FACULTAD DE INGENIERÍA - Trabajo Final de Grado

Ingeniería en Telecomunicaciones

Comunicaciones en tiempo Real en el navegador Web mediante el uso de SIP sobre Websocket para señalizar y WebRTC para proporcionar soporte de media.

INSTITUTO UNIVERSITARIO AERONÁUTICO

Alumnos:

Cioccatto, Maciel

Pignataro, Fabián

Profesor:

Fernández, Javier

...

2015

Dedicado a familiares y amigos

AGRADECIMIENTOS

Ha llegado el gran día y con él se cierra una etapa académica maravillosa y a su vez, intensa; por ello tenemos que agradecer a muchas personas sin las cuales es posible que este momento no hubiese llegado, por ellas valgan estos agradecimientos, por el pasado y también por el futuro que juntos nos espera.

Tenemos el gusto de poder agradecer de manera compartida:

a Javier Fernandez, nuestro tutor de tesis, que estuvo siempre dispuesto a ayudar en lo que fuera necesario. Gracias por tus buenas ideas, trabajo y dedicación.

a todos los profesores de la Carrera de Ingeniería en Telecomunicaciones que en un momento u otro, de una forma u otra nos han dado una mano y nos han alentado para continuar.

a todas las personas de la División de Alumnos que nos ayudaron durante toda la carrera y al personal del Departamento de Desarrollo Profesional, que nos guiaron en todo el desarrollo de la Tesis.

a nuestros familiares, amigos y compañeros de trabajo, por haber estado, por el interés y por saber entender ausencias.

Y por último una dedicación especial, por la cual le damos nombre a quienes, por sobre todo, hicieron posible este momento:

de Fabian, a mis padres Héctor Pignataro, Liliana Carranza, a mi hermano Marcelo Pignataro, a mi tía Evelyn Carranza y a mi abuela Ema Paredes.

de Maciel, a mis padres Enildo Cioccatto, Mirtha Beltramo, y a mis hermanas Ivana y Micaela Cioccatto.

Gracias, sin ustedes esta tesis no sería posible.Fabián – Maciel.

ÍNDICE

PREÁMBULO 1
Capitulo 1.- INTRODUCCIÓN 5
1.1.- Tema a desarrollar 5
1.2.- Objetivos 5
1.2.1.- Objetivo general 6
1.2.2.- Objetivos específicos 6
1.3.- Marco científico de la tesis 7
1.4.- Origen de la RTC 8
1.5.- Situación actual de la RTC 9
1.6.- Beneficios 10
1-7.- Destinatarios 11
1.8.- Índice comentado 12
Capitulo 2.- ESTUDIO TÉCNICO 15
2.1.- Estándares y protocolos a utilizar 15
2.2.- Tecnologías 19
2.2.1.- WebRTC 19
2.2.1.1.- Introducción 19
2.2.1.2.- Arquitectura de WebRTC 20
2.2.1.3.- El funcionamiento de WebRTC 25
2.2.1.4.- Señalización 26
2.2.1.5.- WebRTC y el NAT 28
2.2.1.6.- Seguridad 30
2.2.2.- La web 31
2.2.3.- HTML5 33
2.2.4.- JavaScripts 34
2.2.5.- CCS3 35
2.2.6.- jsSIP 35
2.2.7.- sipML5 35
2.2.8.- Gnu/Linux 36
2.2.9.- Apache, Asterisk y Kamailio 36

2.3.- Protocolos 37
2.3.1.- El protocolo HTTP 37
2.3.2.- Websockets 38
2.3.2.1.- Introducción 38
2.3.2.2.- Descripción general 39
2.3.2.3.- Ventajas 39
2.3.2.4.- Modela de seguridad 40
2.3.3.5.- Relación con el TCP y HTTP 40
2.3.3.- El protocolo SIP 41
2.3.3.1.- Introducción 41
2.3.3.2.- Arquitectura SIP 41
2.3.3.3.- Componentes de una red SIP 43
.- Usuario 44
.- Servidores de red 45
2.3.3.4.- URI – Las direcciones SIP 48
2.3.3.5.- El diálogo SIP 48
2.3.3.6.- La Cabecera del paquete SIP (Message Header) 52
2.3.3.7.- El cuerpo del paquete SIP 53
2.3.4.- El protocolo RTP 57
2.4.- Ws como transporte para SIP 58
2.4.1.- Introducción 58
2.4.2.- Definiciones 59
2.4.3.- SIP sobre Websockets 60
2.5.- Requisitos de hardware y software 63
Capitulo 3.- DESARROLLO DE LA TECNOLOGÍA WEBRTC 65
3.1.- Resumen técnico 65
3.1.1.- Diagrama de la “Web tradicional” 66
3.1.2.- Diagrama general de la parte teórica 66
3.1.3.- Diagrama en bloques del sistema propuesto 70
3.1.4.- Funcionamiento del sistema propuesto 71
3.2.- Diseño de la tecnología WebRTC 75
3.2.1.- Escenarios de comunicación 77

