



FACULTAD DE INGENIERIA

Universidad de Buenos Aires

MAESTRÍA EN SISTEMAS EMBEBIDOS

MEMORIA DEL TRABAJO FINAL

Diseño e implementación de un videoteléfono sobre una plataforma ARM Cortex-A

Autor:

Ing. Ericson Joseph Estupiñan Pineda

Director:

Mg. Ing. Sergio Scaglia (FIUBA)

Jurados:

Dr. Ing. Ariel Lutenberg (FIUBA, CONICET)

Ing. Mariano Llamedo Soria (UTN-FRBA)

Esp. Ing. Marcelo E. Romeo (UNSAM, UTN-FRBA)

*Este trabajo fue realizado en las Ciudad Autónoma de Buenos Aires,
entre marzo de 2019 y abril de 2020.*

Resumen

En la presente memoria se describe el desarrollo de un sistema embebido de recepción de videollamadas realizado para la empresa SURIX S.R.L. La terminal embebida está construida sobre una plataforma ARM Cortex-A y se hace uso del proyecto Yocto para la generación y compilación de un sistema operativo personalizado junto con bibliotecas de código abierto glib, sofia-sip y gstreamer.

En el desarrollo del trabajo se aplicaron conocimientos en GNU/Linux para sistemas embebidos, protocolos de red, programación de aplicaciones con interfaz gráfica de usuario y uso de bibliotecas multimedia para el sistema embebido de código abierto.

Agradecimientos

Quiero expresar mi gratitud a mis padres, mis amigos, mi novia y mis tutores que de una u otra forma han ayudado a que este sueño se haga realidad.

Índice general

Resumen	III
1. Introducción General	1
1.1. Sistemas de video sobre IP	1
1.2. Descripción del trabajo realizado	2
1.3. Motivación	3
1.4. Objetivo y Alcance	3
1.5. Sistemas de video portería en el mercado	4
1.5.1. Commax	4
1.5.2. Sica	5
1.5.3. Dahua Technology	5
1.5.4. Hikvision Technology	6
1.5.5. Resumen de las características de los equipos	7
2. Introducción Específica	9
2.1. Descripción técnica	9
2.1.1. Terminal embebida	10
Interfaz de video	10
Interfaz de audio	10
2.1.2. Aplicación de videollamada	11
2.1.3. Teléfono móvil	11
2.1.4. Servidor PBX	11
2.1.5. Flujo de funcionamiento	11
2.2. Sistemas de videollamada o videoconferencia	11
2.2.1. Protocolo RTP	12
2.2.2. Codificación de audio	12
2.2.3. Codificación de video	12
2.3. Construcción sistema Linux embebido	13
2.3.1. Herramientas para la construcción de un sistema Linux embebido	13
2.3.2. Biblioteca multimedia gstreamer	13
2.4. Requerimientos	13
2.4.1. Asociados al sistema (RQ0XX)	13
2.4.2. Asociados a la terminal embebida (RQ1XX)	14
2.4.3. Asociados al servidor de videollamadas (RQ2XX)	14
2.4.4. Asociados al generador de videollamada (RQ3XX)	14
2.5. Planificación del trabajo realizado	14
3. Diseño e Implementación	17
3.1. Hardware	17
3.1.1. Placa de desarrollo	17
3.1.2. Compatibilidad de hardware	18
3.2. Software	18

3.2.1.	Diseño de aplicación de videollamada	18
	Lenguaje de programación C	18
	Biblioteca base de desarrollo	19
	Biblioteca de multimedia	19
	Biblioteca de SIP	19
	Aplicación multihilo	20
	Diagrama de flujo de la aplicación de videollamada	20
3.2.2.	Proyecto OpenSTLinux	20
	Preparar computadora	21
	Descarga e instalación del ecosistema de desarrollo	21
	Iniciar SDK	22
	Descargar código fuente Linux	22
	Compilar Linux Kernel	23
3.2.3.	Implementación del sistema operativo	23
	Agregar biblioteca Sofia SIP	24
3.2.4.	Implementación del protocolo SIP	25
3.2.5.	Implementación de gstreamer	26
3.2.6.	Funcionamiento de aplicación de videollamada	28
3.3.	Servidor en la nube	31
3.3.1.	Instalación de servidor Asterisk	32
3.3.2.	Configuraciones y recomendaciones	33
4. Ensayos y Resultados		35
4.1.	Escenario para ensayos y resultados	35
4.2.	Resultados del establecimiento de videollamada	35
4.3.	Resultados codificación H264	38
4.4.	Resultados codificación de audio G711	41
4.5.	Resultado final	45
4.6.	Análisis de resultados	45
5. Conclusiones		47
5.1.	Conclusiones generales	47
5.2.	Próximos pasos	47
Bibliografía		49

Índice de figuras

1.1.	Plataforma de desarrollo STM32MP157C-DK ¹	3
1.2.	Portero visor Commax CDV-43KM ²	4
1.3.	Portero visor Sica 915181 ³	5
1.4.	Video portero VTH5241DW ⁴	5
1.5.	Video portero DS-KH6320-WTE1	6
2.1.	Diagrama técnico del proyecto.	9
2.2.	Diagrama de flujo de funcionamiento.	12
2.3.	Diagrama AoN. Los nodos marcados en color rojo resaltan el camino crítico del proyecto	15
3.1.	Componentes de software de la aplicación de video llamada	19
3.2.	Diagrama de flujo de la aplicación de videollamada	20
3.3.	Contenido carpeta openstlinux-20-02-19	24
3.4.	Procesamiento de vídeo digitalizado	26
3.5.	Procesamiento de transmisión de audio digitalizado	28
3.6.	Procesamiento de recepción de audio digitalizado	28
3.7.	Aplicación de videollamada gsof	29
4.1.	Escenario para ensayos y toma de resultados.	36
4.2.	Flujo de paquetes SIP.	36
4.3.	Tramas SIP capturas con Wireshark.	37
4.4.	Mediciones de audio desde la terminal móvil a la terminal embebida	39
4.5.	Transmisión de video en H264	40
4.6.	Mediciones de audio desde la terminal móvil a la terminal embebida	42
4.7.	Mediciones de audio desde la terminal embebida a el teléfono móvil	43
4.8.	Transmisión de audio en G711	44
4.9.	Resultado final del trabajo.	45

Índice de Tablas

1.1.	Resumen de características de los equipos presentados	7
2.1.	Listado de tareas	16
3.1.	Comparación de placas de desarrollo	18
4.1.	Comparación con equipos en el mercado	46

Capítulo 1

Introducción General

En este capítulo se presenta al lector el objetivo del trabajo realizado, conceptos técnicos sobre los sistemas de videollamada y productos en el mercado que están relacionados con el prototipo desarrollado en esta memoria.

1.1. Sistemas de video sobre IP

Los sistemas de comunicaciones y la Internet han abierto el paso a nuevas tecnologías, Internet de las Cosas o IoT (*Internet of Things*). De la mano con los sistemas embebidos se desarrollan nuevas aplicaciones en diferentes áreas industriales y de consumo doméstico. En el área de los sistemas de video sobre IP (*Internet Protocol*, protocolo de internet), las aplicaciones de videollamadas en equipos de portería toman más presencia en el día a día de las personas, siendo una alternativa para edificios residenciales y de oficinas.

Los sistemas de IPTV (*TV over IP*, televisión sobre IP) son otra aplicación muy importante que a diferencia de la televisión analógica que transmite el video de forma analógica por medio de radio frecuencia y cable, se centra en la transmisión de programación televisiva sobre una red IP. Sumado a esto también están las aplicaciones de *streaming* de video. Algunos ejemplos son plataformas muy conocidas como: Netflix, Amazon Prime Video, HBO Now, entre otros.

Estas nuevas tecnologías han dado paso a la forma en que los servicios multimedia llegan a las personas. Así, ver contenido multimedia o recibir una videollamada en un celular móvil o en la computadora de otra parte del mundo es ahora algo habitual.

A continuación se incluyen algunas definiciones importantes para la comprensión del trabajo realizado:

- Terminal: dispositivo que el usuario usa para comunicarse con otro usuario.
- Señal de voz: ondas eléctricas producidas por un sensor de voz.
- Digitalización de señal de voz: proceso en el cual la señal de voz es transformada en datos binarios.
- Señal de video: datos digitales capturados por el sensor de video.
- Codificación de voz: proceso por el cual la señal de voz digitalizada es procesada por algoritmos para reducir el tamaño y/o cambio de formato de la información.

- Compresión de video: reducir el número de bits que son requeridos para representar una imagen de video.
- Transporte: proceso en el cual la señal de video y voz es transportada por la red de Internet.
- Servidor de conmutación: elemento de red encargado de recibir y redirigir los datos de voz entre dos usuario.
- Aplicación de videollamada: software que permite la recepción y/o transmisión de una videollamada.

1.2. Descripción del trabajo realizado

El trabajo aquí documentado es una terminal embebida con sistema operativo GNU/Linux con la capacidad de recibir una videollamada, que permite la comunicación con una terminal móvil con sistema operativo Android o iOS. Para este fin, se diseñó e implementó una terminal embebida basada en una plataforma con un procesador de la familia ARM Cortex-A, y se usó de una aplicación softphone y un servidor PBX (*Private Branch Exchange*, Central Privada Automática) Asterisk. Con estos elementos en conjunto se logra que el sistema embebido reciba una videollamada desde la terminal Android pasando por el servidor.

La terminal embebida fue diseñada e implementada con el objetivo de ser escalable a nivel de software y hardware. Con este fin se eligió como sistema operativo GNU/Linux. Se abstrajo la capa de hardware de la aplicación de software pero manteniendo la familia de procesadores. También se buscó que el proceso de desarrollo de nuevas funcionalidades sea rápido y escalable. Por esto se eligió como bibliotecas de desarrollo glib, gstreamer y sofia-sip.

La terminal embebida es la plataforma de desarrollo STM32MP157C-DK2 de la empresa STMicroelectronics [1]. En la figura 1.1 se ilustra este hardware. Fue elegida por sus características técnicas y herramientas de desarrollo que facilitan la construcción de los diferentes elementos de software necesarios para la terminal. Otras plataformas embebidas también son soportadas por este trabajo, estas son:

- NanoPI Fire 2A.
- Tinker Board S.
- Raspberry PI 3.

Para la terminal móvil se usa la aplicación Linphone [2] para teléfono móvil Android y como servidor PBX Asterisk, el cual se ha configurado para registro y videollamadas entre las terminales.

El funcionamiento del sistema en conjunto usa el protocolo SIP (*Session Initiation Protocol*, Protocolo de inicio de sesión) [3] para el establecimiento de una videollamada entre el usuario A (sistema embebido) y el usuario B (teléfono móvil). Así, el usuario B realizará una petición de videollamada con destino al usuario A, el cual contesta automáticamente y se inicia una sesión multimedia; el usuario A como sistema embebido muestra el video y la reproducción *full-duplex* de audio.



FIGURA 1.1: Plataforma de desarrollo STM32MP157C-DK¹.

1.3. Motivación

Los sistemas videollamadas actualmente están presentes en numerosas aplicaciones domésticas e industriales. Para la empresa SURIX S.R.L es importante mantener sus equipos de video portería competitivos y a la vanguardia de las necesidades de consumo. Por esta razón se ha desarrollado un prototipo de un video portero con sistema operativo GNU/Linux y protocolos de red estándar que amplíen las funcionalidades del producto.

1.4. Objetivo y Alcance

El objetivo del proyecto es diseñar e implementar un prototipo operativo de una terminal embebida que permita contestar una videollamada, con capacidad de visualización de video y reproducción del audio.

El trabajo que se presenta en este documento pretende ser una base conceptual y práctica para futuros proyectos y productos finales para la empresa SURIX S.R.L. Se listan los alcances el trabajo:

- Implementar una terminal embebida con capacidad de reproducción de video.
- Implementar una terminal embebida con captura y reproducción de audio.
- Procesar señales de audio y video, codificación y decodificación.
- Procesar señales de video en codificación H264.
- Procesar señales de audio en codificación G711.
- Enviar y recibir paquetes RTP (*Real-time Transport Protocol*).
- Implementar protocolo SIP (*Session Initiation Protocol*).

¹Imagen tomada de https://wiki.st.com/stm32mpu/wiki/Getting_started/STM32MP1_boards/STM32MP157C-DK2/Lef27s_start/Unpack_the_STM32MP157C-DK2_board

1.5. Sistemas de video portería en el mercado

En el mercado nacional argentino de la seguridad y el control de acceso se encuentran diversas soluciones y marcas, entre ellas SURIX S.R.L. Muchos de los productos ofrecidos son importados. Por esta razón la empresa SURIX S.R.L tiene un fuerte interés en ofrecer una solución competitiva en control de acceso con videollamada.

Este trabajo es un prototipo para un producto final; es por esta razón que este aparatado analiza los productos encontrados en el mercado, definiendo requerimientos y características que harían a un producto competitivo en el mercado.

1.5.1. Commax

Commax [4] es una empresa coreana con presencia en el mercado argentino que ofrece el kit de portero visor CDV-43KM, ver figura 1.2. Las siguientes son las características de este portero que implementa un sistema de videollamada:

- Pantalla táctil LCD de 4.3”.
- Altavoces para llamadas de acceso a puerta.
- Apertura remota de puerta.
- Soporte para un segundo monitor en caso que el monitor principal no atienda la videollamada.
- Almacenamiento de vídeo en memoria microSD.



FIGURA 1.2: Portero visor Commax CDV-43KM².

El precio del kit que se compone de un frente antivandalico y un monitor *indoor* esta alrededor de los \$ 250 dólares.

