el turno de procesamiento de cada recurso se invoca a este método quien, a partir de los datos recibidos, realiza las tareas asignadas y al finalizar entrega al siguiente recurso el resultado obtenido.

5.7.2. Conexión de video y sus recursos asociados

Como se explicó al comienzo de la sección, una de las partes fundamentales del procesamiento de una sesión de medios es la conexión entre los dos agentes de usuario que establece cuál es el camino para el flujo de tráfio que se genera una vez que la llamada está establecida.

Un primer paso para establecer una conexión, es crear un vínculo entre los puertos de cada agente de usuario que van a formar parte de la sesión de medios. Esto se realiza durante la negociación de la llamada a través de los mensajes SIP, como ya se explicó en secciones anteriores.

Una vez establecida la conexión a nivel de puertos, es necesario establecer el camino que debe seguir el flujo de medios previo a ser enviado por la red, y luego cuando es recibido por la red. Dado que la conexión de audio es independiente de la de video y se transmiten por dos canales RTP diferentes, es necesario determinar los caminos para cada una de dichas sesiones.

Para la transmisión de audio, sipXezPhone ya dispone de un objeto que modela esta conexión y que determina cuál es el procesamiento que recibe cada paquete de información que viaja por la red. Este objeto se representa con la clase `MpConnection`, que dispone de todos los métodos y atributos necesarios para modelar una conexión de audio. Entre estos atributos, se encuentran los recursos de procesamiento de medios quiénes se encargan de realizar el procesamiento de los datos en cada etapa del camino.

Para disponer de transmisión de video en la aplicación surge la necesidad de implementar un objeto adecuado para modelar una conexión de video. Siguiendo el modelo de diseño elegido, se procedió a crear una clase análoga a `MpConnection` pero que permitiera representar la sesión de video. Ésta es la clase `MpVideoConnection`, la cual determina el camino de procesamiento seguido por los paquetes de video a la entrada y a la salida de la red. Al igual que en

la conexión de audio, `MpVideoConnection` tiene aociados los recursos de video adecuados para llevar a cabo las diferentes tareas de procesamiento.

Al momento de crear la conexión de video, fue necesario determinar qué recursos formarían parte de la misma. Se consideró que para conformar la `MpVideoConnection` era necesario disponer de los siguientes recursos:

- `MprVideoFromNet`
- `MprVideoDejitter`
- `MprVideoDecode`
- `MprVideoEncode`
- `MprVideoToNet`

Las tareas desarrolladas por cada recurso se describieron con anterioridad y más adelante se explica su funcionamiento con mayor detalle.

Para cada conexión de video que se desea establecer, se crea una nueva `MpVideoConnection` y con ella todos los recursos asociados. En el caso de sipXvideoPhone, la aplicación cuenta con una única instancia de `MpVideoConnection` y una única instancia de cada uno de sus recursos de procesamiento de medios.

Al crearse la instancia de `MpVideoConnection`, se crean con ella las instancias correspondientes a los recursos previamente mencionados. Estos a su vez son interconectados según el orden que se creyó conveniente para que el procesamiento de los datos resultara el deseado.

Las figuras 5.14 y 5.15 muestran los dos posibles caminos para los datos de video.

Figura 5.14: Camino de la conexión en la recepción

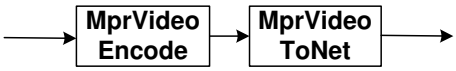


Figura 5.15: Camino de la conexión en la transmisión

Por una parte se encuentra el camino que siguen los datos cuando son recibidos por la red. Éstos se reciben en el recurso `MprVideoFromNet`, se depositan en el `MprVideoDejitter`, y finalmente son procesados por el `MprVideoDecode` quien entrega las imágenes ya decodificadas

al resto de la aplicación.

Por otra parte se tiene el camino que deben seguir los datos que son generados en el usuario local y que luego van a ser enviados por la red. Aquí los datos son procesados por el recurso `MprVideoEncode` quien se encrga de codificar y fragmentar la información según se cree conveniente, y luego el recurso `MprVideoToNet` toma el control para enviar mediante RTP, los datos hacia el usuario remoto.

A partir de este momento, queda establecida la conexión de video que determina los caminos por donde deben pasar los datos previo a ser enviados por la red y luego de ser recibidos por la misma.

A modo resumen, en sipXvideoPhone para modelar la llamada SIP con varios flujos de medios se utilizan conexiones a niveles de puertos representativas de las sesiones de medios que se desea establecer. Más explícitamente se utiliza una conexión que modela la sesión de audio (representada por el objeto `MpConnection`) y otra que modela la sesión de video (representada por el objeto `MpVideoConnection`).

5.7.3. Gráfica de video y sus recursos asociados

En la aplicación de SIPfoundry, todo el procesamiento de medios se modela con una gráfica de recursos de procesamiento donde se encuentran *Dejitter*, *Decode*, *Encode*, entre otros. Esta gráfica se utiliza para controlar la ejecución de cada recurso de modo que uno por vez realice sus tareas.

La gráfica se encarga de que los recursos se ejecuten en el orden correcto, asegurando así que los recursos que generan buffers de salida se invoquen antes que aquellos recursos que deben consumir esos buffers. Una vez establecido el orden de ejecución de los recursos, se comienza la cadena de procesamiento a través de los métodos `doProcessFrame` que presenta cada uno.

En el desarrollo de la aplicación sipXezPhone, a esta gráfica se le denomina `MpCallFlowGraph`, y modela el procesamiento de una sesión únicamente de audio. Para integrar a la aplicación el procesmiento de video, se construyó una gráfica similar denominada `MpVideoFlowGraph` que se encarga de interconctar y manejar los recursos de procesamiento de video de forma ordenada. La aplicación sipXvideoPhone dispone de una única instancia de cada una de estas gráficas a través de las cuales se controla la ejecución de los recursos de cada sesión de manera independiente.

Si hacemos referencia al comienzo de esta sección allí se explicó que el procesamiento de una sesión de medios se separaba en dos grupos: la conexión y el procesamiento local en cada agente de usuario.

Según lo explicado en 5.7.2 para la conexión de video, ésta presenta un cantidad de cinco recursos de procesamiento de medios que determinan el camino de entrada y de salida de los

datos desde y hacia la red. Si se observan estos caminos se nota la ausencia de algún tipo de interfaz con los agentes de usuario, así como también de un objeto que sea capaz de interconectar los dos caminos de la conexión.

Para cumplir con estas necesidades, se utilizan tres recursos más de procesamiento de video. Estos recursos pertenecen al objeto `MpVideoFlowGraph`, y se listan a continuación:

- `MprFromCamera`
- `MprToScreen`
- `MprVideoBridge`

Los recursos `MprFromCamera` y `MprToScreen` se encrgan de realizar el procesamiento local de la información en cada agente de usuario. `MprFromCamera` es el recurso que permite acceder a los datos generados por la cámara para pasarlos al recurso `MprVideoEncode` y a partir de éste continuar con el camino de la conexión. Del mismo modo, el recurso `MprToScreen` se encarga de que los datos a la salida del `MprVideoDecode` sean entregados a la aplicación para desplegar el video correctamente en pantalla.

En cuanto al recurso `MprVideoBridge`, éste se encargará de interconectar los dos posibles caminos de procesamiento que puede seguir la información de video durante la llamada. Al hablar de dos posibles caminos, se hace referencia al siguiente esquema:

1. Cuando los datos provienen de la red, éstos ingresan a la gráfica a través del recurso `MprVideoFromNet` y seguen el camino establecido en la conexión de video (`MpVideoConnection`). Finalmente llegan al recurso `MprToScreen` quién se encargará de entregar los datos para que el video sea mostrado en la pantalla del usuario local.
2. Cuando los datos van a ser enviados por la red hacia el usuario remoto, ingresan a la gráfica de procesamiento a través del recurso `MprFromCamera` y luego continúan por el segundo camino establecido en la conexión de video (`MpVideoConnection`). Finalmente son entregados a la red mediante del recurso `MprVideoToNet`.

A modo aclarativo se ilustra en la figura 5.16 lo antes descrito.

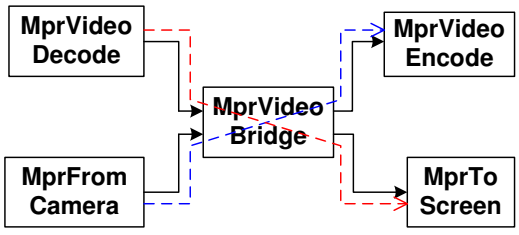


Figura 5.16: Conexión de recursos a través de MprVideoBridge

Al interconectar estos dos caminos mediante el `MprVideoBridge`, se obtiene la gráfica `MpVideoFlowGraph` con todos los recursos de procesamiento de medios que hacen posible la transmisión un flujo de video durante la llamada.

La gráfica `MpVideoFlowGraph` es instanciada durante la inicialización del teléfono y permanece activa mientras la llamada está establecida. Al momento de crearse la instancia, se crean también las instancias correspondientes a los recursos propios de la gráfica como lo son el `MprFromCamera`, `MprVideoBridge`, y `MprToScreen`.

Una vez creada la gráfica, el `MpVideoFlowGraph` es el encargado de instanciar la `MpVideoConnection` y de interconectar los recursos de la misma con los tres recursos propios de la gráfica.

Finalizadas las etapas de inicialización de los recursos, se dispone de una gráfica `MpVideoFlowGraph` capaz de manejar el comportamiento de todos los recursos asociados a la sesión de video. A continuación, en la figura 5.17 se ilustra la interconexión final de todos los recursos:

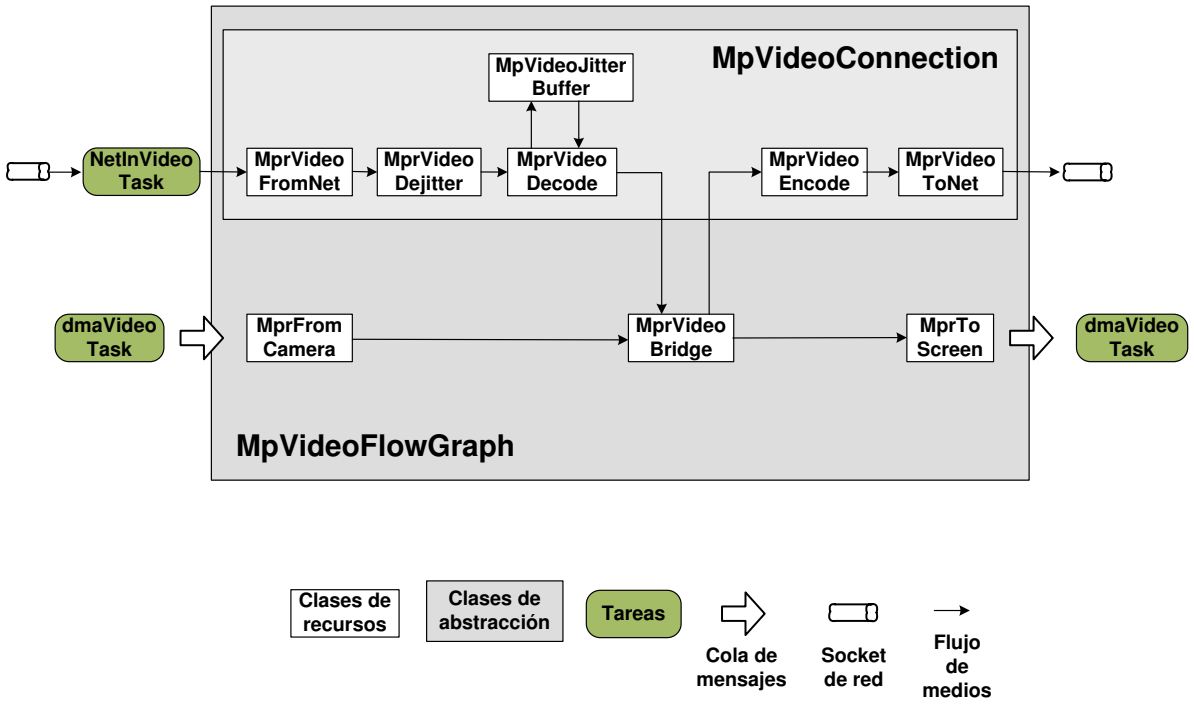


Figura 5.17: VideoFlowGraph

5.7.4. Otros elementos que forman parte de la arquitectura

En esta sección se describen las funcionalidades básicas de algunos elementos del programa que también resultan de importancia en el desarrollo. Los detalles de implementación se encuentran en el punto 8.2.1.

MpVideoJitterBuffer

Uno de los elementos que forma parte de la arquitectura básica de sipXvideoPhone es el denominado `MpVideoJitterBuffer`. Como su nombre lo indica, es un objeto del tipo buffer y es utilizado para almacenar imágenes de video decodificadas.

El objetivo del `MpVideoJitterBuffer` es disponer de un lugar de memoria donde ir depositando las imágenes de video en la medida que van siendo decodificadas, para luego retirarlas a la tasa deseada en el momento que se crea conveniente. Este elemento es miembro de la conexión de video (`MpVideoConnection`) e interactúa únicamente con el recurso `MprVideoDecode` según como se describe a continuación:

Cuando `MprVideoDecode` recibe los paquetes RTP con información de video retira el contenido de carga útil para entregarle al decodificador. A la salida de éste se obtiene una imagen completa decodificada la cual es almacenada directamente en el `MpVideoJitterBuffer`. Un vez que la imagen es depositada en este buffer, permanece allí hasta que el mismo `MprVideoDecode` considere que es el momento de retirarla.

MpMisc

`MpMisc` es una estructura que, como su nombre lo indica, no cumple una función específica si no que se utiliza para definir distintos tipos de elementos que son utilizados a lo largo de todo el desarrollo de la aplicación.

Una de las funcionalidades de `MpMisc` es que en ella se definen una gran cantidad de buffers de memoria que son reservados durante la inicialización el teléfono y que luego son utilizados con fines específicos durante la ejecución del programa. Estos bloques de memoria se reservan de a grupos en los denominados *pools* de buffers, y permiten almacenar alguna estructura de datos determinada.

El modo en que se utilizan los *pools* de memoria es el siguiente: cuando en el programa se precisa almacenar algún tipo de información para que quede disponible en cualquier momento a cualquier objeto de la aplicación, se solicita un bloque de memoria de los disponibles en el *pool* correspondiente. Los datos son copiados en dicho buffer y permanecen allí hasta que en otra parte del programa se requiera obtener esos datos. Para poder acceder a este bloque de datos se utiliza un puntero correspondiente con el tipo de datos del que se trate y una vez que los datos son retirados del buffer, el puntero es liberado.

CameraThreadWnt

`CameraThreadWnt` es el hilo responsable de entregar a la aplicación las imágenes de video capturadas por la cámara. Interactúa con `sipXmediaFactoryImpl`, quien cada vez que tiene una imagen capturada la pone en la cola del hilo. Estas imágenes se van guardando en un arreglo y luego el hilo las saca para procesarlas. Una vez sacada una imagen, se la guarda en una cola propia del primer recurso de la gráfica que refiere a la transmisión, `MprFromCamera`.

ScreenThreadWnt

`ScreenThreadWnt` es el hilo que establece el vínculo entre la gráfica de procesamiento y el despliegue del video en pantalla. Una vez recepcionadas y procesadas las imágenes de video, éstas deben ser desplegadas en la pantalla de visualización.

El último recurso de procesamiento presente en el camino de recepción es el MprToScreen y es quien dispone de las imágenes de video ya decodificadas. `ScreenThreadWnt` interactúa con este recurso de donde va retirando las imágenes de video para ir mostrándolas en la pantalla del agente de usuario.

DmaVideoTaskWnt

`DmaVideoTaskWnt` es el *task* correspondiente con las tareas de procesamiento en los extremos de la conexión en cada uno de los agentes de usuario. Es la clase encargada de manejar los dispositivos finales de la conexión de video; la pantalla de visualización y la cámara web.

Aquí se definen constantes de utilidad en toda la aplicación. Las mismas se relacionan con el tamaño de pantalla elegido, la cantidad de bits con que se representa cada píxel y la cantidad de cuadros por segundo que se desea transferir.

Otra función de la clase es inicializar dos hilos de ejecución y darle a cada uno un nivel de prioridad. Los hilos manejados aquí son los que actúan de interfaz entre la gráfica de procesamiento de video y los dispositivos finales.

Por una parte se maneja el hilo `CameraThreadWnt`, el cual hace de interfaz entre `MpVideoFlowGraph` y la cámara de video. Del mismo modo, se controla la ejecución de `ScreenThreadWnt`, quien interactúa con las pantallas donde se muestra el video.

NetInVideoTask

`NetInVideoTask` es una clase que maneja la recepción de los datos que provienen de la red. Aquí se encuentra implementado todo lo relacionado con los *sockets* de video que intervienen en la aplicación.

La clase cuenta con un método `run()` que se está ejecutando permanentemente y donde se maneja la utilización de los *sockets* para que éstos puedan recibir los datos en el momento

adecuado. El método está implementado de manera que los *sockets* se encuentran escuchando en todo momento y así detectar la llegada de los datos desde la punta remota.

MpMediaTask

`MpMediaTask` (tarea de procesamiento de medios) es el elemento responsable de controlar la ejecución de las gráficas de procesamiento y en consecuencia de todos los recursos que las componen. Esto se aplica tanto para el caso de audio con el `MpCallFlowGraph` como para el de video con el `MpVideoFlowGraph`.

A lo largo de todo el programa los datos son procesados durante intervalos de tiempo denominados *frame intervals*. En cada uno de estos intervalos, la tarea de procesamiento de medios recibe una notificación indicando que puede comenzar a procesar un nuevo bloque de datos. En el caso de video la notificación es enviada por el hilo `ScreenThreadWnt` al momento de finalizar con el procesamiento de cada imagen. En lo que corresponde a la sesión de audio, se lleva a cabo en el hilo análogo, `SpeakerThreadWnt`.

Dado que la aplicación cuenta con una gráfica de procesamiento por cada tipo de medio disponible en la sesión, es necesario establecer un orden de ejecución de las mismas de modo que puedan coexistir adecuadamente. Únicamente una gráfica por vez puede tener el foco provocando que solamente se encuentren accesibles sus recursos de procesamiento. El objeto responsable de manejar las gráficas de procesamiento es `MpMediaTask`, quien asigna los turnos de ejecución.

5.7.5. Testeo de la arquitectura básica utilizando una imagen de prueba

Una vez implementada la arquitectura básica de sipXvideoPhone, se procedió a testear su comportamiento para verificar si realmente se cumplían los objetivos deseados.

El testeo consistió en enviar por la red una secuencia de datos fija que representaran una imagen cualquiera pero distinguible en el agente de usuario remoto. Se eligió una imagen de prueba suficientemente pequeña para poder enviarla sin comprimir en un único paquete RTP y así tener un mejor aprovechamiento del ancho de banda. En esta etapa se excluye la captura de las imágenes y las etapas de codificación y decodificación.

Tomando como referencia el diagrama del `MpVideoFlowGraph` y la `MpVideoConnection`, se hizo un seguimiento de los datos verificando que los mismos fueran los correctos en cada tramo del camino. Para ello fue necesario detenerse en cada recurso que compone el camino de procesamiento para analizar si los datos de entrada eran los esperados, cómo se estaban realizando las tareas propias del recurso, y finalmente que los datos entregados al siguiente recurso fueran los correctos.

A continuación se detalla paso a paso el procesamiento que recibe el flujo de video tanto en la transmisión como en la recepción.

Transmisión

Para transmitir una imagen hacia el agente de usuario remoto es necesario que la misma sea procesada previos a ser enviada por la red. El camino a seguir por los datos que se generan en la máquina local y son enviados a la aplicación remota se observa en la figura 5.18.

A continuación se describe cómo el flujo de datos es enviado de recurso en recurso hasta llegar a transmitir los datos por la red.

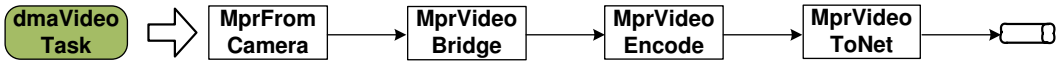


Figura 5.18: Camino de los datos generados localmente

1. El primer lugar en que se dispone de datos para ser enviados es `sipXmediaFactoryImpl`, donde se generó una imagen con un conjunto de valores conocidos para testear el funcionamiento de la aplicación. Esta imagen debe ser enviada al hilo de ejecución `Camera ThreadWnt`, quien establece el vínculo con los recursos de procesamiento.

   La imagen recibe un procesamiento en el hilo `CameraThreadWnt` y es almacenada en una cola propia del primer recurso de la gráfica. A su vez el manejador de recursos `MpMediaTask`, es notificado de que una nueva imagen está disponible para ser procesada.

2. El primer recurso por el que pasan los datos luego de ser capturados por la cámara es `MprFromCamera` y esto se hace a través de la cola propia de la clase, `mpCameraQ`. Cada vez que a este recurso le llega el turno para procesar revisa que la cola no tenga muchas imágenes acumuladas y se pase de rango, en caso de que esto ocurra descarta las más antiguas. Verificado esto pasa a sacar una imagen, según un formato de cola del tipo FIFO, y la envía al próximo recurso.

3. `MprVideoBridge` es el próximo que recibe datos y lo hace a través de su método `doProcess Frame`. Los datos ingresan por el camino que corresponde a la transmisión y deben salir por el camino que los lleva hacia la red. La única funcionalidad de este recurso es interconectar dos flujos de medios. Uno de estos proviene de la parte local de aplicación y debe ser encaminado hacia la red. El otro flujo proviene de la aplicación remota y tiene que ser enviado a quien localmente procesa los datos para desplegarlos en pantalla. Para la parte de transmisión éste conecta el camino que viene desde la camara hacia donde se debe codificar y enviar datos a la red.

4. Una vez que los datos generados localmente están encaminados hacia el primer recurso perteneciente a la conexión de video, se llega a `MprVideoEncode`. Cuando los datos son recibidos en `doProcessFrame`, se verifica que esten llegando correctamente y de la misma forma que son recibidos se encaminan al recurso que los envía a la red. Cabe señalar que en el caso de haber etapa de codificación, este recurso sería el encargado de llevarla a cabo.

5. En `MprVideoToNet` los datos son encapsulados en un paquete RTP siguiendo las especificaciones de la RFC 3550. Siguiendo la recomendación se crea un encabezado que resulta de utilidad en la recepción para poder determinar cierta información. Por ejemplo el valor del campo `Payload Type` se utiliza para identificar el tipo de medios que transporta el paquete. También es posible detectar la pérdida de un paquete mediante un seguimiento del valor `Sequence Number`, el mismo aumenta en una unidad por cada paquete RTP enviado que pertenezca al mismo flujo de medios.

   Luego de crear el paquete RTP, éste debe ser escrito en el socket *socket* correspondiente a la sesión RTP de video. La escritura del paquete en el *socket* se hace mediante la ejecucución de varios métodos que culminan en funciones de la biblioteca *winsock* [39], ampliamente utilizada para trabajar con *sockets*. Así el paquete queda enviado por el canal adecuado hacia el agente de usuario remoto y se da por finalizada la etapa de transmisión.

Recepción

Cuando los datos de video arriban al agente de usuario remoto a través de la red, deben recibir cierto procesamiento de modo de poder finalmente mostrarse en pantalla.

Según el `MpVideoFlowGraph`, el camino establecido para el flujo de video cuando este proviene de la red es el que indica la figura 5.19.

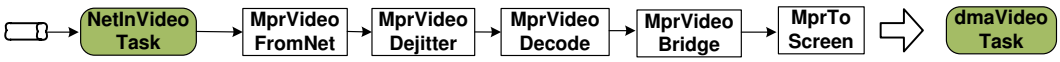


Figura 5.19: Camino de los datos provenientes de la red

1. Los paquetes arriban a la aplicación a través de `NetInVideoTask`. Este *task* lleva el control de la información que proviene de la red y se encarga de verificar continuamente que hayan datos para procesar en la llegada. Cuando encuentra que hay datos, la aplicación comienza a procesarlos.

   A partir del *socket* correspondiente a la transmisión de video, es posible determinar la dirección IP y el puerto en cuestión para proceder a leer los paquetes UDP recibidos. Cuando se encuentra que por el puerto de la conexión se recibió un paquete UDP, la información contenida en el mismo comienza a seguir el camino del `MpVideoFlowGraph`. Para esto, se le entrega al primer recurso del camino un puntero al paquete UDP recibido.

2. El primer recurso de la gráfica que recibe la información de video es el `MprVideoFromNet`, quien a través de los paquetes UDP se encarga de extraer el contenido RTP correspondiente. Se realiza un análisis de los distintos campos del encabezado RTP por ejemplo determinando si el paquete utiliza *padding*, calculando la posición en donde comienza la carga útil, el tipo de carga útil que contiene el paquete.

   Cada paquete RTP recibido, es almacenado en un lugar de memoria previamente reservado para ello. Este lugar de memoria es un buffer del tipo `MpRtpBuf` especialmente

diseñado para guardar paquetes RTP, y se obtiene del *pool* de buffers reservado en la clase `MpMisc` durante la inicialización del teléfono.

Las estructuras de datos `MpRtpBuf` constan de un encabezado y una carga útil que permiten un un fácil manejo de la información recibida en cada paquete RTP. Por su parte, el encabezado presenta la posibilidad de almacenar la información correspondiente a todos los campos de un encabezado RTP según se define en la RFC 3550.

A medida que los paquetes RTP van siendo almacenados en memoria van quedando referenciados por un puntero del tipo `MpRtpBufPtr`, también definido especialmente para este tipo de datos. Este puntero es entregado al siguiente recurso de la gráfica.

3. El siguiente recurso es el `MprVideoDejitter`. Este es el responsable de almacenar todos los punteros que referencian los paquetes RTP que han sido recibidos, de manera de luego poder retirarlos en orden según el menor número de secuencia disponible. Para esto, el `MprVideoDejitter` dispone de un arreglo de tamaño fijo y dos métodos que permiten a otros recursos ingresar un nuevo puntero y retirar uno existente. El mecanismo de almacenamiento de `MprVideoDejitter` es un paliativo al posible arribo de los paquetes RTP en desorden.

   Los paquetes RTP que van arribando a la aplicación van quedando almacenados en lugares de memoria conocidos por el `MprVideoDejitter` y permanecen allí hasta que el siguiente recurso los retira.

4. Una vez que la referencia a los paquetes RTP queda registrada en el `MprVideoDejitter`, el siguiente recurso de la gráfica debe continuar con el proceso. Este es el `MprVideoDecode` quien debe retirar del `MprVideoDejitter` el puntero al próximo paquete RTP para realizar las tareas correspondientes con el contenido de carga útil que el mismo contiene. El objetivo del recurso `MprVideoDecode` es reunir la carga útil correspondiente a cada imagen de video que va arribando por la red, decodificar dicha imagen, y entregarla al siguiente recurso.

   `MprvideoDecode` interactúa con el `MpVideoJitter- Buffer`, donde va depositando las imágenes ya decodificadas. Dado que aquí no se utiliza ningún codec de compresión, lo que se deposita en el arreglo son los datos de video directamente retirados del paquete RTP.

   El procesamiento del `MprVideoDecode` se lleva a cabo en su método doProcessFrame y puede separarse básicamente en dos etapas:

   - La primera etapa consiste en retirar los paquetes RTP del buffer de memoria según un orden creciente de número de secuencia. El contenido de carga útil correspondiente con las imágenes de video recibidas se almacena en el `MpVideoJitterBuffer`. Este procedimiento se repite hasta que el recurso anterior no disponga de más referencias a paquetes RTP arribados.
   - Se procede a retirar una imagen de video de las disponibles en el `MpVideoJitter-Buffer`. Los datos correspondientes con la imagen retirada se almacenan en uno de los bloques de memoria que fueron reservados con este fin. Este bloque es del tipo

`MpVideoBuf` y queda referenciado por un puntero del tipo `MpVideoBufPtr` según como se definió en la clase `MpMisc`.

Finalizado el procesmiento descrito, el recurso entrega al `MprVideoBridge` el puntero al bloque de memoria donde se encuentra la imagen recién procesada.

5. `MprVideoBridge` no realiza ningún tipo de procesamiento con los datos y solamente actúa de puente entre otros recursos de la gráfica. En el caso de la recepción, debe entregar los datos de video al recurso `MprToScreen`.

6. Finalmente cuando los datos son recibidos por el último recurso del camino, `MprToScreen`. Éste dispone de una cola en la cual se van depositando todas las imágenes procesadas que se reciben de los recursos anteriores. Interactúa con uno de los hilos que permanecen corriendo a lo largo de la aplicación (`ScreenThreadWnt`), el cual hace de interfaz con la pantalla del agente de usuario final. Cuando la aplicación está pronta para desplegar una nueva imagen en pantalla, este hilo recurre a la cola del `MprToScreen` para retirar la próxima imagen de video y hacer que se despliegue en pantalla, tal como fue explicado al describir el funcionamiento del hilo.