3.2.2.- Componentes de Software y Hardware 78
3.2.3.- Descripción de cada componente 79
3.2.4.- Instalación/configuración de los servidores 80
.- Instalación del Web Server Host 81
.- Instalación SIP Router Kamailio 84
.- Instalación Asterisk IP PBX 90
3.2.4.- Instalación/configuración de las plataformas de comunicaciones 95
.- WebRTC con Asterisk 95
Pasos para instalar y configurar WebRTC con Asterisk 97
.- WebRTC con Kamailio 101
Pasos para instalar y configurar WebRTC con Kamailio 102
3.2.5.- Pruebas de comunicaciones Real -Time 105
3.2.5.1.- Pruebas en la plataforma Asterisk 105
3.2.5.2.- Pruebas en la plataforma Kamailio 124
3.3.- Resultados 132
3.4.- Dificultades que se han presentado 133
3.5.- Actividades realizadas 134
Capitulo 4.- VIABILIDAD COMERCIAL 139
Capitulo 5.- CONCLUSIONES 143
Capitulo 6.- BIBLIOGRAFÍA 147
Capitulo 7.- ANEXOS 149
7.1.- Lista de Figuras 149
7.2.- Lista de tablas 152
7.3.- Abreviaturas y acrónimos 153
7.4.- Definiciones 155
7.5.- Resumen general de la tesis 160
7.6.- Anexos del Capítulo 1 160
7.6.1.- Resumen del Capítulo 1 160
7.7.- Anexos del Capítulo 2 162
7.7.1.- Arquitectura global de WebRTC 162
7.7.2.- Evolución de la comunicación vía Web 163
7.7.3.- Modelo del navegador 164

7.7.4.- Javascripts 165
7.7.5.- Resumen Capítulo 2 166
7.8.- Anexos del Capítulo 3 167
7.8.1.- El archivo kamailio.cfg 167
7.8.2.- El archivo /etc/kamailio/kamailio.cfg 183
7.8.3.- Configuración de la plataforma WebRTC en Jitsi Web Meet 191
.- Pruebas de WebRTC en Jitsi Web Meet 192
7.9.- Instalación del servidor Linux 193

PREÁMBULO

"El software se va a comer el mundo" es la célebre frase del co-fundador de la empresa Netscape Communications Corporation y creador de Mosaic, uno de los primeros navegadores web de la historia. Esta cita se enmarca perfectamente como preámbulo a lo que será el desarrollo de este trabajo final de grado.

Vivimos en un mundo en donde se atraviesa un cambio generacional en el cual hay una marcada tendencia hacia el uso de "la web" para todo. Para fundamentar dicha tendencia, se procede con la ejemplificación de diferentes situaciones de la vida cotidiana en donde el hombre y sobre todo el joven de estos tiempos están reemplazando formatos y costumbres por el uso de la web:

- El DVD o VHS: *los video-clubs se extinguieron a lo largo del mundo, siendo las plataformas basadas en la web su reemplazo. Aquí se pueden citar servicios de streaming de video como Netflix, iTunes, Redbox y Amazon entre otros los responsables de brindar todo tipo de videos (películas, documentales, musicales, etc.) bajo demanda y sin salir de casa.*

- La compra: *hoy es moneda corriente comprar a través de plataformas web, desde un celular o electrodoméstico hasta un juego de cocina. De nuevo se plantea la situación de comprar a la hora deseada y sin salir de casa.*

- La televisión: *los jóvenes prácticamente no encienden más la TV, en lugar de ello prefieren usar servicio en la web como Netflix, Youtube, Vimeo, Hulu, etc. Los mismos son accesibles a través de un Smartphone, notebook, tableta.*

- La Música y Los Libros: *si bien siempre va a existir el coleccionismo y apego por los formatos físicos en cuestiones de arte como lo es la música y los libros, hay una tendencia desaforada por el consumo de música a través de servicios en internet como Spotify, Music by Apple, Google Play Music, Xbox Music, los cuales permiten escuchar cualquier artista, de nuevo en el momento deseado y sin salir de casa para adquirir su disco en formato físico. En el campo de la literatura lo mismo, servicios como Amazon Kindle están siendo cada vez más adoptados a la hora de leer un libro.*

- La publicidad: *si bien sigue siendo un mecanismo efectivo la publicidad callejera, en radio o TV, cada vez más empresas destinan recursos para publicitar sus productos o servicios a través de Google Adwords, Facebook, Linkedin o Youtube. Los nuevos hábitos de consumo de la población arrastra también la forma de hacer publicidad, llevando la misma también al campo de la web.*

- Diarios y Revistas: *La digitalización de dichos medios fue uno de los primeros pasos en esta tendencia a consumir todo desde la web.*

Invitamos al lector a repasar una vez más la frase “El software se va a comer el mundo” luego de leer los escenarios planteados. La pregunta es, qué pasa en materia de comunicaciones, ya que las comunicaciones son parte de este mundo en que el software planea devorar. El tema elegido para desarrollar en este trabajo final de grado gira en torno a nuestra frase célebre, ya que plantea llevar las comunicaciones (telefonía, video-conferencia, mensajería, etc.) al plano de la web. Dicha tecnología se llama WebRTC o Web Real Time Communication.

Si evaluamos el contexto actual de la telefonía y comunicaciones en el ámbito corporativo, se observa una firme tendencia de migrar las comunicaciones de "tiempo real" sobre las redes de conmutación de paquetes (IP) es un hecho y día a día son más las organizaciones que optan por implementar sus comunicaciones Voz, Video, Mensajería Instantánea sobre redes IP. Además el constante impulso hacia el "Cloud Computing" gracias a la evolución de la calidad del acceso a internet por parte de usuarios y empresas es el motor generador de nuevas tendencias como el "home work" y las "virtual offices".