²Imagen tomada de <https://www.commax.com/eng/index.php?cate1=112&cate2=57&cate3=70&nnum=2180>

1.5.2. Sica

Sica [5] es una empresa argentina que ofrece una amplia gama de productos de seguridad y control de acceso, ofrece el portero visor Sica 915181 (ver figura 1.3). Las características de este producto son:

- Pantalla táctil LCD de 7".
- Altavoces para llamadas de acceso a puerta.
- Apertura remota de puerta.



FIGURA 1.3: Portero visor Sica 915181³.

1.5.3. Dahua Technology

Dahua Technology [6] es un proveedor de soluciones de videovigilancia con presencia en mas 180 países. Ofrece una amplia gama de vídeo porteros de los cuales se destacan sus productos IP; uno de estos productos es el VTH5241DW Wi-Fi Indoor Monitor (ver figura 1.4).



FIGURA 1.4: Video portero VTH5241DW⁴.

³Imagen tomada de <https://sicaelec.com/seguridad/>

⁴Imagen tomada de <https://www.dahuasecurity.com/la/products/All-Products/Video-Intercoms/IP-Products/VTH5241DW>

Los productos de intercomunicación de Dahua vienen con una versátil línea de soluciones de IP y WiFi, con protocolo SIP abierto y estándar que facilita la integración con terceros. Las siguientes son las características del video portero VTH5241DW:

- Sistema operativo LINUX embebido
- Protocolos RTSP, RTP.
- Codificación de video H264.
- Codificación de audio G711.
- Pantalla táctil LCD de 10".
- Conexión WiFi.
- Conexión Ethernet.
- Almacenamiento de video en memoria microSD.

1.5.4. Hikvision Technology

Hikvision Technology [7] es una corporación establecida en Shenzhen China, proveedor a nivel mundial de productos y soluciones de videovigilancia. Entre sus productos de video portero se encuentra DS-KH6320-WTE1 (ver figura 1.5).



FIGURA 1.5: Video portero DS-KH6320-WTE1⁵.

Este video portero que es una solución IP cuenta con las siguientes características:

- Sistema operativo LINUX embebido
- Protocolos SIP, RTSP, RTP.

⁵Imagen tomada de <https://www.hikvision.com/es-la/Products/Video-Intercom/Indoor-Station-/DS-KH6320-WTE1>

- Codificación de video H264.
- Codificación de audio G711.
- Pantalla táctil LCD de 7".
- Conexión WiFi.
- Conexión Ethernet.
- Almacenamiento de video en memoria microSD.

1.5.5. Resumen de las características de los equipos

En la tabla 1.1 se resumen las características de los equipos anteriormente mencionados. Se puede observar que los equipos de video portería basados en IP cumplen con las mismas características de los equipos de transmisión de video analógico y las amplían brindando protocolos de red estándar.

TABLA 1.1: Resumen de características de los equipos presentados

Característica	Commax	Sica	Dahua	Hikvision
Pantalla táctil	x	x	x	x
Altavoces para llamadas de acceso a puerta	x	x	x	x
Apertura remota de puerta	x	x	x	x
Almacenamiento de vídeo	x	x	x	x
GNU/LINUX			x	x
SIP			x	x
RTSP			x	x
RTP			x	x
H.264			x	x
G711			x	x
Conexión WiFi			x	x
Conexión Ethernet.			x	x

Capítulo 2

Introducción Específica

En este capítulo se presenta una visión global con una descripción más detallada y técnica del trabajo realizado, además de los requerimientos del proyecto, definición de tareas y planificación.

2.1. Descripción técnica

En la figura 2.1 se ilustra el diagrama técnico del proyecto. En este diagrama se pueden identificar las partes fundamentales del proyecto: terminal embebida, teléfono móvil y servidor PBX. El sistema embebido esta compuesto por una plataforma ARM Cortex-A, un sistema operativo GNU/LINUX personalizado, bibliotecas de multimedia gstreamer, biblioteca de protocolo SIP y por último una aplicación de videollamada. El teléfono móvil cuenta con una aplicación *softphone* y por último un servidor PBX.

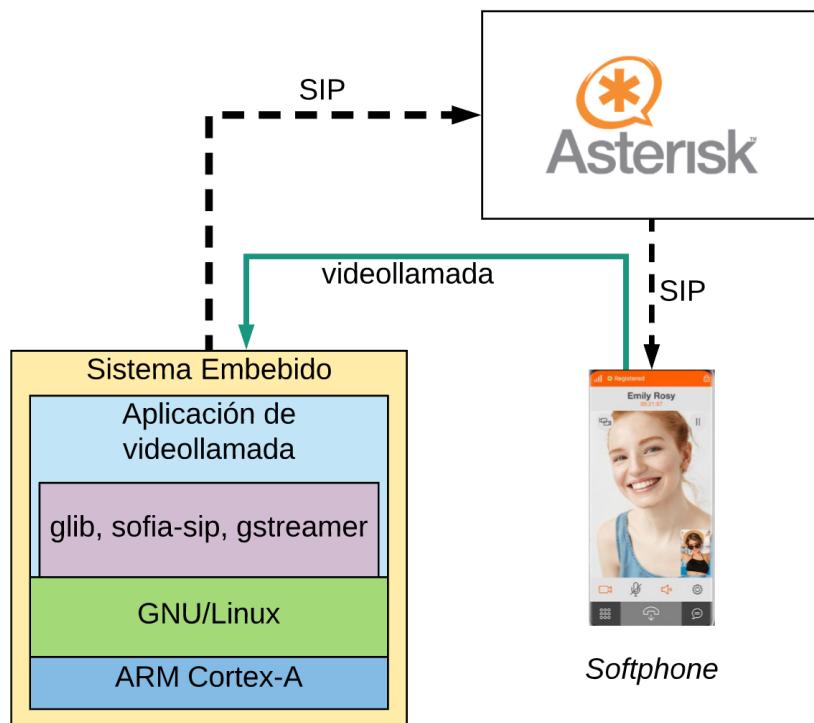


FIGURA 2.1: Diagrama técnico del proyecto.

A continuación, de forma más específica se expone el funcionamiento técnico del proyecto describiendo cada uno de los elementos de red y el flujo de funcionamiento del trabajo realizado.

2.1.1. Terminal embebida

Para el desarrollo de la terminal embebida se eligió la plataforma de desarrollo STM32MP157C-DK2 de la empresa STMicroelectronics [1]. Esta plataforma cuenta con las siguientes características:

- Procesador de dos núcleos, Cortex-A7 a 650 MHz y Cortex-M4 a 209 MHz, ambos de 32 bits.
- Memoria RAM 4 Gb DDR3L, 16 bits, 533 MHz.
- Conexión Ethernet de 1 Gbps.
- Controlador electrónico de audio (*Audio Codec*) con conector Jack de audio.
- Pantalla de 4 pulgadas TFT (*Thin-film-transistor*).
- Wi-Fi 802.11b/g/n.

De la plataforma de desarrollo se hace uso de la pantalla de 4 pulgadas, para mostrar la señal de video recibida en un videollamada, y del conector de audífonos. Estos periféricos son usados por medio del sistema operativo GNU/Linux y las bibliotecas de multimedia gstreamer [8] y se describen en los siguientes apartados.

Interfaz de video

La plataforma de desarrollo cuenta con una interfaz de video DSI (*Display Serial Interface*), que se utiliza para la conexión de una pantalla. Esta interfaz es especificada por la MIPI Allince (*Mobile Industry Processor Interface*) que es una organización global con el objetivo de diseñar y promover interfaces de hardware y software que simplifiquen la integración de los componentes integrados en un dispositivo, desde la antena y el módem hasta los periféricos y el procesador de la aplicación [9].

Otras interfaces de video pueden ser compatibles con el trabajo desarrollado con mínimos cambios, esto a razón de la utilización del sistema operativo GNU/Linux que integra diferentes controladores de video para numerosas plataformas.

Interfaz de audio

La plataforma de desarrollo cuenta con el circuito integrado CS42L51-CNZ de la empresa Cirrus Logic [10]. Este es un codificador de audio estéreo con amplificador de auriculares, se conecta al microcontrolador STM32MP157C por la interfaz SAI (*Serial Audio Interface*) y a los auriculares por medio de un conector Jack 3,5 mm.

Otras interfaces o codificadores de audio son soportados por la aplicación de videollamada de forma transparente, ya que se hace uso de la biblioteca de audio

ALSA (*Advanced Linux Sound Architecture*). Esta biblioteca provee un API (*Application Programming Interface*) que abstrae el uso de diferentes controladores de audio.

2.1.2. Aplicación de videollamada

Desarrollo de software en lenguaje C para el sistema operativo GNU/Linux que se encuentra instalada en la terminal embebida o sistema embebido. Su principal función es la recepción de una videollamada. Como se ilustra en la figura 2.1 hace uso de las bibliotecas, GLib [11], Sofia-SIP [12] y gstreamer [8].

2.1.3. Teléfono móvil

Equipo que tiene instalada una aplicación *softphone*. Softphone es un elemento de software que se utiliza para la realización de llamadas o videollamadas sobre IP, lo que se conoce como VoIP (*Voice over IP*, Voz sobre IP)[13]; este puede estar basado en el protocolo estándar SIP [3].

Para el trabajo realizado se instala aplicación Linphone en un teléfono inteligente Android, para más información sobre este *softphone* ver la siguiente referencia [2].

2.1.4. Servidor PBX

Como servidor PBX se elige Asterisk por estar ampliamente difundido en el mercado y ser una referencia mundial [14], se publica como software libre bajo la licencia GPL (*GNU General Public License*, Licencia Pública General de GNU) [GPL] y porque soporta protocolo SIP y videoconferencias. Para más información sobre este servidor PBX dirigirse a la siguiente referencia [15].

2.1.5. Flujo de funcionamiento

El flujo de funcionamiento del trabajo realizado se ilustra en la figura 2.2. En un primer paso las terminales tanto el teléfono móvil como la terminal embebida por medio de la aplicación de videollamada al ser activadas se registran ante el servidor PBX siguiendo el protocolo SIP. Como segundo paso el teléfono móvil realiza una videollamada con destino a la plataforma embebida. Como tercer paso la plataforma embebida contesta la videollamada de forma automática, se inicia el intercambio de multimedia y como ultimo y quinto paso el teléfono móvil termina la videollamada.

2.2. Sistemas de videollamada o videoconferencia

A continuación se exponen conceptos relacionados con sistemas de videollamada o videoconferencia. Éstos son necesarios en el diseño e implementación del proyecto y se aplican en el capítulo 3.

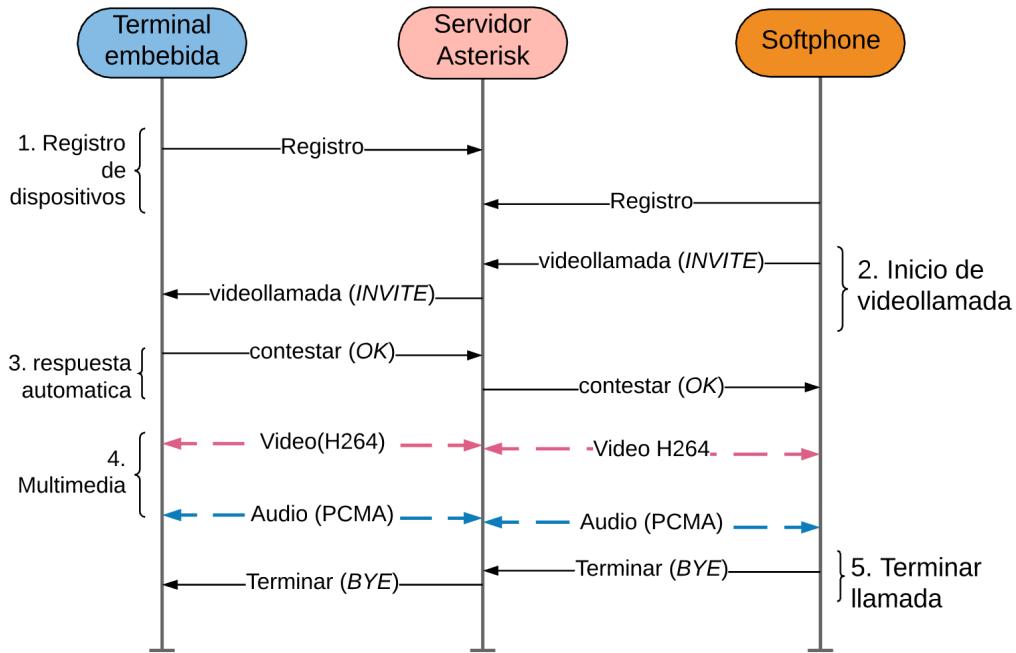


FIGURA 2.2: Diagrama de flujo de funcionamiento.

2.2.1. Protocolo RTP

RTP (*Real-Time Transport* [16], protocolo de transporte en tiempo real) es un protocolo de nivel de aplicación que proporciona una comunicación de extremo a extremo para datos con características de tiempo real, tales como audio y/o video interactivo. RTP es solo un protocolo de empaquetado que por medio de encabezados describe la información de su carga útil, para mas información sobre este protocolo ir a la referencia [17].

2.2.2. Codificación de audio

En el trabajo desarrollado se elige la codificación de audio G711 [18]. Es un estándar de la ITU (*International Telecommunications Union*) usado en la telefonía digital moderna. Cuenta con dos versiones A-law y U-law o PCMA y PCMU respectivamente. MU-law es originaria del estándar T1 en Norteamérica y Japon; mientras que, A-law es originaria del estándar E1 usado en el resto del mundo. G711 hace uso de una tasa de transferencia de 64 Kbits/s y es compatible con la mayoría de los proveedores de VoIP.