Con este análisis realizado en cada tramo de la gráfica, se verificó que el pasaje de los datos se realizaba correctamente y que la arquitectura diseñada permite transmitir información por los caminos creados para video.

A partir de este momento, se dispone entonces de la arquitectura básica para poder transmitir información de video entre dos agentes de usuario sipXvideoPhone. De ahora en más, resta incorporar al desarrollo otras funcionalidades propias del video como por ejemplo la codificación y decodificación del mismo.

5.8. Manejo de dispositivos de hardware

Con el fin de incorporar el soporte de video se debe previsualizar la captura de la cámara y obtener las muestras de video para luego procesarlas y enviarlas. El usuario que está empleando el teléfono debe poder ver lo que captura su cámara. Esto es muy útil ya que de esta forma se puede saber si la aplicación está manejando la cámara de forma correcta y además, al momento de establecer una llamada, saber qué se le va a enviar al otro agente de usuario.

Además es necesario en el extremo receptor desplegar el video que llega de la red para que la video llamada se pueda realizar.

5.8.1. Elección de la plataforma

Para la implementación de la previsualización se analizaron distintas variantes en cuanto a la plataforma a emplear. Entre ellas se estudiaron *Video For Windows* y *DirectShow*.

Video For Windows

Video For Windows (VFW) es una plataforma multimedia desarrollada por Microsoft que permite el procesamiento de video. Su surgimiento se debió al gran crecimiento de cámaras digitales de bajo costo que hubo en ese entonces [40].

Es un conjunto de interfaces de programación de aplicaciones (API) que permiten un simple manejo de dispositivos de audio y de video. Introdujo el formato digital para almacenamiento de video llamado Audio Video Interleaved (AVI) o Audio y Video Entrelazados.

VFW provee una interfaz de programación para que desarrolladores de software que trabajen sobre la plataforma Windows manejando dispositivos de video digital en sus aplicaciones. Si bien para muchos desarrolladores fue suficiente, tiene una serie de limitaciones que incentivaron la búsqueda de una nueva plataforma. Una de sus principales limitaciones es que no permite el soporte del formato de video MPEG, ni de ningún tipo de codificación que durante el proceso de compresión altere el orden de los cuadros.

DirectShow

Con el surgimiento de Windows 95, Microsoft comenzó un proyecto llamado Quartz para proveer nuevas APIs para el soporte de MPEG a Video For Windows. Dado que no era posible imaginar todos los nuevos escenarios ocasionados por las nuevas tecnologías para hacer una única API y como se pretendía un nivel de entendimiento mayor, la idea de agregarle funcionalidades a VFW fue descartada y se optó por desarrollar una nueva arquitectura [41].

Así con Quartz se buscó desarrollar una arquitectura modular que permitiera agregar componentes básicos desarrollados, llamados filtros, para desempeñar distintas funciones [42].

Usando *Common Object Model* (COM) se conectaron estos filtros mediante gráficas produciéndose un flujo de bits desde el dispositivo de captura, pasando por procesos intermedios, hasta el dispositivo de salida. Mediante COM cada filtro era capaz de averiguar capacidades y disponibilidades de otros filtros al ir conectándose en la gráfica. El hecho de que los filtros sean objetos auto contenidos COM, permite a desarrolladores crear sus propios filtros dependiendo de sus necesidades particulares tanto de dispositivos de hardware como de software.

De esta forma Quartz evolucionó a una API que provee procesamiento de audio y video llamada **DirectShow**.

Consideraciones para la elección

A modo de evaluar las distintas posibilidades para implementar software para la previsualización y captura de video, se estudiaron las características de ambas plataformas y se analizaron diversas aplicaciones existentes.

Como ejemplo de aplicación de VFW se consideró **AVICap**. Para esto fue necesario compilar el código obtenido de *CodeGuru* y luego proceder a analizarlo [43].

La clase CAviCap provee las funcionalidades de AviCap. Es un ejemplo del uso de Microsoft Foundation Classes (MFC) que son básicamente un contenedor de la API Win32. Entre sus capacidades se encuentra el listado de drivers, la conexión a los mismos, el seteo de tamaño de los cuadros, de resolución y de formato. Permite el acceso a dispositivos de adquisición de audio y video.

Para utilizar estas clases se debe crear un objeto CAviCap y llamar al método *Create* para crear la ventana. Luego se deben encontrar los drivers mediante el método `GetDriverList` y conectarse con el elegido usando `ConnectWithDriver`. El driver debe ser preferentemente compatible con VFW 1.1. Si no es del todo compatible se debe setear la bandera `m_DoQuickConnection` para que no se hagan ciertas verificaciones a la hora de conectarse con el driver. Esta condición pone limitaciones sobre las cámaras.

Se proveen métodos para el cambio de tamaño de los cuadros, el seteo de la resolución de color y el formato. Éstos son: `SetFrameSize`, `SetBitResolution` y `SetFormat`.

Para incluir a la aplicación del teléfono se debe hacer una función *callback* usando `SetFrameCallBack` para acceder a los datos de las imágenes. El driver llama a esa función cada vez que un nuevo cuadro es capturado en respuesta a un evento generado por un temporizador interno. El driver pasa entonces un puntero a la estructura `VIDEOHDR` del callback. La estructura `lpData` apunta al buffer de datos de imagen.

Si bien la inclusión al código de esta solución era viable, sus limitaciones en cuanto a performance en tiempo real, llevaron a la consideración de otras soluciones.

Como aplicación basada en DirectShow, se estudió **AMCap** incluida dentro de los ejemplos de captura de Microsoft SDK [44]. Es una aplicación de captura de video que posibilita grabar un archivo, previsualización en tiempo real, despliegue de propiedades del dispositivo, alocación de memoria, enumeración de dispositivos y control del flujo.

Para incluirla en el código del teléfono se debió encontrar el driver de la cámara, inicializar los filtros (componentes) y luego conectarlos. Al poseer DirectShow una arquitectura modular se puede interpretar el código más fácilmente en comparación con el de *AviCap*.

Otra aplicación que se consideró fue **VirtualDub** descargado desde *SourceForge* [45]. En este caso se puede ver una aplicación que usa tanto Video For Windows como DirectShow. Es un muy buen ejemplo de aplicación de cada una de estas herramientas de manejo de dispositivos de hardware.

Un factor que también se tuvo en cuenta para la elección, fue la documentación disponible para ambas herramientas. Si bien la ayuda del Microsoft SDK tiene información de las dos, la

de DirectShow es más profunda y clara. Esto tuvo gran importancia en la decisión final debido a la falta de experiencia en el desarrollo de software para manejo de dispositivos.

Entre la documentación a disposición también se tuvo en cuenta la cantidad de foros y desarrolladores de software trabajando con la plataforma. Para el caso de DirectShow hay disponibles (y sumamente activos) el foro de MSDN de Microsoft [46] y un grupo de Google de video y DirectX [47]. Al ser herramientas muy especifícas resultó importante tener la posibilidad de preguntarle a personas más experimentadas.

Cabe señalar que existe también un grupo de Google para VFW. En éste se investigó sobre tutoriales o documentos que explicaran Video For Windows de forma más teórica y se señaló que no se encontraban tutoriales, sino que se debía partir de algún ejemplo para entender. Se nombra *VirtualDub* como un buen ejemplo del cual partir.

A su vez como Video For Windows evolucionó a DirectShow y las cámaras web más recientes se basan en el modelo de *Windows Driver Model* (no soportado por VFW) se consideró más conveniente trabajar con lo último.

Por todos estos motivos se decidió utilizar DirectShow para el manejo de dispositivos de video. Se tomó como ejemplo de partida *AMCap*.

5.8.2. Descripción de DirectShow

Generalidades y capacidades

Microsoft DirectShow es una interfaz que permite el manejo de medios y dispositivos en la plataforma Microsoft Windows. Soporta una gran variedad de medios como ASF, MPEG, AVI, MP3, WAV y MIDI entre otros.

Entre sus capacidades se destacan la captura de medios, la transformación y reproducción de los mismos en dispositivos tanto de audio como de video. Soporta la captura de dispositivos digitales y analógicos basados en Windows Driver Model (WDM) o en Video For Windows.

Estas tres principales áreas reflejan los tres tipos básicos de filtros de DirectShow (un filtro es el componente básico de DS).

La captura puede ser de audio desde un micrófono, de audio y video desde una fuente en tiempo real como puede ser una cámara web. Esta captura se hace por los llamados *source filters* o filtros fuente. Una vez que los medios han sido capturados, es posible transformarlos mediante los llamados *transform filters* o filtros de transformación. Por ejemplo, se podría pasar de video a color a blanco y negro, cambiar el tamaño de imágenes de video o pasar de RGB a YUV. Se pueden conectar tantos filtros de transformación como sean necesarios. Luego de que se han hecho todas las transformaciones deseadas, queda únicamente entregar los medios a la pantalla, al parlante o algún otro dispositivo. También es posible escribir archivos. Esto se

hace mediante los *render filters* o filtros de despliegue. Por lo tanto estos últimos se encargan de mandar los datos a la tarjeta de video, de audio o al disco.

Dependiendo de la aplicación se utilizan o no las distintas capacidades proporcionadas por DirectShow. Como ejemplo de una aplicación desarrollada a través de DirectShow, se encuentra el reproductor Windows Media Player. En este caso, no es tan importante la captura de medios, lo que realmente importa es la capacidad de reproducir distintos formatos: MP3, WAV, AVI, MPEG.

DirectShow fue diseñado básicamente para aplicaciones desarrolladas en C++ o en Visual Basic. Se basa en el Common Object Model (COM) por lo que para escribir aplicaciones utilizando DirectShow es recomendable entender un poco de éste. Por este motivo se hace una breve explicación en el apéndice C.

Arquitectura

DirectShow posee un diseño modular el cual le da gran flexibilidad. Define un conjunto de interfaces COM para filtros que luego son conectados mediante gráficas.

En la figura 5.20 se puede ver un panorama general de una aplicación típica de DirectShow.

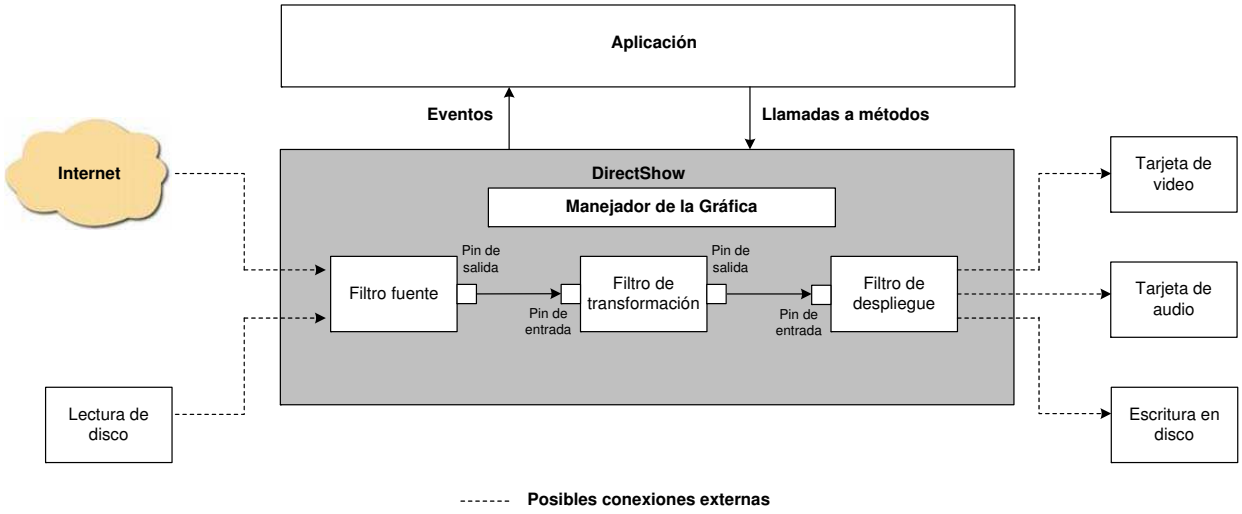


Figura 5.20: Arquitectura de DirectShow

Componentes

Los componentes principales de DirectShow son los filtros y las gráficas de filtros llamadas *filter graphs*.

Los filtros son los átomos de DirectShow. Son componentes básicos que permiten realizar distintas operaciones sobre el flujo de medios. Estas operaciones pueden ser la lectura de un archivo, la captura de video de una cámara web, la decodificación de medios o pasar datos a la tarjeta gráfica o de sonido.

Una gráfica de filtros está compuesta por varios filtros conectados entre sí. Se conecta la salida de un filtro con la entrada de otro. Las conexiones se producen por medio de pines.

Los filtros se manejan por medio de interfaces que permiten la comunicación entre ellos. Mediante las interfaces se negocian parámetros de la conexión como pueden ser el tipo de medios (audio o video) o el formato.

a. Filtros

Un filtro es una entidad completa en sí misma. Si bien pueden desempeñar distintas funciones deben tener al menos un método para recibir medios o uno para despacharlos.

Cada filtro está compuesto al menos por un *pin* que le permite conectarse a otro filtro. Un *pin* es entonces o bien una entrada o una salida del filtro.

Existen tres tipos de filtros: los filtros fuente o *source filters*, los filtros de transformación o *transform filters* y los filtros de despliegue o *render filters*. Todo filtro pertenece a alguna de estas tres categorías.

Los filtros fuente son los que introducen el flujo de medios en la gráfica. Cuentan tan solo con pines de salida. Todos los drivers WDM instalados son accesibles por DirectShow por lo que es posible obtener los datos de un micrófono o una cámara.

Los filtros de transformación, como su nombre lo indica, transforman los medios. Es decir, procesan los medios de acuerdo a requisitos o necesidades de la aplicación. Cuentan con al menos un pin de entrada y uno de salida, ya que reciben los medios de un filtro (puede ser un filtro de transformación o un filtro fuente) y luego los pasan a un filtro de despliegue. Entre las posibles operaciones que pueden realizar los filtros de transformación se encuentra la codificación y decodificación de medios, multiplexación, obtención de medios de la gráfica o simplemente pasaje de medios.

Los filtros de despliegue, son los últimos en la cadena de filtros conectados en la gráfica y se encargan de sacar los medios entregándolos por ejemplo a la tarjeta de sonido o la tarjeta gráfica.

No es posible conectar todo pin de salida a todo pin de entrada. Para que dos filtros se conecten es necesario que primero acuerden el tipo de medios que pasa entre ellos y la forma en que éstos se transportan. Por este motivo es que los pines tienen que publicar una lista de medios soportados y un mecanismo de transporte entre los pines de entrada y de salida de cada filtro.

Una vez que los filtros se ponen de acuerdo tanto en el tipo de medios como en su transporte, es que uno de los pines debe crear una reserva de memoria llamada *allocator* para poder pasar los datos. Esta reserva de memoria se puede encontrar en el pin de entrada o en el de salida que se están conectando.

Cuando se completa la negociación de medios, del transporte, y se crea la reserva de memoria se repite esta operación para todos los filtros pertenecientes a la gráfica.

b. Gráficas de filtros

Las gráficas de filtros organizan los filtros en una unidad. No es suficiente conectar los filtros, además se debe poder controlar el pasaje de medios entre ellos. Para esto se crea el manejador de gráficas de filtros, llamado *Filter Graph Manager*, que indica a los filtros cuándo comenzar a mandar los medios, detener o pausar los mismos. El manejador de filtros es un objeto COM que controla los filtros.

Las funciones de control cubiertas por el manejador de gráficas son las siguientes:

- Coordinación de estados entre filtros
- Reloj de referencia
- Comunicación de eventos a la aplicación
- Métodos para construir las gráficas de filtros

El reloj de referencia es necesario ya que los filtros deben estar sincronizados debido a que están trabajando con muestras. Este reloj es accesible por todos los filtros que componen la gráfica y les permite tener el flujo de medios en orden.

Los métodos que se proveen son para agregar, conectar y desconectar filtros a la gráfica.

Construcción de gráficas

En este punto se describen las ideas básicas de cómo construir una gráfica. Como se explicó anteriormente, una gráfica consiste en un conjunto de filtros conectados mediante pines.

Cuando dos filtros se conectan los pines negocian el tipo de conexión. Generalmente deben acordar el tipo de medios a conectar (si es audio o video, en qué formato, etc.), el tamaño

y cantidad de buffers de memoria necesarios y cuál de los dos filtros va a guardar a dichos datos.

Hay cuatro posibles estados para una gráfica: pausada, corriendo, detenida y en transitorio. En este último la gráfica está pasando de un estado a otro pero aún no ha completado la transición. Esto se debe a la naturaleza de hilos múltiples (multithreading) de DirectShow.

Lo primero que se debe hacer para construir una gráfica es instanciar al manejador de la gráfica mediante `IFilterGraph` o `IGraphBuilder`. Luego se deben agregar y conectar los filtros.

La interfaz `IGraphBuilder` provee métodos para llamar al manejador de la gráfica. Permite conectar los filtros en forma directa mediante sus pines usando el método `Connect`.

Para casos en que se trabaja con gráficas de captura, ya sea de audio o de video, se utiliza la interfaz `ICaptureGraphBuilder`. Permite conectar una secuencia de filtros usando `RenderStream`. Este método conecta el filtro fuente con el filtro de despliegue conectando los filtros intermedios necesarios.

El manejador de la gráfica usa el mecanismo de conexión inteligente (*Intelligent Connect*) que mediante métodos como `RenderStream` emplea algoritmos para conectar los filtros que fueron agregados a la gráfica.

En la aplicación se construye una gráfica de captura de video. La captura de video se refiere a que se recibe video de un dispositvo de hardware como por ejemplo una cámara web o una tarjeta de TV.

La figura 5.21 muestra la relación entre la gráfica de captura y el manejador de la gráfica.

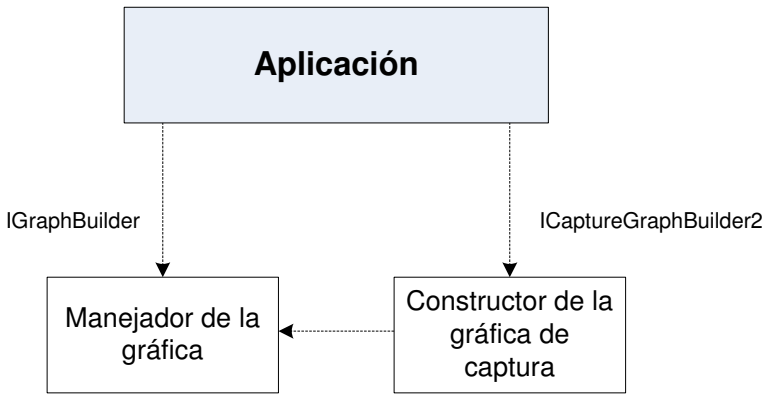


Figura 5.21: Relación entre la gráfica de captura y el manejador de la gráfica

Primero se instancia al manejador de la gráfica, luego se instancia a la gráfica de captura y por úlitmo se realiza el vínculo entre ellos mediante el método de `ICaptureGraphBuilder2`, `SetFiltergraph`.

Los filtros de captura son filtros fuente que tienen generalmente dos pines, uno de previsualización y otro de captura. Como ambos pines trabajan con el mismo tipo de medios para distinguirlos se debe usar la categoría que los define `PIN_CATEGORY_PREVIEW` para previsualización y `PIN_CATEGORY_CAPTURE` para captura. Generalmente el pin de previsualización se usa para desplegar el video mientras que el de captura se usa para escribir en disco.

Todos los filtros presentan la interfaz `IBaseFilter` que le provee al manejador de la gráfica métodos para el control de los mismos. Se puede enumerar pines y obtener información de los filtros. Para cambiar el estado de los filtros se usa la interfaz `IMediaControl`.

GraphEdit

GraphEdit es una herramienta que permite testear gráficas de DirectShow. Se pueden agregar los filtros y conectarlos para ver el resultado. Es posible construir una gráfica usando GraphEdit y luego importarla a la aplicación, sin embargo esto no es recomendable ya que GraphEdit es una herramienta básicamente para testear y depurar, no para usuarios finales.

Convertir una gráfica de GraphEdit (.grf) a código C++ es una solución muy frágil. Si la gráfica no está en perfectas condiciones puede no correr en la aplicación. Además GraphEdit no fue desarrollado con ese fin, sino con el de testear y analizar.

La principal utilidad de GraphEdit es la de analizar y verificar la gráfica construida en la aplicación. Esto se hace registrando la gráfica, corriendo la aplicación y conectándose desde GraphEdit a la gráfica remota para ver exactamente qué filtros están conectados. Como la conexión de los filtros es realizada mediante un método que usa una conexión inteligente y conecta no es completamente transparente al programador.

Para emplear esta funcionalidad de GraphEdit la instancia de la gráfica se debe registrar en una tabla llamada Running Object Table (ROT). Esta tabla es accesible globalmente y mantiene un registro de objetos que están corriendo. Los objetos se registran con un identificador llamado *moniker*.

Cuando se instancia al manejador de la gráfica en la aplicación se debe llamar a una función que la registre en la tabla antes descripta y le de un identificador. Luego de que ya no se va a emplear la gráfica se llama a otra función que borra a la gráfica de la tabla.

Para descargar una gráfica en GraphEdit se usa la función `Connect to Remote Graph` y se elige el identificador de proceso de la gráfica (pid - process ID). Luego se puede ver la gráfica de una manera muy clara, donde se encuentran todos los filtro que intervienen y cuáles pines están conectados.

5.8.3. Previsualización

Objetivo

Se busca construir una gráfica a modo de poder previsualizar lo que captura la cámara. Se desea que lo capturado se muestre en la ventana de previsualización del teléfono (ventana inferior).

Se consideró necesaria la previsualización ya que de esta forma el usuario del teléfono puede saber si su cámara está capturando bien y además qué está capturando. Así una vez establecida la comunicación con otro usuario, puede ver qué está enviando.

Para previsualizar es necesario detectar el driver de la cámara, establecer el tipo de medios que se captura y desplegarlos en la ventana deseada.

También se deben poder extraer los datos de la gráfica para posteriormente codificarlos y enviarlos por la red.

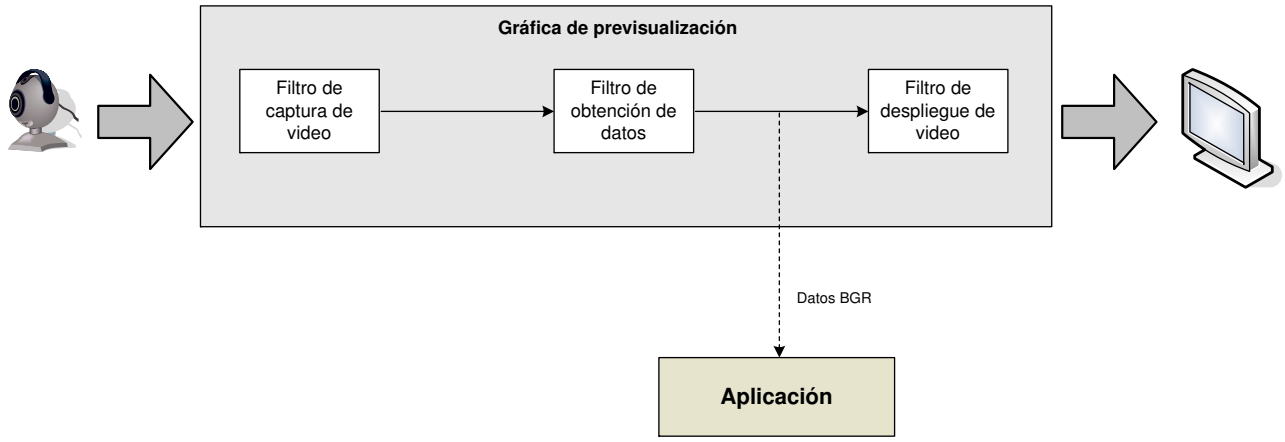


Figura 5.22: Gráfica de previsualización

Descripción de los filtros

Esta gráfica contiene los tres tipos de filtros básicos:

- Filtro fuente: es el filtro de captura de video.
- Filtro de transformación: es un filtro cuya función es permitir la obtención de datos de la gráfica para su posterior procesamiento.
- Filtro de despliegue: es el filtro que proporciona los medios a la tarjeta de video para la previsualización.

a. Filtro de captura de video

Para obtener el filtro de captura de video primero se debe encontrar el driver de la cámara. Esto se logra usando el enumerador de dispositivos, `System Device Enumerator`. Este objeto devuelve un grupo de identificadores de los dispositivos (moniker). Un *moniker* es un objeto COM que contiene información de otro objeto COM, lo que le permite a la aplicación obtener información de este último sin tener que crearlo. Después si la aplicación necesita crearlo puede usar el *moniker* para hacerlo.

Primero se encuentran todos los dispositivos de captura mediante la interfaz `ICreateDevEnum`. Si no hay dispositivo de captura conectado se da un mensaje de advertencia y la gráfica de previsuzlización no se construye ya que el filtro de captura queda en `NULL`. Si hay más de un dispositivo, se crea una lista en donde se identifica cada uno. Esta lista se puede usar por ejemplo para desplegar en un menú y darle la opción al usuario de escoger el dispositivo. En la aplicación no se consideró que fuera necesario, si hay más de un dispositivo de captura de video conectado se elige el último de la lista.

DirectShow provee métodos para controlar el stream de salida de un filtro de captura. Para esto se usa la interfaz `IAMStreamConfig` que permite indicar el tipo y formato de los medios. En este caso es video RGB de 24 bits y de tamaño QCIF (176 x 144 píxeles).

b. Filtro de obtención de datos

Este filtro le permite a la aplicación obtener los datos para su posterior procesamiento. Se sacan los datos a medida que van pasando por la gráfica para usarlos en el resto del código.

En una primera instancia se construyó únicamente la gráfica con el filtro de captura y el de previsualización. Se quiso empezar por la gráfica más sencilla posible para saber si esta solución era adecuada. Como se pudo previsualizar correctamente en la ventana del teléfono, se procedió a encontrar la forma de sacar los datos de la gráfica.

Los datos en las gráficas de DirectShow corren en hilos distintos al de la aplicación. La mayoría de los filtros fuente corren un hilo distinto para cada pin de salida. En este hilo se hace un bucle en el cual se llena el pin de salida con los datos y se los pasa al siguiente filtro. En cambio la mayoría de los filtros de transformación no crean hilos sino que trabajan sobre el hilo del filtro upstream. Los filtros de despliegue no crean threads.

No resulta trivial encontrar la forma de sacar los datos fuera de la gráfica de DirectShow. Es necesario un mayor entendimiento de cómo fluyen los datos.

Los datos se almacenan en arreglos de memoria (buffers) que están contenidos en objetos COM llamados *media samples* o muestras de medios. Estas muestras implementan la interfaz `IMediaSample`. Las muestras son creadas por un objeto llamado *allocator* que implementa la interfaz `IMemAllocator`.

Para acceder a los datos se consideraron dos alternativas: utilizar un filtro estándar de DirectShow llamado *Sample Grabber* o utilizar un filtro similar implementado en los ejemplos de DirectShow llamado *Grabber Sample* que hace algunas mejoras sobre el ya existente. Entre las mejoras se encuentra el soporte de una estructura para almacenar datos llamada `VIDEOINFOHEADER2` que permite el soporte de MPEG. El *Sample Grabber* únicamente permite trabajar con su predecesora, la estructura `VIDEOINFOHEADER` que se describe más adelante.

Debido a que las mejoras introducidas para la aplicación en cuestión no ofrecían beneficios se decidió trabajar con el *Sample Grabber*. En la gráfica de previsualización no se pasan los datos codificados en MPEG sino que se usa RGB24 sin comprimir.

El *Sample Grabber* es un filtro de transformación incluido en Microsoft SDK. Tiene un pin de entrada y uno de salida. Pasa los datos al filtro downstream sin alterarlos por lo que se puede conectar a la gráfica de previsualización sin afectar el despliegue de video logrado conectando únicamente el filtro de captura y el de despliegue. Para acceder a los datos se llaman a métodos de la interfaz `ISampleGrabber`.

Para usar este filtro se lo debe agregar a la gráfica y setearle el tipo de medios que pasa por él. Como todo filtro implementa la interfaz `IBaseFilter` y para instanciarlo se debe llamar al método de COM, `CoCreateInstance`. Luego se lo agrega a la gráfica llamando al método `AddFilter` del manejador de la gráfica.