WebRTC; se trata de tecnología de código abierto (Open Source) que dota tanto a los fabricantes de navegadores web así como también a los desarrolladores de aplicaciones web y móviles, de APIs y componentes que permiten a los fabricantes de navegadores web, implementar capacidades multimedia (procesamiento de voz y video) y los desarrolladores de aplicaciones web y móviles de herramientas para que puedan construir nuevas aplicaciones que permitan la comunicación en tiempo real; servicios de Voz, Video, Mensajería, Compartición de escritorio, transferencia de archivos, sin la necesidad de ejecutar ningún “plugin” sobre el sistema operativo de escritorio o móvil. La idea planteada es comunicarse desde un navegador web o desde “la web”, impulsando la costumbre de las personas de hoy, de hacer todo desde la web, inclusive COMUNICARSE en tiempo real.

A partir de esta tecnología se está frente a un nuevo horizonte, el cual permite que dichas comunicaciones pasen a ser tan simples, eficientes y libres como mandar un correo electrónico, solicitar el contenido de una página web, escuchar música online, ver películas o leer un libro, siempre desde “la web”.

CAPÍTULO 1.- INTRODUCCIÓN

En este capítulo se pretende centrar al lector respecto al tema de la Tesis.

En primer lugar se exponen los objetivos a cumplir y se hace un desarrollo de los diferentes marcos en los que se ha desarrollado la Tesis.

Tras ello se describe de forma general los orígenes y la situación actual de las telecomunicaciones en el campo RTC a través del navegador web.

Se continua el capitulo con los beneficios de la tesis y para quienes está dirigida la nueva tecnología.

Se concluye con unas breves explicaciones sobre el contenido global de cada capítulo.

1.1.- Tema a desarrollar

Comunicaciones en tiempo real en el navegador web mediante el uso de SIP sobre websocket para señalizar y webrtc para proporcionar soporte de media.

1.2.- Objetivos

Los objetivos de la tesis se definen dentro del Objetivo General como guía para la implementación de la nueva tecnología de comunicación. Además se plantean unos Objetivos Específicos, que complementan al Objetivo General con la finalidad de extraer resultados tangibles y evaluables.

A continuación se detalla el objetivo general y los objetivos específicos.

1.2.1.- Objetivo general

Aplicar la tecnología WebRTC y la combinación de nuevos protocolos de comunicaciones que permiten implementar el concepto de cliente de comunicaciones unificadas en un navegador web, a través de una aplicación web.

1.2.2.- Objetivos específicos

- Analizar y comprender la tecnología WebRTC: definición, alcance, posibles escenarios de uso, así como también los fundamentes científicos y técnicos sobre los cuales se asienta la tecnología.

- Analizar y comprender el stack de protocolos propuestos para implementar la señalización en entornos de WebRTC puros (usuario WebRTC - usuario WebRTC) e híbridos (usuario de sistema de comunicaciones convencional - usuario WebRTC).

- Implementar una arquitectura cliente-servidor que utilice las tecnologías "WebRTC" y "The WebSocket Protocol as a Transport for the Session Initiation Protocol (SIP)" para pruebas de concepto y análisis.

- Implementar una “arquitectura hibrida” conformada por terminales convén-cionales de voz, video, mensajería y tecnología WebRTC.

- Estudiar y documentar esta nueva tecnología en busca de:

  - *Fomentar e impulsar el uso de “estándares abiertos” de internet.*

  - *Fomentar e impulsar a la web estandarizada (W3C e IETF) como motor de comunicaciones de los usuarios de internet.*

  - *Ampliar las posibilidades de interacción entre usuarios de internet y los sistemas de comunicaciones convencionales.*

- *Fomentar e impulsar la convergencia de todos los tipos posibles de comunicaciones hacia estándares abiertos de internet, sin la necesidad de pagar por costosas soluciones privativas de software/hardware, ni tener que utilizar sistemas gratuitos pero acotados a subscriptores en donde se debe aceptar la intercepción de la información intercambiada en el sistema.*

- *Innovar en características como la movilidad y nomadismo, posibilidades de tele-trabajo y educación a distancia, a partir de implementar comunicaciones unificadas en cualquier sistema web (CRM, ERP, Campus Virtual, Páginas web, etc.).*

1.3.- Marco científico de la tesis

Esta Tesis se encuadra dentro de dos áreas de conocimiento:

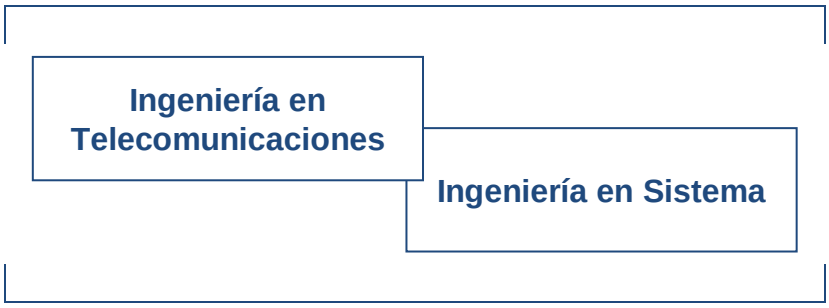


Figura 1.- Áreas de conocimiento.