2.2.3. Codificación de video

MPEG-4 (*Moving Picture Experts Group*) [19] es un estándar creado por la ISO (*International Organization for Standardization*, Organización Internacional de Normalización) para la codificación de fuentes multimedia, esto corresponde a imágenes, audio y video. En este trabajo, para la codificación de video, se usa el protocolo H264, que corresponde a MPEG-4 Parte 10 o MPEG-4 AVC, todos estos son sinónimos de este formato de video.

2.3. Construcción sistema Linux embebido

En esta sección, se explica la construcción de un sistema operativo GNU/Linux personalizado, ya que es importante tener en cuenta la metodología y proceso de compilación de las fuentes del *Kernel* GNU/Linux.

2.3.1. Herramientas para la construcción de un sistema Linux embebido

Para la construcción de las herramientas necesarias para ejecutar un sistema operativo GNU/Linux en una terminal embebida se usó el proyecto Yocto y el *framework* OpenEmbedded. Los herramientas necesarias son:

- Compilador de código C para arquitectura del procesador.
- Sistema de arranque U-Boot
- *Kernel* GNU/Linux
- Sistema de archivos (*File System*)
- SDK (*software development kit*, kit de desarrollo de software)

El capítulo 3 se detallan los pasos para la construcción y despliegue de estos elementos de *software*.

2.3.2. Biblioteca multimedia gstreamer

Para el manejo de video en la plataforma embebida, el trabajo desarrollado hace uso de la biblioteca y entorno de trabajo *gstreamer* [8]. Es una biblioteca muy completa de código abierto bajo la licencia LGPL y soporta múltiples codificaciones de audio y video, de las cuales se destacan G711, G729, G722 para audio y H264, VP8 y VP9 para video.

Esta biblioteca se elige por su escalabilidad, y permite agregar nuevos codificadores de video o audio rápidamente. Está ampliamente utilizada y posee una comunidad de usuarios y desarrolladores muy activa.

Es importante mencionar que la biblioteca gstreamer debe ser compilada en el momento de la construcción del sistema operativo GNU/Linux personalizado.

2.4. Requerimientos

Para la realización de este proyecto, se debieron cumplir con los siguientes requerimientos:

2.4.1. Asociados al sistema (RQ0XX)

- RQ001: el sistema debe establecer una videollamada entre dos terminales, una plataforma embebida y un celular móvil Android.

2.4.2. Asociados a la terminal embebida (RQ1XX)

- *RQ101*: la terminal embebida debe recibir y contestar una videollamada sobre protocolo SIP.
- *RQ102*: la terminal embebida debe permitir la captura y reproducción de audio digital PCM 16 bits a 8 KHz.
- *RQ103*: la terminal embebida debe mostrar video en formato H.264 y/o VP8.
- *RQ104*: la terminal embebida debe permitir la codificación y decodificación de audio en el estándar G711.
- *RQ105*: la terminal embebida debe enviar y recibir datagramas RTP sobre el protocolo de transporte UDP.
- *RQ106*: la terminal embebida debe implementar protocolo SIP y registrarse en un servidor PBX Asterisk.
- *RQ107*: la terminal embebida debe permitir la conexión de salidas y/o entradas de audio. (ej: auriculares o altavoces).
- *RQ108*: la terminal embebida debe poder silenciar el micrófono.
- *RQ109*: la terminal embebida debe contar con una interfaz de usuario.
- *RQ110*: la terminal embebida debe tener la capacidad de decodificación de video H264 y/o VP8.
- *RQ111*: la terminal embebida debe integrar una cámara para la realización de video.
- *RQ112*: la terminal embebida debe tener la capacidad de codificación de vídeo H264 y/o VP8.

2.4.3. Asociados al servidor de videollamadas (RQ2XX)

- *RQ201*: Se requiere la instalación o uso de un servidor Asterisk con soporte para videollamadas con codificación de video H.264 ó VP8.

2.4.4. Asociados al generador de videollamada (RQ3XX)

- *RQ301*: Se necesita una aplicación softphone para teléfono inteligente con sistema operativo Android que tenga la funcionalidad de videollamada en formato de video H.264 ó VP8.

2.5. Planificación del trabajo realizado

Para el seguimiento del proyecto se utilizó el programa Trello¹ que ayudó a la medición de tiempo y orden de las tareas a realizar. La tabla 2.1 se lista las tareas y

¹<https://trello.com/>

la figura 2.3 ilustra el diagrama AoN (*Activity on node*) donde los nodos marcados en color rojo resaltan el camino crítico del proyecto.



FIGURA 2.3: Diagrama AoN. Los nodos marcados en color rojo resaltan el camino crítico del proyecto

TABLA 2.1: Listado de tareas

Tarea	Descripción
Tarea	Descripción
1	Gestión de proyecto
1.1	Definición de proyecto, alcances, requerimientos, WBS y plan de trabajo.
1.2	Presentación y exposición del proyecto.
2	Marco teórico
2.1	Búsqueda de bibliografía técnica
2.2	Organización y clasificación de información
2.3	Escritura de referencias técnicas del proyecto.
3	Hardware
3.1	Definición de plataforma Cortex-A
3.2	Definición de interfaz de audio.
3.3	Definición de interfaz de video.
3.4	Definición de display de video.
3.5	Definición de controles de llamada.
3.6	Conexiones y armado de prototipo.
4	Firmware
4.1	Programación de funciones de audio, captura y reproducción.
4.2	Programación de funciones de video, reproducción de video.
4.3	Definición de protocolo de codificación de audio.
4.4	Definición de protocolo de codificación de video.
4.5	Implementación y/o programación de codificadores de audio.
4.6	Implementación y/o programación de codificadores de video.
4.7	Programación de protocolo RTP.
4.8	Implementación o programación de protocolo SIP.
4.9	Programación de lógica de firmware.
4.10	Pruebas de funcionalidad.
5	Servidor
5.1	Instalación de servidor Asterisk
5.2	Configuración de servidor Asterisk para video llamadas.
6	Aplicación Softphone
6.1	Búsqueda de aplicación softphone con capacidad para videollamadas.
7	Validación y verificación
7.1	Planificación de pruebas de funcionamiento
7.2	Pruebas de campo
7.3	Validación y verificación
8	Memoria
8.1	Escritura de memoria
8.2	Preparación de presentación

Capítulo 3

Diseño e Implementación

En este capítulo se presenta el diseño e implementación del trabajo desarrollado, que se divide en tres partes: hardware, software y servidor en la nube.

3.1. Hardware

En esta sección se describen las razones por las que se elige la placa de desarrollo STM32MP157C-DK2 para la realización del trabajo. Además, los parámetros de diseño que el hardware debe tener para cumplir con los requerimientos mencionados en 2.4.

3.1.1. Placa de desarrollo

Para la realización de este trabajo se debía elegir un hardware con ciertos requerimientos. En la tabla 3.1 se hace una comparación de las plataformas que disponía la empresa SURIX S.R.L y las características que son necesarias para el trabajo. Como se puede observar se comparan tres plataformas: STM32MPU157C, NanoPI [20] y Tinker Board [21]. De estas placas de desarrollo, Tinker Board no cuenta con una pantalla táctil (esta se debe ser comprada por aparte) y NanoPI no soporta entrada y salida de audio, estas características son fundamentales para el desarrollo del proyecto. Por ultimo queda la placa de desarrollo STM32MP157C-DK2 que cumple con las especificaciones necesarias.

Es importante resaltar que como el resultado de este trabajo es un prototipo, es la razón por la que se elige usar un placa de desarrollo para la realización del proyecto. Adicionalmente, la placa de desarrollo elegida (STM32MP157C-DK2) cuenta con una amplia documentación para el desarrollo de software desde la implementación de un sistema Linux embebido hasta el desarrollo de software para la placa, esto puede ser encontrado en el siguiente link https://wiki.st.com/stm32mpu/wiki/Main_Page.

Las características que se requieren en la tabla 3.1 son necesarias para el desarrollo de un videoteléfono que funcione como un monitor de video portería IP como los que se describieron en 1.5.3 y 1.5.4. Características fundamentales son: un procesador ARM de la familia Cortex-A que incluye una MMU (*Memory Management Unit*) para la implementación de sistemas operativos modernos como GNU/Linux o Android, conexión Ethernet o WiFi para la conexión a Internet, salida y entrada de audio para altavoces y micrófono respectivamente.

TABLA 3.1: Comparación de placas de desarrollo

Características	STM32MP157C	Kit NanoPi	Tinker Board
Procesador ARM Cortex-A	x	x	x
Soporte sistema Linux embebido	x	x	x
Pantalla tactil	x		
Salida y entrada de audio	x		x
Conexión GPIO	x	x	x
Conexión Ethernet	x	x	x
Conexión WiFi (opt.)	x	x	x
Conexión bluetooth (opt.)	x		x

3.1.2. Compatibilidad de hardware

Es importante la implementación del sistema operativo GNU/Linux en la placa de desarrollo. Esto es a razón de que el trabajo realizado se espera que sea cambiada a otra plataforma diseñada por la empresa SURIX S.R.L. El diseño de hardware de esa plataforma se escapa de los alcances de este trabajo.

3.2. Software

Esta sección describe el desarrollo a nivel de software del trabajo, el cual comienza con la presentación del diseño de la aplicación de videollamada, instalación de la herramienta de software OpenSTLinux, implementaciones de los elementos de software y por ultimo la implementación del servidor en la nube.

3.2.1. Diseño de aplicación de videollamada

La aplicación de videollamada es un desarrollo de software para sistemas operativos GNU/Linux. En la figura 3.1 se ilustran los componentes de software que integran esta aplicación. Como nombre clave de la aplicación de videollamada se usa gsوف.

A continuación se explican cada uno de los lineamientos de diseño, y la elección de los componentes de software que componen la aplicación de video llamada.

Lenguaje de programación C

Al utilizarse un sistema operativo, se tiene la posibilidad de elegir un lenguaje de alto nivel orientado a objetos (Java, C++, Python, etc), pero la empresa SURIX S.R.L requiere portar código en C que ha sido desarrollado con anterioridad para

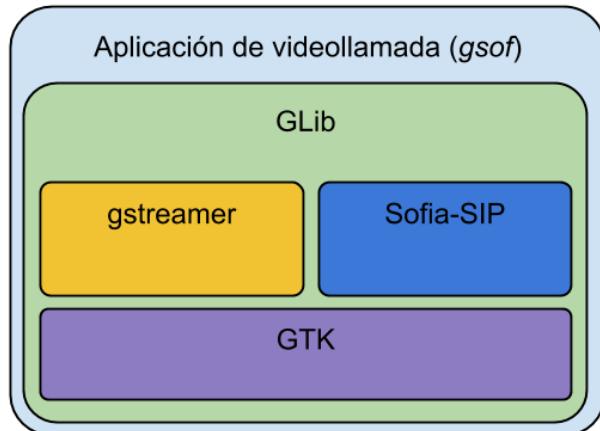


FIGURA 3.1: Componentes de software de la aplicación de video llamada

plataformas embebidas. Adicionalmente, se requiere que el lenguaje de programación sea eficiente en tiempo de ejecución. Por estas razones se elige para el desarrollo de la aplicación de videollamada el lenguaje de programación C.

Biblioteca base de desarrollo

Por sí solo, el lenguaje C no cuenta con bibliotecas avanzadas para el manejo de hilos, memoria, cadenas de caracteres, colas y control de errores. Es por esto que, para el desarrollo de la aplicación de videollamada, se hace uso de la biblioteca GLib. GLib proporciona los bloques de construcción de aplicaciones centrales para bibliotecas y aplicaciones escritas en C [11].

Biblioteca de multimedia

La aplicación de videollamada debe soportar la recepción y reproducción de video en formato H264, transmisión de audio capturado del micrófono y reproducción de audio recibido. Para esto se elige la biblioteca gstreamer, ya que incluye importantes elementos que son necesarios para el proyecto, como son: protocolo RTP, codificación de video H264 y codificación de audio G711. Adicionalmente, es una biblioteca con soporte para múltiples sistemas operativos, basada en GLib y extensible por plugins [22].

Biblioteca de SIP

Para la integración del protocolo SIP se hace uso de la biblioteca Sofia-SIP [12]. Se elige esta biblioteca sobre PJSIP ya que tiene una versión desarrollada sobre GLib y se centra solo en el protocolo SIP. Por otro lado PJSIP implementa protocolo RTP el cual ya es implementado con la biblioteca gstreamer.

Aplicación multihilo

La ampliación de videollamada debe ser una aplicación multihilo ya que en una videollamada debe mantener la reproducción de video, transmisión y recepción de audio y eventos de protocolo SIP. Para la implementación de multihilos se hace uso de *Threads* de la biblioteca GLib.

Diagrama de flujo de la aplicación de videollamada

En la figura 3.2 se ilustra el diagrama de flujo diseñado para el funcionamiento de la aplicación de videollamada. Los bloques se de color anaranjado son procesos que se ejecutan con el uso de la biblioteca Sofia-SIP, y los de color azul, por la biblioteca gstreamer. Cada proceso es ejecutado en un hilo diferente.