Es importante configurarle el tipo de medios al *Sample Grabber* ya que por defecto no tiene ningún tipo de medios preferido. Si se lo dejara sin un tipo de medios en particular se podría conectar a cualquier tipo de datos sin tener control sobre éstos.

Cuando se va a conectar con otro filtro compara el tipo de medios que tiene seteado contra el del otro filtro para conectarse con los medios adecuados o no conectarse si éste último no tiene esa posibilidad.

El tipo de medios viene dado por tres atributos de la estructura `AM_MEDIA_TYPE`: el *format type*, el *major type* y el *subtype*, donde el tipo de formato especifica la estructura de los bloques donde se almacenan los datos, el tipo de datos especifica básicamente si es audio o video y el subtipo el formato en que están éstos.

En la gráfica en cuestión, el formato debe ser `VIDEOINFOHEADER` (que es el que utiliza el *Sample Grabber*), el tipo de datos es video y el subtipo es RGB de 24 bits (1 byte para cada componente de color). Se eligió este formato debido a que es ampliamente soportado por las cámaras web. Igualmente es posible setear la profundidad de bit utilizando las capacidades del dispositivo de despliegue empleado.

Luego se setean las propiedades del stream de video usando el atributo de formato. Estas propiedades son por ejemplo, las dimensiones de la ventana y la tasa de cuadros (*frame rate*).

En realidad se pueden setear todas las propiedades de la estructura `VIDEOINFOHEADER` de DirectShow. Esta estructura almacena información del tamaño, color y posición de las imágenes de video.

Entre los atributos de la estructura `VIDEOINFOHEADER` se encuentra el bit rate, el tiempo medio por cuadro (*frame rate*) y el encabezado `BITMAINFOHEADER` que contiene información de color y dimensiones del video como por ejemplo el tamaño en bytes requerido, el ancho y el alto en píxeles de la imagen y la cantidad de bits por píxel.

Previo a conectar el Sample Grabber a la gráfica, se le debe setear que copie las muestras que pasan por él en un buffer interno, usando `SetBufferSamples`.

El *Sample Grabber* puede funcionar en dos modos de operación: callback o buffering. El primero llama a una función cada vez que pasan las muestras mientras que el segundo (el que se está usando) guarda una copia antes de mandarlas al filtro downstream.

Se debe tener en cuenta que este es un filtro de transformación del tipo *trans-in-place*, es decir que usa el mismo buffer de entrada que de salida. Para el despliegue de video, el buffer de salida se encuentra generalmente en la tarjeta gráfica, por lo que el filtro lee la memoria de video que es mucho más lenta que el resto de la memoria.

c. Filtro de previsualización

Es un filtro de despliegue de DirectShow. Como todo filtro de despliegue presenta únicamente pines de entrada. Hay varios filtros de este tipo: *video renderer*, *VMR7* y *VMR9*. El primero es compatible con versiones de Windows anteriores a XP. Los dos últimos son distintas versiones del Video Mixing Renderer. Se decidió considerar únicamente a éstos por ser los más nuevos.

Algunas de las cualidades del VMR son la posibilidad de mezclar varios streams en tiempo real, trabajar en un modo sin desplegar los datos y reproducción de video en varias ventanas.

La figura 5.23 muestra los componentes del filtro de despliegue VMR.
Los componentes en gris pueden ser provistos por la aplicación para personalizarlos.

- Mixer: objeto COM descargado por el filtro cuando se deben manejar varios streams.
- Compositor de imagen: objeto COM que hace la unión de los streams para DirectDraw.
- Memoria de presentación: objeto COM que guarda los objetos de DirectDraw y maneja la comunicación con la tarjeta gráfica.
- Manejador de ventana: se usa solamente cuando el filtro trabaja en modo ventana. Maneja una ventana que crea el VMR.

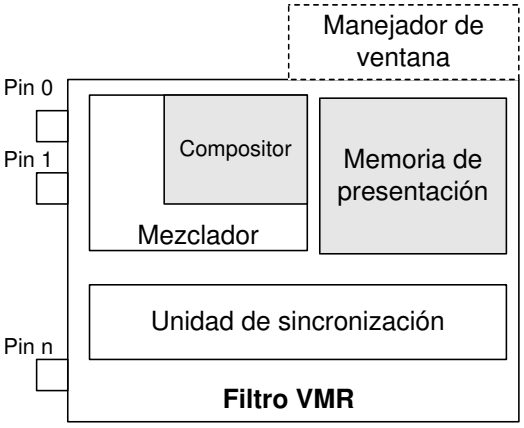


Figura 5.23: Filtro VMR

Tanto el filtro VMR7 como el VMR9 pueden trabajar en dos modos de operación: con ventana y sin ventana. Cuando se trabaja en modo ventana, el filtro crea su propia ventana para desplegar el video llamando al *Window Manager* que expone las interfaces `IBasicVideo` e `IVideoWindow`. Cuando se trabaja en modo sin ventana, el filtro no crea su propia ventana sino que se le pasa la misma desde la aplicación. Igualmente si se trabaja en modo ventana se puede desplegar el video en la ventana de la aplicación utilizando las interfaces `IBasicVideo` e `IVideoWindow`.

Si el puntero al filtro se deja en `NULL` se utiliza el filtro por defecto que es *VMR7*, en modo ventana con un solo pin de entrada.

Debido a que se tomó como ejemplo de partida AMCap, en una primera instancia se usó el modo ventana que funcionó correctamente. Sin embargo se decidió cambiar al modo sin ventana por ciertas ventajas que éste presenta. En el modo ventana está la posibilidad de deadlocks cuando se mandan mensajes entre hilos, además pueden existir problemas para recortar la imagen al colocarla en la ventana. Es necesario que la imagen tenga el estilo correcto.

Al usar el modo sin ventana la posibilidad de deadlocks por llamadas de hilos disminuye ampliamente y ya no hay problemas en cuanto al estilo de la imagen. Estos problemas se evitan al dibujar directamente sobre la ventana de la aplicación usando DirectDraw.

DirectDraw forma parte de la API de DirectX de Microsoft y se usa para desplegar imágenes en aplicaciones que requieren buena performance. Utiliza aceleración de hardware y permite acceder a la memoria de video.

Construcción de la gráfica de previsualización

Para construir la gráfica de previsualización, se agregan los filtros, se los conecta y luego se corre la gráfica. En la sección anterior se describieron los filtros que participan y cómo agregarlos.

Para agregar los filtros a la gráfica fue necesario primero instanciar al manejador de la gráfica *IGraphBuilder* y al constructor de la gráfica de captura `ICaptureGraphBuilder2` y vincularlos mediante `SetFiltergraph`.

Si bien lo que se busca es la previsualización del video y no la captura en disco, se usó el pin de captura (PIN_CATEGORY_CAPTURE) para realizar la conexión previendo que a futuro se le pueda incorporar la grabación de las video llamadas. El pin de captura además de permitir salvar en disco permite previsualizar.

La conexión se realizó mediante el método `RenderStream` de la gráfica de captura, tomando como pin el de captura, tipo de medios video, filtro fuente el de captura, filtro de transformación el *Sample Grabber* y filtro de previsualización el VMR7 en modo sin ventana.

Después de conectados los filtros se le configura el rectángulo de la ventana donde se debe previsualizar al filtro de previsualización.

El último paso es correr la gráfica usando la interfaz `IMediaControl` y liberar las interfaces una vez que se dejan de usar mediante `Release`.

Obtención de los datos en la aplicación

Para obtener los datos de video es necesario hacer una función *callback* que los devuelva a la aplicación para su procesamiento. Esto se hace mediante una función que use la interfaz `ISampleGrabber` del filtro de otención de datos a la cual se le pasa un puntero en donde copiar las muestras.

Esta función usa la interfaz `ISampleGrabber` y llama al método `GetCurrentBuffer` que devuelve las muestras que están actualmente en el buffer y las guarda en el lugar de memoria apuntado desde la aplicación. Para saber el tamaño del buffer se llama a esta misma función con el puntero al buffer en `NULL`. Se devuelve la cantidad de bytes necesarios para almacenar lo que está en el buffer.

Cada píxel de la imagen se guarda en memoria en tres bytes donde el primer byte es la componente de azul, el segundo byte es la componente de verde y el tercer byte es la componente de rojo. Los valores van de 0 a 255.

Al realizar la obtención de datos se observó que con una de las cámaras usadas no se obtenían la cantidad de imágenes por segundo deseadas. Este problema se debe a que si bien

se le puede pedir al driver que entregue una determinada cantidad de imágenes por segundo puede pasar que éste no pueda hacerlo debido a que no tenga suficiente ancho de banda. Si por ejemplo se está utilizando mucha CPU para el procesamiento de los cuadros.

Debido a que el *Sample Grabber* utiliza el mismo buffer de entrada que de salida, la lectura de los datos se hace en la memoria de video que se caracteriza por se más lenta que el resto de la memoria. Por este motivo se decidió agregar un filtro entre el *Sample Grabber* y el filtro de previsualización de video que copiara todas las muestras para que no se leyera de la tarjeta gráfica.

Se colocó entonces un conversor de espacios de colores llamado *Color-Space Converter* que hace una copia de cada muestra sin aumentar el uso de CPU. Este filtro no tiene porqué configurarse, simplemente se lo debe agregar a la gráfica y él hace la conexión de la forma más eficiente.

Este filtro es un filtro de transformación que convierte formatos de RGB.

Análisis de la gráfica con GraphEdit

Para analizar la gráfica de previsualización se utilizó GraphEdit. En el código se agregaron unas funciones para registrar la gráfica en la tabla de objetos corriendo. Luego se corrió la aplicación y se abrió GraphEdit en donde se descargó la gráfica.

La figura 5.24 muestra la gráfica obtenida al analizarla con GraphEdit.

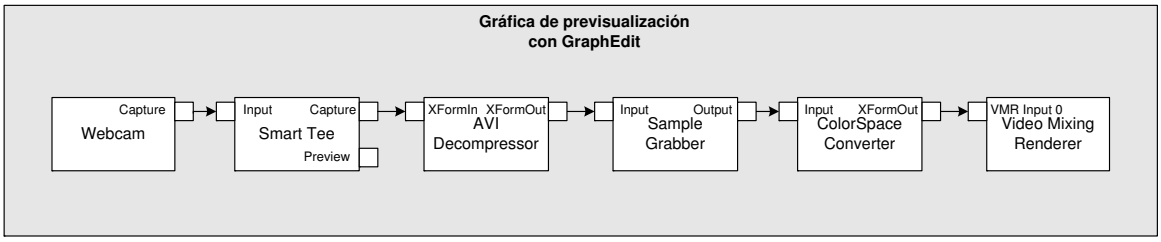


Figura 5.24: Gráfica de previsualización ensayada en GraphEdit

Como se puede apreciar, al llamar a *RenderStream* se agregaron el filtro *Smart Tee* y el filtro *AVI Decompressor*.

El filtro *Smart Tee* se usa en las gráficas de captura de video para separar los streams de previsualización y captura. Este filtro es útil cuando el filtro fuente de captura de video no posee pines distintos para la captura y previsualización. Los medios soportados son los mismos que los del filtro conectado *downstream*.

El filtro *AVI Decompressor* habilita a unirse a la gráfica al manejador de codecs de video, *Video Compression Manager*.

5.8.4. Visualización

Objetivo

Para que la video llamada sea posible se debe poder ver el video enviado por el otro usuario del teléfono. Dado que ya se tenía experiencia en el manejo de DirectShow y éste permite realizar dicho objetivo, se decidió usarlo como herramienta.

La gráfica debe realizar las siguientes funciones:

- Recibir imágenes RGB
- Regular el *frame rate*
- Desplegar en la ventana de visualización dichas imágenes (ventana superior del teléfono)

Descripción de los filtros

Luego de analizar los filtros provistos por DirectShow se llegó a la conclusión de que no existía ningún filtro fuente que cumpliera los requisitos, por lo que se tuvo que implementar uno.

Esto consistió en un nuevo desafío ya que hasta el momento se habían empleado filtros ya diseñados. Se tomaron como ejemplos los filtros incluidos en DirectShow llamados *PushSource* y *Ball*.

La figura 5.25 muestra la gráfica de previsualización construida.

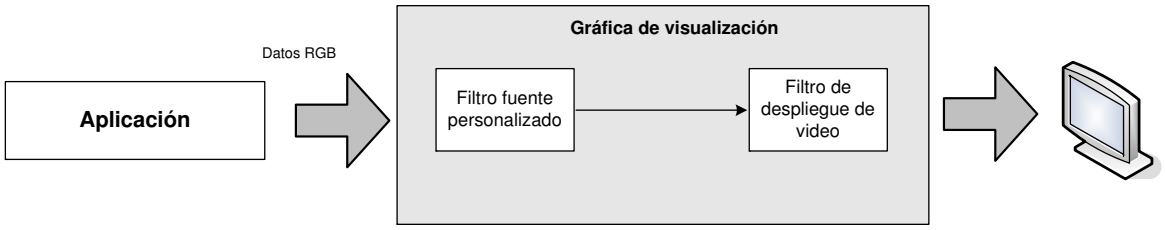


Figura 5.25: Gráfica de visualización

1. Filtro fuente

Se diseñó un filtro fuente que recibe las imágenes sin comprimir en el formato RGB de 24 bits y luego saca un stream de video que le pasa al filtro de visualización VMR7.

Para diseñar el filtro fuente se tuvieron que implementar dos clases, una para el filtro y otra para el pin de salida del filtro. Estas dos clases heredan de `CSource` y de `CSourceStream`.

`CSource` es la clase base para implementar filtros fuente y contiene uno o más pines de salida derivados de `CSourceStream`. DirectShow tiene una guía de pasos a seguir para la implementación de ambos. Además provee dos ejemplos de filtros fuente personalizados: *PushSource* y *Ball*.

Primero se debe implementar la clase base para los pines y reescribir tres métodos: `GetMediaType`, `DecideBufferSize` y `FillBuffer`. El primero devuelve el tipo de medios soportado por el pin, el segundo acuerda los requerimientos de buffer del pin y el tercero llena una muestra de medios con el video.

El método `CSourceStream::GetMediaType` devuelve el tipo de medios preferido por el filtro. Hay dos métodos `GetMediaType` uno con dos parámetros de entrada que se usa en el caso de que se tengan varios pines y otro con un solo parámetro de entrada que se usa cuando el filtro tiene un único pin como en el filtro fuente deseado.

El parámetro de entrada al método es un puntero a la estructura de medios `CMediaType`. Esta estructura está compuesta por el tipo de medios, el subtipo, el formato, el tamaño de las muestras y el tipo de compresión.

En el filtro fuente que se diseñó se fijó que el tipo de medios es video, RGB24, con encabezado `VIDEOINFOHEADER`, sin comprimir, las muestras son de tamaño 176 x 144 x 3 bytes y los datos están sin comprimir. En el método `GetMediaType` se especifican todas estas características y además se destina memoria para el encabezado `VIDEOINFOHEADER` con el método `AllocFormatBuffer`.

El segundo método que se implementó fue `CSourceStream::DecideBufferSize`. Este método configura los requerimientos del buffer de memoria del pin. Se define el *allocator* y sus propiedades. Tiene dos parámetros de entrada: el puntero a la interfaz `IMemAllocator` y el puntero a las propiedades de éste definidas en la estructura `ALLOCATOR_PROPERTIES`.

La interfaz del *allocator* ubica las muestras para moverlas entre los pines. La usan los pines que comparten *allocator* ya que no todos los filtros crean su propio *allocator*.

La estructura `ALLOCATOR_PROPERTIES` describe la cantidad de buffers, el tamaño, el alineamiento y propiedades del prefijo del *allocator*.

Por último se implementó el método `CSourceStream:FillBuffer`. Este método se llama cada vez que llega una nueva muestra del stream de video y llena `IMediaSample`. Como parámetro de entrada tiene un puntero a la muestra de medios.

La interfaz `IMediaSample` setea y devuelve propiedades de las muestras. Una muestra de medios es un objeto COM que contiene un bloque de datos. Las muestras soportan el uso de buffers compartidos entre filtros.

En este método se obtiene el puntero a las muetras y se copian los datos que le llegan al

filtro. Los datos que le llegan al filtro vienen dados por un puntero (variable global) que se setea desde la aplicación. A cada nueva muestra se la configura como punto de sincronismo mediante `SetSyncPoint`.

Luego de implementadas estas clases e instanciadas en la gráfica de visualización, no se llaman a los métodos en forma directa sino que DirectShow los llama en un hilo que tiene corriendo para el pin derivado de CSourceStream. En ese hilo el bucle `CSourceStream::DoBufferProcessingLoop` recibe las muestras y las manda downstream llamando a `CSourceStream::FillBuffer`. Este bucle está corriendo en forma continua.

2. Filtro de visualización

Para el filtro de previsualización se utilizó el VMR7 al igual que para el filtro de previsualización.

Construcción de la gráfica

Para construir esta gráfica se siguen los mismos pasos que para la de previsualización: primero se hace una inicialización de los objetos COM y se instancia al manejador de la gráfica. Segundo se agrega el filtro fuente. Tercero se conectan todos los filtros desde el pin de salida del filtro fuente usando el método *Render*.

En este caso no se verifica la gráfica en GraphEdit ya que para hacer esto se debe registrar el filtro fuente diseñado en la lista de objetos COM. Como el filtro se usa únicamente dentro de la aplicación no se consideró necesario registrarlo.

5.8.5. Integración a la aplicación

Previsualización

La integración de todo lo desarrollado acerca del manejo de hardware en la previsualización se hace en la clase `sipXmediaFactoryImpl`. Aquí inicialmente se identifica el dispositivo de captura de video y se crea la gráfica de previsualización.

Periódicamente, cada $\frac{1}{cuadrosporsegundo} = \frac{1}{25} = 40ms$ desde `sipXmediaFactoryImpl` se solicita a la gráfica de previsualización que ésta le provea la imagen que está capturando en ese momento. La imagen recibida en formato BGR24 es enviada al hilo de ejecución `CameraThreadWnt`.

Cuando la imagen capturada por la cámara es recibida por el hilo, éste la envía a la cola del primer recurso de procesamiento de video. Luego los datos siguen el camino de los recuros explicado en 5.9.

Visualización

Lo primero que se realiza en cuanto al manejo del hardware en la etapa de visualización es la creación de la gráfica de visualización, en `sipXmediaFactoryImpl`.

Una vez creada la gráfica que permite visualizar imágenes en pantalla es necesario indicar qué es lo que se desea mostrar. Cuando las imágenes de video son recibidas por la red y luego decodificadas a formato BGR24 llegan al último recurso de la gráfica que es `MprToScreen`. Como ya se explicó en 5.9 las imágenes que llegan a este recurso se guardan en una cola específica del recurso.

Desde el hilo `ScreenThreadWnt` también periódicamente (cada 40ms) se toma una imagen de la cola del recurso `MprToScreen` y se la envía a la gráfica de visualización. La gráfica toma la imagen y la despliega en pantalla.

5.9. Integración de MPEG-1 a la aplicación

Hasta el momento la aplicación sipXvideoPhone dispone, por una parte de una plataforma base que soporta transmisión de video y por otra de un codec de compresión de video que permite reducir el ancho de banda consumido al transmitir los datos. Esto aún no resulta suficiente para lograr transmitir la señal de video comprimida a través de la red, requiriéndose un procesamiento adicional que permita paquetizar correctamente el flujo de video. A su vez, en el agente de usuario remoto la información recibida debe ser reconstruida de manera adecuada para lograr una buena comunicación entre las dos puntas.

Para aplicaciones de tiempo real comunmente se utiliza el protocolo RTP (Real Time Protocol), definido en la RFC 3550, como protocolo de transporte. Este es también el protocolo utilizado en sipXvideoPhone para transportar ambos flujos de tiempo real, el de audio y el de video, y en cada uno de estos casos se debe paquetizar el tráfico de datos según la recomendación que corresponda.

Esta etapa del desarrollo, se focalizó en cómo se lleva a cabo la transmisión de un flujo de video MPEG-1 utilizando RTP como medio de transporte. Tomando como base el análisis teórico realizado para el codec MPEG-1 y su encapsulado en RTP, se procedió a integrar el codec MPEG-1 a la aplicación sipXvideoPhone. Para ello fue necesario primero realizar un estudio de la librería que nos proporcionó las herramientas de codificación y decodificación, y así poder integrarla adecuadamente a la aplicación.

En esta sección se exponen los detalles de la librería elegida (Libavcodec), y luego se explica en detalle cómo se implementó el procesamiento de los datos de video previo a ser transmitidos en el origen y al ser recibidos en destino.

5.9.1. Libavcodec

Introducción

Libavcodec es una librería que proporciona la implementación de varios codecs tanto de audio como de video. En el caso concreto de la aplicación en desarrollo, la librería fue utilizada

para codificar y decodificar solamente señales de video dado que la transmisión de audio ya se encuentra completamente desarrollada.

Libavcodec forma parte de un proyecto de código abierto denominado FFmpeg, sumamente utilizado en aplicaciones licenciadas por la GPL (General Public License). FFmpeg es un desarrollo que permite grabar archivos de voz y video, hacer streaming, y convertir audio y video a diferentes formatos. A tales efectos FFmpeg, se compone de varias librerías, cada una de las cuales provee una funcionalidad determinada.

Las principales librerías que comprenden el proyecto FFmpeg son:

- ffmpeg: es una herramienta que permite convertir un video de un formato a otro, capturar y codificar en tiempo real los datos provenientes de una tarjeta de televisión.
- ffserver: implementa un servidor de streaming multimedia.
- ffplay: implementa un reproductor multimedia.
- libavcodec: es una librería que contiene todos los codecs de audio y de video soprotados por las distintas funcionalidades del proyecto FFmpeg.
- libavformat: es una librería que contiene multiplexores y demultiplexores para manejar archivos multimedia.

Si bien el proyecto FFmpeg provee varias librerías, la única que resulta de interés en nuestra aplicación es Libavcodec dado que es quien proporciona las funcionalidades de codificador y decodificador de video necesarias para continuar con el desarrollo de sipXvideoPhone.

Libavcodec dispone de implementaciones para varios codecs de compresión de video, entre ellos los siguientes estándares:

- H.261
- H.263
- H.264
- MPEG-1
- MPEG-2
- MPEG-4

En el caso concreto de sipXvideoPhone, se utilizó la implementación para MPEG-1 dado que fue el codec elegido para comprimir la señal de video.

Estructuras y métodos proporcionados

En lo que refiere a libavcodec, esta utiliza estructuras de datos especialmente diseñadas para almacenar información característica de la aplicación como lo son por ejemplo los codecs y las imágenes. A continuación se proporciona un listado de las estructuras que fue necesario utilizar para lograr el objetivo deseado:

`AVCodec:` Libavcodec utiliza esta estructura de datos para modelar los codecs de compresión. Esta tiene campos que permiten definir las características propias de un determinado codec, como por ejemplo el nombre, tipo de codec (audio o video), formatos de píxel soportados, entre otros.

`AVCodecContext:` esta estructura es utilizada para definir el resto de los parámetros configurables para la codificación y que no son propios del codec. Con esto se hace referencia por ejemplo al formato de la imagen (CIF, QCIF, entre otros), frame rate, formato píxel (RGB24, YUV, entre otros), bit rate de salida, etc.

`AVFrame:` esta estructura de datos permite representar los frames de video. La misma presenta campos para identificar el tipo de imagen que está representando, y a su vez tiene punteros que permiten acceder a los datos que componen la imagen.

Además de estas estructuras básicas la librería presenta métodos específicos para realizar las tareas relacionadas con la compresión y descompresión de video, como lo son la codificación, decodificación, conversión entre distintos tipos de imágenes, entre otros.

A continuación se detallan los métodos que resultaron de interés y fueron utilizados en la aplicación sipXvideoPhone:

`avcodec_find_encoder:` este método genera a partir de un identificador correspondiente con el codec, una estructura de datos representativa del mismo. Esta estructura es del tipo `AVCodec`.

`avcodec_open:` una vez disponible el codec y seteada la información en el `AVCodecContext`, es necesario asociarlas de alguna manera para poder codificar acorde a la configuración especificada. Para esto, Libavcodec dispone de este método que permite vincular el codec de compresión con las características predefinidas.

`avcodec_encode_video:` este es el método que tiene implementado el procedimiento de codificación de una imagen de video. En base a la configuración predefinida en el `AVCodecContext`, el método codifica la imagen y la deja almacenada en un buffer de salida del cual entrega un puntero que lo referencia. El método también proporciona a la salida, el tamaño que utilizó del buffer para almacenar la imagen codificada.

`avcodec_decode_video:` este método, como su nombre lo indica, es quien tiene implementada la decodificación de una imagen de video. A partir de una imagen guardada en buffer y de los parámetros seteados previamente en el `AVCodecContext`, el método genera la imagen decodificada correspondiente que queda almacenada en una estructuura del tipo

`pFrame`. Cada vez que se ejecuta este método, retorna un identificador `frameFinished` que permite saber si fue posible decodificar esa imagen o no.

`img_convert:` este es otro de los métodos utilizados, el cual realiza la conversión de una imagen de un formato a otro. En el caso concreto de nuesta aplicación, este método se utilizó para convertir la imagen decodificada al formato BGR24, formato con el que se trabaja en el resto del programa para reproducir correctamente la imagen en pantalla.

`av_register_all:` con este método se inicializa la librería, y con ella todos los codecs.

Funcionamiento de la librería

Una vez elegida la librería a utilizar para codificar y decodificar video, se procedió a realizar un testeo de la misma con el objetivo de comprender en detalle su funcionamiento y de asegurarnos que realmente proporcionara las funcionalidades buscadas. Las únicas tareas que se pretendó obtener de la librería fueron la codificación y la decodificación propiamente dichas. Todo otro procesamiento previo o posterior de la información de video que fuera necesario se implementa en el resto del desarrollo de sipXvideoPhone, sin hacer uso de las herramientas proporcionadas por Libavcodec.

Para testear la librería, se trabajó con una solución de ejemplo independiente de la aplicación sipXvideoPhone. La aplicación se desarrolló de manera tal que, a partir de una secuencia de imágenes, generara un archivo codificado con el codec de video MPEG-1. Luego, a partir de este archivo se invoca al decodificador para que retorne los frames de video decodificados, también impresos en archivos separados a modo de poder visualizar si efectivamente se estaban procesando correctamente los datos.

Las pruebas realizadas con el codec de compresión MPEG-1, resultaron exitosoas pudiendo obtener videos codificados y a su vez los frames originales, resultantes de la decodificación.

Con la librería funcionando correctamente para MPEG-1, se realizaron sucesivos testeos para lograr comprender cómo se utiliza correctamente la librería y así definir la manera de integrala a la aplicación sipXvideoPhone. Para ello se hizo hincapié en los siguientes puntos:

- ¿Cómo se deben pasar los datos al codificador?
- ¿Cómo se deben pasar los datos al decodificador?
- ¿Qué atributos deben setearse necesariamente en el `AVCodecContext`? Cómo se hace?
- ¿Cómo retorna los datos el codificador?
- ¿Cómo retorna los datos el decodificador?

A continución se pasa a explicar la codificación y decodificación propiamente dichas.

a. Codificador

En lo que refiere al codificador, se encontró que la funión de Libavcodec que se encarga de codificar un frame de video necesita como parámetros de entrada el `AVCodecContext`, el tamaño de memoria que se reservó para guardar los datos codificados y un puntero a donde deben ser escritos estos datos. También es necesario pasarle como parámetro la imagen a codificar en una estructura del tipo `AVFrame`.

En el parámetro `AVCodecContext` es donde se setean las características que determinan cómo es el codec a utilizar. Para que el codificador funcione correctamente se deben setear como mínimo el ancho y alto de la imagen a codificar(`AVCodecContext->width`, `AVCodecContext->height`), la cantidad de cuadros por segundo que llegan al codificador (`AVCodecContext->frame_rate`) y el tamaño de GOP deseado (`AVCodecContext->gop_size`).

Si bien se tiene un parámetro para configurar la cantidad de cuadros por segundo a codificar, el único valor que soporta la librería para el caso de MPEG-1 es 25. En cuanto al tamaño de la imagen, tanto el ancho como el alto pueden tomar cualquier valor.

El formato de la imagen de entrada al codificador no es configurable ya que el codificador MPEG-1 que proporcionado por Libavcodec solamente acepta imágenes YC_rC_b4:2:0. Dado que los datos capturados por la cámara se pasan a través de los recursos en formato BGR24 en este punto es de utilidad hacer uso de la función `img_convert` para convertir la imagen capturada BGR24 al formato aceptado por Libavcodec.

Cada vez que una imagen YC_rC_b4:2:0 es pasada al codificador de Libavcodec el mismo codifica la imagen y la pone en el lugar reservado para ello. Si bien los datos obtenidos son del tipo MPEG-1 el codificador no está completamente especificado y el formato de los datos de Libavcodec es algo a aclarar.