Dentro del área de las Telecomunicaciones se obtienen las técnicas, medios de desarrollos, protocolos y tecnologías para lograr el funcionamiento de la nueva tecnología.

Y dentro del área de Sistemas, esta tesis se vincula con los procesos de desarrollo (instalación y configuración), simulación y diseño del producto. Con el objetivo principal de poder lanzarlo al mercado.

1.4.- Origen de la RTC

En este apartado se hace una reseña histórica en donde se resume el origen de las comunicaciones en tiempo real a través de navegadores web.

.- Introducciónn

World Wide Web desde la década de 1990, ha evolucionado de manera exponencial, adaptándose a las nuevas velocidades de enlace e integrándose paralelamente a los nuevos avances tecnológicos (por ejemplo, cámaras de video). Por lo tanto, si sumamos los avances y la creciente demanda de aplicaciones de integración para la WWW son algunas de las razones detrás de las Comunicaciones en Tiempo Real para la Web.

.- Orígen

El primer anuncio de la W3C sobre WebRTC se realizó en mayo de 2011 [1], que plantea el objetivo de definir un proyecto público para la implementaciónn de APIs del lado del cliente que permitan Real-Time Communications en los navegadores Web. El borrador público de W3C WebRTC se publicó el 27 de octubre de 2011 [2], en donde se especifica como enviar (audio y video) por medio de navegadores web a través de la red de comunicación a otros usuarios, sin el uso de ningún plugin o software externo.

El IETF se relaciona con el proyecto WebRTC en mayo de 2011 [3]. Los principales objetivos del grupo de trabajo IETF son la definición del modelo de comunicación, gestión de sesiones, la seguridad, la solución de NAT transversal, los formatos de medios, el acuerdo de códec y transporte de datos. Y en junio de 2011 Google lanzó públicamente el código fuente de su aplicación API WebRTC [4].

1.5.- Situación actual de la RTC

La tecnología WebRTC está ganando adeptos y marcando tendencia en el mundo de las comunicaciones. Grandes nombres de las comunicaciones e internet están apostando por dicha tecnología.

A continuación se citan algunas noticias que denotan la evolución y adopción de WebRTC en el mercado.

- El Servicio de Google Plus + Hangouts migró hacia WebRTC, dejando de ser necesario la instalación del Complemento para el navegador, que permitía utilizar antes el servicio.

- Firefox lanzó el servicio “Hello” basado en WebRTC, el mismo permite mantener una comunicación de video-llamada con cualquier usuario de Internet que acceda tanto desde un navegador Firefox como Google Chrome. *https://www.mozilla.org/es-AR/firefox/hello/.*

- Telefónica extendió su servicio de “TU Go” (*http://www.movistar.com.ar/tugo/*) hacia los navegadores Web utilizando WebRTC como tecnología. *http://go.tu.com*.

- Ericsson lanza al mercado “OpenWebRTC”. Se trata de un “Framework” para desarrollar aplicaciones WebRTC. *http://www.openwebrtc.io/.*

- Facebook migró su servicio de video-chat hacia WebRTC, permitiendo así la posibilidad de enviar mensajes de audio y video sin tener que instalar ningún complemento.

- AT&T pone a disposición su “API” para desarrollar aplicaciones WebRTC que puedan interactuar con su infraestructura de Carrier. *http://developer.att.com/apis/enhanced-webrtc.*

1.6.- Beneficios

Con el desarrollo e implantación de esta nueva tecnología de comunicación, se espera obtener los siguientes beneficios:

- *Dotar de mayor funcionalidad e integración entre los sistemas de información y los sistemas de comunicaciones de las organizaciones.*

- *Concentrar la tarea de administración de los sistemas de comunicaciones de una organización.*

- *Ahorrar dinero y espacio físico en cada estación de trabajo de una organización.*

- *Proporcionar servicios de video-llamadas y mensajería instantánea a sitios de internet como redes sociales, campus universitarios, páginas web de organizaciones que usan estándares abiertos.*

- *Lograr establecer comunicaciones telefónicas a costo cero (usando internet), para aquellas personas que dispongan con acceso a internet y que quieran comunicarse con organizaciones tanto públicas como privadas.*

- *Fomentar la convergencia de las comunicaciones hacia una arquitectura libre, estandarizada y personalizable.*

1.7.- Destinatarios

El trabajo final de grado estará dirigido a todas las empresas y personas que deseen introducirse en esta nueva tecnología de manera tal que puedan proponer y/o desarrollar soluciones que permitan incrementar las comunicaciones con su público objetivo brindando una manera más fácil, ágil, flexible, integrable y menos costosa frente a las tecnologías tradicionales.

Dicho trabajo final de grado puede ser el punto de partida para una empresa que desee encarar un diseño de una solución basada en WebRTC o una integración de WebRTC como una vía más de comunicación dentro de sus medios.

Empresas con aplicaciones basadas en tecnología web, como por ejemplo: help desk, ventas online, atención de clientes online, campus universitarios virtuales, etc.

También es un gran avance para aquellas empresas que están comenzando a migrar su infraestructura hacia una "Cloud Híbrida", ya que entre los servicios típicos como: correo electrónico, intranet, servidores de archivos, etc. a partir de implementar WebRTC se puede migrar a servicios de telefonía, video conferencias, mensajería instantánea, etc.