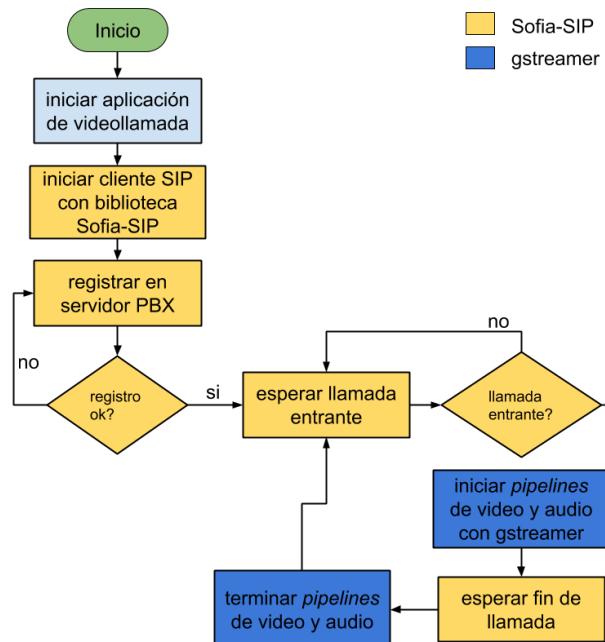


FIGURA 3.2: Diagrama de flujo de la aplicación de videollamada

3.2.2. Proyecto OpenSTLinux

Se usa el proyecto OpenSTLinux para la construcción del sistema operativo GNU/-Linux personalizado. Este proyecto recopila diferentes proyectos como Yocto, OpenEmbedded, Poky, BitBake y Metadata. En esta memoria se describen los pasos para la compilación del sistema operativo GNU/Linux con las librerías necesarias para la plataforma STM32MP157C-DK2. Se usa una distribución Ubuntu en la máquina *host*¹.

¹computadora donde se compila los códigos fuente para la plataforma objetivo

Preparar computadora

Para empezar se deben instalar los siguientes paquetes en el *host*:

```

1 $ sudo apt-get install sed wget curl cvs subversion git-core coreutils
    unzip texi2html texinfo docbook-utils gawk python-pyslirite2 diffstat
    help2man make gcc build-essential g++ desktop-file-utils chrpath
    libxml2-utils xmllt docbook bsdmainutils iutils-ping cpio python-
    wand python-pycryptopp python-crypto
2 $ sudo apt-get install libsdl1.2-dev xterm corkscrew nfs-common nfs-
    kernel-server device-tree-compiler mercurial u-boot-tools libarchive
    -zip-perl
3 $ sudo apt-get install ncurses-dev bc linux-headers-generic gcc-multilib
    libncurses5-dev libncursesw5-dev lrzsz dos2unix lib32ncurses5 repo
    libssl-dev
4 $ sudo apt-get install default-jre

```

Por defecto, en los sistemas Linux se permite un maximo de 8 particiones para un dispositivo MMC (*MultiMediaCard*), para cambiar esto ejecutar los siguientes comandos:

```

1 $ echo 'options mmc_block perdev_minors=16' > /tmp/mmc_block.conf
2 $ sudo mv /tmp/mmc_block.conf /etc/modprobe.d/mmc_block.conf

```

Descarga e instalación del ecosistema de desarrollo

Para descargar las herramientas de software necesarias se ejecutaron los siguientes comandos:

```

1 $ cd $HOME/STM32MPU_workspace/tmp
2 $ wget https://www.st.com/content/ccc/resource/technical/software/
    sw_development_suite/group0/51/d4/96/18/27/ab/49/93/
    stm32mp1dev_yocto_sdk/files/SDK-x86_64-stm32mp1-openstlinux
    -20-02-19.tar.xz/jcr:content/translations/en.SDK-x86_64-stm32mp1-
    openstlinux-20-02-19.tar.xz
3 $ tar xvf en.SDK-x86_64-stm32mp1-openstlinux-20-02-19.tar.xz

```

Posterior a la descarga se ejecutaron los siguientes comandos de instalación:

```

1 $ mkdir -p $HOME/STM32MPU_workspace/STM32MP1-Ecosystem-v1.2.0/Developer
    -Package/SDK
2 $ chmod +x $HOME/STM32MPU_workspace/tmp/stm32mp1-openstlinux-20-02-19/
    sdk/st-image-weston-openstlinux-weston-stm32mp1-x86_64-toolchain
    -2.6-openstlinux-20-02-19.sh
3 $ $HOME/STM32MPU_workspace/tmp/stm32mp1-openstlinux-20-02-19/sdk/st-
    image-weston-openstlinux-weston-stm32mp1-x86_64-toolchain-2.6-
    openstlinux-20-02-19.sh -d /home/stm32mpu/STM32MPU_workspace/
    STM32MP1-Ecosystem-v1.2.0/Developer-Package/SDK/

```

Iniciar SDK

El SDK (*software development kit*) son todos los componentes de software que se necesitan para iniciar la construcción del sistema operativo GNU/Linux personalizado; para iniciar se ejecutan los siguientes comandos:

```

1 $ cd $HOME/STM32MPU_workspace/STM32MP15-Ecosystem-v1.2.0 / Developer-
    Package
2 $ source SDK/environment-setup-cortexa7t2hf-neon-vfpv4-ostl-linux-
    gnueabi

```

Se revisa que las variables de entorno, el compilador y la versión de SDK se hayan configurado correctamente:

```

1 $ echo $ARCH
2 $ echo $CROSS_COMPILE
3 $ $CC --version
4 $ echo $OECORE_SDK_VERSION

```

Descargar código fuente Linux

En este momento se tiene todas las herramientas de software para poder compilar una aplicación en la computadora *host* y que esta se ejecute en la plataforma STM32MP157C-DK2 con un sistema operativo GNU/Linux por defecto. Para descargar el código fuente de Linux se ejecutan los siguientes comandos:

```

1 $ wget https://www.st.com/content/ccc/resource/technical/sw-updater/
    firmware2/group0/73/ca/80/91/b4/13/45/f8/
    STM32cube_Standard_A7_BSP_components_kernel/files/SOURCES-kernel-
    stm32mp1-openstlinux-20-02-19.tar.xz/jcr:content/translations/en.
    SOURCES-kernel-stm32mp1-openstlinux-20-02-19.tar.xz
2 $ cd $HOME/STM32MPU_workspace/STM32MP15-Ecosystem-v1.2.0 / Developer-
    Package
3 $ tar xvf en.SOURCES-kernel-stm32mp1-openstlinux-20-02-19.tar.xz
4 $ cd stm32mp1-openstlinux-20-02-19/sources/arm-ostl-linux-gnueabi/linux-
    stm32mp-4.19-r0
5 $ tar xvf linux-4.19.94.tar.xz

```

Para que se pueda compilar el código fuente de Linux sin errores es necesario aplicar los siguientes cambios:

```

1 $ cd linux-4.19.94/
2 $ for p in `ls -1 ../*.patch`; do patch -p1 < $p; done
3 $ make ARCH=arm multi_v7_defconfig "fragment*.config"
4 $ for f in `ls -1 ../fragment*.config`; do scripts/kconfig/merge_config.
    sh -m -r .config $f; done
5 $ yes '' | make ARCH=arm oldconfig

```

Compilar Linux Kernel

Para compilar por primera vez el Kernel y los programas necesarios se deben ejecutar los siguientes comandos. Esto puede llegar a tomar varios minutos:

```

1 $ make ARCH=arm uImage vmlinux dtbs LOADADDR=0xC2000040
2 $ make ARCH=arm modules
3 $ mkdir -p $PWD/install_artifact/
4 $ make ARCH=arm INSTALL_MOD_PATH="$PWD/install_artifact" modules_install

```

3.2.3. Implementación del sistema operativo

En la sub-sección 3.2.2, se describe la forma de como trabajar con las herramientas de software para generar una imagen del sistema operativo GNU/Linux para la plataforma STM32MP157C-DK2. En esta sección se describe como generar el SDK para compilar una imagen personalidad del sistema operativo GNU/Linux.

El primer paso es ejecutar los siguientes comandos para iniciar el paquete de distribución; este paquete esta basado en el proyecto OpenEmbedded.

```

1 $ mkdir $HOME/STM32MPU_workspace/STM32MP15-Ecosystem-v1.2.0 / Distribution
   -Package
2 $ cd $HOME/STM32MPU_workspace/STM32MP15-Ecosystem-v1.2.0 / Distribution-
   Package
3 $ mkdir openstlinux-20-02-19
4 $ cd openstlinux-20-02-19
5 $ sudo apt-get install repo
6 $ repo init -u https://github.com/STMicroelectronics/oe-manifest.git -b
   refs/tags/openstlinux-20-02-19
7 $ repo sync

```

Los comandos anteriores pueden tomar varios minutos para completarse. Como resultado en la figura 3.3 se detalla el contenido de la capeta openstlinux-20-02-19.

Una vez se haya iniciado el repositorio se inician las variables de entorno con el siguiente comando:

```

1 $ DISTRO=openstlinux-weston MACHINE=stm32mp1 source layers/meta-st/
   scripts/envsetup.sh

```

El comando anterior habilita la herramienta bitbake para la modificación y generación de un nuevo SDK y/o una nueva imagen del sistema operativo. Para generación de una nueva imagen con la configuración por defecto ejecutar el comando:

```

1 $ bitbake st-image-weston

```

```

openstlinux-20-02-19 OpenSTLinux distribution
└── layers
    ├── meta-openembedded          Collection of layers for the OpenEmbedded-Core universe (OpenEmbedded standard)
    └── meta-st
        ├── meta-st-openstlinux      STMicroelectronics layer that contains the frameworks and images settings for the OpenSTLinux distribution
        ├── meta-st-stm32mp          STMicroelectronics layer that contains the description of the BSP for the STM32 MPU devices
        ├── recipes-bsp
            ├── alsas                Recipes for ALSA control configuration
            ├── drivers               Recipes for Vivante GCNANO GPU kernel drivers
            ├── trusted-firmware-a    Recipes for TF-A
            └── u-boot                Recipes for U-Boot
        ├── recipes-extended
            ├── linux-examples        Recipes for Linux examples for STM32 MPU devices
            ├── m4coredump             Recipes for script to manage coredump of cortexM4
            └── m4projects              Recipes for firmware examples for Cortex M4
        ├── recipes-graphics
            ├── gcnano-userland       Recipes for Vivante libraries OpenGL ES, OpenVG and EGL (multi backend)
            └── [...]
        ├── recipes-kernel
            ├── linux                Recipes for Linux kernel
            └── linux-firmware         Recipes for Linux firmwares (example, Bluetooth firmware)
        ├── recipes-security
            └── optee                 Recipes for OPTEE
        ├── recipes-st
            └── images                Recipes for the bootfs and userfs partitions binaries
            └── [...]
        ├── meta-st-stm32mp-addons   STMicroelectronics layer that helps managing the STM32CubeMX integration
        ├── scripts
            └── envsetup.sh           Environment setup script for Distribution Package
            └── [...]
        ├── meta-timesys             Timesys layer for OpenEmbedded (standard)
        └── openembedded-core        Core metadata for current versions of OpenEmbedded (standard)

```

FIGURA 3.3: Contenido carpeta openstlinux-20-02-19

Agregar biblioteca Sofia SIP

El SDK que se genera por defecto contiene las siguientes bibliotecas que son necesarias para la creación de la ampliación de videollamada:

- gio-2.0
- gstreamer-1.0
- gstreamer-video-1.0
- x11
- gtk+-3.0

A parte de las bibliotecas anteriormente mencionadas es necesario disponer de la biblioteca Sofia-SIP (sofia-sip-ua-glib). Para poder agregar esta biblioteca es necesario crear y agregar un nuevo *layer*² al proyecto Yocto para generar un nuevo SDK y a su vez una nueva imagen del sistema operativo GNU/Linux.

Para la integración de la biblioteca Sofia-SIP se busca el correspondiente *layer* en <https://layers.openembedded.org/layerindex/branch/master/layers/>. Seguidamente se ejecutan los siguientes comandos para agregar el nuevo *layer*:

```

1 $ cd layers/meta-st
2 $ git clone git://git.osmocom.org/meta-telephony

```

El *layer* que se agregó es meta-telephony. Este *layer* tiene bibliotecas que no son necesarias por lo que se pueden retirar siguiendo los siguientes comandos:

```

1 $ cd layers/meta-st/meta-telephony
2 $ rm -rf recipes-devtools

```

²en el proyecto Yocto un *layer* es un conjunto de archivos que describen la integración de una aplicación o bibliotecas

```

3 $ rm -rf recipes-devtools
4 $ rm -rf recipes-telephony

```

Una vez realizados estos pasos se agrega el nuevo *layer* con los siguientes comandos:

```

1 $ cd ~/STM32MPU_workspace/STM32MP15-Ecosystem-v1.2.0 / Distribution-
    Package/openstlinux-20-02-19/build-openstlinuxweston-stm32mp1
2 $ bitbake-layers add-layer ../layers/meta-st/meta-telephony

```

Posteriormente a esto se debe generar de nuevo el SDK, con el siguiente:

```
1 $ bitbake -c populate_sdk st-image-weston
```

Y una nueva imagen con el comando:

```
1 $ bitbake st-image-weston
```

3.2.4. Implementación del protocolo SIP

La implementación del protocolo SIP hace uso de la biblioteca Sofia-SIP. Esta biblioteca es de código abierto, cumple con la especificación IETF RFC3261 [3] y es basada en un desarrollo realizado por Nokia Research Center [12].

Sofia-SIP esta disponible para su instalación en los repositorios de diferentes distribuciones GNU/Linux o también su código fuente puede ser descargado desde la url <http://sofia-sip.sourceforge.net/download.html>. Por ejemplo para la distribución Ubuntu se puede instalar con el comando:

```
1 $ sudo apt install libsofia-sip*
```

Posterior a la instalación para el uso de la misma en el software de la aplicación de videollamada se recomienda usar el siguiente comando:

```
1 $ pkg-config --libs sofia-sip-ua-glib
```

Como resultado del comando se obtienen las bibliotecas que el compilador debe usar.

```
1 -lsofia-sip-ua-glib -lsofia-sip-ua -lglib-2.0
```

3.2.5. Implementación de gstreamer

En esta sección se explica la implementación del modulo de audio y video de la aplicación de videollamada. Se hizo uso de GStreamer, una biblioteca *open source* para el manejo de componente de multimedia [8]. Esta biblioteca puede ser encontrada en la mayoría de paquetes de distintas distribuciones GNU/Linux, Windows, Mac OS X, Android y iOS.