Respecto a algunas de las capas de MPEG-1, la librería genera:

1. Una rebanada por cada imagen
2. Como primer imagen del GOP una imagen I y el resto P o B de acuerdo al tamaño especificado de GOP
3. Un GOP por cada seguencia de video

A modo de ejemplo, en el caso en que el tamaño del GOP sea cinco y se codifique una imagen I cada cuatro imágenes P el esquema de capas queda como el representado en la figura 5.26.

b. Decodificador

El decodificador, como se mencionó anteriormente, requiere que en sus parámetros de entrada se pase por una parte el puntero al lugar de memoria donde se encuentra la imagen de

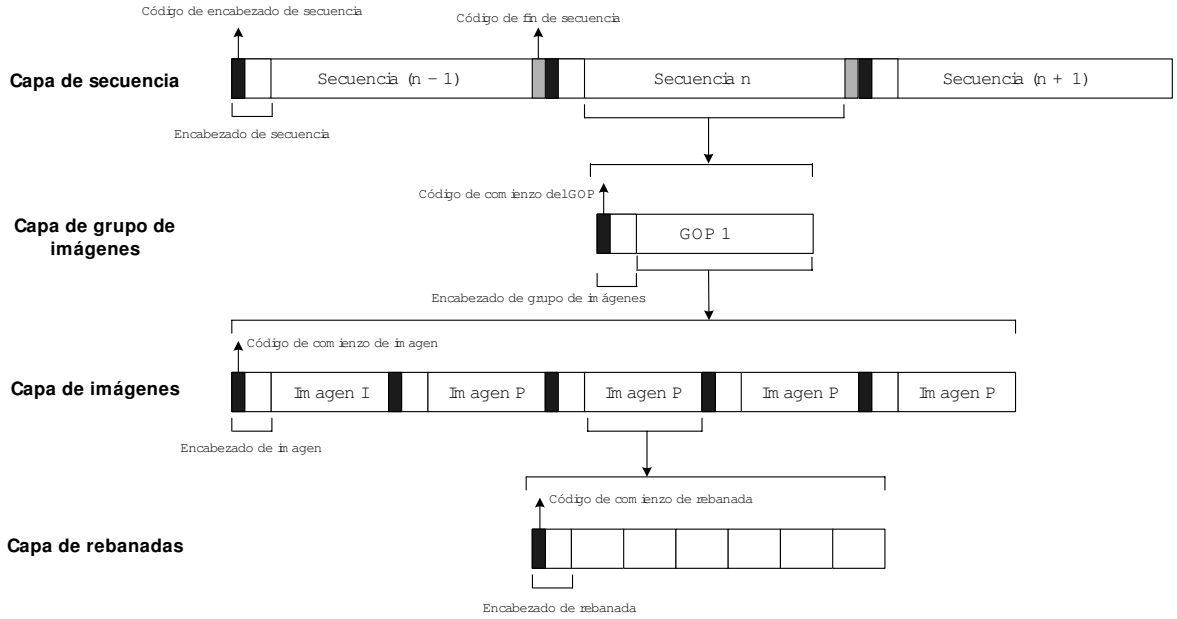


Figura 5.26: Capas MPEG-1 según Libavcodec

video a decodificar, y por otra parte el `AVCodecContext` con los parámetros que determinan las características de la decodificación. Además de indicarle al decodificador dónde se encuentra la imagen de video, es necesario indicarle el tamaño de la misma. Este parámetro se mide en bytes y permite que el decodificador procese los datos guardados en todo ese bloque de memoria.

Para que el decodificador logre procesar una determinada imagen codificada, es necesario que el mismo disponga de la siguiente imagen que forma parte del stream de video. Esto implica que, el decodificador recién es capaz de propocionar una imagen de video decodificada la siguiente vez que este es invocado. Este mecanismo de funcionamiento se explica con la existencia de un buffer interno al propio decodificador, donde se almacena cada imagen codificada que recibe cada vez que es invocado. Cada vez que se ejecuta el método, el decodificador retorna el frame decodificado correspondiente a la imagen anterior, que tiene disponible en su buffer y procede a almacenar la nueva imagen que le ingresó.

En cuanto a la información que se recibe a través del `AVCodecContext`, el decodificador solamente requiere que este parámetro tenga seteados los valores que indican las dimensiones de la imagen. Estos se especifican en los campos ancho y altura de la estructura (`AVCodecContext->width, AVCodecContext->height`).

A la salida del decodificador, se obtienen los frames de video correspondientes a las imágenes originales que envió el agente de usuario remoto. Estos datos son entregados por el decodificador al resto de la aplicación en estructuras del tipo `AVFrame`, a partir de las cuales nuestra aplicación deberá ser capaz de obtener y procesar los datos que conforman la imagen. Luego de

estudiar esta estructura de datos, se encontró cómo acceder al puntero que referencia los datos buscados. Con este puntero ya sería suficiente para continuar el procesamiento de la imagen recibida en el resto del programa.

Sin embargo se consideró de gran utilidad el método `img_convert` disponible en Libavcodec para convertir la imagen ya decodificada al formato BGR24. Nuestra aplicación, fue preparada para trabajar con imágenes en este formato y es por tanto necesario que la imagen decodificada se convierta a BGR24 en alguna etapa del procesamiento. Dado que Libavcodec provee esta funcionalidad, se consideró conveniente hacerlo inmediatamente después de obtener el frame decodificado. De esta manera, para el procesamiento restante en él programa se trabaja con la imagen en su formato final (BGR24).

En base a todo lo explicado sobre el funcionamiento de la librería, fue necesario implementar métodos extras en el código de sipXvideoPhone que sirvieron de interfaz para poder reunir la información que precisa el tanto el codificador como el decodificador de forma correcta. Más adelante se explica cómo se implementó esta interfaz que permitió integrar Libavcodec al resto de la aplicación.

5.9.2. Transmisión de imágenes codificadas

Para implementar la transmisión de imágenes codificadas a través de la red, fue necesario desarrollar clases accesorias que actuaran de interfaz entre nuestra aplicación y la librería Libavcodec. Para implementar esta interfaz, se creyó conveniente crear un objeto *codificador* que modelara todo lo relativo al codec MPEG-1 en lo que a la transmisión respecta. A tales efectos se desarrolló la clase `MpeVideo` que contiene todos los métodos y atributos necesarios para que el codificador MPEG-1 se integre de manera adecuada.

A continuación se explican los detalles de la clase `MpeVideo` y luego se realiza un análisis paso a paso del procesamiento que reciben los datos previo a ser enviados por la red.

MpeVideo

Visto el funcionamiento de las herramientas para codificación que provee la librería Libavcodec se está en condiciones de crear una clase, `MpeVideo`, que modele al codificador MPEG-1. La interfaz entre esta clase y las otras clases de la aplicación es genérica. La misma hereda de una clase que represente a un codificador de video genérico de la cual puedan heredar otros codecs de video, siempre pensando en hacer un programa que se preste para futuras mejoras como ser la inclusión de otro codec. La misma idea fue aplicada para el decodificador.

`MpeVideo` se encarga de transformar la imagen recibida en formato BGR24 a una imagen en formato MPEG-1. De acuerdo con el resto del desarrollo, previo a codificar es necesario transformar el formato de la imagen, de BGR24 a YC_rC_b4:2:0 para luego utilizar las funciones de la librería Libavcodec y obtener el resultado deseado.

Además de codificar la imagen de video en MPEG-1, la clase realiza el paquetizado en RTP según se describe en la RFC 2250. Para ello se debe formar el encabezado específico de video con información acerca del fragmento de video que se está encapsulando.

La clase `MpeVideo` mantiene información característica del flujo de video que se debe procesar, como por ejemplo el tamaño de la imagen, tamaño del GOP, y *frame rate*. Estos parámetros se encuentran seteados de acuerdo con la elección de diseño previa y sus valores son:

- Ancho de la imagen: 178 píxeles
- Alto de la imagen: 144 píxeles
- *Frame rate*: 25 cuadros por segundo
- Tamaño del GOP: 3 (una imgen I cada dos imágenes P)

El tamaño del GOP es arbitrario y puede cambiarse a cualquier valor. Aumentar el tamaño del GOP tiene un compromiso entre el tiempo de procesamiento, el tamaño de las imágenes codificadas y la calidad de las mismas. Cuanto más grande sea el valor de GOP se codifica una mayor cantidad de imágenes P que de imágenes I trayendo como consecuencia un mayor tiempo de procesamiento. La situación inversa ocurre con el tamaño de las imágenes y el ancho de banda requerido. A mayor valor de GOP menor ancho de banda es utilizado dado que las imágnes P tienen un tamaño muy inferior al de las I.

Interacción del codificador con los recursos

Habiendo introducido las tareas más importantes que se realizan en la codificación en lo que sigue se pasa a explicar el procesamiento que reciben los datos previo a ser enviados hacia el agente de usuario remoto. El camino de procesamiento seguido por los datos se ilustra en la figura 5.27.

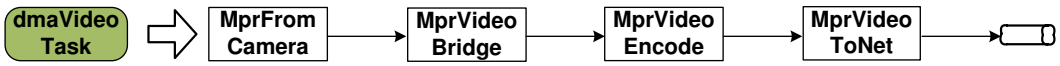


Figura 5.27: Camino de la transmisión de video

Las imágenes recorren un camino similar al explicado en la sección 5.7.5 al mencionar la arquitectura básica de video. La incorporación de MPEG-1 se ve reflejada básicamente en los recursos `MprVideoEncode` y `MprVideoToNet`. En lo que sigue se hace una descripción del camino que toman los datos detallando la parte que involucra el procesamiento de video MPEG-1.

El recurso `MprVideoEncode` recibe la imagen en formato BGR24 transformandola al formato del codec de video elegido, MPEG-1. Aquí se fragmenta cada imagen en la cantidad de tramos necesarios para que cada uno pueda ser enviado por la red. Quien limita este tamaño es la MTU que en nuestro caso toma el valor de 1500 bytes. Luego cada fragmento es paquetizado en la carga útil de RTP adecuadamente y enviado al siguiente recurso.

A continuación a quien llegan los datos es `MprVideoToNet`. Éste recibe la carga útil de un paquete RTP y se encarga de armar el encabezado RTP para armar el paquete completo y enviarlo por la red con destino al agente de usuario remoto con la cual se estableció la comunicación.

1. Se envía a la instancia del codificador MPEG-1, una imagen sin codificar en formato BGR24. Previo a codificar es necesario transformar el formato de la imagen recibida a YC_rC_b 4:2:0 dados los requerimientos de las herramientas de Libavcodec. Una vez obtenido esto se pasa a codificar la imagen mediante `avcodec_encode_video` obteniendo la imagen MPEG-1 así como el tamaño de la misma. Esto se devuelve a quien solicitó la tarea para que continúe con el curso de los datos.

   Mediente variables miembro se guardan valores de utilidad para otras funciones de la clase, como ser el tipo de imagen que se codifico y la cantidad de imágenes codificadas hasta el momento.

2. Previo a paquetizar la información MPEG-1, ess necesario fragmentar el paquete de video codificado en sub-paquetes de tamaño menor tal que quepan en un paquete RTP. Mediante una función bucle para cada sub-paquete se van a invocar dos funciones, una que escriba el encabezado específico de video y arme la carga útil del paquete RTP y otra que escriba el encabezado RTP y arme éste paquete para enviarlo por la red. Los dos puntos que siguen corresponden a las dos funciones mencionadas.

3. Para cada fragmento de imagen MPEG-1 se arma el encabezado específico de video según lo indica la RFC 2250. Para realizar esta tarea correctamente se debe identificar lo siguiente:

   a) cantidad de imágenes codificadas hasta el momento
   b) si existe algún encabezado MPEG-1 y cuál
   c) tipo de imagen a la que pertenece el fragmento
   d) si el fragmento es el último de la imagen

   Con esta información es posible formar el encabezado específico de video para imégenes del tipo I y para imágenes de otro tipo también se debe obtener información del encabezado de rebanada.

   Juntando el encabezado especifico de video con el fragmento de imagen MPEG-1 se obtiene la carga útil del paquete RTP.

4. A partir de la carga útil obtenida en el paso anterior, se construye el paquete RTP según lo establecido en la RFC 3550. Para esto es necesario escribir el encabezado del mismo. Algunos de los campos se relacionan con el tipo de carga útil que se está transportando, con un número de secuencia que identifica al paquete, con una referencia temporal que identifica la imagen que se está transportando y con un número que identifica una fuente de sincronización.

   Una vez obtenido el paquete RTP el mismo se escribe en el *socket* que corresponde a la sesión de video establecida para que el agente de usuario remoto lo reciba.

5.9.3. Recepción de imágenes codificadas

En la etapa de recepción de las imágenes codificadas, se procedió de manera análoga. Se implementó un objeto *decodificador* que actúa de interfaz entre la aplicación sipXvideoPhone y la librería Libavcodec. Este objeto es la clase `MpdVideo`, y modela la decodificación de imágenes codificadas en MPEG-1 cuando estas son recibidas en paquetes RTP. Aquí no solamente se decodifican los datos de video, sino también se reconstruye la información completa de cada imagen que fue recibida de manera distribuida en varios paquetes RTP.

A continuación se explican los detalles de la clase `MpdVideo` y luego se realiza un análisis paso a paso del procesamiento que reciben los datos MPEG-1 recibidos.

MpdVideo

Para implementar la etapa de decodificación fue necesario definir una clase que modelara el decodificador MPEG-1 de acuerdo con los estándares que lo regulan y el análisis realizado previamente. Se procedió entonces a implementar la clase `MpdVideo` en la cual se desarrollan los métodos necesarios para realizar una correcta reconstrucción de los datos de video enviados por el agente de usuario remoto.

En `MpdVideo` se hace uso de las herramientas proporcionadas por Libavcodec, para lo cual fue necesario definir métodos y atributos específicos que permitieran utilizarlas de manera adecuada. Además de integrar la librería, se procedió a implementar otros métodos que fueran de utilidad al momento de reconstruir la información de video recibida.

La clase se encarga de reunir el contenido completo de cada imagen de video que fue enviada desde la punta remota y luego realiza la etapa de decodificación.

Dado que las imágenes de video transmitidas se encuentran fragmentadas en varios paquetes RTP, es necesario reunir el contenido completo correspondiente a cada imagen para luego pasar a la etapa de decodificación. Esta tarea es realizada por `MpdVideo`, quien analiza los campos del encabezado de video (RFC 2250) para obtener las características del flujo recibido. Se detecta si los datos de video recibidos corresponden a la misma imagen que se recibió en la pasada anterior, o si es el comienzo de una nueva. También es posible obtener el tipo de imagen del que se trata, ya que la misma puede ser del tipo I, P, ó B.

Una vez reunido el contenido completo de una imagen de video MPEG-1, `MpdVideo` realiza la decodificación. A partir de un flujo MPEG-1 se obtiene una imagen en formato BGR24 para luego desplegar en pantalla. Aquí se utilizan las herramientas proporcionados por Libavcodec que permiten llevar a cabo estas tareas.

Interacción del decodificador con los recursos

Como se explicó anteriormente cuando los paquetes RTP de video son recibidos por la red, estos son procesados por los recursos que conforman el `MpVideoFlowGraph` y corresponden al

camino de recepción. Este camino se muestra en la figura 5.28.

Figura 5.28: Camino de la recepción de video

El procesamiento de los datos en el camino de recepción es similar al explicado con anterioridad en la sección 5.7.5. La incorporación de MPEG-1 a la aplicación resulta en nuevas tareas de procesamiento que deben realizarse según las especificaciones de este codec de compresión.

En lo que refiere la reconstrucción del flujo de video MPEG-1, se lleva a cabo en el recurso `MprVideoDecode`. En el turno de procesamiento de este recurso se ejecuta el método `doProcessFrame` donde son invocados diferentes métodos que permiten, a partir de los paquetes RTP recibdos, obtener las imágenes de video reconstruidas y prontas para finalmente mostrar en pantalla. A tales efectos, el recurso interactúa con otros dos objetos que forman parte del desarrollo de la aplicación: el `MpVideoJitterBuffer` y el `MpdVideo`.

A continuación se describe paso a paso cómo se realiza el procesamiento de la información de video en el método `MprVideoDecode::doProcessFrame` y su interacción con los dos objetos mencionados:

1. En primer lugar, el `MprVideoDecode` obtiene el puntero al próximo paquete RTP que se debe procesar, tomando como referencia el número de secuencia.

   Una vez disponible el paquete RTP, se obtiene de su encabezado el tipo de carga útil que el mismo trae. Si el paquete contiene datos de video codificado en MPEG-1 entonces el programa pasa a ejecutar una serie de pasos implementados específicamente para este codec de compresión de video. En caso contrario, se ejecuta la aplicación para el caso por defecto. El procesamiento de los datos descrito de aquí en adelante es específicamente para video codificado en MPEG-1.

2. Se envía el contenido de carga útil del paquete RTP al decodificador para que reconstruya la información de video que se originó en el agente de usuario remoto. De acuerdo al funcionamiento del decodificador proporcionado por Libavcodec, es necesario que previamente se reúna la información correspondiente con una imagen completa de video.

   Se almacena el contenido de sucesivos paquetes RTP de video MPEG-1 hasta completar toda la información correspondiente a una misma imagen. Por otra parte, se toman acciones correctivas para contemplar los casos en que haya pérdidas de paquetes RTP. Al perderse un paquete RTP con contenido de video MPEG-1, es posible encontrarse con la ausencia de información correspondiente al encabezado de video (video header) o información propia de las imágenes que se enviaron. En ambas situaciones se consideró que la mejor opción era descartar toda la información ya recibida y próxima a recibir que pertenezca a una misma imagen.

Para cumplir con estos objetivos, se analiza el contenido de la carga útil recibida en cada paquete RTP, la cual puede ser información de encabezado (*video header* definido en la RFC 2250) y/o datos propios de las imágenes codificadas.

En el caso de recibir información de encabezado, esta es analizada para poder procesar correctamente el flujo de video que se recibirá luego. A tales efectos, se obtiene el contenido de los campos que resultan de utilidad:

- El comienzo de una nueva imagen de video codificada, que se detecta con la presencia del *Begin of slice*.
- El fin de cada imagen codificada, que se detecta con la presencia del *End of slice*.
- Tipo de imagen que se está procesando (I, P, ó B).
- Marca de tiempo del paquete de video, identificada con el parámetro *Time Reference.*

Una vez analizado este encabezado se procede a copiar la carga útil de video en un arreglo del objeto decodificador `MpdVideo`, que fue definido con este propósito. El tamaño de este arreglo es fijo y se eligió de manera de poder almacenar cualquier imagen de video codificada, independientemente de su tamaño.

Se retorna un parámetro indicador de si ya se recibió la imagen completa o no. Mientras no haya sido completada, el `MprVideoDecode` sigue retirando paquetes RTP de memoria y el procedimiento se repite. Cuando en el arreglo se dispone de los datos correspondientes a una imagen completa, entonces ya es posible pasar a la etapa de decodificación.

3. Para poder pasar al decodificador la información correspondiente a una imagen de video, es necesario primero obtenerla del arreglo donde fue almacenada. Esto no resulta trivial dado que el arreglo de almacenamiento es propio de la calse `MpdVideo` y por tanto no accesible desde afuera, como se requiere en este caso. Más aún, al retirar una imagen de este vector se deben actualizar correctamente los punteros del mismo de modo de dejar registrado el movimiento de los datos.

   En esta etapa de procesamiento el objetivo es dejar el contenido de la imagen disponible en un bloque de memoria, al igual que se hizo en su momento con los paquetes RTP. En este caso se obtendrá del *pool* de *frames* de video (reservado a en la clase `MpMisc` durante la inicialización de la aplicación) un buffer del tipo `MpArrayBuf` que es rellenado con la imagen de video codificda que se juntó previamente en `MpdVideo`.

   En consecuencia, los datos correspondientes con la imagen quedan referenciados en memoria por un puntero del tipo `MpArrayBufPtr`. Con el, el recurso `MprVideoDecode` puede tener acceso al *frame* de video recibido cuando sea necesario.

4. En este momento, el recurso `MprVideoDecode` ya dispone del puntero al bloque de memoria que contiene una imagen completa de video codificado y está pronto para pasarle esta información al decodificador MPEG-1. Sin embargo, la invocación al decodificador no se realiza directamente en el recurso `MprVideoDecode`, sino en el objeto `MpVideoJitterBuffer` donde luego quedará gurdada la imagen decodificada.

Para decodificar las imágenes de video nuevamente se hace uso de las implementaciones en la clase `MpdVideo`. Luego de obtener la imagen decodificada y en el formato correcto, esta se almacena en un arreglo específico del objeto `MpVideoJitterBuffer`. Finalmente, se actualizan los punteros que manejan este arreglo de modo de dejar pronta la referencia al próximo lugar disponible para colocar una nueva imagen decodificada.

5. Realizadas estas tareas, el `MprVideoDecode` pasa a retirar una imagen de las disponibles en el `MpVideoJitterBuffer` para entregarla al siguiente recurso de la gráfica.

   Nuevamente se utiliza uno de los bloques de memoria del *pool* de video reservado durante la inicialización del teléfono. La idea es obtener un buffer del tipo `MpVideoBuf` para dejar almacenado en él la imagen decodificada que se desea entregar al siguiente recurso de la gráfica. De esta manera, a través del puntero `MpVideoBuf` correspondiente es posible indicarle al próxio recurso el lugar de memoria donde se encuentran los datos.

   Recordemos que el pasaje de información entre algunos recursos sucesivos se realiza a través de los parámetros definidos en el método doProcessFrrame, en los cuales es posible especificar el puntero a la dirección de memoria donde se encuentran los datos. Este es el caso de la interconexión entre el `MprVideoDecode` y el `MprVideoBridge`.

5.9.4. Problemas encontrados

Al momento de incorporar el codec de compresión a la aplicación, nos enfrentamos con dos problemas que se describirán a continuación.

Integración de Libavcodec

En primer lugar, resultó complicado hacer funcionar de manera correcta la librería en conjunto con el resto del programa.

Cuando en etapas previas se testeó el comportamiento de Libavcodec nunca se tuvo en cuenta el tiempo de procesamiento adicional que se suma en sipXvideoPhone, ni tampoco la infinidad de veces que los métodos de la librería son invocados. Estos dos factores resultaron un problema a la hora de incorporar el codec de compresión a la solución.

Nos encontramos que el método codificador (`avcodec_encode_video`) era invocado una infinidad de veces, generando que en un determinado punto el programa se colgara. Todo nos condujo a pensar que la librería tiene un tope de veces en que puede ser ejecutado, que los tiempos en que cada vez era invocado el método eran demasiado distantes debido al procesamiento extra de toda la aplicación, o que se estaba olvidando liberar alguna memoria luego de la ejecución del método.

Dado que la librería que estábamos utilizando ya se encontraba pre-compilada, no era posible depurar su implementación y así detectar con facilidad dónde se encontraba el error. En consecuencia se debió seguir otro tipo de camino que nos permitiera detectar cuál era el

problema y cómo solucionarlo.

Se retornó a una solución de ejemplo en la que fuera posible simular exactamente la manera en que el método es invocado en sipXvideoPhone y de esta manera testear su comportamiento de manera aislada. Se logró reproducir el problema observando que la ejecución del método codificador efectivamente caía llegadas aproximadamente las 5000 ejecuciones. Con este resultado se procedió a hacer lo siguiente:

1. Verificar que todas las memorias que se estuvieran reservando para almacenar imágenes o algún otro tipo de dato fueran correctamente liberadas. Esto efectivamente se estaba realizando bien por tanto aquí no se encontraba el problema.

2. Testear la ejecución del método una gran cantidad de veces seguidas veces eliminando ahora los tiempos adicionales que se estaban simulando entre una ejecución y la siguiente. Con esto se observó que la librería seguía generando error, aproximadamente en el mismo número de invocación.

3. Investigar si era un problema conocido a los usuarios de Libavcodec. Con esto se encontraron varios casos en los que la librería ya había presentado un comportamiento similar, y se debía básicamente a la manera en que la librería había sido compilada.

Luego de este análisis, se nos presentaron dos posibles caminos a seguir:

- Conseguir la solución completa e intentar compilarla.
- Buscar otra solución Libavcodec pre-compilada pero que fuera compatible con nuestro entorno de trabajo y nos permitiera trabajar sin problemas.

La primera opción implicaría ponerse a trabajar nuevamente con una solución agena lo cual probablemente resultara una tarea engorrosa. Sumado a esto, sería necesario aprender a manejar otro entorno de trabajo dado que Libavcodec no presenta posibilidades de ser compilada en Visual Studio.

Se consideró que la primera opción podría complicarnos aún más el trabajo, motivo por el cual decidimos proceder a buscar una nueva solución pre-compilada de Libavcodec.

Luego de una exhaustiva búsqueda se encontró una versión diferente para el paquete compilado de Libavcodec. Este nuevo compilado fue testeado del mismo modo observando que el problema seguía ocurriendo pero era posible alcanzar una mayor cantidad de ejecuciones del método. Realizando una codificación a 25 frames por segundo, se alcanzaba a codificar 30 minutos de video sin que la aplicación fallara. Se consideró que estos resultados podían darse por válidos y por lo tanto se procedió a utilizar esta versión de Libavcodec.

Pérdida de paquetes en la aplicación

Otro de los problemas a los que nos enfrentamos fue a la pérdida de paquetes RTP una vez que estos ya habían arribado a la aplicación.

Luego de procesar los datos y en destino, se observó que las imágenes obtenidas presentaban falta de información. Todo hacía pensar que se estuvieran perdiendo paquetes RTP en alguna etapa del camino. Procedimos a capturar el flujo de paquetes a través de la red, observando que todos los enviados eran recibidos en la otra punta. De esta manera se concluía que el problema se encontraba en alguna etapa del procesamiento de los datos en nuestra aplicación.

Analizando el comportamiento del programa se encontró que la información faltante se estaba perdiendo ni bien entraba a la aplicación. Como se explicó anteriormente, los datos recibidos por la red se procesan por la red se procesan por primera vez en el task `NetInVideoTask` y en el recurso `MprVideoFromNet`. El task se encarga de leer la información del socket y almacenarla en una estructura correspondiente con el formato de paquetes UDP. Luego en el primer recurso de la gráfica se obtiene a partir del paquete UDP, el RTP correspondiente. Se detectó que cuando en `NetInVideoTask` se trabaja con los paquetes UDP, algunos no se encontraban disponibles.

Este problema comenzó a ocurrir a partir del momento que se integró el codec de compresión y con el la transmisión de imágenes de video completas. Hasta entonces se enviaban solamente trozos de imágenes para poder encapsularlas en un único paquete RTP. Todo conducía a pensar que debido al incremento en la cantidad de paquetes transmitidos, el socket no estaba pudiendo recibirlos todos debido al buffer de recepción que no tenía el tamaño adecuado.

Se analizó en profundidad el manejo de sockets en aplicaciones de *Windows*, encontrando que posiblemente el problema se debiera a que:

- el tamaño de buffer de recepción del socket de video fuera demasiado chico y por tanto no apropiado para almacenar la gran cantidad de infomración que se estaba transmitiendo.
- el tiempo entre veces consecutivas que se leen los sockets era demasiado grande como para poder procesar a tiempo toda la informacíón recibida en el puerto.

En base a estos resultados, se intentó localizar el lugar del programa donde fuera posible setear estos parámetros para que el socket de video se comportara de la manera adecuada. Luego de un estudio profundo sobre el desarrollo de sipXvideoPhone, se encontró que el buffer de recepción de los datos de video efectivamente era muy pequeño. La implementación de los sockets se había implementado de manera análoga a lo existente para el caso de audio en sipXezPhone y por tanto se encontraba todo seteado para el caso en que los datos transmitidos sean muestras de audio, las cuales se envían en paquetes RTP bastante más pequeños.

Aumentando considerablemente este buffer se logró eliminar por completo la pérdida de informcación en la recepción.

5.10. Testeo

Fue fundamental para la integración de la arquitectura de video al código realizar varias etapas de testeo a medida que se fueron efectuando modificaciones y agregados en la aplicación.

Durante la etapa de integración de la arquitectura básica se testeó paso a paso el recorrido que seguían los datos por todos los recursos de procesamiento. Aquí se logro desarrollar un módulo que permitió establecer una sesión de medios entre dos agentes de usuario SIP. En principio por esta sesión solamente fueron enviados datos de prueba y se detectaron posibles errores de implementación más fácilmente. La transmisión de datos de prueba también permitió determinar cuándo la arquitectura base daba resultados satisfactorios para comenzar a agregar más complejidad al desarrollo.