Finalmente, aquellos entusiastas que deseen comenzar a trabajar con la tecnología planteada, podrán encontrar en este trabajo final de grado un buen punto de partida, ya que el trabajo es descriptivo y todas las herramientas utilizadas son de naturaleza "Open Source" logrando que los costos de inversión sean nulos en cuanto a licencias de software.

1.8.- Índice comentado

Se comentan aquí, de forma general, los temas que se tratarán en cada capítulo de esta tesis y se resume el contenido de los mismos:

Capítulo 1.- Introducción:

Se presenta el tema a desarrollar, los objetivos a cumplir y se hace un desarrollo de los diferentes marcos en los que se ha desarrollado la Tesis.

Luego, se pone de manifiesto, de forma general los orígenes y la situación actual de las telecomunicaciones en el campo RTC a través de navegadores web.

En otro apartado de este capítulo se citan los beneficios que se esperan obtener con el desarrollo de la tesis.

Se continúa con el apartado “destinatarios” en donde se describe para quienes está dirigido el desarrollo de esta nueva tecnología.

Se concluye con una síntesis sobre el contenido de cada capítulo.

Capítulo 2.- Estudio técnico:

Contiene una descripción breve de las distintas Tecnologías, Protocolos y procesos que se utilizaran luego para el desarrollo de la tecnología planteada. Este capítulo contiene la parte teórica de la Tesis.

Capítulo 3.- Desarrollo de la tecnología webrtc:

Se sintetizan con dos diagramas conceptos teóricos del Capítulo 2 y luego se expone un diagrama en bloque con el sistema propuesto y varios diagramas que detallan el funcionamiento de este sistema propuesto.

Tras ello se describe en forma detallada el diseño de la tecnología WebRTC. Se plantea un escenario de comunicación en tiempo real, en donde se configura e instala cada uno de los componentes, de modo tal que le permita a los desarrolladores, bajo dichas directrices poder desarrollar e implementar esta nueva tecnología.

Se continua el capitulo con las pruebas de comunicaciones Real-Time que demuestran que el sistema propuesto funciona.

Después se citan los resultados alcanzados.

Finalmente se esbozan las dificultades que se han presentado y las actividades realizadas en el desarrollo del Informe Final de Grado.

Capítulo 4.- Viabilidad comercial:

Contiene un análisis sobre la viabilidad comercial de la nueva tecnología de comunicación.

Capítulo 5.- Conclusiones:

Se resume, desde un enfoque totalmente técnico los aportes relevantes alcanzados, se hacen recomendaciones para desarrolladores que deseen implementar la tecnología planteada y finalmente se hace una percepción sobre la tecnología WebRTC.

Capítulo 6.- Bibliografía:

Contiene la bibliografía más relevante que se ha consultado y a la que es oportuno acceder en caso de querer ampliar la información aquí presentada.

Capítulo 7.- Anexos:

Contiene los anexos de esta tesis, en donde se amplía la información a modo de complemento de lo que hay en cada capítulo.

Referencias:

[1] Web Real-Time Communications Working Group. *http://www.w3.org/2011/04/webrtc/*, Mayo 2011.

[2] Adam Bergkvist, Daniel C. Burnett, Cullen Jennings, and Anant Narayanan. WebRTC 1.0: Real-time Communication Between Browsers. *http://www.w3.org/TR/2011/WD-webrtc-20111027/*, Octubre 2011.

[3] Magnus Westerlund, Cullen Jennings, and Ted Hardie. Real-Time Communication in WEB-browsers charter. *http://tools.ietf.org/wg/rtcweb/charters?item=charter-rtcweb-2011-05-03.txt*, Mayo 2011.

[4] Google release of WebRTC source code. *http://lists.w3.org/Archives/Public/public-webrtc/2011May/0022.html*, Junio 2011.

CAPÍTULO 2.- ESTUDIO TÉCNICO

En este capítulo se definen las distintas técnicas, tecnologías, protocolos, normas y estándares que se usan para el desarrollo de la alternativa de comunicación que se planteada, se resalta sus funcionamientos, utilidades y aplicabilidad.

Por último, se presentan los recursos de hardware y de software, que se utilizan en la arquitectura de comunicación.

2.1.- Estándares y protocolos a utilizar

Las **normas y estándares** le permiten a los desarrolladores disponer de documentos técnicos que definen y estandarizan las tecnologías y protocolos de comunicación, y a su vez, evitan problemas de licencias de códec. El desarrollo del informe final, está basado en la IETF y en W3C.

IEFT: *El Grupo de Trabajo en Ingeniería de Internet, es una gran comunidad internacional abierta de diseñadores de red, operadores, proveedores e investigadores con la misión de hacer que Internet funcione mejor mediante la producción de documentos técnicos (RFC).*

RFC: *son una serie de publicaciones del (IETF) que describen diversos aspectos sobre el funcionamiento de Internet (protocolos, redes de computadoras, procedimientos, etc.).*

W3C: *El Consorcio World Wide Web, ayuda a que la Web alcance su máximo potencial mediante el desarrollo de normas que promuevan su evolución y que aseguran la interoperabilidad. A su vez, define las APIs para que los desarrolles Web utilicen en sus aplicaciones.*

API: *Una* interfaz de programación de aplicaciones, es *un conjunto de rutinas, estructuras de datos, clases de objetos, y variables para la creación de aplicaciones y protocolos, y que son implementadas como bibliotecas. Estas API permiten mayor versatilidad y facilidad al desarrollar programas, porque los programadores no tiene la necesidad de programar desde un principio cada detalle.*