Por ejemplo, para la distribución Ubuntu se puede instalar con el comando:

```
1 $ sudo apt-get install libgstreamer1.0-0 gstreamer1.0-plugins-base
    gstreamer1.0-plugins-good gstreamer1.0-plugins-bad gstreamer1.0-
    plugins-ugly gstreamer1.0-libav gstreamer1.0-doc gstreamer1.0-tools
    gstreamer1.0-x gstreamer1.0-alsa gstreamer1.0-gl gstreamer1.0-gtk3
    gstreamer1.0-qt5 gstreamer1.0-pulseaudio
```

Posterior a la instalación para el uso de la misma en el software de la aplicación de videollamada se recomienda usar el siguiente comando:

```
1 $ pkg-config --cflags --libs gstreamer-1.0
```

Como resultado del comando se obtienen las bibliotecas que el compilador debe usar.

```
1 -pthread -I/usr/include/gstreamer-1.0 -I/usr/include/glib-2.0 -
    gstreamer-1.0 -lgobject-2.0 -lglib-2.0
```

Siguiendo las sugerencias de implementación de la biblioteca GStreamer, en la figura 3.4 se ilustra el *pipeline* que es implementado para el procesamiento de video desde la recepción en protocolo RTP hasta su reproducción.

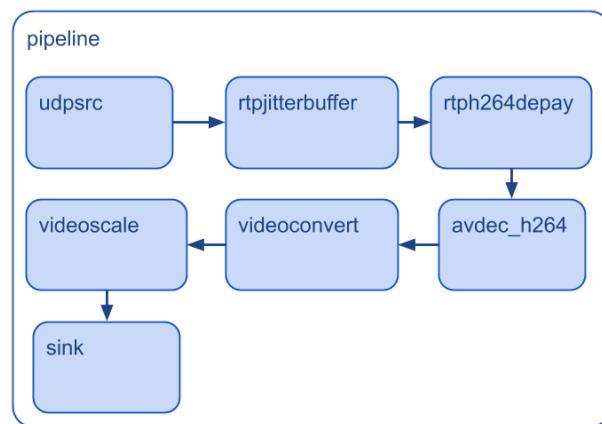


FIGURA 3.4: Procesamiento de vídeo digitalizado

El *pipeline* ilustrado en la figura 3.4 se compone de los siguientes objetos:

1. **udpsrc**: primer objeto del *pipeline* destinado a la apertura del socket UDP y recepción de paquetes RTP de vídeo, para mas información ver <https://gstreamer.freedesktop.org/documentation/udp/udpsrc.html?gi-language=c>.

2. **rtpjitterbuffer**: objeto destinado a almacenar en un *buffer* una cantidad de paquetes para mitigar el efecto de *jitter*³ en la red, para mas información ver <https://gstreamer.freedesktop.org/documentation/rtpmanager/rtpjitterbuffer.html?gi-language=c>.
3. **rtpH264Depay**: objeto que extrae la carga útil de los paquetes RTP, en este caso datos de vídeo codificados en H.264, para mas información ver <https://gstreamer.freedesktop.org/documentation/rtp/rtpH264Depay.html?gi-language=c>.
4. **avdecH264**: objeto que decodifica los datos de vídeo digitalizado, para mas información ver https://gstreamer.freedesktop.org/documentation/libav/avdec_h264.html?gi-language=c.
5. **videoconvert**: objeto que prepara los datos de vídeo para que puedan ser transformados, para mas información ver <https://gstreamer.freedesktop.org/documentation/videoconvert/index.html?gi-language=c>.
6. **videoscale**: objeto que aplica un escalamiento a los datos de vídeo, para mas información ver <https://gstreamer.freedesktop.org/documentation/videoscale/index.html?gi-language=c>.
7. **sink**: objeto que reproduce el video mostrándolo en pantalla.

La programación de audio se implementó siguiendo los *pipeline* que se ilustran en la figuras 3.5 y 3.6 para la transmisión y recepción de audio respectivamente. La transición de audio tiene los siguientes objetos:

1. **alsasrc**: primer objeto en la transmisión de audio encargado de la lectura de datos de la tarjeta de audio usando ALSA (*Advanced Linux Sound Architecture*), para mas información ir a <https://gstreamer.freedesktop.org/documentation/alsa/alsasrc.html?gi-language=c>.
2. **audioconvert**: objeto que prepara los datos de audio para que puedan ser transformados, para mas información ver <https://gstreamer.freedesktop.org/documentation/audioconvert/index.html?gi-language=c>.
3. **audioresample**: objeto que cambia la frecuencia de muestreo de los datos de audio, para mas información ver <https://gstreamer.freedesktop.org/documentation/audioresample/index.html?gi-language=c>.
4. **mulawenc**: objeto que comprime los datos de audio en formato G711 PC-MU, para mas información ver <https://gstreamer.freedesktop.org/documentation/mulaw/mulawenc.html?gi-language=c>.
5. **rtppcmupay**: objeto que encapsula los datos de audio comprimidos en paquetes RTP, para mas información ver <https://gstreamer.freedesktop.org/documentation/rtp/rtppcmupay.html?gi-language=c>.
6. **udpsink**: objeto que crea un *socket* UDP y envía los paquetes RTP, para más información ver <https://gstreamer.freedesktop.org/documentation/udp/udpsink.html?gi-language=c>.

La recepción de audio tiene los siguientes elementos:

³variación en el tiempo de retardo en milisegundos (ms) entre paquetes de datos a través de una red

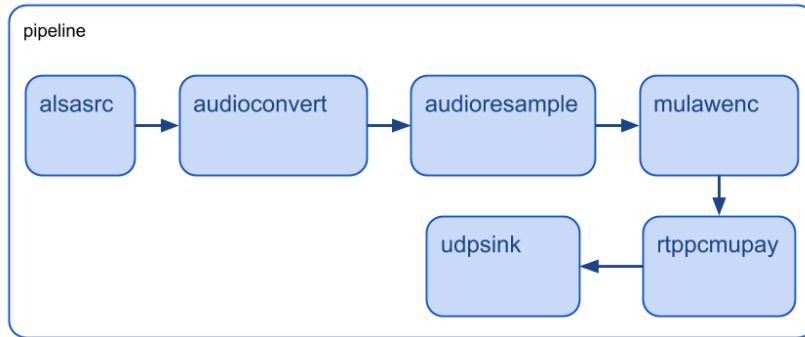


FIGURA 3.5: Procesamiento de transmisión de audio digitalizado

1. **udpsrc**: primer objeto del *pipeline* destinado a la apertura del socket UDP y recepción de paquetes RTP de audio.
2. **rtpjitterbuffer**: objeto destinado a almacenar en un *buffer* una cantidad de paquetes para mitigar el efecto de *jitter*.
3. **rtppcmudepay**: objeto que extrae los datos de audio comprimido en PCMU de los paquetes RTP, para mas información ver <https://gstreamer.freedesktop.org/documentation/rtp/rtppcmudepay.html?gi-language=c>.
4. **mulawdec**: objeto que descomprime los datos de audio digitalizado, para mas información ver <https://gstreamer.freedesktop.org/documentation/mulaw/mulawdec.html?gi-language=c>.
5. **autoaudiosink**: objeto que reproduce el audio, para mas información ver <https://gstreamer.freedesktop.org/documentation/autodetect/autoaudiosink.html?gi-language=c>.

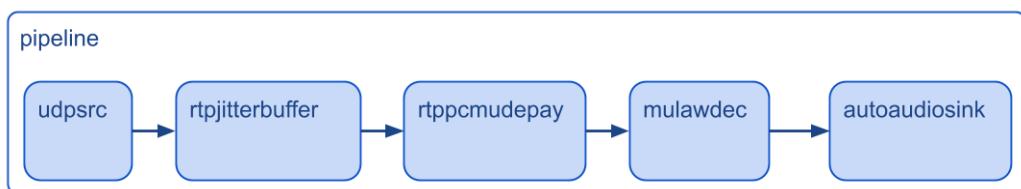


FIGURA 3.6: Procesamiento de recepción de audio digitalizado

3.2.6. Funcionamiento de aplicación de videollamada

La aplicación de videollamada es una ampliación de escritorio que se ejecuta desde la terminal con el siguiente comando:

1 gsof

El comando lanza una ventana la cual esta lista para recibir una videollamada y mostrar el video correspondiente. En la figura 3.7 se ilustra la aplicación de videollamada con el nombre de proyecto: gsof; esta aplicación abre una ventana que ocupa toda la pantalla, dentro de esta ventana hay un *layout* principal para



FIGURA 3.7: Aplicación de videollamada gsof

la reproducción de video (a) y un campo de texto en la parte inferior en el que se muestran mensajes de la aplicación (b).

Al momento de ejecutar la aplicación en la salida estándar *stdout* se muestran los mensajes de ejecución. Estos mensajes ayudan para hacer seguimiento y control de errores. A continuación se muestra un segmento de estos mensajes cuando se recibe una videollamada resaltando la negociación SDP (*Session Description Protocol*).

```

su_source_port_create() returns 0x556285be7500
    ::tag_null: 0
    ::tag_null: 0
REGISTER response 401 Unauthorized
    ::tag_null: 0
REGISTER response 200 OK
INDICATION INVITE
    ::tag_null: 0

===== Audio Address: 186.182.120.213:16850 =====

===== Video Address: 186.182.120.213:54000 =====
INDICATION STATE
    nua::callstate: 5
    soa::active_audio: 0
    soa::active_video: 0
    soa::active_image: 0
    soa::active_chat: 0
    nua::offer_recv: true
    soa::remote_sdp: v=0
o=root 1428681114 1428681114 IN IP4 45.77.160.236
s=Asterisk PBX 14.7.5
c=IN IP4 45.77.160.236
b=CT:384
t=0 0
m=audio 16420 RTP/AVP 8 0 3 101

```

```

a=rtpmap:8 PCMA/8000
a=rtpmap:0 PCMU/8000
a=rtpmap:3 GSM/8000
a=rtpmap:101 telephone-event/8000
a=fmtp:101 0-16
a=maxptime:150
m=video 16422 RTP/AVP 99 34 100
a=rtpmap:99 H264/90000
a=fmtp:99 profile-level-id=42801F
a=rtpmap:34 H263/90000
a=rtpmap:100 VP8/90000
a=rtcp-fb:* ccm fir
    soa::remote_sdp_str: "v=0
o=root 1428681114 1428681114 IN IP4 45.77.160.236
s=Asterisk PBX 14.7.5
c=IN IP4 45.77.160.236
b=CT:384
t=0 0
m=audio 16420 RTP/AVP 8 0 3 101
a=rtpmap:8 PCMA/8000
a=rtpmap:0 PCMU/8000
a=rtpmap:3 GSM/8000
a=rtpmap:101 telephone-event/8000
a=fmtp:101 0-16
a=maxptime:150
m=video 16422 RTP/AVP 99 34 100
a=rtpmap:99 H264/90000
a=fmtp:99 profile-level-id=42801F
a=rtpmap:34 H263/90000
a=rtpmap:100 VP8/90000
a=rtcp-fb:* ccm fir
"
    ::tag_null: 0
INDICATION STATE
    nua::callstate: 7
    soa::active_audio: 3
    soa::active_video: 3
    nua::answer_sent: true
    soa::local_sdp: v=0
o=- 2600643136511232947 364405289222003252 IN IP4 192.168.100.3
s=-
c=IN IP4 186.182.120.213
t=0 0
m=audio 16850 RTP/AVP 0
a=rtpmap:0 PCMU/8000
m=video 54000 RTP/AVP 99
a=rtpmap:99 H264/90000
    soa::local_sdp_str: "v=0
o=- 2600643136511232947 364405289222003252 IN IP4 192.168.100.3
s=-
c=IN IP4 186.182.120.213

```

```

t=0 0
m=audio 16850 RTP/AVP 0
a=rtpmap:0 PCMU/8000
m=video 54000 RTP/AVP 99
a=rtpmap:99 H264/90000
"
    ::tag_null: 0
gsofsip UA:
Not implemented yet event 'nua_i_ack' (3): [200] OK
    ::tag_null: 0
INDICATION STATE
    nua::callstate: 8
    soa::active_audio: 3
    soa::active_video: 3
    ::tag_null: 0

----- MEDIA -----
gsofsip UA:
Not implemented yet event 'nua_i_bye' (9): [200] Session Terminated
    ::tag_null: 0
INDICATION STATE
    nua::callstate: 10
    soa::active_audio: 3
    soa::active_video: 3
    ::tag_null: 0
INDICATION TERMINATED
TERMINATED
    ::tag_null: 0
TX End-Of-Stream reached.
RX End-Of-Stream reached.

```

Es importante resaltar que el resultado de la negociación SDP es un flujo de multimedia con audio en codificación G711 y vídeo en H264.

3.3. Servidor en la nube

En este apartado se describe la instalación y puesta en marcha del proyecto Asterisk en un servidor.

Se destina un servidor VPS (*virtual private server*) contratado del portal Vultr⁴, a continuación se listan sus características:

- Nombre de dominio: sip.ericsonj.net.
- Sistema operativo: CentOS.
- Memoria RAM: 512 MB.
- Espacio en disco: 20 GB.
- Número de núcleos: 1.