Concluida esta etapa se pasó a integrar la codificación y decodificación de video según el codec MPEG-1. La transmisión de este tipo de datos implicó realizar tareas extra a la codificación y decodificación, como ser el paquetizado en paquetes RTP, la recepción de paquetes RTP y la correcta fusión de los mismos para obtener una imagen MPEG-1, entre otras.

Durante la implementación de las funciones específicas que a cada recurso atañe, las mismas fueron testeadas independientemente. Aquí fue imprescindible afinar las tareas de cada recurso, ya que en la arquitectura base no se tomaron en cuenta cuestiones de performance, tiempos de procesamiento, capacidad de adquisición de datos en la recepción, y otros.

El siguiente nivel de testeo involucró la verificación de que cada recurso estuviese interactuando correctemente con el recurso previo y el siguiente.

Seguido se procedió a comprobar el correcto funcionamiento extremo a extremo de la transmisión de imágenes y luego de la recepción de las mismas.

Al establecer una llamada entre dos agentes de usuario SIP incluyendo la funcionalidad de videollamada fue donde se encontraron problemas relativos a la transmisión en tiempo real y problemas propios de la aplición. En la solución de estos problemas aún nos encontramos trabajando.

5.11. Resultados y conclusiones

Como resultado de esta etapa se logró establecer una llamada VoIP entre dos agentes de usuario que se encuentran conectados a través de una LAN, y establecen dos sesiones de medios. La primera, una sesión de audio, ya implementada previamente permite elegir entre codificar el audio con el codec G.711 o el G.729. La segunda sesión, que corresponde a un flujo de video tiene como opción de codecs únicamente MPEG-1.

SipXvideoPhone contiene dos pantallas que despliegan video; una en que se muestra la captura del video local, y la otra donde se observa lo capturado en la aplicación remota y transmitido a través de la red.

Respecto a la previsualización de la imagen capturada localmente no hubo mayor problema y la aplicación permite visualizarla correctamente.

Es en la visualización del video perteneciente a la parte remota es donde se encontraron la mayoría de las dificultades. Aún esta facilidad no funciona adecuadamente debido a problemas propios de la aplicación. El hecho de que la aplicación sea para utilizar en tiempo real implica tener un cuidado muy importante en los tiempos de procesamiento. Si la velocidad con que los datos son tratados y procesados no es exactamente la requerida se produce una pérdida muy importante en la performance que se refleja en el usuario final al querer visualizar el video recibido.

Mejorar la velocidad con que los datos son tratados en todo momento de la aplicación así como en la velocidad con que los datos son tomados y entregados a la red no es nada trivial.

Además de estos problemas existen problemas referentes a la utilización del lenguaje de programación, C/C++ donde debe haber especial cuidado en el manejo de las memorias que la aplicación utiliza.

SipXvideoPhone sigue en pleno desarrollo de manera de intentar solucionar los problemas antes descritos y así generar un prototipo de aplicación que funcione adecuadamente.

Capítulo 6

Sincronización de audio y video

6.1. Introducción

En esta sección se plasma el estudio sobre cómo sincronizar dos sesiones de medios, una de audio y otra de video. Dado que nuestra aplicación espera establecer una llamada entre dos agentes de usuario involucrando la transmisión de audio y video, es necesario relacionar las dos sesiones para sincronizarlas. La sincronización se refiere a poder hacer que el video y el audio no se encuentren desfazados. Típicamente en aplicaciones de tiempo real sobre Internet, se observa que un video está desincronizado cuando el movimiento de los labios de una persona filmada no concuerda con las palabras que son escuchadas.

En este capítulo se explica cuál es la mejor alternativa aplicada al desarrollo de sipXvideoPhone y cómo la misma puede ser aplicada. Dado que la aplicación desarrollada presenta una sincronización aceptable entre el flujo de audio y video debido a que se trabaja en una LAN, solamente se dejó el camino avanzado para posteriores desarrollos fuera de una LAN. Debido a que en un principio no se sabía qué tan buena iba a ser la sincronización ya hay una gran parte implementada pronta para ser incorporada al código.

Al momento esta funcionalidad no está completamente testeada debido a que la sesión de video no presenta retardos considerables. Esto lleva a que no se pudo observar si los resultados son los deseados o aún es necesario afinar alguno de los cálculos efectuados.

6.2. Aspectos generales de la sincronización

En aplicaciones que combinan audio y video es necesario que haya una sincronización entre los mismos. Dado que la aplicación que se desarrolló transmite audio y video en tiempo real surgen dos complicaciones: primero que las sesiones RTP y RTCP para audio y video son distintas y segundo que los medios son transmitidos en forma separada por la red. Sumado a esto, la utilización de distintos codificadores provoca una variación en el desfasaje en la reproducción de los medios en el receptor.

El término que se emplea para este tipo de sincronización es sincronización de labios o en inglés *Lip Syncrhonization* [50]. Esto se debe a que el movimiento de los labios debe concordar con el habla.

Generalmente lo que ocurre es que el audio precede al video debido a que el procesamiento necesario para la codificación y envío del video es mayor. Por este motivo muchas técnicas, en particular la que se implementó, se basan en este hecho y simplemente calculan el tiempo en que el audio debe retrasarse.

El motivo por el cual se sincroniza es la sensibilidad del ser humano ante diferencias entre el tiempo de reproducción del audio y el video. A continuación se presenta un panorama general de las características que se deben tener en cuenta a la hora de sincronizar.

6.3. Percepción humana

Para la sincronización de audio y video se tiene en cuenta el umbral de percepción humana. Un desfasaje menor a 20 ms entre el audio y el video se considera que es imperceptible. Al acercarse a los 50 ms se puede percibir la falta de sincronización pero no es posible determinar si el audio precede al video o viceversa. A medida que la diferencia entre el video y el audio aumenta se hace posible determinar su relación de precedencia.

Una vez que se superan los 50 ms de diferencia, el video comienza a distraer al participante de la conferencia y por lo tanto al llegar a un desfasaje de aproximadamente 1 segundo, termina siendo ignorado. Esto ocasiona que no tenga ninguna ventaja la transmisión del mismo.

El Advanced Television System Committee (ATSC) provee lineamientos para que sistemas de transmisión *broadcast* tengan un cierto nivel de calidad. Se establece que para una calidad aceptable en la recepción el audio y el video no deben desfasarse por más de +/- 15 ms.

El mecanismo para la sincronización de audio y video busca entonces que la diferencia entre el tiempo de presentación de cada uno de estos medios sea lo más cercano a cero posible.

6.4. Retardos

Durante una sesión en la cual se transmite audio y video los retardos desde el momento en que se envían los medios hasta su recepción se van acumulando. Se analizaron las posibles causas de retardos:

- Retardos en el transmisor: ocasionado por la captura, procesamiento y codificación de los datos.

- Retardos en la red.
- Retardos en el receptor: debidos a la utilización de buffers, decodificación y presentación de los datos en los dispositivos.

Dado que los retardos pueden variar mucho durante la sesión y es difícil estimar cada uno, se considera un retardo general suma de todos ellos.

La figura 6.1 contiene un esquema de los dispositivos de captura y reproducción, codificadores, la red, decodificador. Cada etapa introduce un cierto retardo. Generalmente donde se introducen los retardos mayores para el video son en la paquetización en RTP luego de la codificación, en la red (si se trabaja en Internet) y en la reconstrucción de imágenes para pasarle al decodificador.

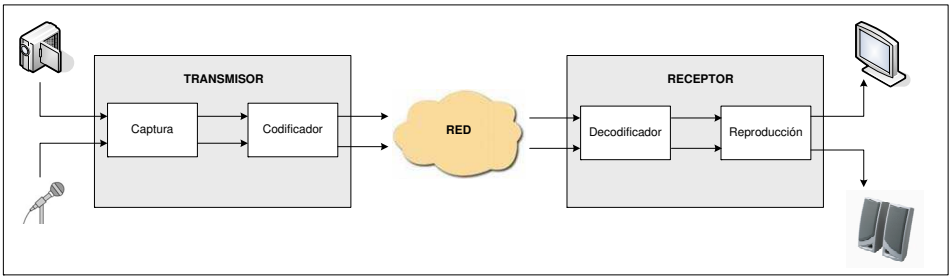


Figura 6.1: Componentes de la sesión

6.5. Sincronización de labios

Existen varias formas de lograr sincronización de labios. Una de las técnicas considera que los retardos son constantes y conocidos. Otra considera que que no lo son y se basa en un reloj común a los distintos medios. En este caso es posible aplicar la segunda técnica.

En la sincronización que utiliza un reloj de referencia común el método empleado para sincronizar no puede pretender calcular todos los retardos sino que debe intentar calcular el retardo total.

Para poder llevar a cabo la sincronización es necesario entender el comportamiento del transmisor, de la red y del receptor que se describen en secciones posteriores.

6.5.1. Conceptos necesarios de RTP y RTCP

Como ya se ha hecho anteriormente una breve descripción de estos dos protocolos, esta sección se remite únicamente a la aplicación de los mismos para la sincronización de medios.

RTP

Todos los paquetes de medios llevan un encabezado IP, uno UDP y luego se paquetizan en RTP. El uso de paquetes RTP permite la reconstrucción y reproducción de medios en la recepción sin *glitches* pero no permite una sincronización entre los mismos. Dado que las sesiones de audio y video RTP son diferentes, surge la necesidad de vincularlas de alguna forma.

Cada encabezado RTP lleva una marca de tiempo de 32 bits llamada *RTP timestamp* que se incrementa de acuerdo a la frecuencia de muestreo del medio contenido en el paquete. Si un stream de audio utiliza un reloj de frecuencia igual a su frecuencia de muestreo, por ejemplo para audio de 8kHz se lo muestrea a 8kHz y en cada paquete hay 300 muestras, el reloj se incrementa en 300. Para una señal de video de 90kHz si se desean 25 cuadros por segundo, el aumento del reloj es de 1/25 x 90000.

El valor inicial de las marcas de tiempo es elegido aleatoriamente, por lo tanto no es posible encontrar un vínculo entre las marcas de tiempo de las sesiones de audio y de video utilizando únicamente las marcas de tiempo de los encabezados RTP.

Si bien RTP no posibilita la sincronización en la recepción, permite hacer un manejo a nivel de buffer.

RTCP

Para que las marcas de tiempo de los paquetes RTP puedan ser relacionadas se generan además paquetes RTCP para cada una de las sesiones. Dichos paquetes contienen una marca de tiempo de referencia que es común a ambas sesiones. A la vez, tienen una marca de tiempo que se corresponde a la marca de tiempo de los paquetes RTP. Los paquetes RTCP permiten hacer un mapeo entre las marcas de tiempo de los paquetes RTP y una referencia común.

La referencia común, también llamada reloj de pared *wall clock*, viene dada por la etiqueta *NTP timestamp*. NTP viene de *Network Time Protocol* y es simplemente un reloj común al transmisor medido en segundos.

NTP

Es el reloj de pared que provee referencia a los medios que salen de un punto final.

En la RFC 3550 se especifica que la etiqueta *NTP timestamp* tiene un tamaño de 64 bits. Los primeros 32 bits corresponden a los segundos y los restantes 32 bits a las fracciones de segundo. Por lo tanto la precisión de este reloj es de +/- 0,1 nanosegundos [51].

Se suele mal interpretar que RTCP necesita de un servidor de tiempo NTP para generar el reloj del transmisor. Este servidor de tiempo permite sincronizar relojes y mide los segundos que pasaron desde el 1 de enero de 1970. Sin embargo, tal como está definido en la especificación de RTP, el uso de este servidor no es obligatorio. De hecho la mayoría de las aplicaciones

de video no utilizan un servidor de tiempo NTP como reloj de pared.

El transmisor puede generar la etiqueta *NTP timestamp* directamente a partir de cualquier reloj de referencia. Es usual que se reutilice el reloj del dispositivo de captura de audio como base de tiempo NTP.

En la aplicación del video teléfono se utiliza un reloj provisto por Windows que genera el tiempo transcurrido desde el 1 de enero de 1970 hasta el momento expresado en segundos y fracciones de segundo (se usa la estructura *timeb*).

La especificación de NTP está en la RFC 1305 donde se describe el uso de este protcolo. El *Network Time Protocol* se utiliza para sincronizar tiempos entre un grupo de servidores y clientes distribuidos.

También se definen algunos conceptos que resultan útiles a la hora de trabajar con relojes de dos fuentes distintas. La *estabilidad* de un reloj indica qué tan bien se mantiene una frecuencia constante. La *exactitud* es qué tan bien se conserva la frecuencia y el tiempo en comparación con los estándares internacionales. La *precisión* es qué tan bien se conservan estas cantidades con el tiempo. El *offset* entre dos relojes es la diferencia de tiempo entre uno y otro mientras que el *corrimiento* o *skew* es la diferencia de frecuencia. Los relojes sufren variaciones de corrimiento llamadas *drift*.

6.5.2. Comportamiento del transmisor

El transmisor se encarga de la captura de medios, codificación, paquetización en RTP y envío de paquetes a la red. Para la paquetización del audio deben agruparse varias muestras en un paquete RTP, mientras que para la de video se deben fragmentar los cuadros para poder encapsularlos en RTP. En la figura 6.2 se pueden apreciar las distintas etapas de la transmisión.

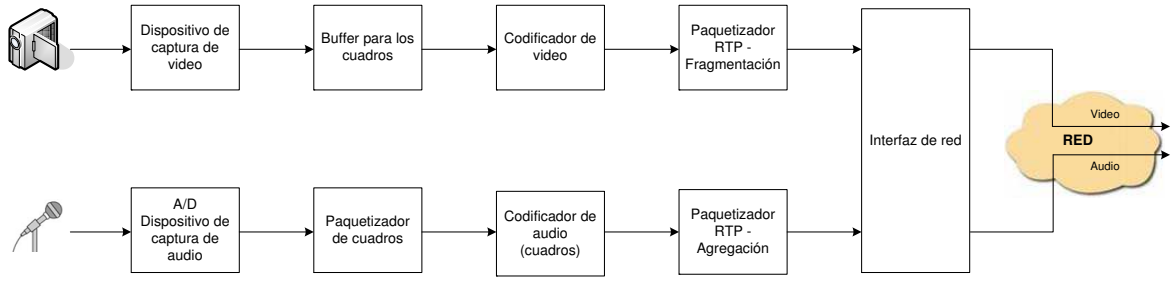


Figura 6.2: Comportamiento del transmisor

En cada una de las etapas se generan retardos:

- **Retardo debido a la paquetización en la captura de audio**
  La mayoría de los dispositivos de captura de audio proporcionan el audio capturado en paquetes formados por una cantidad fija de muestras. Así el retardo se puede calcular como el número de muestras por paquete dividido por la frecuencia de muestreo (muestras por segundo).

- **Retardo en la paquetización del audio para su codificación**
  Los codificadores necesitan que les sean ingresados grupos de datos (chunks) conocidos como cuadros de audio para producir datos a la salida. El tamaño de estos cuadros depende del codec de audio empleado. Para G.729 los cuadros son de 10 ms.

- **Retardo en el procesamiento del audio en el codificador**
  Este retardo es provocado por el algoritmo utilizado para codificar los datos. Este retardo no puede ser mayor a la duración de un cuadro de audio.

- **Retardo en la paquetización RTP del audio**
  Para formar los paquetes RTP se deben recolectar varios cuadros de audio y luego generar el correspondiente encabezado. El retardo viene dado por el tiempo desde que se junta el primer frame de audio hasta que el paquete RTP es enviado a la interfaz de red.

- **Retardo en la interfaz de red**
  Este retardo es muy pequeño en comparación con los antes mencionados.

Para el caso de video existen únicamente dos fuentes de retardos:

- **Retardo de codificación del video**
  Es el tiempo transcurrido desde que se capturó el cuadro hasta que se codificó el mismo.

- **Retardo en la paquetización RTP del video**
  Para paquetizar los cuadros de video es necesario que los mismos sean fragmentados. Esta fragmentación no se puede hacer de cualquier manera se debe respetar lo especificado por la RFC del codec de video usado. Para MPEG-1 la norma que especifica el encabezado de video a agregar al paquete RTP y la forma de fragmentación del video codificado es la RFC 2250.

6.5.3. Comportamiento del receptor

El receptor se encarga de almacenar los datos, decodificarlos y mandarlos a los dispositivos de reproducción de audio y video. La figura 6.3 muestra el procesamiento realizado en el receptor.

En cada una de las etapas se generan retardos.

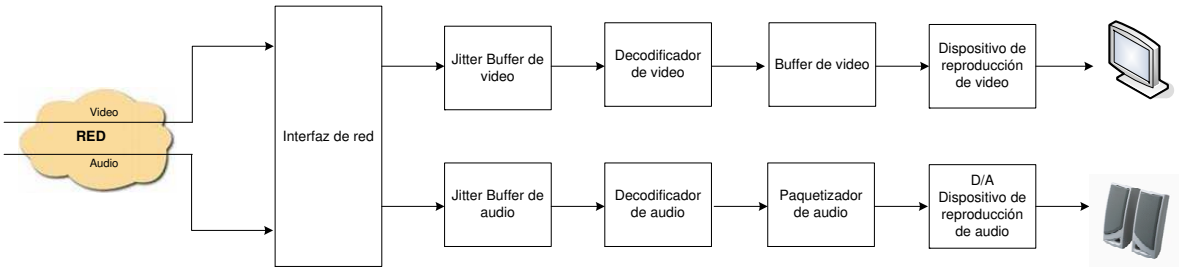


Figura 6.3: Comprotamiento del receptor

- **Retardo debido a la eliminación del jitter de audio**
  Se usa para absorber las variaciones en los tiempos de llegada de paquetes al receptor. Éste agrega un retardo que es igual al nivel nominal del buffer medido en milisegundos.

- **Retardo en la decodificación del audio**
  Es similar al retardo de codificación de audio.

- **Retardo en la paquetización de audio**
  Este retardo se debe a que se juntan varios paquetes de audio para enviar al dispositivo de reproducción de audio.

Los retardos para video son similares.

- **Retardo de paquetización de video**
  Este retardo se debe a que antes de pasarle al decodificador los datos se deben juntar todos los paquetes que forman un cudadro. Además puede existir un retardo adicional si el decodificador necesita que se le pase un GOP entero para empezar a decodificar.

- **Retardo de decodificación de video**
  Impuesto por el decodificador de video.

- **Retardo de despliegue del video**
  Es el tiempo que transcurre desde que se copian los datos hasta que se despliegan en pantalla.

Para el cálculo del tiempo de presentación, el receptor debe tener en cuenta los siguientes factores:

- El offset entre el reloj del transmisor y el del receptor cuando se hace el mapeo con el tiempo local.

- La variación en los tiempos de llegada de los distintos paquetes.

- Los retardos ocasionados por el procesamiento en la recepción.

6.6. Sincronización utilizando RTCP

La figura 6.4 muestra la relación de tiempo de envío, llegada y presentación de los paquetes RTP de audio y video. Se pueden ver también los retardos.

Para hacer la sincronización entre el audio y el video se hace un mapeo entre las etiquetas RTP de ambos medios y el reloj de referencia usando la información contenida en los paquetes RTCP. De esta forma se obtienen los tiempos de envío absolutos, es decir en unidades NTP. Después una vez recibidos los paquetes se obtiene el tiempo de llegada expresado en las mismas unidades (reloj local). Se hace una estimación del retardo de ambos medios y se calcula el retardo de cada uno. Se obtiene un delta que es la diferencia entre el retardo de video y el de audio. Si es mayor que cero se retarda el video y si es menor que cero se retarda el audio.

Por cómo está implementado el teléfono se puede asumir que el audio nunca va a tener retardos mayores que el video, por lo tanto el delta siempre es mayor que cero y se debe retardar únicamente el audio.

Antes de que lleguen los primeros paquetes RTCP de audio y video no es posible sincronizar. Una vez que llegan estos paquetes se puede hacer el mapeo entre las marcas de tiempo de los paquetes RTP de audio y video con el reloj de referencia.

6.6.1. Análisis de los paquetes de audio

Se analizan los paquetes RTP y RTCP de audio y se calcula el retardo desde el envío hasta la presentación de las muestras [52].

Primero se calcula el tiempo de envío mapeado en unidades absolutas (NTP).

$$TE_a = TE_{RTCPa} + \frac{T_{RTPa} - T_{RTPrtcpa}}{Ra} = T_{NTPrtcpa} + \frac{T_{RTPa} - T_{RTPrtcpa}}{Ra}$$

Donde:

TE_a es el tiempo de envío del paquete RTP mapeado en unidades NTP
TE_{RTCPa} es el tiempo NTP de envío del paquete RTCP de audio
$T_{RTPrtcpa}$ es el tiempo RTP del paquete RTCP de audio
$T_{RTPrtpa}$ es el tiempo del paquete RTP de audio
R_a es la frecuencia del reloj de audio en Hz. Para PCM es 8000Hz.

Después se calcula el tiempo de presentación de la muestra de audio.

$$TP_a = TL_a + Offset\ de\ llegada\ de\ audio$$

Donde:

TP_a es el tiempo de presentación de la muestra de audio

TL_a es el tiempo de llegada en unidades de NTP (reloj local)

El offset de llegada de audio se debe a al offset entre el reloj local y el reloj remoto (reloj de pared) y al offset debido al jitter. Dado que el offset entre los relojes del transmisor y el receptor varía a lo largo de una sesión, una forma de estimar las imprecisiones entre éstos es calcular las diferencias entre el tiempo de llegada y el de envío del paquete RTP para varios paquetes (se toma una ventana) y considerar que el offset es la diferencia mínima. El offset debido al jitter se puede calcular como el tiempo medio que está un paquete de audio en el *dejitter*.

Por último, cuando ya se tiene el tiempo de llegada y el de presentación del audio, se estima el retardo como la diferencia entre estos tiempos.

$Delay_a = TP_a - TL_a$

6.6.2. Análisis de los paquetes de video

El análisis para video es análogo al de audio. Se calculan los tiempos de llegada y presentación y en base a estos se estima el retardo.

$TE_v = T_{NTPrtcpv} + \frac{T_{RTPv} - T_{RTPrtcpv}}{Rv}$

Donde:

TE_v es el tiempo de envío del paquete RTP mapeado en unidades NTP
$T_{NTPrtcpv}$ es el tiempo NTP de envío del paquete RTCP de video
$T_{RTPrtcpv}$ es el tiempo RTP del paquete RTCP de video
$T_{RTPrtpv}$ es el tiempo del paquete RTP de video
R_v es la frecuencia del reloj de video en Hz. Típicamente 90000Hz.

Después se calcula el tiempo de presentación del cuadro de video.

$TP_v = TL_v + Offset\ de\ llegada\ de\ video$

Donde:

TP_v es el tiempo de presentación del cuadro de video
TL_v es el tiempo de llegada en unidades de NTP (reloj local)

El offset de llegada de video se debe a los mismos factores que el offset de llegada de audio más otros factores. Se le debe sumar el tiempo que lleva rearmar un cuadro de video (debido a que éstos se fragmentan para su transmisión) y el tiempo de procesamiento de los cuadros.

Finalmente se calcula el retardo de video como la diferencia entre estos dos tiempos.

$$Delay_v = TP_v - TL_v$$

6.6.3. Cálculo del retardo

Para calcular el retardo entre el audio y video se restan los retardos de cada uno y se calcula un delta.

$$\Delta = Delay_v - Delay_a$$

Si Δ es mayor que cero se debe retardar el audio en Δ * R_a. Si es menor que cero se retrasa el video Δ * R_v. En el caso del video teléfono se puede asumir que siempre es mayor el retardo de video por lo que se retrasa únicamente el audio.

6.6.4. Implementación e integración al código

Dada la arquitectura ya existente, se decidió implementar un recurso que tuviera dos pines de entrada y dos pines de salida. Un par de pines de entrada y salida para el audio y otro para el video. Dado que este recurso retardará únicamente el audio se decidió que fuera un recurso de audio por lo que hereda de `MpAudioResource` y es creado en la conexión de audio `MpConnection`. En la figura 6.5 se puede ver cómo interactúa el recurso de sincronización `MprSynchronize` con el resto de los recursos. Los pines 0 son para el audio y los pines 1 son para el video.

Para integrar el nuevo recurso al código se debieron cambiar las conexiones de audio y de video. Este recurso se crea en la conexión de audio al igual que el resto de los recursos de audio. Una vez creado se lo agrega a la gráfica de audio `MpCallFlowGraph` y se lo conecta a ésta usando los pines 0. Luego se le pasa un puntero al recurso `MprFromNet` del recurso de sincronización ya que ahora va estar conectado a éste y no al `MprDejitter`. También se conecta el `MprSynchronize` al `MprDejitter` por el pin 0. De esta forma el recurso queda conectado a la gráfica de audio tal como se muestra en la figura 6.5.

Como este recurso interviene tanto en la gráfica de audio como en la de video, fue necesario vincularlas de alguna forma. Para esto se implementó un método en la gráfica de audio que devuelve el puntero a la instancia de la conexión de audio, `getConnectionPtr`. En la conexión de audio hay un método que devuelve el puntero al recurso de sincronización. Así la conexión de video puede obtener la instancia al recurso de sincronización. Como se explicó anteriormente, las conexiones de audio y de video se crean en la clase `CpPhoneMediaInterface` y es en el método `createConnection` donde se llama a un método de la conexión de video que conecta al recurso `MprSynchronize` con los recursos de video `MprVideoFromNet` y `MprVideoDejitter` por los pines 1, como se muestra en la figura 6.5.

Con el recurso conectado a ambas gráficas por medio de los pines con el método `doProcessFrame` se procedió a implementar los métodos para el procesamiento de los paquetes RTCP y RTP con el fin de sincronizar. Para el procesamiento de los paquetes RTCP se siguió una línea

similar a la del procesamiento de los RTP.

A `MprFromNet` y `MprVideoFromNet`, como se describió en otras secciones, le llegan los paquetes UDP desde `NetInTask` y `NetInVideoTask` respectivamente. Estos recursos le sacan el encabezado UDP a los paquetes y hacen un cierto procesamiento de los paquetes RTP que luego envían a los siguientes recursos en sus respectivas gráficas (antes de que se diseñara el recurso de sincronización se conectaban a los recursos que eliminan el *jitter*). El método de los recursos `FromNet` que es llamado desde `NetInTask` y `NetInVideoTask` es `pushPacket` que tiene como parámetros de entrada un puntero al paquete UDP y un entero que indica si se trata de un paquete RTP y RTCP. Hasta el momento no se hacía ningún analísis de paquetes RTCP, simplemente se identificaba cuando llegaban y se los descartaba incrementando una cuenta de paquetes RTCP descartados. Ahora los paquetes RTCP se deben analizar para extraer la información de tiempo que vincula las marcas de tiempo de audio y de video.

Cabe señalar que si bien hasta el momento el código del teléfono no analizaba los paquetes RTCP en su recepción, sí los enviaba. Hay un hilo corriendo que envía los paquetes RTCP aproximadamente cada 5 segundos (tal como se recomienda en la RFC 3550). Se envían tanto *SenderReport* como *SDES* y *ReceiverReport*.

El análisis de los paquetes RTP dentro de los recursos `FromNet` se hace en el método `parseRtpPacket`. Ahí se obtienen las marcas de tiempo, el *padding*, las extensiones, la lista de *CSRC* y se guardan los paquetes RTP sin el encabezado UDP en una estructura del tipo `MpRtpBuf`. Para el análisis de los paquetes RTCP se implementó un método análogo llamado `parseRtcpPacket` y se diseñó una estructura análoga a `MpRtpBuf` para almacenar los paquetes RTCP llamada `MpRtcpBuf`.

Para la sincronización se deben analizar los paquetes RTCP *SenderReport* y *SDES*. Los primeros son los que permiten hacer el mapeo de tiempos y los segundos permiten saber si los medios tienen la misma procedencia mediante el atributo *CNAME*. En una aplicación en donde hay varios participantes hay que identificar qué medios se deben sincronizar. Para esto es necesario saber la procedenica de los paquetes RTP de audio y video. Dado que el *CNAME* es común a las sesiones de audio y video de un mismo participante permite identificar las fuentes a sincronizar. Además este identificador, a diferencia del identificador SSRC, es único para toda la comunicación. El SSRC puede cambiarse dentro de una misma sesión en caso de que haya colisión entre dos participantes.

Como por ahora el teléfono trabaja solo con dos participantes (todavía no se implementó la video conferencia) se asume que los datos recibidos son de la misma fuente por lo que se analizan únicamente los paquetes RTCP *SenderReport*. El tipo de paquete se obtiene del campo *PT* del encabezado RTCP siendo 200 el correspondiente para los paquetes *SenderReport*.