Los **protocolos** utilizados para el desarrollo de esta alternativa, son los siguientes:

Protocolos		**Especificación**
DTLS	*Datagram Transport Layer Security*	RFC 6347
HTTP	*Hyper-Text Transport Protocol*	RFC 2616
ICE	*Interactive Connectivity Establishment*	RFC 5245
IP	*Internet Protocol version 4 y version 6*	RFC 791 y 2460
SCTP	*Stream Control Transport Protocol*	RFC 2960
SDP	*Session Description Protocol*	RFC 4566
SIP	*Session Initiation Protocol*	RFC 3261
SRTP	*Secure Real-Time Transport Protocol*	RFC 3711
STUN	*Session Traversal Utilities for NAT*	RFC 5389
TCP	*Transmission Control Protocol*	RFC 793
TLS	*Transport Layer Security*	RFC 5246
TURN	*Traversal Using Relays around NAT*	RFC 5766
UDP	*User Datagram Protocol*	RFC 768
	The WebSocket Protocol	RFC 6455

A continuación se describe cada protocolo:

DTLS: *Es un protocolo que proporciona privacidad en las comunicaciones en UDP. En WebRTC, este protocolo permite a las aplicaciones cliente/servidor comunicarse de manera que se eviten las escuchas no deseadas (eavesdropping), accesos no permitidos, o modificación de mensajes. El protocolo DTLS está basado en el protocolo TLS y proporciona garantías de seguridad equivalentes. La semántica de los datagramas de los protocolos subyacentes no es modificada al utilizar DTLS.*

DTLS se utiliza para intercambiar la clave utilizada para la encriptación SRTP de la "media" intercambiada.

HTTP: *Es el protocolo utilizado en cada transacción realizada en la World Wide Web.*

ICE: *Protocolo implementado en los clientes WebRTC para intentar resolver la dirección IP correcta para iniciar una transacción WebRTC en ambientes de NAT.*

SCTP: *Stream Control Transport Protocol, RFC 2960.*

SDP: *Es un protocolo para describir los parámetros de inicialización de los flujos multimedia. Corresponde a un protocolo actualmente redactado en el RFC 4566 por la IETF.*

SIP: *Es un protocolo de señalización creado para administrar sesiones multimedia entre dos o más participantes.*

SRTP: *Es un protocolo que define un perfil de RTP con la intención de proporcionar cifrado, autenticación del mensaje e integridad. Estamos hablando de criptografía de clave simétrica.*

STUN: *Encuentra la IP pública que "nateó" al ordenador que genera el SDP y cambia esa dirección privada con la IP pública que descubrió.*
Resuelve casi todos los tipos de NAT.

TCP: *Protocolo de Internet responsable de gran parte de las transacciones de datos que ocurren en Internet. Es un protocolo orientado a la conexión, capaz de dar ciertas garantías de la información que transporta.*

TLS: *Protocolo criptográfico para proporcionar comunicaciones seguras por una red, comúnmente Internet.*

TURN: *Cuando STUN no alcanza, directamente se envía la media a este servidor y el la reenvía al destino, es un media server. Es caro y crea delay.*

UDP: *Protocolo de Internet responsable de gran parte de las transacciones de datos que ocurren en Internet. Es un protocolo de transporte no orientado a la conexión, es decir que no hay garantías sobre los datos que transporta.*

2.2.- Tecnologías

2.2.1.- Webrtc

2.2.1.1.- Introducción

WebRTC (Web Real-Time Communication) es una tecnología de código abierto que permite añadir capacidades multimedia a los navegadores Web. Básicamente consta de una API y un conjunto de protocolos que hacen posible que navegadores web “de serie” puedan soportar comunicaciones de voz, video, mensajería instantánea, compartición de pantalla y transferencia de archivos P2P (peer to peer), entre otros servicios sin la necesidad de instalación de software complementario. Con WebRTC se pueden desarrollar sistemas de comunicaciones utilizando tan solo los navegadores web como “endpoints” de la comunicación.

Otro aspecto significativo en la utilización de este tipo de comunicación es que una aplicación web que incluya WebRTC como tecnología, le permite al navegador web poder acceder a la cámara y micrófono del sistema (PC, notebook, tablet o smartphone), para luego mediante técnicas de señalización poder enviar dichos parámetros a la otra parte de la comunicación (el otro navegador web) y entonces abrir una negociación para poder establecer, mantener y finalizar una sesión multimedia entre navegadores web.

Finalmente, es importante recalcar que WebRTC no es algo “llave en mano”, es decir, no es un sistema listo para utilizar como lo son Skype, Google Hangout, etc. Sino que se trata más de un “framework” con herramientas que permiten construir sistemas de comunicaciones de última generación basados en tecnología web. WebRTC cuenta con las garantías de una API bien documentada y respaldada por importantes entidades tecnológicas como son el W3C, IETF y Google. Además es de código libre en todos sus niveles, a diferencia de otros sistemas como los plugins flash de Adobe o aquellos sistemas basados en códec de audio/video propietarios.