⁴<https://www.vultr.com/>

- Localización: Miami, Estados Unidos.

3.3.1. Instalación de servidor Asterisk

A continuación se listan los comandos que se deben introducir en la consola CentOS para la instalación:

1. Descargar Asterisk y librerías

```

1  cd /usr/src/
2  wget http://downloads.asterisk.org/pub/telephony/libpri/libpri-
   -1.6.0.tar.gz
3  wget http://downloads.asterisk.org/pub/telephony/dahdi-linux-
   -complete/dahdi-linux-complete-2.11.1-rc1+2.11.1-rc1.tar.gz
4  wget http://downloads.asterisk.org/pub/telephony/asterisk/
   asterisk-14.3.0.tar.gz

```

2. Instalar paquetes necesarios

```

1  yum install kernel-devel
2  yum install gcc
3  yum install gcc-c++
4  yum install ncurses-devel.x86_64
5  yum install libuuid-devel.x86_64
6  yum groupinstall "Development Tools"
7  yum install libxml2-devel.x86_64
8  yum install sqlite-devel
9  yum install git

```

3. Instalar jansson

```

1  git clone https://github.com/akheron/jansson.git
2  cd jansson/
3  autoreconf -i
4  ./configure
5  make
6  make install

```

4. Instalar Asterisk

```

1  cd ..
2  tar zxvf asterisk-14.3.0.tar.gz
3  tar zxvf dahdi-linux-complete-2.11.1-rc1+2.11.1-rc1.tar.gz
4  tar zxvf libpri-1.6.0.tar.gz
5  cd dahdi-linux-complete-2.11.1-rc1+2.11.1-rc1
6  uname -r
7  make KVERS=3.10.0-514.6.2.el7.x86_64
8  cd ../libpri-1.6.0
9  make
10 ls
11 cd ../dahdi-linux-complete-2.11.1-rc1+2.11.1-rc1
12 ls
13 make install
14 make install KVERS=3.10.0-514.6.2.el7.x86_64
15 make config KVERS=3.10.0-514.6.2.el7.x86_64
16  cd ../libpri-1.6.0

```

```

17 make && make install
18 ls
19 cd .. / asterisk -14.3.0
20 ./ configure —libdir=/usr/lib64
21 make menuselect
22 make install
23 make samples
24 make config
25 chkconfig asterisk

```

3.3.2. Configuraciones y recomendaciones

Para la activación de soporte de video y configuraciones generales en el servidor Asterisk es necesario agregar las siguientes líneas en el archivo **/etc/asterisk/sip.config**.

```

1 [ general ]
2 bindport=5060
3 transport=udp
4 canreinvite=no
5 directrtpsetup=no
6 externip=0.0.0.0
7 nat=yes
8 videosupport=yes
9
10 [ friends_internal ](!)
11 type=friend
12 host=dynamic
13 context=from-internal
14 allow=ulaw
15 allow=alaw
16 allow=h264
17 allow=vp8

```

ALGORITMO 3.1: Configuración sip.config

Además, para la efectiva comunicación de vídeo y audio se recomienda agregar la configuración `nat=yes` en la sección `[general]`, ver algoritmo 3.1.

Capítulo 4

Ensayos y Resultados

En el presente capítulo se describe el escenario con el que se realizaron ensayos y se obtuvieron resultados satisfactorios en la realización de una videollamada. Además, una tabla donde se compara el resultado del trabajo con los equipos de video portería del mercado.

4.1. Escenario para ensayos y resultados

Para el proceso de ensayos y recopilación de resultados se preparó el escenario ilustrado en la figura 4.1.

- Computadora portátil configurado como acceso de red en la interfaz *ether-net* con servicios de DHCP y NAT. Red local configurada como 192.168.10.0/24.
- Plataforma embebida STM32MP157C-DK2 conectada por cable ethernet a la computadora portátil y con la dirección IP 192.168.10.50.
- Servidor Asterisk en el servidor con nombre de dominio *sip.ericsonj.net*
- Teléfono inteligente Android conectado a internet por medio WiFi o red de datos móvil 4G y con aplicación softphone registrada en el servidor Asterisk (*sip.ericsonj.net*).

4.2. Resultados del establecimiento de videollamada

El establecimiento de una videollamada requiere el uso de protocolo SIP, para evidenciar esta funcionalidad se hizo uso del programa *Wireshark*. Este programa se ejecuto en la computadora portátil capturando la interfaz *ethernet* a la cual esta conectada el sistema embebido STM32MP157C-DK2.

En la figura 4.2 se ilustra el flujo de una videollamada entrante al sistema embebido. Se identifica al servidor Asterisk por su hostname *sip.ericsonj.net* y al sistema embebido por su nombre *stm32mp1.local*. El teléfono inteligente Android es el generador de la videollamada pasando por el servidor en la nube.

El flujo de una videollamada empieza con un mensaje *SIP INVITE* que es enviado por el teléfono inteligente al servidor Asterisk y retransmitido a la plataforma STM32MP157C-DK2. Seguidamente la plataforma contesta con un mensaje *SIP 100 TRYING*, un *SIP 200 OK* y se inicia el intercambio de multimedia. Ver figura 4.2.

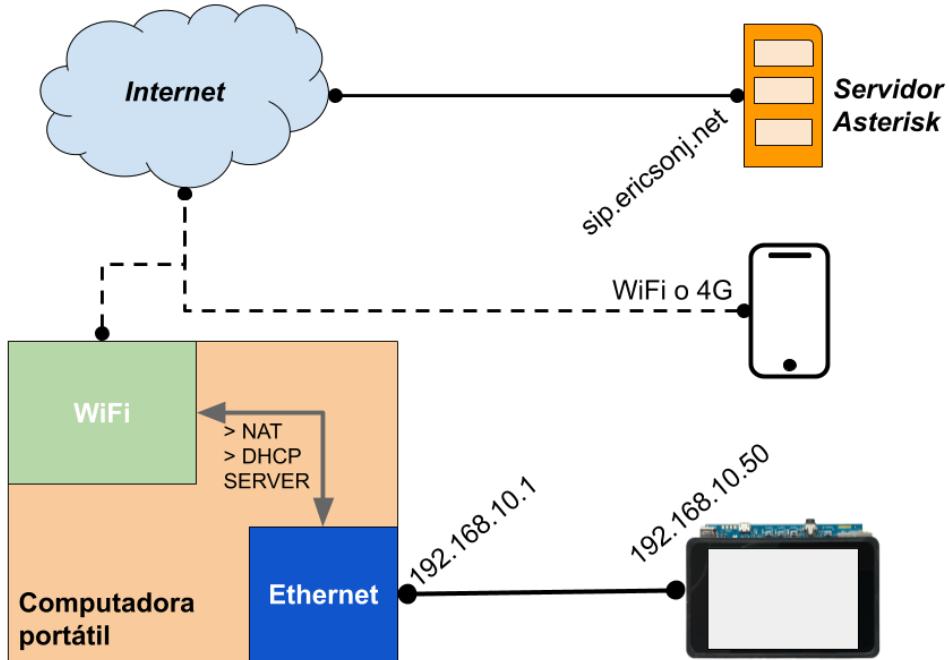


FIGURA 4.1: Escenario para ensayos y toma de resultados.

En la finalización de la videollamada el teléfono inteligente Android envió un mensaje *SIP BYE* al servidor que es retransmitido al sistema embebido; se termina con el intercambio de multimedia y se contesta con un mensaje *SIP 200 OK*.

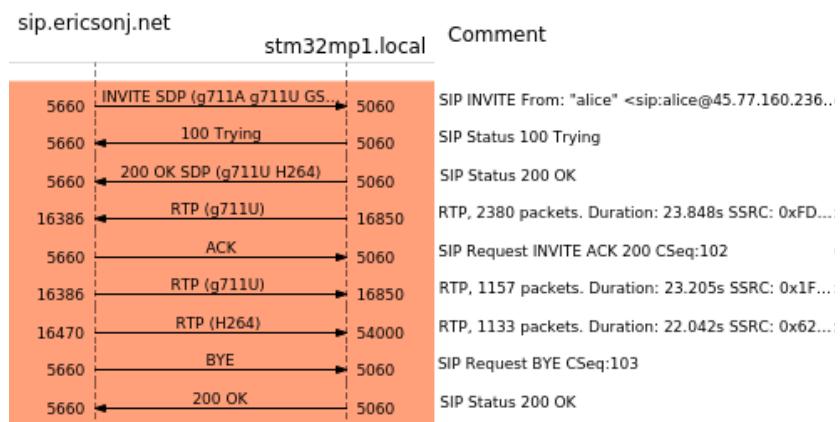


FIGURA 4.2: Flujo de paquetes SIP.

En la figura 4.3 se evidencia los mensajes SIP en una videollamada. Es importante resaltar el nombre de usuario de los interlocutores, el teléfono inteligente Android está registrado con el usuario “alice” y la plataforma embebida STM32MP157C-DK2 con el usuario “bob”.

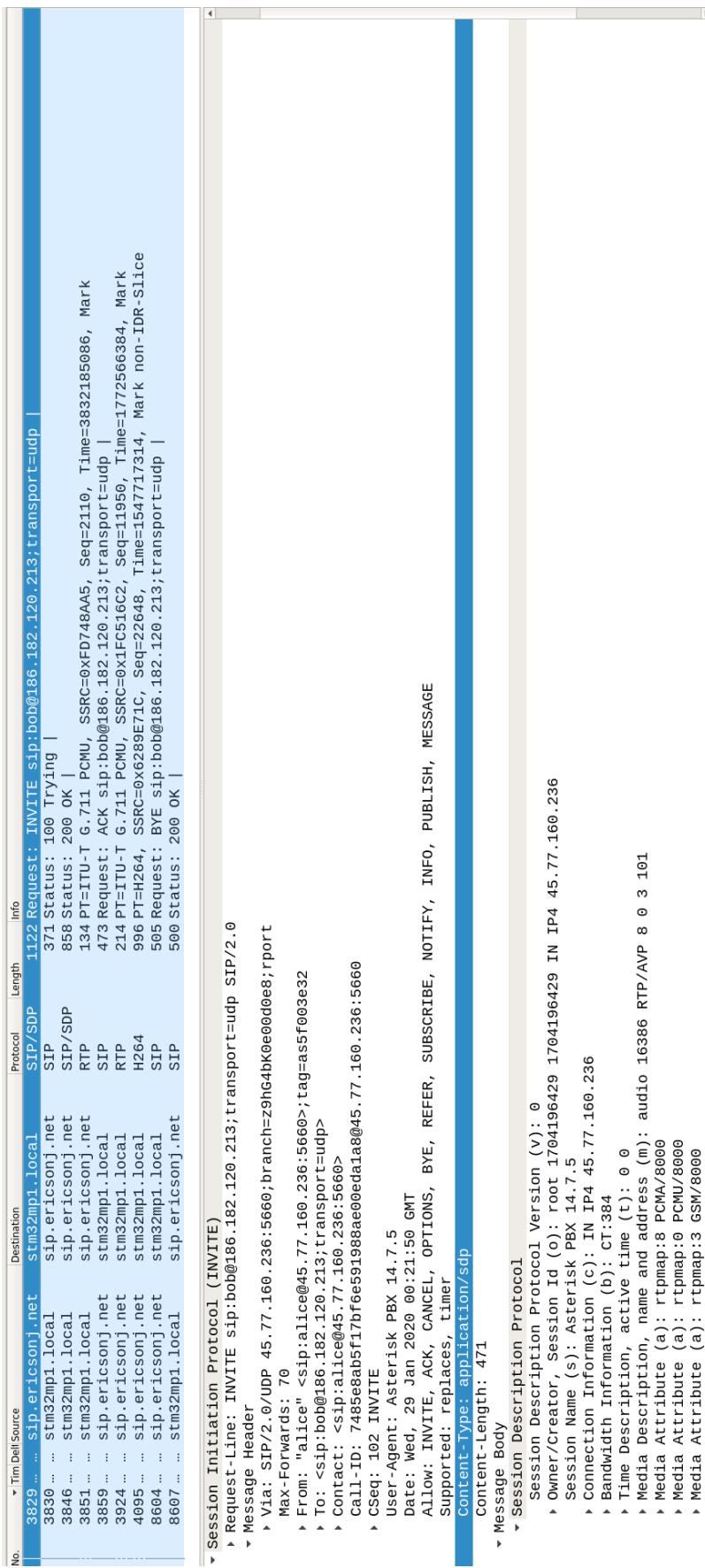


FIGURA 4.3: Tramas SIP capturas con Wireshark.

4.3. Resultados codificación H264

Para la transmisión de video, este se codifica en el estándar H264 y transmitido sobre RTP desde el teléfono móvil a el sistema embebido. En la figura 4.4 se ilustra el grafico de *jitter* y mediciones tomadas por el programa Wireshark de la recepción de video por parte de la terminal embebida. De estas mediciones es importante resaltar que no hay pérdida de paquetes y el *jitter* máximo es de 9,24 ms. Este jitter es compensado por la implementación de un buffer *jitter* descrito en 3.2.5.

En la figura 4.5 se ilustra el flujo de vídeo desde el teléfono inteligente Android a la terminal embebida. Se evidencia la codificación de vídeo en H264.