La estructura `MpRtcpBuf` al igual que la de RTP, hereda de `MpDataBuf` y está compuesta por un encabezado y los datos RTCP. Se le llama encabezado RTCP a la parte común a todos los paquetes RTCP independientemente del tipo. Todos los paquetes RTCP tienen los siguientes

campos:

- `V`: 2 bits. Versión.
- `P`: 1 bit. Padding.
- `IC`: 5 bits. Cuenta de ítems - *Item Count*.
- `PT`: 8 bits. Tipo de paquete - *Packet Type*.
- `Length`: 16 bits. Largo del paquete.

La estructura `MpRtcpBuf` tiene métodos de acceso a los distintos campos del paquete RTCP. Se crea una estructura para almacenar el encabezado RTCP llamada `RtcpHeader` que contiene los campos recién descriptos. El campo *Length* indica el tamaño en bytes del paquete RTCP incluyendo los bytes de *padding*.

En el método `parseRtcpPacket` de los recursos `FromNet` se analiza el tipo de paquete RTCP y se guarda en un buffer `MpRtcpBuf` el contenido del paquete si es un *SenderReport*. Después de analizado el paquete RTCP se pasa al recurso de sincronización con el método `pushRtcpAudioPacket` o `pushRtcpVideoPacket` dependiendo si se trata de audio o video.

El recurso `MprSynchronize` tiene como atributos las marcas de tiempo de los paquetes RTCP de audio y de video ya que a partir de éstas es que se calcula el retardo. También se guardan las marcas de tiempo de los primeros paquetes RTP de audio y video que llegan después de los RTCP de sincronización. Se decidió que la resincronización se haría cada 15 segundos aproximadamente, por lo que se analiza solamente un paquete RTCP de cada tres que llegan.

Usando las marcas de tiempo y mediante las ecuaciones descritas en la sección anterior, el recurso de sincronización calcula los retardos de audio y video y el Δ. Después se setea el retardo para los paquetes de audio en la clase `MprDeijitter`. Las muestras de audio son retardadas recién en el recurso `MprDecode` luego de que el *jitter buffer* le devuelve las muestras decodificadas. En `MprDecode` usando el puntero al recurso `MprDejitter` se accede al retardo de audio y se duerme el hilo durante ese tiempo antes de colocar las muestras en la cola de salida para ser reproducidas.

Para calcular el tiempo local expresado en las mismas unidades que el tiempo NTP de los paquetes RTCP se usó la estructura de Windows `_timeb` y la función `_ftime` que devuelve el tiempo local almacenado en una estructura `_timeb`. Esta última estructura tiene los siguientes campos:

- `time`: tiempo en segundos transcurridos desde el 1 de enero de 1970.
- `millitm`: fracciones de segundo transcurridas desde el 1 de enero de 1970 en milisegundos.
- `timezone`: diferencia entre el tiempo local y el GMT (Greenwich Mean Time).

- `dstflag`: no es cero si *daylight savings time* está en efecto para esta zona (ahorro de energía - cambio de horario en verano).

Para trabajar con las marcas de tiempo NTP, como son de 64 bits, se debieron usar dos variables *long* de 32 bits cada una y trabajar en punto flotante para realizar las cuentas. Se hicieron las conversiones necesarias del tiempo local y se calculó el tiempo de presentación como el tiempo local más un offset. Este offset como se explicó anteriormente se debe a diferencias en los relojes de transmisión y recepción y al procesamiento realizado para audiio y video en el receptor.

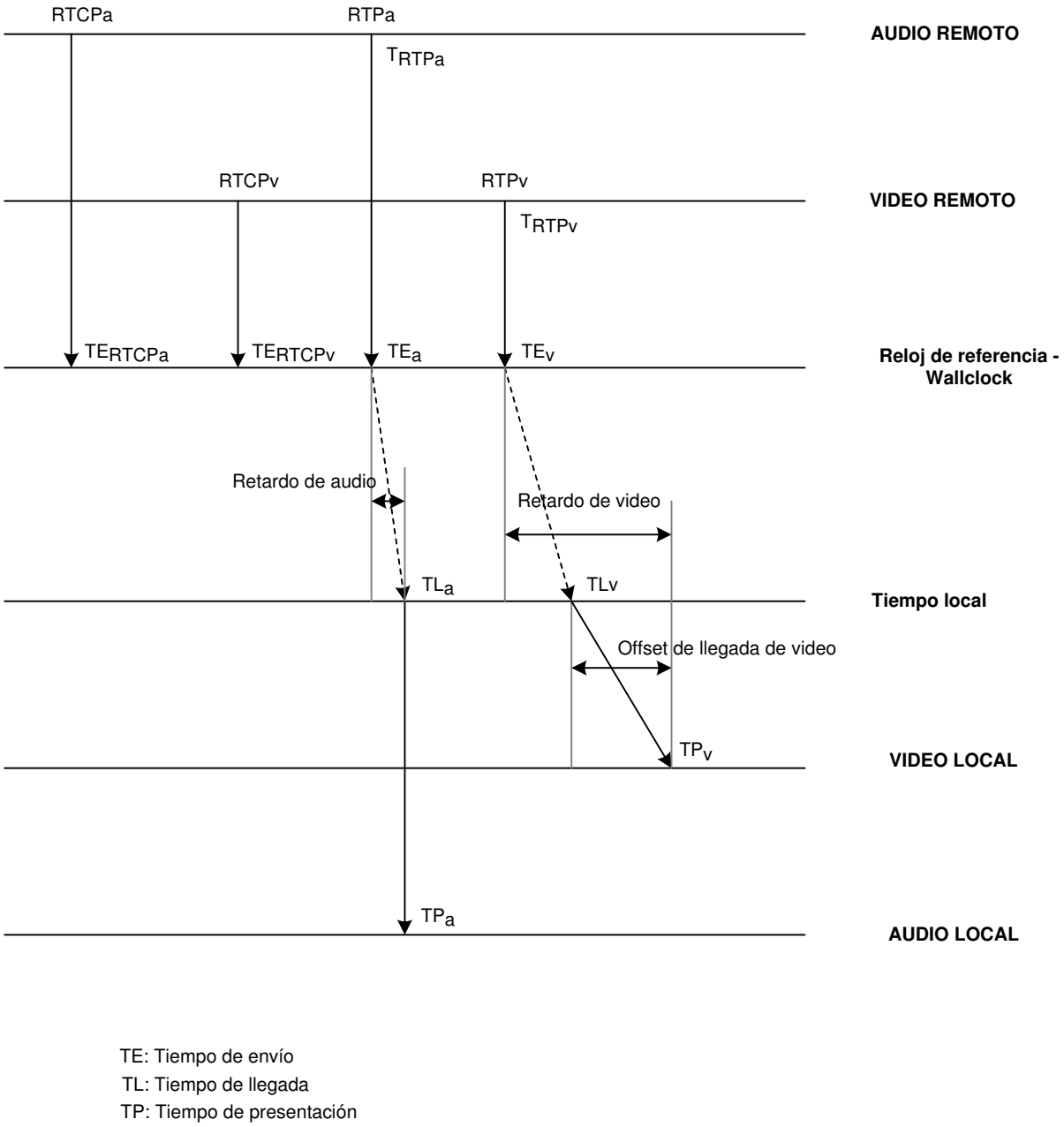
6.7. Resultados y conclusiones

Se logró implementar un recurso para la sincronización del audio y el video que pertenece a los recursos de audio. Se modificaron las conexiones de audio y video y pudo conectarse el recurso a ambas gráficas.

Se diseñó una estructura para el almacenamiento de los paquetes RTCP análoga a la estructura para el almacenamiento de paquetes RTP. En vez de descartar los paquetes RTCP, se analizaron dentro de los recursos `MprFromNet` y `MprVideoFromNet` y si son del tipo *SenderReport*se guardan en la estructura `MpRtcpBuf`. Se pasan los paquetes RTP y RTCP al recurso de sincronización que obtiene la información de marcas de tiempo y hace un mapeo para el cálculo de retardos.

Para el cálculo del offset de presentación de las muestras de audio y video se considera que una opción puede ser hacer una primera aproximación teórica y luego ajustarlo en base a ensayos. Esto se deja como sugerencia para en un futuro continuar implementando la sincronización entre audio y video. Se desarrolló una arquitectura básica y se planteó un posible camino a seguir.

Se puede concluir que si bien no se concluyó con la implementación de esta funcionalidad, se avanzó mucho en la tarea y se comprendió la forma de realizarla e integrarla al código existente.



TE: Tiempo de envío
TL: Tiempo de llegada
TP: Tiempo de presentación

Figura 6.4: Sincronización utilizando RTCP

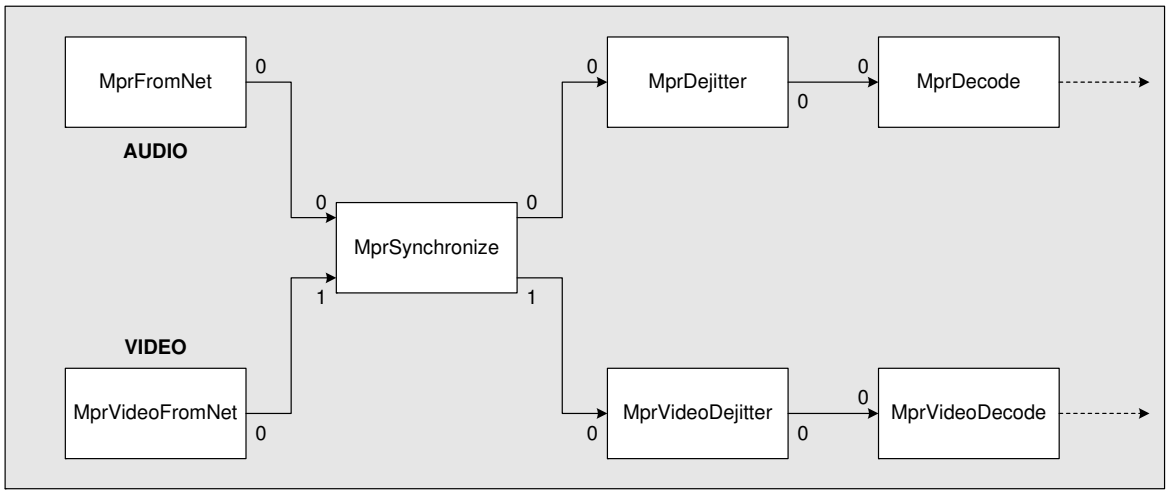


Figura 6.5: Conexiones de audio y video con el recurso de sincronización

Capítulo 7

Modificación de la interfaz

7.1. Introducción

Como último agregado a la aplicación sipXvideoPhone se cambió el aspecto de la interfaz gráfica. La solución original disponía de la interfaz que fue mostrada en la sección 3.1.2. Dado que cambiar la misma era un tema pendiente para los desarrolladores de la solución original se creyó interesante realizarlo como parte del proyecto.

En esta sección se explica a grandes razgos cómo está desarrollada la interfaz y qué relación tiene con el resto de las librerías que componen la aplicación. Finalmente se muestra el resultado obtenido.

7.2. Cambio de interfaz

La intefaz de la aplicación sipXezphone esta desarrollada con wxWidgets, que es una biblioteca libre utilizada para el desarrollo de interfaces gráficas programadas en lenguaje C++. La forma en que las clases que pertenecen a la interfaz se relacionan con el resto de la aplicación es mediante una clase llamada `sipXmgr`. Esta clase es invocada desde las instancias que contienen a los botones y ventanas cada vez que hay algún cambio en el estado de los mismos. A su vez `sipXmgr` es quien interactúa con la librería sipXtapi. Recordando, esta librería es el API que permite desarrollar aplicaciones SIP, siendo la librería de más alto nivel como lo indica la figura 3.3.

Las clases más importantes que refieren a la interfaz y permiten comprender los cambios son:

`MainPanel:` es la clase donde se crea el contenido de la ventana principal. Dentro de los elementos que crea encontramos:

- Controles de volumen de micrófono y parlante.
- Panel donde se encuentra la botonera para discar.

- Panel para ingresar el URI con el cual se desea comunicar.
- Botón de discar.
- Panel con botones para atender, cortar, transferir, poner mudo y hold.
- Botón que despliega la agenda.
- Botón que despliega las pantallas de visualización y previsualización.

`SipXezPhoneFrame`: aquí se crea el marco que contiene a todos los elementos de la interfaz gráfica. Se crean la barra de menús principales, el `MainPanel`, la ventana de agenda y las ventanas de previsualización y visualización. Los menús más importantes son:

- *Configuration*
- *Audio Settings*
- *Video Settings*

`SipXezPhoneApp`: esta es la clase que representa a la aplicación principal. Aquí se carga la configuración inicial del teléfono y se crea `SipXezPhoneFrame`.

7.2.1. wxWidgets

Dentro de las clases explicadas es donde se crean las ventanas, paneles, botones y demás elementos visibles. Para cambiar el aspecto de la aplicación fue necesario estudiar brevemente algunas de las herramientas que wxWidgets provee. Las herramientas más utilizadas fueron:

- `wxFrame`: representa una cuadro del tamaño y en la posición que se desee. Crea ventanas como las típicas que contienen a todas las aplicaciones y dentro de ella se pueden alojar otras ventanas.
- `wxPanel`: representa una ventana donde se ubican elementos a controlar.
- `wxDialog`: un diálogo es similar a una ventana donde el elemento a controlar que son elementos que permiten al usuario elegir una opción entre muchas.
- `wxMenu`: es un menú del tipo *popup* donde se despliega una lista de items para elegir. Luego de que un elemento de la lista es elegido el menú se oculta.
- `wxGridSizer`: representa una estructura del tipo matriz en donde luego se coloca un elemento en cada entrada.
- `wxSlider`: es un control del tipo barra deslizadora, en donde se puede deslizar el control hacia arriba o hacia abajo para obtener el valor deseado.
- `wxComboBox`: esta función representa una lista donde el usuario puede editar y los datos ingresados son tomados para tomar alguna acción.
- `wxBitmapButton`: representa un botón en donde se despliega un bitmap.

- `wxStaticBitmap`: es una función que despliega una imagen del tipo bitmap dentro de diálogos.
- `wxColor`: sirve para representar un color en algún elemento de la interfaz.
- `wxSize`: es una estructura que permite definir un ancho y alto para luego ser asignado a algún elemento de wxWidgets.
- `wxPoint`: es utilizado para representar las coordenadas de los elementos. Se compone de dos valores.

7.3. Resultados

Luego de comprender como se utiliza wxWidgets en la aplicación se realizaron cambios en la diagramación de los elementos visibles, en las figuras que los representan, los tamaños y en el tipo de objeto que los contienen. Como resutlado se obtuvo la interfaz que se observa en la figura 7.1.

Figura 7.1: Interfaz gráfica de sipXvideoPhone

Capítulo 8

Algunos detalles de implementación

El objeto de este capítulo es explicar en un mayor grado de detalle algunas de las clases que resultan claves para la aplicación. Aquí se describen las funciones más importantes de estas, las cuales permiten que el programa tenga el funcionamiento deseado. En primer lugar se detallarán las clases que se relacionan con las funcionalidades de audio y luego las que lo hacen con video.

8.1. Audio

En esta sección se describen las clases que ofician de codificador y decodificador de audio G.729. Como ya se explicó estas actúan de interfaz entre la aplicación y la librería que que implementa el codec.

8.1.1. MpeG729

`MpeG729` modela el comportamiento del codificador G.729. Esta clase utiliza las herramientas que provee Voice Age y refieren a la codificación, adaptándolas de forma adecuada a la aplicación sipXvideoPhone.

La clase `MpeG729` hereda de `MpEncoderBase` e implementa sus métodos. Los mismos son:

`initEncode:` este método debe ser invocado previo a que se codifique el primer flujo de datos de audio. Aquí se inicializan todas las variables e instancias utilizadas durante la etapa de codificación. Haciendo referencia a las herramientas de Voice Age, resulta adecuado invocar aquí al método `va_G.729a_init_encoder()` ya que este también debe ejecutarse previo a la primer codificación de audio.

`encode:` este es el método que implementa la codificación de un flujo de audio, que en este caso concreto permite obtener los datos en formato G.729. Aquí se reciben paquetes de audio de 10ms de duración y se ejecuta el método `va_G.729_encoder` de Voice Age para

codificarlos. A la salida se obtiene un paquete de datos codificados de tamaño igual a 10 bytes.

`freeEncode:` aquí se libera toda la memoria reservada durante la inicialización. Este método debe ser invocado una vez que ya no sea necesario volver a codificar información de audio.

Como se explicó en el punto 4.4.2 en esta clase se crea una estructura estática a la cual se le definieron las características del codec que se detallan el el cuadro 8.1.

Parámetro	**Valor**
versión del codec	G.729
frecuencia de muestreo de los datos PCM esperados	8000Hz
cantidad de canales soportados por el codec	1
tasa de bits del codec	8 kbps
cantidad de muestras PCM en el paquete entrante	80
cantidad de bits en un paquete codificado	80 bits

Cuadro 8.1: Características del codec

8.1.2. MpdG729

Análogamente se procedió a implementar la clase `MpdG729` que modela el decodifcador G.729. Esta clase actúa de interfaz entre la librería de Voice Age y la aplicación sipXvideoPhone.

En lo que sigue se especifican los métodos que la clase `MpdG729` contiene:

`initDecode:` en este método se inicializan todas las variables y memorias que son utilizadas durante el proceso de decodificación. Para integrar el aplicativo de Voice Age, aquí es necesario invocar al método `va_G.729a_init_decoder()`, el cual debe ser ejecutado antes de invocar cualquier otro método de la librería que se relacione con la decodificación.

También se inicializa la instancia del *jitter buffer* utilizado en la recepción. Como se explicó en el punto 4.4.2 este buffer permite almacenar los paquetes recibidos por la red para luego enviarlos a decodificar cada intervalos fijos de tiempo. De esta manera se logra reproducir el audio a una cadencia fija.

`decode:` este método es el que realiza la decodificación de audio cuando es recibido en formato G.729. Actúa de interfaz entre el método decodificador proporcionado por la librería de Voice Age, `va_G.729a_decoder()`, y el resto de la aplicación. Aquí se recibe un paquete con datos de audio (80 bytes) codificado según la recomendación G.729 y a través del método `va_G.729a_decoder()` se obtiene un paquete de audio en formato PCM de 10ms de duración.

`freeDecode`: al igual que en el codificador este método se utiliza para liberar las memorias reservadas y utilizadas durante el proceso de decodificación.

Aquí se define una estructura de datos que caracteriza los parámetros del codec G.729 al igual que en la clase `MpeG729`.

8.2. Video

La presente sección explica varios elementos relativos al agregado de la funcionalidad de video a la aplicación. En primera instancia se detallan elementos importantes que no forman parte del conjunto de recursos de video. Luego se explica cómo es el formato del método `doProcessFrame` que tienen todos los recursos por heredar de `MpResource`. Por último se describen las clases que implementan al codificador y decodificador MPEG-1.

8.2.1. Elementos complementarios a los recursos

MpVideoJitterBuffer

`MpVideoJitterBuffer` es un objeto que oficia de *jitter buffer* en la recepción y se utiliza para almacenar las imégenes ya decodificadas. A tales efectos, dispone de un arreglo (`JbQ`) de tamaño fijo donde es posible almacenar elementos del tipo `MpVideoSample`, diseñados para modelar las muestras de video que conforman las imágenes decodificadas.

Cada imagen de video decodificada está formada por una determinada cantidad de muestras, que en el caso de sipXvideoPhone se corresponde con los componenetes Azul (B), Verde (G), y Rojo (R) del formato de imagen BGR24 que utiliza 8 bits para representar cada componente. La cantidad de muestras utilizadas para representar una imagen decodificada se deriva del tamaño de la imagen multiplicado por tres para contemplar los tres componentes de color, y en la aplicación resulta en un valor de 176 x 144 x 3.

El objeto `MpVideoJitterBuffer` presenta, además del arreglo `JbQ`, tres parámetros que permiten llevar un control de la cantidad de elementos que se encuentran almacenados en cada momento. Estos parámetros se denominan `JbQIn`, `JbQOut` y `JbQCount` y cumplen las siguientes funciones:

`JbQIn`: es un puntero que en todo momento indica la posición del arreglo a partir de la cual se deben comenzar a colocar las siguientes muestras que ingresan al `MpVideoJitterBuffer`. El valor de este puntero se actualiza cada vez que ingresa una nueva imagen deocdificada al arreglo, incrementándose su valor en una cantidad correspondiente con la cantidad de muestras que presenta la imagen. De esta manera, el puntero queda referenciando al póximo lugar disponible para continuar almacenando muestras.

`JbQOut`: es un puntero que en todo momento indica la posición del arreglo a partir de la cual se deben comenzar a retirar las muestras que se encuentran en el `MpVideoJitterBuffer`.

El valor de este puntero se actualiza cada vez que se retiran muestras del arreglo de modo de quedar apuntando al próximo bloque de muestras que se deben retirar. Su valor es incrementado en una cantidad correspondiente con la cantidad de muestras que presenta la imagen que se retiró del arreglo.

`JbQCount:` es una variable que en todo momento lleva la cuenta de la cantidad de elementos `MpVideoSamples` que se encuentran almacenados en el arreglo. Esta variable es incrementada cada vez que ingresan imágenes decodificadas al `MpVideoJitterBuffer`, y es decremetada cada vez que se retiran muestras del mismo.

Los métodos mediante los cuales es posible colocar y retirar imágenes del `MpVideoJitterBuffer` son:

```
ReceivePacket(MpArrayBufPtr &videoFramePacket, int videoFrameSize, int payloadType
              ,MpVideoDecoderBase* videoDecoder)

GetSamples(MpVideoSample *videoSamples, JB_Video_size *pLength)
```

MpMisc

La clase `MpMisc` se utiliza básicamente para definir varios elementos que son utilizados a lo largo de todo el desarrollo. Una de las funcionalidades que presenta es definir buffers de memoria que se reservan durante la inicialización del teléfono. Estos buffers son reservados en bloques y para cada tipo de datos se dispone de un *pool* de buffers adecuado.

A modo de ejemplo, se listan a continuación algunos de los *pools* de buffers que se utilizaron para la implementación de video.

`RawVideoPool:` este *pool* de buffers está formado por una cantidad determinada de bloques de memoria, cada uno del tamaño de una imagen de video en formato BGR24. Estos bloques de memoria son del tipo `MpVideoBuf` y quedan referenciados por un puntero del tipo `MpVideoBufPtr`.

`VideoFramePool:` este *pool* de buffers se definió con el objetivo de poder almacenar cuadros de video codificados. El tamaño de estos cuadros es variable y por tanto se eligió un tamaño de bloque suficientemente grande capaz de almacenar cualquier imagen codificada que se envíe entre los agentes de usuario. Estos bloques de memoria son del tipo `MpArrayBuf`, y quedan referenciados por los punteros `MpArrayBufPtr`.

`RtpVideoPool:` para almacenar los paquetes RTP de video, se consideró conveniente crear un *pool* de buffers con bloques de memoria del tamaño de un paquete RTP. Este valor queda determinado por la constante `NETWORK_MTU`, que en el programa está seteada en un valor de 1500 bytes. Los bloques de memoria son del tipo `MpRtpBuf`, y quedan referenciados por punteros `MpRtpBufPtr`.

Para llevar a cabo la inicialización de todos estos espacios de memoria, y luego liberarlos al finalizar la llamada, la estructura `MpMisc` cuenta con métodos específicos que se detallan a continuación:

```
extern OsStatus mpStartUp(int sampleRate, int samplesPerFrame,
 int numBuffers, OsConfigDb* pConfigDb)

extern OsStatus mpShutdown(void)
```

El primero de estos métodos es invocado al abrirse la aplicación, para ininicializar la estructura `MpMisc` y con ella todos los *pools* de memoria definidos. Por otra parte, al finalizar la aplicación, se invoca al método `mpShutDown` que se encarga de liberar todas las memorias previamente resevadas en `mpStartUp`.

Además de estos dos métodos, `MpMisc` cuenta con dos más a través de los cuales es posible manipular la inicialización y el fin de las tareas que se ejecutan durante el programa. Estos métodos son los siguientes:

```
extern OsStatus mpStartTasks(void)

extern OsStatus mpStopTasks(void)
```

Al igual que en el caso anterior, uno de los métodos cumple la fución de inicializar y el otro de cerrar o liberar todo lo previmente inicializado. En este caso no se trata de espacios de memoria, sino de *tasks* donde se definen tareas específicas de algunas partes de la aplicación.

CameraThreadWnt

`CameraThreadWnt` es el hilo de ejecución a través del cual las imágenes de video capturadas son entregadas al resto de la aplicación para ser procesadas.

Las funciones más importantes y que ayudan a entender la mecánica del hilo son:

`cameraOutCallBackProc:` es la función mediante la cual se avisa desde `sipXmediaFactoryImpl` que hay una imagen capturada y se envía para que el método la guarde en la cola del hilo.

`CameraThread:` esta es la función principal del hilo la cual es llamada reiteradas veces. Cada vez que se ejecuta, se invocan las funciones `inPostVideoUnprep` e `inPreVideoPrep` en este orden.

`inPostVideoUnprep:` recibida una imagen por el hilo de ejecución la misma es pasada a esta función, que es realmente quien interactúa con la gráfica de procesamiento mediante el recurso `MprFromCamera`. La imagen recibida es guardada en una estructura del tipo `MpVideoBufPtr` obtenida al pedir un buffer de memoria al *pool* `RawVideoPool`. El objetivo final es almacenar el buffer en que se guardó la imagen en la cola `mpCameraQ`, propia de la clase `MprFromCamera`. Previo a esto se debe verificar que no se esté fuera de rango en `mpCameraQ`, en cuyo caso se pasará a descartar los datos más antiguos. Una vez asegurado que se estará dentro del rango, el buffer del tipo `MpVideoBufPtr` que contiene la imagen se guarda en la cola del primer recurso en la recepción, a través de la función `MpMisc.pCameraQ->send`.

`inPreVideoPrep`: mediante esta función se reservan las estructuras de memoria necesarias para un correcto funcionamiento del hilo.

`cleanVideoPrep`: mediante esta función se liberan las estructuras de memoria antes reservadas para el funcionamiento del hilo.

ScreenThreadWnt

`ScreenThreadWnt` es el hilo de ejecución a través del cual las imágenes de video son desplegadas en la pantalla de visualización del agente de usuario.

Para comprender el funcionamiento del mismo a continuación se explican las funciones más importantes:

`cleanOutVideoPrep`: esta función libera las estructuras de memoria previamente reservadas para el funcionamiento del hilo.

`TimerVideoCallbackProc`: esta función es invocada cada $\frac{1}{cuadrosporsegundo} = \frac{1}{25} = 40ms$, para poner en la cola del hilo un mensaje notificando que una nueva imagen debe ser desplegada en pantalla. Utilizando ese intervalo de tiempo la función es invocada 25 veces por segundo, lo que coincide con la cantidad de cuadros por segundo que se desea ver en pantalla.

`outVideoPrePrep`: aquí se reservan las estructuras de memoria a utilizar la primera vez que se ejecuta el método. Las siguientes veces se recibe una imagen de la cola propia del último recurso de recepción, `MprToScreen`, mediante `MpMisc.pScreenQ->receive` y se guardan en una estructura del tipo `MpVideoBufPtr`. Una vez obtenido este buffer el mismo es retornado para que se siga procesando su contenido.

`ScreenThread`: es el método principal del hilo y cada vez que tiene un mensaje anunciando que debe desplegar una imagen en pantalla invoca a `outVideoPrePrep`. Luego se invoca a la función `setpFrame` pasándole como parámetro el puntero a la imagen que debe desplegar en pantalla. Como se explica en la sección 5.25 la función `setpFrame` hace que los datos que se le pasan sean desplegados en pantalla. Por último es necesario decirle al manejador de recursos que debe empezar a procesar la recepción de una nueva imagen mediante la invocación a `MpMediaTask::signalFrameStart()`.

DmaVideoTaskWnt

La clase `DmaVideoTaskWnt` modela el *task* correspondiente con las tareas de procesamiento en los extremos de la conexión en cada uno de los agentes de usuario. Estos son la pantalla de visualización de video y la cámara web.