2.2.1.2.- Arquitectura de WebRTC

WebRTC ofrece comunicación multimedia en tiempo real, y al ser de código abierto permite a desarrolladores la capacidad de implementar nuevas aplicaciones. La arquitectura de WebRTC es expone en la siguiente figura:

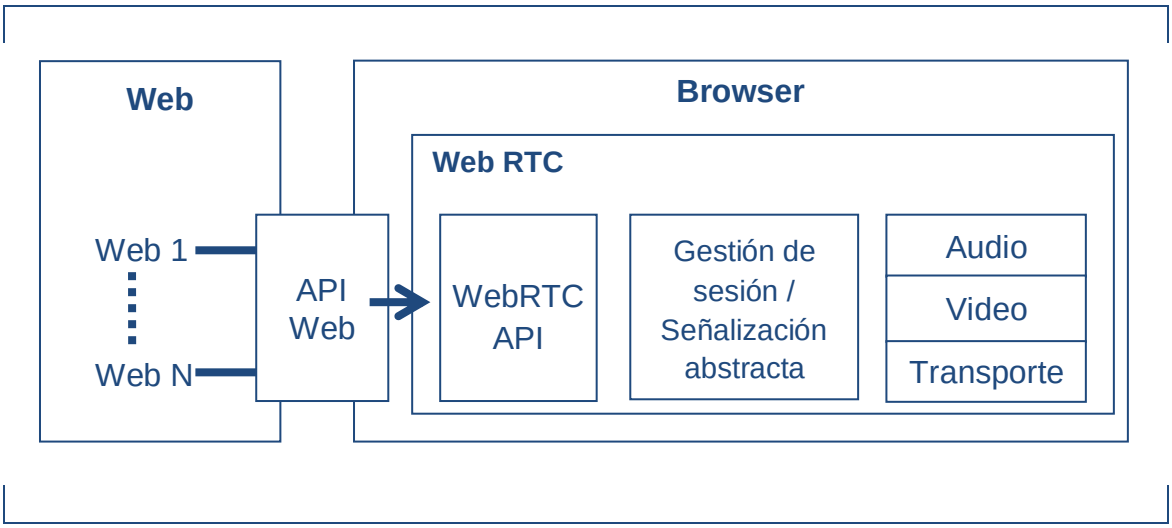


Figura 2.- Arquitectura WebRTC.

El esquema de la Arquitectura WebRTC, se divide en dos secciones que componen la tecnología WebRTC y estas son:

Web: *Capa avocada a los desarrolladores de aplicaciones WebRTC. Se tratan de APIs javascript mediante las cuales los desarrolladores disponen métodos ágiles para construir aplicaciones WebRTC abstrayéndose del tipo de sistema operativo y/o navegador que consuma dicha aplicación.*

Browser: *stack WebRTC que implementa un navegador web que soporta dicha tecnología.*

A su vez, en la arquitectura WebRTC, la aplicación Web se ejecuta por desarrolladores que desean ofrecer aplicaciones multimedia en tiempo real, y que pueden utilizar los Web API para navegadores. Web API, es un API que habilita fácilmente las propuestas de Web multimedia de desarrolladores. Y WebRTC API permite a los desarrolladores de navegadores, implementar más fácilmente las propuestas de Web API. La aplicación WebRTC se encuentra en el servidor del servicio y se ejecuta en el navegador.

Por su parte la Gestión de Sesión es un nivel abstracto, que permite la gestión de llamadas y a su vez, una gestión por medio de señalización, que le permite a los desarrolladores de aplicaciones la decisión del protocolo de implementación. El transporte de Media en WebRTC es seguro desde su diseño por lo tanto se realiza a través de SRTP.

Y por último se encuentran los motores de voz y video que son marcos referenciales, tanto para el audio desde el micrófono y el de video de la cámara, hacia la red, y desde la red hacia los parlantes y la pantalla.

En el siguiente esquema se detallan los elementos de cada motor de A/V y de transporte:

Audio	**Video**	**Transporte**
Códec iSAC / iLBC	Códec VP8	SRTP
Buffer para jitter NeTEQ	Buffer para jitter de Video	Multiplexación
AEC / NC	Mejora de imagen	STUN + TURN + ICE
Captura de Audio	*Captura de Video*	*Red I/O*

A continuación, se describen cada uno de los componentes que forman parte del A/V y transporte para una comunicación Web RTC.

.- Audio

El motor de audio, debe ser capaz de tomar el audio de la placa de sonido y transmitirlo hacia la red.

Los códecs de audios soportados por la tecnología WebRTC son:

- iSAC: *El Internet Speech Audio Códec (iSAC) es un códec de banda ancha especialmente diseñado para trabajar con audio en formato de voz. Es un códec adecuado para aplicaciones VoIP y audio en streaming.*

- iLBC: *El Internet Low Bitrate Códec (iLBC) es un códec de banda estrecha diseñado para aplicaciones de audio en formato de voz. Aunque esté diseñado para voz principalmente, también resulta un buen códec para audio en streaming.*

  El principal atractivo de iLBC recae en su capacidad para mantener la calidad aunque se produzcan pérdidas de muestras. Una pérdida de muestra se produce cuando se pierde o se retrasa un paquete IP.

Además de los codecs, otros componentes son necesarios para lograr comunicaciones de voz. Esto incluye el software basado en la cancelación de eco acústico, el control automático de ganancia y la reducción del ruido.