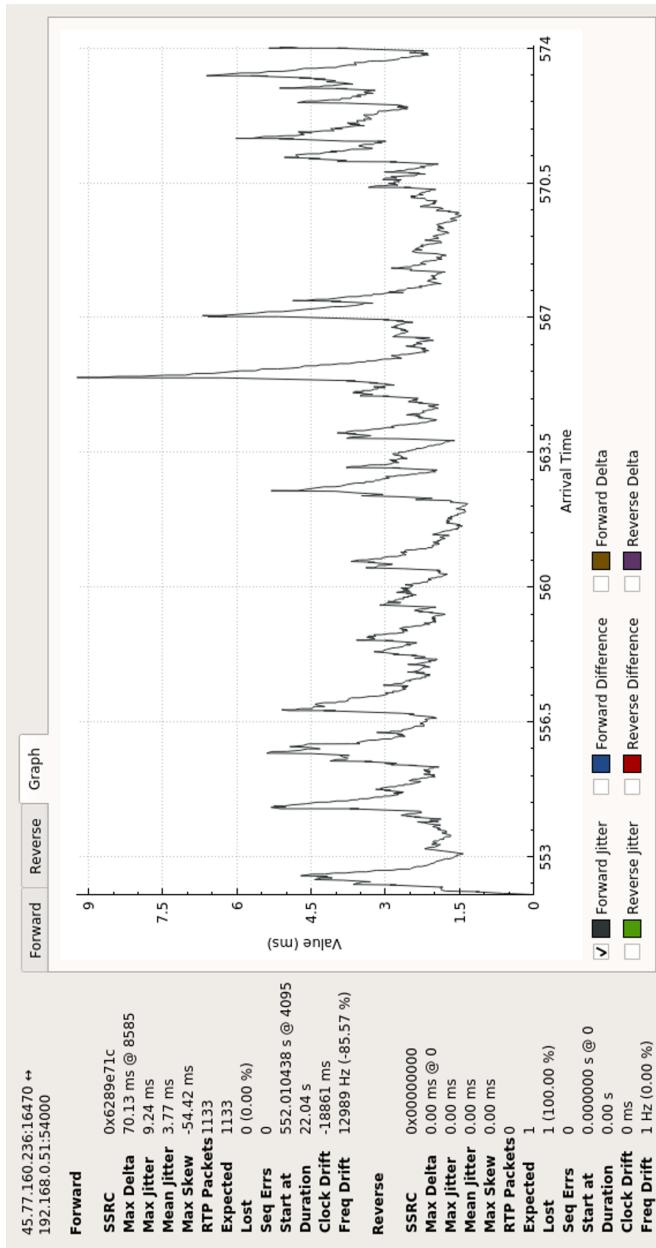


FIGURA 4.4: Mediciones de audio desde la terminal móvil a la terminal embebida

No.	▼ Tim Deli Source	Destination	Protocol	Length	Info
4095	... sip.ericsongj.net	STM32mp1.local	H264	996	PT=H264, SSRC=0x6289E71C, Seq=22648, Time=15477177314, Mark non-IDR-Slice
4101	... sip.ericsongj.net	STM32mp1.local	H264	1293	PT=H264, SSRC=0x6289E71C, Seq=22649, Time=1547720914 FU-A Start:non-IDR-Slice
4102	... sip.ericsongj.net	STM32mp1.local	H264	1294	PT=H264, SSRC=0x6289E71C, Seq=22650, Time=1547720914 FU-A
4104	... sip.ericsongj.net	STM32mp1.local	H264	1294	PT=H264, SSRC=0x6289E71C, Seq=22651, Time=1547720914 FU-A
4105	... sip.ericsongj.net	STM32mp1.local	H264	932	PT=H264, SSRC=0x6289E71C, Seq=22652, Time=1547720914, Mark FU-A End
4110	... sip.ericsongj.net	STM32mp1.local	H264	1293	PT=H264, SSRC=0x6289E71C, Seq=22653, Time=1547723614 FU-A Start:non-IDR-Slice
4111	... sip.ericsongj.net	STM32mp1.local	H264	1294	PT=H264, SSRC=0x6289E71C, Seq=22654, Time=1547723614 FU-A
4112	... sip.ericsongj.net	STM32mp1.local	H264	958	PT=H264, SSRC=0x6289E71C, Seq=22655, Time=1547723614, Mark FU-A End
4117	... sip.ericsongj.net	STM32mp1.local	H264	1293	PT=H264, SSRC=0x6289E71C, Seq=22656, Time=1547726314 FU-A Start:non-IDR-Slice
4118	... sip.ericsongj.net	STM32mp1.local	H264	834	PT=H264, SSRC=0x6289E71C, Seq=22657, Time=1547726314, Mark FU-A End
4124	... sip.ericsongj.net	STM32mp1.local	H264	947	PT=H264, SSRC=0x6289E71C, Seq=22658, Time=1547729014, Mark non-IDR-Slice
4131	... sip.ericsongj.net	STM32mp1.local	H264	992	PT=H264, SSRC=0x6289E71C, Seq=22659, Time=1547732614, Mark non-IDR-Slice
4137	... sip.ericsongj.net	STM32mp1.local	H264	1293	PT=H264, SSRC=0x6289E71C, Seq=22660, Time=1547735314 FU-A Start:non-IDR-Slice
4138	... sip.ericsongj.net	STM32mp1.local	H264	826	PT=H264, SSRC=0x6289E71C, Seq=22661, Time=1547735314, Mark FU-A End
4147	... sip.ericsongj.net	STM32mp1.local	H264	1293	PT=H264, SSRC=0x6289E71C, Seq=22662, Time=1547738914 FU-A Start:non-IDR-Slice
4148	... sip.ericsongj.net	STM32mp1.local	H264	120	PT=H264, SSRC=0x6289E71C, Seq=22663, Time=1547738914, Mark FU-A End
4151	... sip.ericsongj.net	STM32mp1.local	H264	774	PT=H264, SSRC=0x6289E71C, Seq=22664, Time=1547741614, Mark non-IDR-Slice
► Frame 4101: 1293 bytes on wire (10344 bits), 1293 bytes captured (10344 bits) on interface 0					
► Ethernet II, Src: 92:91:cf:17:3f:47 (92:91:cf:17:3f:47), Dst: stm32mp1.local (00:00:00:00:00:00)					
► Internet Protocol Version 4, Src: sip.ericsongj.net (45.77.160.236), Dst: stm32mp1.local (192.168.0.51)					
► User Datagram Protocol, Src Port: 54000, Dst Port: 16470, Protocol: Real-Time Transport Protocol					
► [Stream setup by SDP (frame 3829)]					
10. = Version: RFC 1889 version (2)					
..0. = Padding: False					
...0 = Extension: False					
.... 0000 = Contributing source identifiers count: 0					
0...i... = Marker: False					
Payload type: H264 (99)					
Sequence number: 22649					
[Extended sequence number: 88185]					
Timestamp: 1547720914					
Synchronization Source identifier: 0x6289e71c (1653204764)					
► H 264					

FIGURA 4.5: Transmisión de video en H264

4.4. Resultados codificación de audio G711

Para la transmisión de audio, este se codifica con el estándar G711 y es transmitido sobre RTP. En la figura 4.6 se ilustra el gráfico de *jitter* y mediciones tomadas desde el programa Wireshark del envío de audio desde el teléfono móvil a la terminal embebida. De este gráfico es importante resaltar que como resultado se obtienen un flujo de audio con ningún paquete perdido y un *jitter* máximo de 5,33 ms inferior al tiempo entre frames de audio que es de 20 ms. De la misma forma, en la figura 4.7 se ilustra el gráfico de *jitter* y mediciones del envío de audio desde la terminal embebida al teléfono móvil. Como resultado, se obtiene un flujo de audio con 2 paquetes perdidos, que representa el 0,17% de error y un *jitter* máximo de 9,12 ms; inferior al tiempo entre frames de audio que es de 20 ms. Estos datos evidencian que no se afecta la calidad del audio por temas relacionado a la red.

Por último en la figura 4.8 se ilustra el flujo de audio desde el teléfono inteligente Android a la terminal embebida y viceversa; también se evidencia la codificación de audio en G711.