Es utilizada para definir una gran cantidad de constantes que son empleadas a lo largo de todo el código y de esta manera mantener los valores parametrizados. Dado que `DmaVideoTaskWnt` debe manejar todo aquello relacionado con la pantalla de visualización y la cámara de video,

las constantes aquí definidas son básicamente acerca de los tamaños de pantalla, cantidad de muestras por imagen, entre otros. A modo de ejemplo:

```
QCIF_WIDTH = 176
QCIF_HEIGTH = 144
VIDEO_BITS_PER_SAMPLE = 8
VIDEO_SAMPLES_PER_FRAME = QCIF_WIDTH*QCIF_HEIGTH*VIDEO_BITS_PER_SAMPLE/8
VIDEO_FRAME_RATE = 25
```

Otro objeto de la clase es inicializar dos hilos de ejecución y darle a cada uno un nivel de prioridad. El método que hace esto es `dmaVideoStartup`. Aquí se inicializa el hilo llamado `CameraThreadWnt` el cual hace de interfaz entre `MpVideoFlowGraph` y la cámara de video. También se inicia el hilo de ejecución llamado `ScreenThreadWnt` quien actúa de interfaz entre `MpVideoFlowGraph` y las pantallas donde se muestra el video.

También existe un método de esta clase, `dmaVideoShutdown`, encargado de terminar la ejecucución de los hilos al finalizarse la llamada.

NetInVideoTask

`NetInVideoTask` es el *task* correspondiente con el procesamiento de los datos al momento de recibirlos por la red. Aquí se encuentra implementado todo lo relacionado con los *sockets* de video que intervienen en la aplicación.

Un método de gran importancia en `NetInVideoTask` es el denominado `get1VideoMsg` quien lleva a cabo la lectura en el *socket* de video y de esta manera obtiene los paquetes UDP arribados a la red. El contenido de estos paquetes es depositado en un bloque de memoria del tipo `MpUdpBuf`, reservado previamente durante la inicialización del teléfono. Luego se entrega el puntero de referencia `MpUdpBufPtr` al primer recurso de la gráfica de procesamiento, el `MprVideoFromNet`. Esto se lleva a cabo al invocar el método `pushPacket` disponible en el recurso y quien recibe como parámetro la referencia al paquete UDP recién almacenado en memoria.

MpMediaTask

`MpMediaTask` es el elemento responsable de controlar la ejecución de las gráficas de procesamiento y en consecuencia de todos los recursos que las componen. Esto se aplica tanto para el caso de audio con el `MpCallFlowGraph` como para el de video con el `MpVideoFlowGraph`.

Cuando las gráficas de procesamiento son instanciadas se encuentran por defecto en estado inactivo. La clase `MpMediaTask` es informada de la existencia de una nueva gráfica a través del método `manageFlowGraph` y a partir de ese momento el programa es consciente que debe manejarla.

Para que los recursos de procesamiento lleven a cabo sus tareas, su gráfica asociada debe previamente pasar al estado activo. Esto lo realiza `MpMediaTask` ejecutando el método `startFlowGraph`. Del mismo modo, cuando se desea desactivar las gráficas al finalizar la aplicación se invoca al método `stopFlowGraph`, también de `MpMediaTask`.

`MpMediaTask` dispone del método `signalFrameStart` a través del cual se le informa que comienza un nuevo *frame interval* y por tanto es momento de procesar un nuevo bloque. En el caso de video este método es invocado en el hilo `ScreenThreadWnt` al momento de finalizar con el procesamiento de cada imagen. En lo que corresponde a la sesión de audio, el método es invocado en el hilo análogo denominado `SpeakerThreadWnt`.

`MpMediaTask` se encarga de manejar adecuadamente los turnos de ejecución de las gráficas de audio y video. Para ello dispone del método `setFocus`. Este método es ejecutado reiteradas veces durante el tiempo en que la llamada se encuentra establecida para ir alternando entre el procesamiento del flujo de audio y el de video.

8.2.2. doProcessFrame

El método `doProcessFrame` está presente tanto en los recursos de audio como en los de video, y todos ellos lo heredan de la clase padre `MpResource`. Cabe señalar que si bien es un método común a todos los recursos de procesamiento de medios, se encuentra implementado de manera distinta en cada uno de ellos según el comportamiento buscado en cada ocasión.

Para comprender mejor la interacción de los recursos a través del método `doProcessFrame`, se creyó conveniente mostrar la manera en que éste es declarado:

```
doProcessFrame(MpBufPtr inBufs[], MpBufPtr outBufs[], int inBufsSize, int outBufsSize,
    UtlBoolean isEnabled, int samplesPerFrame, int framesPerSecond)
```

donde cada parámetro significa los siguiente:

- `inBufs[]`: es un puntero al buffer que le está entregando el recurso anterior, si es que lo hay.
- `outBufs[]`: es un puntero al buffer de salida que se le entrega al siguiente recurso, si es que lo hay.
- `inBufsSize`: es el tamaño del buffer de entrada.
- `outBufsSize`: es el tamaño del buffer de salida.
- `isEnabled`: es un booleano que indica si el recurso está habilitado para ejecutarse o no lo está.
- `samplesPerFrame`: indica la cantidad de muestras por bloque de datos correspondientes con el flujo de video que se está procesando.

- `framesPerSecond`: indica la cantidad de cuadros por segundo correspondientes con el flujo de video que se está procesando.

8.2.3. MpeVideo

`MpeVideo` es la clase que modela al codificador MPEG-1. Aquí fue necesario implementar distintos métodos para poder enviar desde el agente de usuario local el flujo de video codificado correctamente. Éstos se describen a continuación.

`initEncode:` este método es invocado una única vez antes de codificar la primer imagen. Inicializa variables y memoria para uso de la clase así como funciones y atributos de Libavcodec que es necesario invocar previo a la codificación. Uno de los atributos que es necesario inicializar es `AVCodecContext` al cual se le asignó la siguiente configuración:

- `AVCodecContext->width:` 178 píxeles
- `AVCodecContext->height:` 144 píxeles
- `AVCodecContext->frame_rate:` 25 cuadros por segundo
- `AVCodecContext->gop_size:` 3 (una imgen I cada dos imágenes P)

`freeEncode:` mediante este método se liberan algunas memorias reservadas tanto en el `initEncode` como en la propia librería Libavcodec.

`encode:` aquí es donde corresponde transformar la imagen recibida en formato BGR24 a una imagen en formato MPEG-1. Previo a codificar es necesario transformar el formato de la imagen, de BGR24 a YC_rC_b4:2:0 para luego utilizar las funciones de la librería Libavcodec y obtener el resultado deseado. Aquí es necesario actualizar variables que mantengan la información de qué tipo de imagen se codificó y cuántas se han codificado hasta el momento.

`getRTPvideoHeaderLength:` con este método se obtiene el tamaño del encabezado de video definido en la RFC 2250, de valor 32 bits.

`writeRtpPayload:` este método recibe un fragmento de imagen MPEG-1 y lo paquetiza en RTP. Para esto se debe formar el encabezado específico de video tal cual lo indica la RFC 2250. Entre la información que debe ser tenida en cuenta para realizar esto se encuentra el tipo de imagen al que corresponde el fragmento recibido, si éste contiene algún tipo de encabezado MPEG-1 y cuántas imágenes han sido enviadas por la red.

8.2.4. MpdVideo

En `MpdVideo` se hace uso de las herramientas proporcionadas por Libavcodec, para lo cual fue necesario definir métodos y atributos específicos que permitieran utilizarlas de manera adecuada. Además de integrar la librería, se procedió a implementar otros métodos que fueran de utilidad al momento de reconstruir la información de video recibida.

La clase cuenta con un arreglo denominado `FrameQ` que fue diseñado con el objetivo de almacenar las imágenes de video que van siendo recibidas por la aplicación. La idea es que el arreglo sea utilizado para ir guardando los datos correspondientes a una misma imagen, mientras se va reuniendo el contenido completo de la misma.

Para llevar el control de los datos que en cada momento están presentes en `FrameQ`, se utilizan la variables `FrameQIn`, `FrameQOut`, y `FrameQCount` cuyos funcionamientos se describen a continuación:

`FrameQIn:` es un puntero que en todo momento indica la posición del arreglo a partir de la cual se deben comenzar a colocar los próximos datos correspondientes con la imagen que se está almacenando. El valor de este puntero se actualiza cada vez que al arreglo ingresa una nueva tanda de datos de la imagen codificada, incrementándose su valor en una cantidad correspondiente con la cantidad de datos que ingresan. De esta manera, el puntero siempre queda referenciando al póximo lugar disponible para continuar almacenando la misma imagen.

`FrameQOut:` es un puntero que en todo momento indica la posición del arreglo a partir de la cual se debe comenzar a retirar una imagen completa. El valor de este puntero se actualiza cada vez que se retira una imagen del arreglo de modo de quedar apuntando a la próxima imagen que se debe retirar. Su valor es incrementado en una cantidad correspondiente con el tamaño de la imagen que se está retirando del arreglo.

`FrameQCount:` es una variable que en todo momento lleva la cuenta de la cantidad de datos que conforman la imagen almacenada en el arreglo (esta cantidad se mide en bytes). Esta variable es incrementada cada vez que ingresan datos correspondientes con una imagen codificada, y es decremetada cada vez que se retira la imagen.

Al desarrollar la clase `MpdVideo`, se encontró la necesidad de implementar los siguientes métodos que sirvan de ayuda para lograr los objetivos buscados:

`addToFrame:` este método tiene como objetivo reunir la información recibida correspondiente con una misma imagen de video. Esto es de utilidad al momento de querer entregarle los datos al decodificador para que realice su tarea. A medida que se recibe información con una imagen de video, la misma se deposita en el arreglo `FrameQ` propio de la clase.

`getFrame:` este método permite obtener del arreglo `FrameQIn`, la imagen de video codificado que se encuentra almacenada. Este método es invocado una vez que se reunió el contenido completo de cada imagen.

`decode:` este es el método que realiza la etapa de decodificación propiamente dicha. Aquí se utilizan los métodos `avcodec_decode_video` e `img_convert` proporcionados por Libavcodec, mediante los cuales es posible obtener las imágenes decodificadas y en el formarto BGR24 para poder luego desplegar en pantalla.

`getVideoHeaderLength:` con este método es posible acceder al tamaño del encabezado de video definido en la RFC 2250. Este encabezado presenta un tamaño fijo de 32 bits.

`initDecode:` aquí se inicializan los métodos y parámetros utilizados por la librería Libavcodec. Este método debe ser invocado una sola vez en toda la aplicación, y permitirá utilizar las herramientas de decodificación disponibles en Libavcodec.

`freeDecode:` análogamente, a través de este método es posible liberar las memorias reservadas y utilizadas por estructuras de la librería. Este método es invocado en el destructor de la clase `MpdVideo`.

Capítulo 9

Conclusiones

En este capítulo se hace un análisis final del proyecto, evaluándose los conocimientos adquiridos y resultados obtenidos. Se describe el proceso de desarrollo de cada una de las etapas del proyecto y su correspondencia con los objetivos planteados. Se sacan conclusiones en cuanto a los logros y los avances realizados.

A su vez se mencionan todos los apartamientos en cuanto a la planificación original, las dificultades encontradas en el camino y los pasos que se siguieron ante esta situación. Finalmente se hace un breve resumen de la gestión del proyecto incluyendo todas las tareas y su dedicación horaria.

9.1. Resultados obtenidos

Uno de los objetivos del proyecto consistió en el estudio del protocolo SIP con el fin de agregarle funcionalidades a un cliente SIP para establecer una comunicación de audio y video entre dos usuarios dentro de una LAN. Con este objetivo en vista se estudiaron en profundidad las características de SIP, sus aplicaciones y todos los protocolos necesarios para el desarrollo de la aplicación en cuestión.

Se adquirieron los conceptos generales de SIP así como también el detalle de algunos aspectos necesarios para su implementación. Dado que la solución de partida (sipXezPhone) es un cliente SIP se pudieron analizar con mayor facilidad los componentes básicos del establecimiento de la sesión: los mensajes.

Dentro de los conocimientos teóricos que fueron necesarios para el desarrollo de la aplicación de tiempo real se encuentra el estudio del protocolo de descripción de sesiones SDP y los protocolos de transmisión de medios en tiempo real RTP y RTCP. Se adquirió una muy buena base de estos protocolos ya que se debieron implementar para el agregado del soporte de video. Si bien en un principio se pensó que estos protocolos eran accesorios, resultaron absolutamente esenciales dado que la mayor dificultad del proyecto se encontró en la implementación de la

transmisión de video, por lo que se los debió estudiar minuciosamente.

En los objetivos del proyecto se encontraba también el agregado de un codec de audio para lo que se debió estudiar el codec elegido previo a su incorporación en el programa. Dado que se agregó el soporte de G.729 se estudió la recomendación y las técnicas empleadas por este codec de compresión de audio. Se logró implementar la negociación de medios mediante SDP así como también la transmisión de audio sobre RTP.

Además de lograr la comunicación de audio mediante G.729 entre dos de los teléfonos sipX-videoPhone, se verificó su compatibilidad con otras implementaciones resultando en un buen funcionamiento del mismo. Esto significó que se respetaron todas las características del establecimiento de la comunicación y el transporte de medios. Este hecho, fue de gran importancia dado que la compatibilidad de una aplicación con otras es de sumo interés.

Para alcanzar el objetivo de la comunicación de video, además de hacer posible su negociación en forma similar a la que se implementó para G.729, se debió desarrollar una arquitectura que permitiera su procesamiento y transporte dado que la aplicación de partida no tenía nada implementado.

Para el desarrollo de la arquitectura de video se consideró primordial la flexibilidad y compatibilidad con el resto de la aplicación. Se logró desarrollar una arquitectura flexible y muy abierta, en donde la integración de codecs de video implica únicamente implementar las características particulares de dichos codecs teniendo una parte común en el procesamiento de los datos de video.

Dentro de los conocimientos adquiridos cabe destacar que se aprendieron a utilizar herramientas de programación para el manejo de dispositivos de hardware como cámaras de video y monitores. Además se obtuvo un conocimiento general del manejo de *sockets* en Windows.

Para la incorporación del soporte de video se estudiaron distintos codecs analizándose las características, ventajas y desventajas de los mismos. Se estudiaron varias técnicas de compresión de video comunes a los distintos codecs. Entre los codecs estudiados se encuentra H.263, Theora (un codec de implementación propietaria de código abierto) y MPEG-1. Finalmente se optó por MPEG-1 estudiándose el mismo en detalle.

No solo se debieron estudiar las características de dicho codec, sino que también se estudió la forma en que se debe enviar a la red. Se estudió e implementó la recomendación sobre el paquetizado en RTP de MPEG-1. Si bien se podrían haber fragmentado los cuadros de video de cualquier forma para su envío, se consideró hacerlo de acuerdo a estándares para obtener compatibilidad con otras aplicaciones.

Dado que la aplicación que se desarrolló trabaja en tiempo real, se tuvieron en cuenta varios aspectos para la toma de decisiones de implementación.

Dentro de los logros del proyecto se destaca el aprendizaje de un nuevo lenguaje de programación, C++, así como cierta experiencia en el desarrollo de soluciones de software. La implemntación de la arquitectura de video implicó un análisis exhaustivo de posibles soluciones aplicándose varias téncias de desarrollo de software.

Se logró desarrollar una aplicación que funciona adecuadamente para la transmisión de audio y video en tiempo real. La arquitectura empleada es flexible, por lo que es posible modificar aspectos de la implementación sin tener que cambiar todo lo ya implementado.

El principal objetivo del proyecto era el desarrollo de un *softphone* basado en SIP que soportara el transporte de video. Dicho objetivo fue alcanzado, obteniéndose una aplicación que permite realizar todas las tareas necesarias. Es posible previsualizar la captura de la cámara en forma correcta, transmitir video codificado en MPEG-1 y visualizar en el otro extremo el video transmitido.

9.2. Imprevistos, desvíos y dificultades

El propósito de esta sección es dar una visión de los imprevistos y dificultades que surgieron a lo largo del proyecto y explicar los paliativos o acciones tomadas frente a los mismos.

La primer dificultad surgida se relaciona, ni más ni menos, con la solución de la cuál se pensaba partir. En un primer momento se eligió el agente de usuario sipXphone, desarrollado por SIPfoundry, como base del desarrollo. Luego de tener instalado el ambiente de desarrollo se procedió a compilar la aplicación y a estudiar la misma para poder comenzar con la inclusión de nuevas funcionalidades. Al poco tiempo de comenzado el estudio, la organización SIPfoundry la dio por obsoleta. Frente a esto surgió el dilema de si continuar o no con el desarrollo de esta aplicación. La opción que se tomó fue de abandonar el teléfono sipXphone ya que este era antiguo, complicado y no se podría recibir mucha ayuda por parte de otros desarrolladores. Este cambio de solución de partida implicó un atraso en los plazos previstos.

Desde un principio se tuvo como segunda opción otro agente de usuario SIP proveniente de la misma organización, sipXezPhone. Elegida la nueva solución de partida se procedió a instalar el nuevo ambiente de desarrollo y compilar dicha aplicación para su posterior desarrollo.

Una deficiencia de la nueva solución elegida fue la falta de documentación de la misma. En un principio se supo que esta solución disponía de menor documentación que sipXphone, pero igualmente se creyó una mejor opción seguir con el desarrollo de una solución en vigencia. La aplicación proveía una descripción general y dada la inexperiencia en el desarrollo de software de aplicaciones de tales dimensiones se creyó que esto sería suficiente. Al momento de estudiar el funcionamiento de la aplicación se vió que eso no era así y que la falta de documentación enlentecería el estudio.

Como forma de disminuir las consecuencias que esto traía se optó por utilizar un programa llamado DoxyGen que a partir de código en distintos lenguajes de programación genera documentación. Ésta fue útil hasta cierto punto logrando comprender algunos aspectos de la arquitectura y dejando por comprender muchos otros.

Sumado al tema de la falta de documentación, está el hecho de que la solución contuviera más de 500 clases. Esto constituyó una de las principales dificultades durante todo el desarrollo. La arquitectura del programa, la relación que tienen las clases entre sí y los distintos niveles por los que debe pasar una simple operación convierten a sipXezPhone una aplicación muy compleja. Esto no quiere decir que no sea posible el desarrollo de la misma, pero sí que es necesario mucho tiempo de adaptación al código y de comprensión del mismo a un nivel tal que permita continuar el desarrollo.

A nivel general existen *bugs* conocidas de la solución, algunos fueron resueltos en el correr del proyecto, y muchas otros aún siguen pendientes. Este tipo de errores puso trabas en el camino de desarrollo pero pudieron ser superadas.

Cabe mencionar que dado que el programa es una aplicación en tiempo real los tiempos de procesamiento deben ser realmente pequeños. El agregar video a la aplicación genera que la información a procesar tenga un gran volumen, haciendo que los tiempos del programa hayan jugado una mala pasada.

Un imprevisto que tuvo mucha implicancia en el proyecto fue la subestimación de horas que consumirían las distintas etapas. Sobre todo la etapa de video, donde viendo la sección 9.4 se observa claramente que el tiempo estimado fue por demás inferior al tiempo consumido.

Como último comentario cabe aclarar que lo mismo ocurrido al comienzo del proyecto cuando se conoció que sipXphone se había dado por obsoleto volvió a ocurrir recientemente con sipXezPhone. Los motivos de esto fueron la complejidad de la solución y la cantidad de *bugs* que el mismo contiene.

9.3. Trabajo a futuro

En cuanto a posibles trabajos a futuro existen muchos dado que a este tipo de aplicaciones siempre se le pueden agregar nuevas funcionalidades, corregir detalles, mejorar la performence, incursionar en cuanto a calidad de servicio, e infinidad de cosas más. Aquí se describen los trabajos a futuro que se consideran más importantes.

Respecto a las funciones con las cuales contaba el programa de partida, existe una que no funciona correctamente. Ésta es la función de transferencia y sería un interesante trabajo solucionar esto ya que implicaría un estudio más a fondo de SIP y la aplicación. El estudio de SIP sería necesario para comprender los mensajes que deberían ser intercambiados durante la

transferencia. Luego el desafío pasaría a ser el intentar plasmar todo ello en la aplicación.

El poder optar por más de un codec de video le da mucho valor a una aplicación así, ya que cada codec es adecuado para una situación específica. Esto sería otro trabajo a tener en cuenta a la hora de continuar con el desarrollo del programa. Durante el agregado del codec MPEG-1 a la aplicación ya fue tenida en cuenta la posible incorporación de otros codecs al momento de diseñar la arquitectura sobre la cual se transmitiría video. Teniendo el desarrollo de video con MPEG-1 solamente es necesario el agregado de código en algunas clases existentes y la inclusión de un módulo que incluya al nuevo codificador y decodificador. Esto incluye el paquetizado del codec sobre RTP, la fusión de los paquetes RTP en la recepción, entre otros.

Una de las ideas que surgió al idear el proyecto fue el agregado de conferencia de llamadas. Esto supone un estudio importante de la manera en que éstas son desarrolladas habitualmente y de qué forma se podría hacer compatible con la aplicación. Para este trabajo se podrían comprimir encabezados de paquetes y así aprovechar más eficientemente el ancho de banda disponible. Existe una recomendación de la IETF, RFC 2508, que describe como comprimir encabezados IP/UDP/RTP. También en la recomendación de SIP se explica cómo implementar una versión compacta de los campos.

Dado que la aplicación sipXvideoPhone supone como ambiente de trabajo una LAN, intentar que la misma funcione sobre Internet lo haría una aplicación muy atractiva. Para lograr esto hay que tener en cuenta la necesidad de utilizar servidores SIP, como por ejemplo servidores de registro. La aplicación original ya cuenta con alguna funcionalidad de este tipo y de cómo solucionar los problemas que trae estar detrás de una LAN al utilizar NAT. Las rutas que recorren los paquetes en Internet y la congestión de la red, generaría otros problemas como por ejemplo mucho *jitter*, retardo, pérdida de paquetes, desorden de los mismos. La aplicación ya cuenta con mecanismos para evitar el *jitter*, tener en cuenta la pérdida de paquetes y el desorden de los mismos. Todo esto junto con los retardos incluidos por la red y la necesidad de trabajar en tiempo real podría llegar a hacer laborioso este trabajo.

9.4. Gestión del proyecto

Al comenzar el proyecto se tomó un curso llamado *Curso de Gestión de Proyecto* dictado en el IIE. Con lo aprendido en el mismo y junto con el tutor se planificó un cronograma de tareas a realizar con plazos y dedicación horaria. A continuación se describe el cronograma planificado y luego el cronograma real con plazos y horas reales.

Planificación

N°	***Tarea***	***Fecha de inicio***	***Fecha de finalización***	***Horas totales***
1	Estudio SIP y protocolos relacionados	01/05/2006	01/07/2006	333
2	Elección softphone	01/07/2006	01/08/2006	167
3	Estudio softphone	02/08/2006	01/09/2006	167
4	Estudio del lenguaje de programación	s/p*		0
5	Implementación servidor SVN	13/01/2007*	21/02/2007	
6	Funcionalidades agregadas de audio	02/09/2006	01/12/2006	500
7	Transmisión de video	02/12/2006	01/03/2007	500
8	Funcionalidades agregadas de video	02/03/2007	01/04/2007	167
9	Documentación y presentación final	02/04/2007	02/05/2007	167
			Total	2001

* Tarea no tomada en cuenta a la hora de la planificación.

Horas reales

N°	***Tarea***	***Fecha de inicio***	***Fecha de finalización***	***Horas totales***
1	Estudio SIP y protocolos relacionados	01/05/2006	28/08/2007	133
2	Elección softphone, compilación, instalación de la LAN	01/03/2006	30/09/2006	167
3	Estudio softphone	01/10/2006	30/11/2006	330
4	Estudio del lenguaje de programación	01/10/2006	28/08/2007	70
5	Implementación servidor SVN			95
6	Funcionalidades agregadas de audio	01/12/07	21/03/07	350
7	Transmisión de video	22/03/2007	28/08/2007	1920
8	Funcionalidades agregadas de video	—	—	0
9	Documentación y presentación final	01/05/2006	28/08/2007	750
			Total	3815

Nota: Las tareas que no cuentan con fecha de inicio y fin fueron llevadas a cabo durante todo el período del proyecto.

Bibliografía

[1] J. Rosenberg et al. *SIP: Session Initiation Protocol*, RFC 3261, Junio 2002.

[2] H. Schulzrinne & S. Casner & R. Frederick & V. Jacobson *RTP: A Transport Protocol for Real-Time Applications*, RFC 3550, Julio 2003.

[3] M. Handley & V. Jacobson & C. Perkins *SDP: Session Description Protocol*, RFC 4566, Julio 2006.

[4] R. Fielding & J. Gettys & J. Mogul *Hypertext Transfer Protocol – HTTP/1.1*, RFC 2616, Junio 1999.

[5] J. Klensin *Simple Mail Transfer Protocol*, RFC 2821, Abril 2001.

[6] A. Johnston *SIP Understanding the Session Initiation Protocol* Segunda Edición, Artech House

[7] J. Rosenberg, *A Presence Event Package for the Session Initiation Protocol (SIP)*, RFC 3856, Agosto 2004

[8] A. B. Roach, *Session Initiation Protocol (SIP)-Specific Event Notification*, RFC 3265, Junio 2002

[9] http://www.cs.uwc.ac.za/ btucker/academic/research/publications/WuRadovanovicTucker-SATNAC2005.pdf

[10] M. Caetano, G. Maetos, F. Morales, SIP & Asterisk, Implementaciones de VoIP basadas en SIP, Setiembre 2005

[11] F. Yergeau, *UTF-8, a transformation format of ISO 10646*, RFC 2279, Enero 1998

[12] J. Rosenberg, H. Schulzrinne, *An Offer/Answer Model with the Session Description Protocol (SDP)*, RFC 3264, Junio 2002

[13] Andrew S. Tanenbaum *Redes de Computadoras*, Pearson:3era, Edición, 1997.

[14] SIPfoundry - http://www.sipfoundry.org/

[15] SIPforum - http://www.sipforum.org/

[16] IETF - http://www.ietf.org/

[17] SIP forum test framework - http://www.sipfoundry.org/sip-forum-test-framework/sip-forum-test-framework-sftf.html

[18] Pingtel - http://www.pingtel.com/

[19] Repositorio del código original del proyecto - http://scm.sipfoundry.org/rep/sipX/branches/sipXtapi-media-update/

[20] wxWidgets - http://www.wxwidgets.org/

[21] ITU-T Recommendation G.729, *CODING OF SPEECH AT 8 kbit/s USING CONJUGATE-STRUCTURE ALGEBRAIC-CODE-EXCITED LINEAR-PREDICTION (CS-ACELP)*, Marzo 1996

[22] ITU-T Recommendation G.712

[23] CELP - http://en.wikipedia.org/wiki/Code_Excited_Linear_Prediction

[24] ACELP - http://en.wikipedia.org/wiki/Algebraic_Code_Excited_Linear_Prediction

[25] Descripción G.729 - http://www.jdrosen.net/e6880/index.html

[26] LSP - http://en.wikipedia.org/wiki/Line_spectral_pairs

[27] http://en.wikipedia.org/wiki/Glottis

[28] Voice Age - http://www.voiceage.com/

[29] DoxyGen - http://www.doxygen.org/

[30] H. Schulzrinne & S. Casner *RTP Profile for Audio and Video Conference with Minimal Control*, RFC 3551, Julio 2003

[31] ISO/IEC 11172, *Coding for moving pictures and associated audio for digital sotrage up to about 1.5 Mbps*, Noviembre 1992

[32] J. Watkinson, *MPEG handbook*

[33] RLC - http://en.wikipedia.org/wiki/Run-length_encoding

[34] VLC - http://en.wikipedia.org/wiki/Variable-length_code

[35] Curso de TV digital, Ing. Rafael Sotelo

[36] D. Hoffman & G. Fernando & V. Goyal & M. Civanlar *RTP Payload Format for MPEG1/MPEG2 Video*, RFC 2250, Enero 1998.

[37] Theora - http://www.theora.org/

[38] L. Barbato *RTP Payload Format for Theora Encoded Video*, draft-barbato-avt-rtp-theora-01, Junio 2006.

[39] Windows Sockets API - http://msdn2.microsoft.com/en-us/library/ms740673.aspx

[40] VFW - http://en.wikipedia.org/wiki/Video_for_Windows

[41] Programming Microsoft DirectShow for Digital Video and Television, Mark D. Pesce

[42] DirectShow en Wikipedia - http://en.wikipedia.org/wiki/DirectShow

[43] AVICap - http://www.codeguru.com/Cpp/G-M/multimedia/video/article.php/c1601

[44] Microsoft Platform SDK Documentation

[45] VirtualDub - http://virtualdub.sourceforge.net/

[46] Foro de DirectShow de Microsoft - http://forums.microsoft.com/MSDN/ShowForum.aspx?ForumID=129&SiteID=1

[47] Foro de DirectShow de Google - http://groups.google.com/group/microsoft.public.win32.programmer.directx.video/topics

[48] Sistema de visión para el equipo de robots autónomos del ITAM, Luis Alfredo Martínez Gómez

[49] http://www.codeproject.com/directx/Paint_your_source_filter.asp

[50] Voice and Video Conferencing Fundamentals, Cisco Press, Scott Firestone & Thiya Ramalingam & Steve Fry

[51] David L. Mills, RFC 1305, *Network Time Protocol* (Version 3), Marzo 1992

[52] Addison Wesley, *RTP audio and video for the internet*, Colin Perkins.