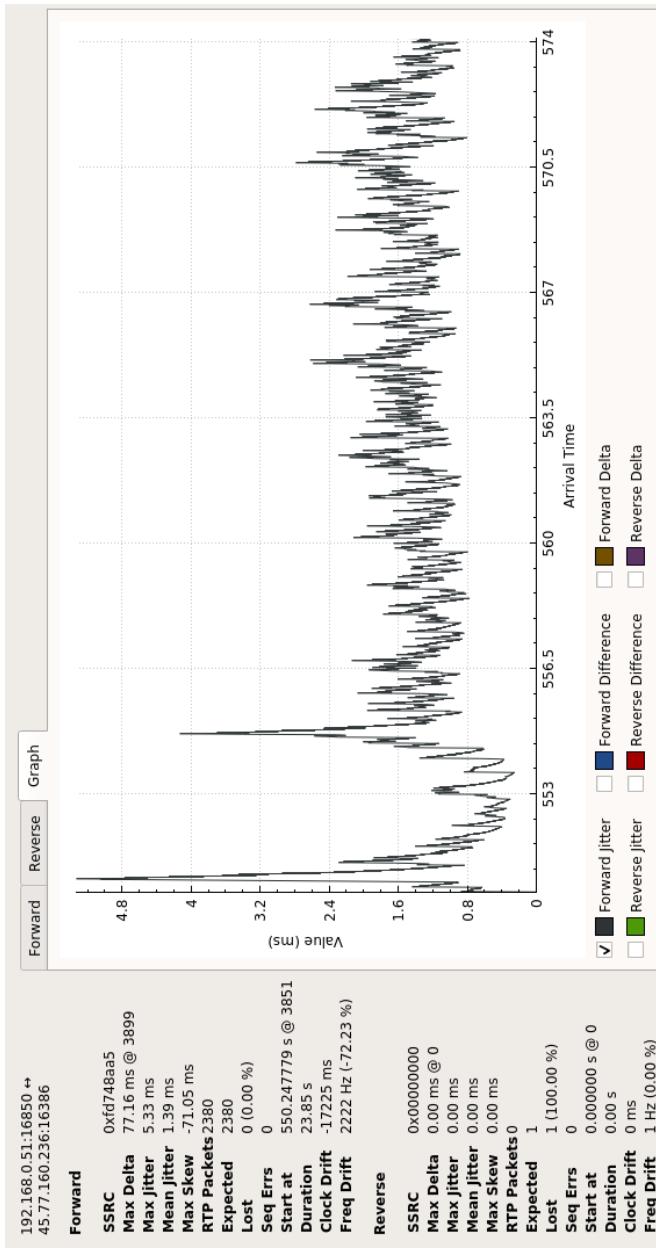


FIGURA 4.6: Mediciones de audio desde la terminal móvil a la terminal embebida

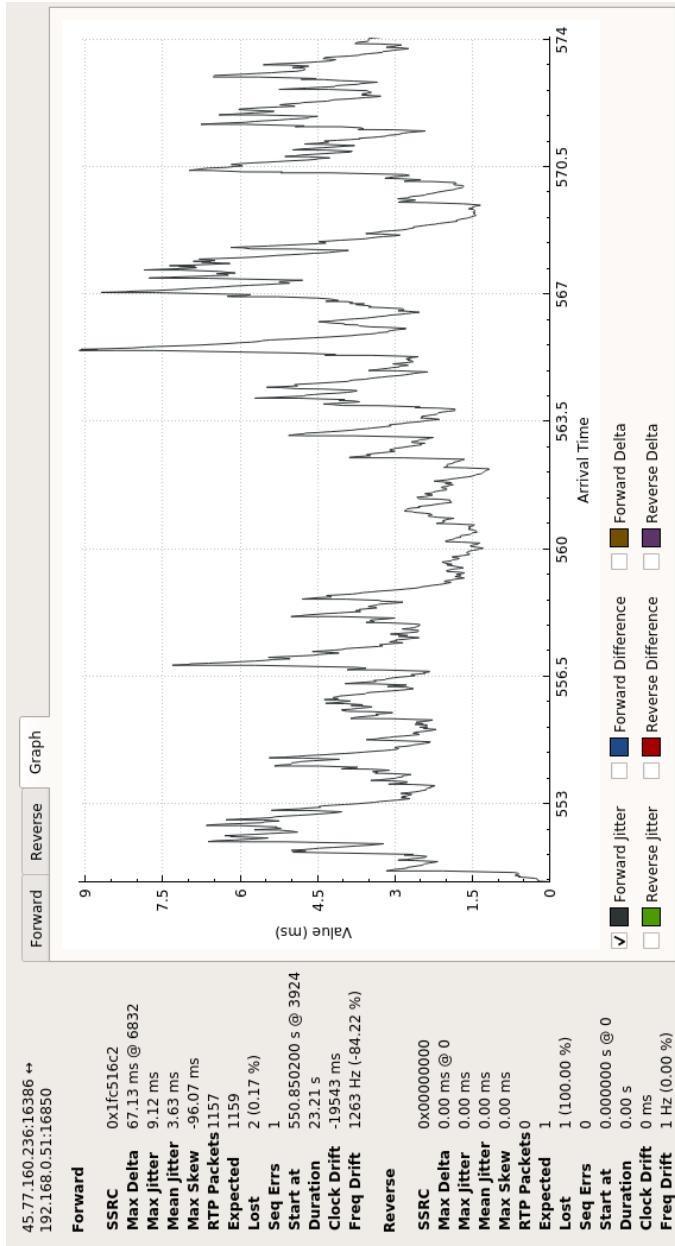


FIGURA 4.7: Mediciones de audio desde la terminal embebida a el teléfono móvil

No.	▼ Tim.Dell Source	Destination	Protocol	Length	Info
4153	... sip.ericonj.net	stm32mp1.local	RTP	214	PT=ITU-T G.711 PCMU, SSRC=0x1fc516c2, Seq=12019, Time=1772577424
4154	... stm32mp1.local	sip.ericonj.net	RTP	134	PT=ITU-T G.711 PCMU, SSRC=0xfd748aa5, Seq=2309, Time=3832201006
4155	... sip.ericonj.net	stm32mp1.local	RTP	214	PT=ITU-T G.711 PCMU, SSRC=0x1fc516c2, Seq=12020, Time=1772577584
4156	... stm32mp1.local	sip.ericonj.net	RTP	134	PT=ITU-T G.711 PCMU, SSRC=0xfd748aa5, Seq=2310, Time=3832201086
4157	... stm32mp1.local	sip.ericonj.net	RTP	134	PT=ITU-T G.711 PCMU, SSRC=0x1fc516c2, Seq=2311, Time=3832201166
4159	... sip.ericonj.net	stm32mp1.local	RTP	214	PT=ITU-T G.711 PCMU, SSRC=0x1fc516c2, Seq=12021, Time=1772577744
4160	... stm32mp1.local	sip.ericonj.net	RTP	134	PT=ITU-T G.711 PCMU, SSRC=0xfd748aa5, Seq=2312, Time=3832201246
4161	... stm32mp1.local	sip.ericonj.net	RTP	134	PT=ITU-T G.711 PCMU, SSRC=0xfd748aa5, Seq=2313, Time=3832201326
4162	... stm32mp1.local	sip.ericonj.net	RTP	134	PT=ITU-T G.711 PCMU, SSRC=0x1fc516c2, Seq=2314, Time=3832201406
4165	... sip.ericonj.net	stm32mp1.local	RTP	214	PT=ITU-T G.711 PCMU, SSRC=0x1fc516c2, Seq=12022, Time=1772577904
4166	... stm32mp1.local	sip.ericonj.net	RTP	134	PT=ITU-T G.711 PCMU, SSRC=0xfd748aa5, Seq=2315, Time=3832201486
4167	... sip.ericonj.net	stm32mp1.local	RTP	214	PT=ITU-T G.711 PCMU, SSRC=0x1fc516c2, Seq=12023, Time=1772578064
4168	... stm32mp1.local	sip.ericonj.net	RTP	134	PT=ITU-T G.711 PCMU, SSRC=0xfd748aa5, Seq=2316, Time=3832201566
4169	... stm32mp1.local	sip.ericonj.net	RTP	134	PT=ITU-T G.711 PCMU, SSRC=0xfd748aa5, Seq=2317, Time=3832201646
4170	... stm32mp1.local	sip.ericonj.net	RTP	134	PT=ITU-T G.711 PCMU, SSRC=0xfd748aa5, Seq=2318, Time=3832201726
► Frame 4153: 214 bytes on wire (1712 bits), 214 bytes captured (1712 bits) on interface 0					
► Ethernet II, Src: 92:91:cf:17:3f:47 (92:91:cf:17:3f:47), Dst: stm32mp1.local (00:80:e1:42:63:11)					
► Internet Protocol Version 4, Src: sip.ericonj.net (45.77.160.236), Dst: stm32mp1.local (192.168.0.51)					
► User Datagram Protocol, Src Port: 16386, Dst Port: 16850					
► Real-time Transport Protocol					
► [Stream setup by SDP (frame 3829)]					
10.. = Version: RFC 1889 Version (2)				
..0. = Padding: False				
...0 = Extension: False				
....	0000 = Contributing source identifiers count: 0				
0... = Marker: False				
Payload type:	ITU-T G.711 PCMU (0)				
Sequence number:	12019				
[Extended sequence number:	77555]				
Synchronization source identifier:	0x1fc516c2 (533910114)				
Payload:	f8ecedf0efffcff6f87e7efef747070726f6e726d706f6e70...				

FIGURA 4.8: Transmisión de audio en G711

4.5. Resultado final

En la figura 4.9 se ilustra el resultado de una videollamada recibida por la plataforma STM32MP157C-DK2; se observa la conexión de red (a), conexión de alimentación (b), conexión para auriculares (c) y pantalla (d).

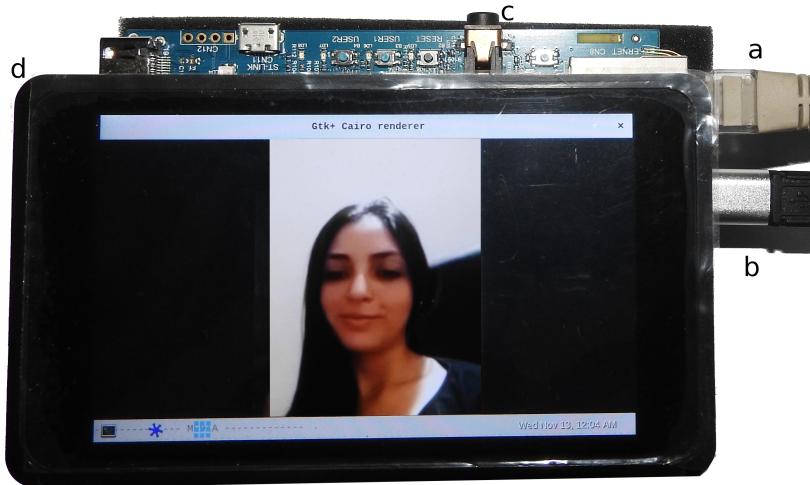


FIGURA 4.9: Resultado final del trabajo.

4.6. Análisis de resultados

Revisando las características de algunos productos que se encuentran en el mercado nacional, se realizó una comparación de éstos con los resultados obtenidos en el trabajo final.

Los productos descritos en 1.5.1 y 1.5.2 transmiten el vídeo de forma analógica y no cuentan con conexión ethernet. Esta condición hace que estos equipos estén dirigidos a solucionar un requerimiento muy específico y la integración con terceros por medio de protocolos de red no es posible. Los productos descritos en 1.5.3 y 1.5.4 son productos IP; esta característica los hace compatibles con tecnologías como IoT y desarrollos de terceros.

En la tabla 4.1 se comparan las caracterizadas que ofrecen los equipos encontrados en el mercado con el trabajo realizado. El trabajo realizado al igual que los equipos analizados cuentan con protocolo de red, recepción de video en H264, conexión Ethernet y WiFi. Las caracterizadoras de altavoces, almacenamiento de video y apertura de puerta no fueron contempladas en el trabajo realizado y se deja para trabajos futuros.

Surix S.R.L, siendo una empresa de más de 20 años en el mercado de vídeo porteros ve en los equipos IP una producto competitivo y versátil para el mercado argentino. Es por esta razón que el trabajo final es un prototipo de un monitor para vídeo portero con las siguientes características:

- Sistema operativo LINUX embebido
- Protocolos SIP, RTSP, RTP.

TABLA 4.1: Comparación con equipos en el mercado

Característica	Dahua	Hikvision	Trabajo realizado
Pantalla táctil	x	x	x
Altavoces para llamadas de acceso a puerta	x	x	
Apertura remota de puerta	x	x	
Almacenamiento de vídeo	x	x	
GNU/LINUX	x	x	x
SIP	x	x	x
RTSP	x	x	x
RTP	x	x	x
H.264	x	x	x
G711	x	x	x
Conexión WiFi	x	x	x
Conexión Ethernet.	x	x	x

- Codificación de video H264.
- Codificación de audio G711.
- Pantalla táctil LCD de 4”.
- Conexión Ethernet.

Capítulo 5

Conclusiones

En este capítulo se encuentra la conclusión general del proyecto y sus próximos pasos. Es importante resaltar que la empresa Surix S.R.L esta interesada en la continuación de este proyecto para la culminación de un producto final.

5.1. Conclusiones generales

La culminación de este trabajo final logra ser un punto de partida y referencia para la empresa Surix S.R.L en los sistemas de videollamada. Se cumple con el propósito de empezar el desarrollo a nivel de software del prototipo de un nuevo producto para la empresa. Se implementó con éxito un sistema operativo GNU/Linux sobre una plataforma embebida ARM Cortex-A y se desarrolló una aplicación nativa de videollamada. A continuación se listan los logros mas característicos del trabajo final:

- Establecimiento de una videollamada entre una terminal móvil Android y la plataforma embebida, la cual reproduce el vídeo de la llamada.
- Se desarrolló una aplicación nativa para sistema operativo GNU/Linux y se implementó con éxito las librerías de protocolo SIP y manejo de multimedia.
- Implementación de un sistema operativo GNU/Linux personalizado para la plataforma STM32MP157C-DK2.
- Se instaló un servidor en la nube (Asterisk PBX).

5.2. Próximos pasos

En esta sección se listan los trabajos futuros a partir del proyecto:

- Diseño y construcción de un SOM (*System On Module*) basado en un procesador ARM Cortex-A.
- Diseño y construcción de una tarjeta electrónica con diferentes periféricos tales como: GPIO (*General Purpose Input/Output*), ethernet, RELES, teclado numérico, entre otros y que soporte el SOM.
- Algoritmos de cancelación de eco para mejorar la experiencia del usuario en una comunicación full-duplex con altavoces.

- Algoritmos de detección de voz para la reducción de ruido ambiente en una comunicación.

Bibliografía

- [1] st.com. *STM32MP157C-DK2*. Disponible: 2018-12. 2019. URL: <https://www.st.com/en/evaluation-tools/stm32mp157c-dk2.html>.
- [2] linphone.org. *Linphone*. Disponible: 2020-03. 2020. URL: <https://www.linphone.org/>.
- [3] J. Rosenberg y col. *SIP: Session Initiation Protocol*. RFC 3261. RFC Editor, 2002.
- [4] commax.com. *VIDEOPHONE SERIES*. Disponible: 2020-03. 2020. URL: <https://www.commax.com/eng/index.php?cate1=112&cate2=57&cate3=70&nnum=2180>.
- [5] sicaelec.com. *Porteros y Video Porteros*. Disponible: 2020-03. 2020. URL: <https://sicaelec.com/seuridad/>.
- [6] dahuasecurity.com. *Video Porteros*. Disponible: 2020-03. 2020. URL: <https://www.dahuasecurity.com/la/products/All-Products/Video-Intercoms/IP-Products/VTH5241DW>.
- [7] hikvision.com. *Intercomunicador de video*. Disponible: 2020-03. 2020. URL: <https://www.hikvision.com/es-la/Products/Video-Intercom/Indoor-Station-/DS-KH6320-WTE1#prettyPhoto>.
- [8] gstreamer.freedesktop.org. *gstreamer, open source multimedia framework*. Disponible: 2019-11. 2019. URL: <https://gstreamer.freedesktop.org/>.
- [9] mipi.org. *MIPI Overview*. Disponible: 2019-11. 2019. URL: <https://www.mipi.org/about-us>.
- [10] *Low-Power, Stereo Codec with Headphone Amp*. CS42L51. Rev. F2. Cirrus Logic. Ago. de 2015.
- [11] developer.gnome.org. *GLib Reference Manual*. Disponible: 2020-03. 2020. URL: <https://developer.gnome.org/glib/>.
- [12] sofia-sip.sourceforge.net. *Sofia-SIP*. Disponible: 2020-03. 2020. URL: <http://sofia-sip.sourceforge.net/>.
- [13] Voip-info.org. *What is VOIP*. Disponible: 2003-07. 2018. URL: <https://www.voip-info.org/what-is-voip>.
- [14] voipreview.org. *Compare the Top 10 Best Open Source PBX Software of 2020*. Disponible: 2020-03. 2020. URL: <https://www.voipreview.org/business-voip/best-open-source-pbx-software>.
- [GPL] *GNU General Public License*. Inglés. Ver. 3. Free Software Foundation, 29 de jun. de 2007. URL: <http://www.gnu.org/licenses/gpl.html>.
- [15] J. Van Megelen, R. Bryant y L. Madsen. *Asterisk: The Definitive Guide: Open Source Telephony for the Enterprise*. O'Reilly Media, 2019. ISBN: 9781492031574. URL: <https://books.google.com.ar/books?id=G6eeDwAAQBAJ>.
- [16] H. Schulzrinne. *RTP: A Transport Protocol for Real-Time Applications*. RFC 3550. Jul. de 2003. DOI: <10.17487/rfc3550>. URL: <https://tools.ietf.org/rfc/rfc3550.txt>.
- [17] Arun Handa. «Chapter 4 - The IMS-related Protocols». En: *System Engineering For IMS Networks*. Ed. por Arun Handa. Burlington:

- Newnes, 2009, págs. 45 -93. ISBN: 978-0-7506-8388-3. DOI:
<https://doi.org/10.1016/B978-0-7506-8388-3.00004-6>. URL: <http://www.sciencedirect.com/science/article/pii/B9780750683883000046>.
- [18] Voip-info.org. *ITU G.711*. Disponible: 2003-07. 2018. URL:
<https://www.voip-info.org/itu-g711>.
- [19] Keith Jack. «Chapter 7 - MPEG-1, MPEG-2, MPEG-4, and H.264». En: *Digital Video and DSP*. Ed. por Keith Jack. Instant Access. Burlington: Newnes, 2008, págs. 165 -199. ISBN: 978-0-7506-8975-5. DOI:
<https://doi.org/10.1016/B978-0-7506-8975-5.00007-8>. URL: <http://www.sciencedirect.com/science/article/pii/B9780750689755000078>.
- [20] wiki.friendlyarm.com. *NanoPi Fire3*. Disponible: 2020-03. 2020. URL:
http://wiki.friendlyarm.com/wiki/index.php/NanoPi_Fire3.
- [21] tinkerboarding.co.uk. *Hardware*. Disponible: 2020-03. 2020. URL:
<https://tinkerboarding.co.uk/wiki/index.php/Hardware>.
- [22] gstreamer.freedesktop.org. *GStreamer Features*. Disponible: 2019-11. 2019. URL: <https://gstreamer.freedesktop.org/features/>.