[53] Video Capture in Microsoft Windows, Yun Teng

Apéndice A

Prueba del codec G.729 Voice Age

Previo a integrar el codec de Voice Age a la aplicación sipXvideoPhone, se procedió a probar la librería de forma aislada. A tales efectos, se utilizaron los archivos ejecutables proporcionados por Voice Age los cuales nos permitieron evaluar si el funcionamiento de la librería era el deseado.

El procedimiento correcto para probar el funcionamiento de la solución de Voice Age es el siguiente:

1. Por una parte verificar que la señal de salida del codificador cumpla con el estándar de G.729
2. Luego verificar que teniendo como entrada al decodificador una señal de audio codificada en G.729, la salida obtenida sea en el formato correcto y descrito anteriormente.

Para realizar estas pruebas se hace necesaria la utilización de un reproductor de archivos codificados en G.729. Dado que éste es un estándar no disponible gratuitamente no fue posible encontrar un reproductor adecuado para procesar este tipo de archivos. Fue necesario entonces encontrar otra alternativa que permitiera evaluar las herramientas de Voice Age.

La alternativa encontrada fue verificar que, dada una señal de audio en el formato correspondiente y pasándola por el codificador daba una salida tal que la misma a la entrada del decodificador retornara la señal original.

Igualmente a la hora de probar sipXvideoPhone (con este producto integrado) contra otros teléfonos que soporten G.729 se puede verificar la compatibilidad del producto de Voice Age con codecs que tienen otra procedencia.

Las pruebas realizadas con el aplicativo de Voice Age resultaron de ayuda para comprender cómo se hace uso de la misma y del API correspondiente.

Prueba codificador-decodificador

A continuación se describe el procedimiento de testeo que mencionado previamente.

1. Se partió de una señal de audio en el formato apropiado para ingresar al codificador G.729. El ejecutable `va_G.729a_encoder.exe` requiere como entrada un archivo de audio en formato PCM 16 bits mono, muestreado a una frecuencia de 8000Hz. Para obtener esta señal se utilizó el grabador de sonidos básico de Windows, el cual permite configurar el formato de salida de la grabación, pero lo entrega encapsulado en un archivo WAV. Una vez grabada la voz de una persona y obtenido el archivo WAV, que llamaremos `entradaCodificador.wav`, se extrajo el encabezado con un editor binario y se guardó para luego utilizarlo.

2. Se ingresó el archivo `entradaCodificador.wav` al codificador obteniendo una señal de audio codificada.

3. La señal codificada se ingresó al decodificador disponible en el ejecutable `va_G.729a_-decoder.exe` obteniendo un archivo, que se llamó `salidaDecodificador.raw`, en el formato que especifica Voice Age. Este formato coincide con el formato de señales de entrada al codificador.

4. Se incorporó al archivo `salidaDecodificador.raw` el encabezado WAV obtenido al comienzo de la prueba obteniendo un archivo WAV audible en cualquier reproductor de sonido. Este archivo resultó en un sonido exactamente igual al audible en `entrada-Codificador.wav`.

Observando que la salida generada por el decodificador se escuchaba igual que la señal original que ingresó al codificador, se pudo afirmar que el procesamiento punta a punta entre codificador y decodifcador se realiza correctamente.

Apéndice B

Problemas de integración del aplicativo G.729 de Voice Age

En este apéndice se presenta un análisis de los problemas encontrados al integrar la solución de Voice Age a la aplicación desarrollada.

Una vez establecida la sesión de audio G.729, se observó un flujo de paquetes RTP en ambas direcciones. Si bien el contenido de estos paquetes RTP se identificaba como codificado en G.729, lo que se escuchaba en ambas puntas receptoras era solamente silencio. Esto mostró un comportamiento erróneo del desarrollo realizado, lo que resultó en un análisis inmediato de los posibles puntos de falla.

Dado que la información recibida en destino se conformaba por silencio puro, existía la posibilidad de que la información no se estuviera codificando o decodificando correctamente. Esto podía ser consecuencia de que las bibliotecas proporcionadas por Voice Age no funcionaran correctamente o que la implementación del codificador y/o decodificador no fueran las adecuadas. La primera opción fue descartada de inmediato dado que el aplicativo de Voice Age fue probado previo a su integración en el programa. En cuanto a la segunda opción, hubo que revisar la información codificada para poder sacar conclusiones y detectar el problema.

Se verificó a qué correspondía la información que se estaba obteniendo a la salida del codificador. Para esto se procedió de la siguiente manera:

1. Se realizó una llamada entre dos agentes de usuario y se capturaron los paquetes RTP intercambiados.

2. A partir de los paquetes RTP se obtuvo la carga útil correspondiente y se formó un archivo con el contenido de varios paquetes RTP consecutivos.

3. Con ese gran contenido de datos, se creó un archivo de audio en formato WAV. Para ello, fue necesario recurrir a las especificaciones de dicho formato donde se detalla cómo debe ser el encabezado de un archivo de audio WAV.

4. Una vez disponible el archivo WAV, se escuchó utilizando un reproductor de audio adecuado. Si los paquetes RTP transmitidos durante la llamada contenían información codificada correctamente, este archivo WAV debía reproducir parte de la conversación mantenida durante la llamada. Este no fue el caso, dado que el audio resultante del archivo WAV no era consistente con ninguna conversación.

Los resultados de esta prueba no permitieron detectar el error del programa, motivo por el cual se procedió a analizar paso a paso la ejecución del código. Con esto se encontró una posible falla en la etapa de recepción de los datos, que está estrechamente relacionada con la decodificación de los mismos. Surgieron dos posibilidades:

1. Existía un problema de *jitter* en las muestras de audio obtenidas luego de pasar por la etapa de decodificación. Esto podía ser consecuencia de que el *jitter buffer* disponible en la recepción no tuviera el tamaño adecuado resultando en el descarte o sobreescritura de algunas muestras.

2. La interacción entre la clase `MpJitterBuffer` y el decodificador `MpdG729` no se estaba realizando de manera correcta. Cabe señalar que en `MpJitterBuffer` es donde se invoca al método decodificador.

Con estas conclusiones se realizaron las correciones correspondientes para que el flujo de audio ingresara de manera adecuada al decodificador G.729. Finalizados estos cambios se hicieron nuevamente las pruebas entre dos agentes de usuario sipXvideoPhone obteniendo, esta vez, resultados exitosos. La señal de audio intercambiada entre las dos puntas de la conversación se logró escuchar correctamente.

Apéndice C

Common Object Model (COM)

COM es una plataforma independiente, orientada a objetos para crear componentes de software binarios que interactúan.

No es un lenguaje orientado a objetos sino un estándar. Los objetos COM pueden ser creados en distintos lenguajes y una vez traducidos a código de máquina binario pueden interactuar (de ahí que se llaman componentes binarios). El único requerimiento de COM es que se use un lenguaje de programación que permita el uso de punteros para acceder a datos e invocar funciones.

Para el manejo de estos objetos se usan interfaces que son simplemente un conjunto de datos y funciones. La única forma de manipular los datos asociados a un objeto es por medio de interfaces.

Una interfaz es en realidad un conjunto de especificaciones de funciones que no están implementadas. Vendrían a ser definiciones de los métodos, es decir qué tipo de variables devuelven, qué parámetros tienen y qué deben hacer.

La implementación de las interfaces es el código que realiza las funciones especificadas en la definición. Muchas implementaciones de interfaces son provistas por librerías de COM. Igualmente, es posible que los implementadores desarrollen sus propias interfaces.

Una instancia de una interfaz es un puntero a un arreglo donde cada elemento es un puntero a uno de los métodos de la misma.

Todas las interfaces COM son de un cierto tipo, GUID (Globally-Unique ID) de 128 bits. Además poseen un identificador (IID) que especifica explícitamente y de modo único las funciones de dicha interfaz. Todo objeto COM es creado llamando a una API Win32 mediante `CoCreateInstance` que devuelve un puntero a una determinada interfaz de él.

La herencia en COM no es usada para la reutilización de código, ya que en realidad no

están implementadas, sino para heredar las definiciones de funciones. Un objeto COM puede implementar varias interfaces que son accedidas mediante el método `QueryInterface` descrito a continuación.

Todas las interfaces heredan sus definiciones de `IUnknown` que contiene tres métodos esenciales para controlar el objeto: `QueryInterface`, `AddRef` y `Release`. Mediante `QueryInterface` es que se consulta a un objeto si soporta una interfaz en particular (recordemos que los objetos pueden tener más de una interfaz). Si es así, este método devuelve un puntero a la misma. `AddRef` incrementa la cuenta interna del objeto cada vez que se le agrega una nueva interfaz, mientras que `Release` decrementa esta cuenta y verifica si llegó a cero la cuenta interna. Si es así significa que nadie más lo está usando por lo que libera la memoria del objeto. Una vez que se produce el `Release` no es posible utilizar ninguna de las interfaces del objeto [53].

Los procedimientos básicos de COM se resumen en el siguiente código:

```
CoInitialize(NULL); //se inicializa COM
IUnknown pUnk* = NULL;
IObject pObject* = NULL; //se crea el objeto COM
HRESULT hr = CoCreateInstance(CLSI_Object, CLSCTX_INPROC_SERVER,
NULL, IID_IUnknown, (void **)&pUnk);

if (SUCCEEDED(hr))
{
//se verifica las interfaces del objeto COM
hr = pUnk->QueryInterface(IID_IObject, (void **)&pObject);

if (SUCCEEDED(hr)) //se llaman a los metodos de dicha interfaz
{
pObject->method1();
pObject->method2();
pObject->Release();
pUnk->Release();
}
}

CoUninitialize(); //se libera el recurso que usa la libreria COM
```

Apéndice D

Manual de usuario

El motivo de este manual explicar cómo se utiliza sipXvideoPhone así como la manera en que se lo debe configurar. También se describen los requisitos que debe cumplir el sistema en que va a ser ejecutado para lograr un correcto funcionamiento.

D.1. Requerimientos del sistema

Los requerimientos mínimos de hardware y software del sistema son:

- Windows XP Service Pack 2
- 500 Mhz de procesador
- 256 MB de memoria RAM
- Tarjeta de sonido full-duplex
- Tarjeta de video
- Micrófono
- Parlantes
- Cámara web

D.2. Instalación

Para poder correr la aplicación como primer paso debe copiar la carpeta /Telefono al disco local de su computador. Una vez hecho esto no es necesario ningún tipo de instalación, solamente se debe hacer doble-click en el archivo sipXezPhone.exe.

D.3. Visión general

Aquí se da una visión general de las distintas pantallas del teléfono así como su función y uso. En la figura D.1 se observa la interfaz gráfica de la aplicación.

Figura D.1: Interfaz gráfica de sipXvideoPhone

D.3.1. Configuración

Al cargar el teléfono es desplegada una pantalla que se utiliza para configurar información del usuario y otros aspectos de la llamada. Ésta se observa en la figura D.2; los campos que es posible configurar y aplican a una llamada dentro de una LAN son:

Identity: es el identificador del usuario. Puede ser un número, un nombre o una combinación de ambos. Este campo se puede definir escribiendo la dirección URI de sip, por ejemplo: sip:maria@192.168.1.104 o solamente el nombre del usuario sin la dirección IP, que en este caso es "maria".

Default SIP port: define el puerto por el que se establece la negociación SIP. Por defecto se utiliza el puerto 5060.

Default RTP port: establece el número de puerto por el que se establece la primera sesión de medios. Se utilizan dos puertos por cada sesión de medios. Para las otras sesiones de medios se utilizan los puertos que siguen al establecido. Por defecto se utiliza el puerto 8000.

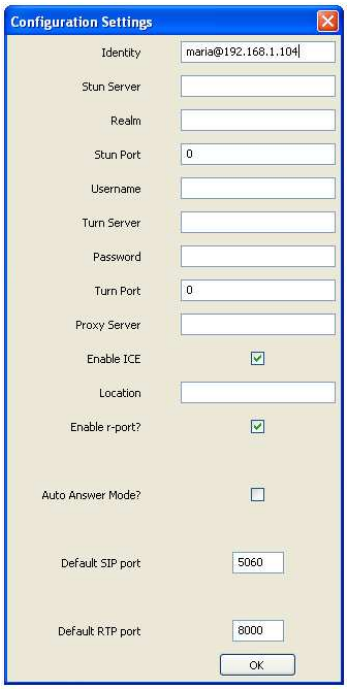


Figura D.2: Pantalla inicial de sipXvideoPhone

D.3.2. Menús

El teléfono cuenta con una barra de menús que contiene dos entradas, "Settings" y "Help". "Settings" contiene distintas entradas que sirven para configurar las opciones con las cuales se desea utilizar el teléfono. El menú "Help" da información sobre el origen del teléfono.

A continuación se describe la función de los ítems del menú que importan para especificar la configuración del teléfono a utilizar dentro de una LAN.

Configuration

Corresponde con la misma pantalla que se despliega al iniciar el teléfono. Permite cambiar los datos del usuario y los puertos por los cuales se establece la llamada en cualquier momento.

Audio Settings

Es donde se permite configurar el codec de audio a utilizar en la llamada. Los codecs disponibles son:

- G.711μ
- G.711a
- G.729

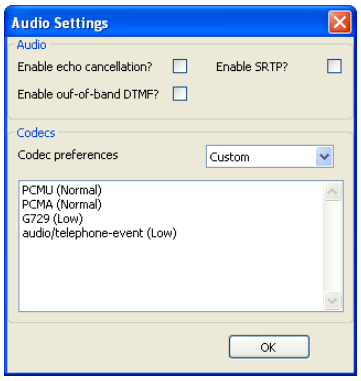


Figura D.3: Audio Settings

Video Settings

Aquí es posible escoger el codec de video que usa la aplicación. Actualmente se dispone de MPEG-1.

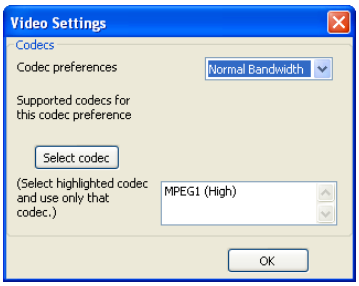


Figura D.4: Video Settings

D.4. Elementos importantes de la interfaz

La aplicación cuenta en su interfaz con una serie de elementos importantes que se detallan en esta sección.

D.4.1. Botontes

Los botones que resultan importantes para un buen manejo del teléfono son el de discar, atender y cortar, entre otros. Éstos son mostrados en la figura D.5 y a continuación se explica brevemente la función de los mismos.

Atender/Cortar: sirve para atender y cortar la llamada

Transferir: se utiliza para transferir la llamada en curso hacia otro teléfono

Hold: pone y saca de pausa una llamada

Mudo: bloquea la transmisión de sonido hacia el teléfono remoto. Volviendo a apretarlo vuelve a transmitir sonido.

Llamar: sirve para iniciar una llamada

Desplegar video: despliega una pantalla con dos subpantallas. La pantalla inferior corresponde al video capturado localmente y en la pantalla superior se despliega el video que se recibe del usuario remoto con el cual se estableció la comunicación.

Agenda: muestra una agenda telefónica donde se pueden registrar datos de contactos

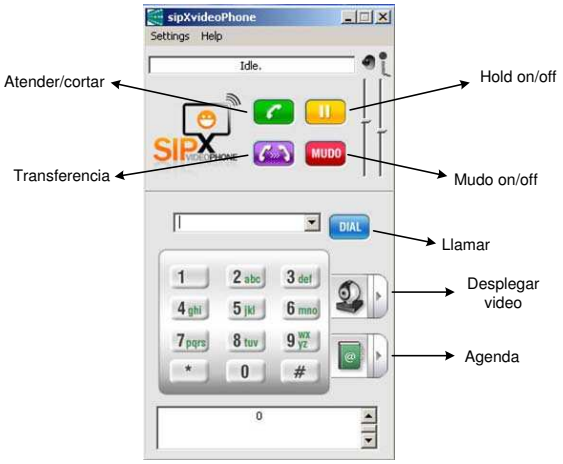


Figura D.5: Botones

D.4.2. Barra de estado

Debajo de la barra de menús existe un recuadro blanco que oficia de barra de estados donde a medida que cambia la situación en que se encuentra el teléfono la misma va cambiando su contenido. Los estados más comunes son:

***Idle*:** dice que el teléfono está en reposo

***Dialing*:** señala que se está discando

***Remote Alerting*:** indica que el teléfono con el cual se quiere establecer la comunicación está timbrando

***Accepted*:** significa que la llamada originada por este teléfono fue aceptada por el usuario llamado

***Connected*:** señala que existe una llamada en curso

***Busy*:** indica que el usuario que fue llamado tiene su linea telefónica ocupada

***sip:usuario@dominio*:** advierte que el teléfono con el URI sip:usuario@dominio quiere establecer una llamada con este teléfono. A la vez que el mensaje es desplegado este teléfono timbra para remarcar la situación

D.4.3. Barra de discado

La barra de discado es el recuadro blanco (pop-up) que se observa a la izquiera del botón azul que dice "DIAL". Este recuadro se utiliza cuando se quiere originar una llamada, de la siguiente forma:

- se escribe el URI de la persona con la cual se quiere hablar.
  Por ejemplo: sip:maria@192.168.1.100

  o

- se selecciona dentro de la lista histórica de URI's discados el deseado

Por lo general luego de haber hecho alguna de las dos cosas anteriores se procede a presionar el botón de llamar efectuándose la acción de llamar.

Apéndice E

SVN

E.1. Motivación

Para tener un mejor control de los avances del proyecto, implementamos un sistema de control de versiones. El objetivo fue tener un repositorio donde guardar las distintas versiones del programa que se fueran desarrollando, y que siempre existiese una única última versión sobre la cual trabajar.

A tales efectos, se consideró de gran utilidad montar un sistema de control de versiones SVN (*Subversion*). Esto consiste en instalar un servidor y los correspondientes clientes para poder acceder de manera adecuada.

E.2. Introducción a SVN

Subversion es un sistema de control de versiones de uso libre y código abierto. UTiliza el concepto de repositorio, que representa un lugar donde se almacena información en forma de versiones a través del tiempo. El repositorio es un servidor que almacena diferencias entre sucesivas versiones en el tiempo, y lleva un registro de cada vez que se realizaron modificaciones en su contenido. Estos cambios los registra con un número de versión. Este mecanismo permite recuperar versiones antiguas de información o también analizar las modificaciones realizadas entre versiones diferentes. Por este motivo, muchas veces estos sistemas con considerados *Máquinas del tiempo*.

Un servidor SVN puede ser accedido de forma remota a través de redes. Esto permite que la información almacenada en el repositorio pueda ser accedida por varias personas independientemente de dónde se encuentren físicamente. De esta manera se puede lograr el trabajo en conjunto a partir de los aportes realizados por cada uno de los usuarios que disponen de permisos para actualizar el contenido del repositorio.

E.3. Instalación del servidor SVN

El servidor SVN se montó sobre una plataforma Linux. Para ello previamente fue necesario instalar el sistema operativo Linux en un PC, y la plataforma elegida fue Ubuntu.

Para poder acceder de forma remota al servidor se eligió utilizar el protocolo SSH (*Secure Shell*), lo que llevó a tener que configurar el protocolo y crear los usuarios de acceso remoto. Una vez disponible y accesible el servidor Linux, se comenzó a instalar y crear el sistema SVN.

El montaje de un servidor SVN consta de dos partes:

1. Descargar e instalar todos los paquetes necesarios para crear una respositorio. Esto se lleva a cabo ejecutando el siguiente comando:

   ```
   apt-get install subversion
   ```

2. Crear el repositorio en el directorio deseado. Para esto se ejecutan los siguientes comandos:

   ```
   mkdir /home/repositorio
   svnadmin create /home/repositorio
   ```

3. Configuración y creación de usuarios SVN. Se creó un grupo de usuarios denominado "subversion"formado por tres usuarios (nosotras) con permisos tanto de lectura como de escritura en el repositorio.

   ```
   chgrp subversion repositorio -R
   chmod g+w
   ```

Con esto se creó en el servidor una directorio denominado repositorio"bajo el cual se encontró luego todo el control de versiones del proyecto sipXvideoPhone. El repositorio pertenece al grupo de usuarios "subversion.[el] cual dispone de los permisos adecuados para poder acceder y modificar el contenido del sistema.

Una vez creados los usuarios SVN, el grupo y los permisos adecuados ya se dispone del servidor SVN buscado. Para poder acceder al mismo se debieron instalar clientes SVN en cada una de las máquinas locales de trabajo.

E.4. El cliente SVN

Al momento de instalar el cliente SVN se eligió la aplicación TortoiseSVN, un cliente diseñado para plataformas Windows. Éste se utiliza en conjunto con el explorador de Windows, resultando sumamente amigable y fácil de utilizar.

Se procedió a instalar el cliente en cada una de las máquinas locales de trabajo, para lo que se descargó un archivo "installz se ejecutó lo siguiente:

`TortoiseSVN-1.4.1.7992-win32-svn-1.4.2.md5`

Al utilizar esta aplicación, se obtuvieron una gran cantidad de problemas de compatibilidad con el servidor SVN instalado. Se encontró que las versiones cliente y servidor eran diferentes y provocaban que el repositorio se dañara constantemente. Se estuvo un tiempo utilizando el TortoiseSVN pero dado que los problemas eran cada vez peores se consideró que sería mejor buscar otro cliente que se adecuara al servidor.

Se procedió a buscar otro cliente que funcionara correctamente con el servidor y se determinó que la mejor opción sería descargar la línea de comandos para SVN. Este es un cliente que se utiliza a través de la consola de comandos y desde allí se ejecutan los comandos necesarios.

En las máquinas de trabajo se instaló la versión 1.3 de *Command Line SVN for Windows*.

E.5. Comunicación entre cliente y servidor

Para acceder al servidor Linux el protocolo utilizado es SSH, para lo cual cada una de nosotras dispone de un usuario de acceso. Para acceder al repositorio se utilizó entonces SVN sobre SSH. Esto implica que la etapa de autenticación es realizada únicamente por los pares openSSH (cliente y servidor), y una vez autorizado el acceso se podrá pasar a utilizar el SVN. Con este entorno, no fue necesario configurar accesos en el propio SVN puesto que todo se regula con el SSH del sistema Linux.

Al utilizar SVN sobre SSH, las URLs para acceder al repositorio tienen la siguiente forma:

`svn+ssh://usuario@direccion_ip/path_al_repositorio_en_el_servidor`

E.6. Utilización del repositorio

El repositorio utiliza el concepto de *revisiones* para guardar registro de las diferentes versiones de la información que son controladas por el sistema. Al comienzo del proyecto, el repositorio almacena la información con el número de versión 1, y se va incrementando a medida que los usuarios registran cambios.

Para empezar a utilizar el repositorio hay que importar por primera vez la información hacia el servidor. Esto se realiza desde la máquina cliente con el comando `import`. Con esta instrucción, el cliente solicita la dirección URL del repositorio destino. Una vez especificada la dirección destino, el servidor responde con la solicitud de la clave de acceso. Si los permisos están correctamente configurados, inmediatamente se sube el código al repositorio señalado y queda registrado con el número de revisión 1.

Cuando un usuario desea trabajar sobre la última versión del proyecto deberá bajar el contenido del mismo del repositorio. Para ello, ejecuta el comando `checkout` y especificando la URL destino y el directorio de destino se guarda una copia en la máquina cliente. Sobre esta

copia local del proyecto el usuario puede trabajar, realizando las modificaciones que crea necesario. Al subir al servidor los cambios realizados, éstos quedarán almacenados bajo un nuevo número de revisión. Para esto se hace uso del comando `commit` disponible en los clientes SVN, y nuevamente especificando la URL del repositorio se logra almacenar la nueva información en el repositorio. Esta nueva versión se guarda con el número de revisión 2.

Básicamente, el mecanismo de uso del repositorio SVN es el explicado anteriormente. A continuación se describen brevemente los comandos más utilizados por el cliente SVN:

Imoprt: con este comando se pone por primera vez el proyecto a disposición del repositorio. Cuando se dispone de la versión original a subir al repsoitorio, ejecutando este comando se sube la revisión 1 al servidor SVN.

Checkout: este comando permite bajar una versión a la máquina cliente. Haciendo un `checkout` se pasa a disponer en el ambiente de trabajo (cliente) de la última revisión almacenada en el repositorio.

Commit: luego de hacer las modificaciones en la máquina cliente, se debe subir nuevamente el código al repositorio para registrar los cambios. Para esto se utiliza el comando `commit`. La nueva versión queda almacenada en el repositorio bajo un nuevo número de revisión (el número se incrementa en 1).

Update: con este comando se actualiza la copia de trabajo de la máquina local según la última revisión presente en el repositorio. Cada vez que es ejecutado, se descargan las diferencias entre la revisión de la copia de trabajo y la útlima revisión disponible en el servidor.

Add: Se utiliza para incorporar un nuevo archivo al control de versiones. Este comando tiene efectos sobre la copia de trabajo y para transmitir estos cambios al repositorio se debe ejecutar seguidamente el comando `commit`.

E.6.1. Recuperación del repositorio

Debido a las facilidades proporcionadas por el SVN siempre es posible volver a versiones anteriores del trabajo. Para esto se deacarga sobre una nueva copia de trabajo (`checkout`) el contenido de la revisión de interés y se continúa trabajndo sobre ella. Si bien esto es fácil de realizar, presenta un inconveniente. Se trata de que al relizar modificaciones sobre una revisión que no es la última almacenada en el repositorio, no es posible elevar esos cambios al servidor. Solamente se deben subir al repositorio aquellas modificaciones que hayan sido efectuadas sobre la última revisión disponible en el servidor. Por esta razón fue necesario crear un nuevo repositorio (para lo cual ya disponíamos de los comandos) y cargar (mediante un `import`) en el mismo la versión de interés que pasaría a ser la revisión número 1.

E.7. Sistema de respaldos

Para aprovechar mejor la utilización del SVN se implementó un sistema de respaldos. Este se basó en la funcionalidad que proporcionan los sistemas SVN con este propósito.

El SVN disponde de un comando `dump` que crea un archivo que contiene la toda la información presente en el repositorio. Este comando realiza una copia de la primer revisión del repositorio y para el resto copia solamente las diferencias. Como resultado obtiene un archivo tal que cuando se carga de nuevo un nuevo repositorio, se obtiene el mismo control de versiones original. Esto provee la posibilidad de tener disponible todo el control de versiones aún cuando se haya dañado el repositorio.

Para realizar los respaldos, se procedió a crear un ejecutable que realizara dos tareas:

- Generar, a través del comando `dump`, el archivo con el respaldo de todas las versiones almacenadas en el repositorio.
- Se crea una archivo comprimido con los datos obtenidos en el paso anterior. Estos archivos comprimidos se nombran con la fecha y hora en que se realiza el respaldo para así tener un mejor control de la información.

Con este mecanismo fue posible obtener respaldos de manera eficiente para utilizar en el caso que fuera necesario.

Apéndice F

CD del proyecto

Es la finalidad de este apéndice describir qué incluye el CD que se encuentra en la contratapa de esta documentación y los requerimientos del sistema a utilizar.

F.1. Contenido del CD

El contenido del CD es:

1. La presente decumentación en versión electrónica.

Ubicación:	/Documentacion
Archivos:	proyecto.pdf

2. La aplicación desarrollada durante el proyecto de fin de carrera, sipXvideoPhone

Ubicación:	/Telefono
Archivos:	sipXezPhone.exe
	sipXtapi.dll
	avcodec.dll
	avformat.dll
	pcre3.dll
	sipxezphone-config
	/res

3. El manual de usuario de la aplicación

   Ubicación: /Manual
   Archivos: Manual.pdf

4. Fuentes de la aplicación desarrollada

   Ubicación: /Fuentes

5. Documento que explica el procedimiento para crear el ambiente de desarrollo necesario para compilar las fuentes del proyecto.

   Ubicación: /Fuentes
   Archivo: instalacion.pdf

6. Archivo que indica el contenido del CD y cómo se utiliza

   Ubicación: .
   Archivo: leeme.txt

F.2. Requerimientos del sistema

Los requisitos que debe cumplir el sistema en el que se ejecute el CD son:

- Lector de CD
- Para leer los documentos: Acrobat Reader y editor de texto
- Para ejecutar la aplicación:
  - Windows XP Service Pack 2
  - 500 Mhz de procesador
  - 256 MB de memoria RAM
  - Tarjeta de sonido full-duplex
  - Tarjeta de video
  - Micrófono
  - Parlantes
  - Camara web