- Buffer para jitter NeTEQ: *Se dispone de un jitter buffer para contrarrestar los efectos negativos del jitter y la pérdida de paquetes.*

- Cancelador de Eco: *Se dispone de un software cancelador de eco, para contrarrestar los efectos del eco causado por la retroalimentación (AEC, Acoustic Echo Canceler).*

- Reductor de Ruido: *Se dispone de un software reductor de ruidos (NR, Noise Reduction).*

.- Video

Motor de video, debe ser capaz de tomar el video de la webcam y enviarlo por la red y/o tomar el video de la red y dibujarlo en pantalla.

Codecs: Aún no está definido que códec de video utilizar, las opciones están entre VP8 y H264, pero si se tiene en cuenta que VP8 es mucho más fácil de trabajar en términos de patentes, ya que es un estándar libre de regalías (H.264 está lleno de patentes de una serie de compañías, incluyendo MPEG LA). De esta manera, VP8 es una alternativa de desarrollo mucho menos costoso, y por lo tanto atractivo para los desarrolladores.

- VP8: *fue diseñado para manejar el formato de imagen utilizado por la mayoría de los vídeos en la web: 4:2:0 con color de 8 bits por canal de profundidad, escaneo progresivo (no entrelazado) y dimensiones de imagen 16383x16383 píxeles.*

WebRTC junto con los Códec, incluye componentes para ocultar la pérdida de paquetes, limpiar las imágenes ruidosas, así como capacidades de captura y reproducción a través de múltiples plataformas.

- Buffer para jitter de Video: *Al igual que en el contexto del audio, se dispone de un jitter buffer para el tratamiento del video y la reducción de los efectos de la pérdida de paquetes y jitter.*

- Mejora de imagen: *Por ejemplo, puede remover el "ruido" captado desde la webcam.*

.- Transporte

- RTP Stack: *Soporte del protocolo RTP y SRTP (Secure RTP) para el intercambio de "media".*

- P2P support: *Soporte para los protocolos STUN, TURN y la tecnología ICE como métodos para afrontar los posibles problemas de NAT.*

2.2.1.3.- El funcionamiento de WebRTC

El funcionamiento de WebRTC, se puede observar en la siguiente figura:

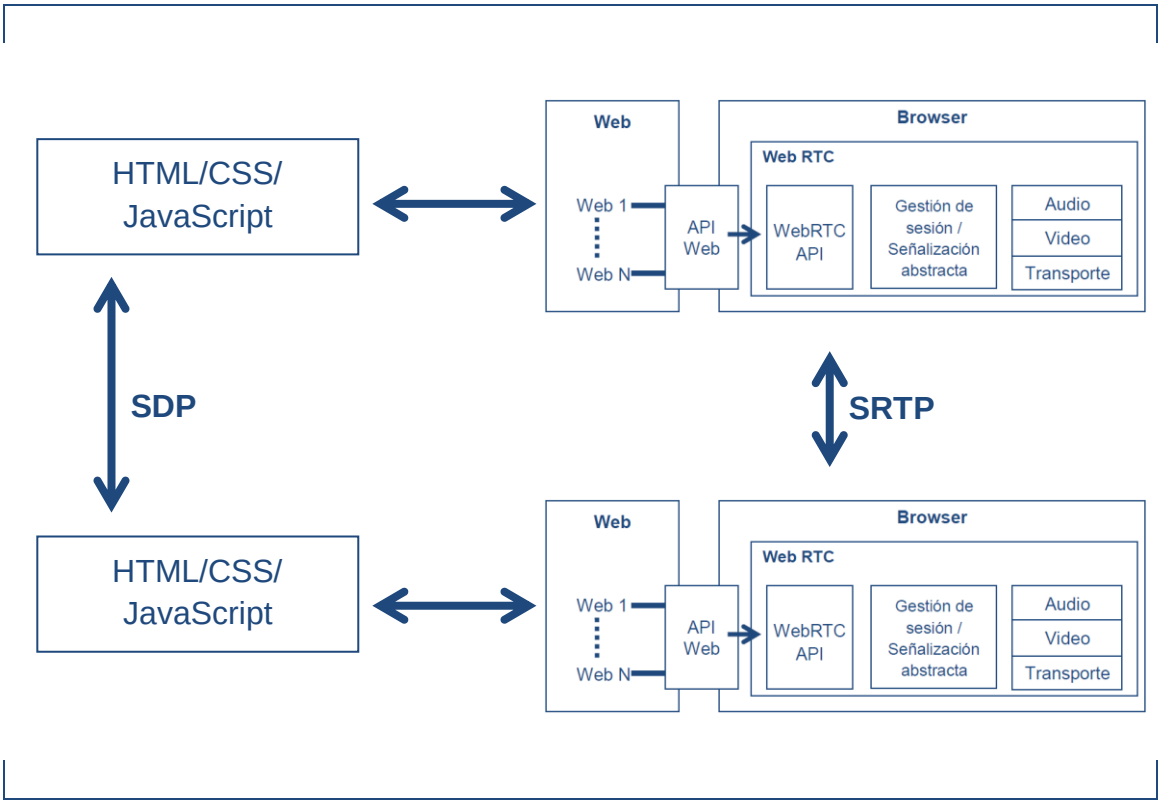


Figura 3.- Funcionamiento de WebRTC.

2.2.1.4.- Señalización

A la hora de intercambiar el mensaje SDP entre los navegadores, se necesita pasar por la etapa de señalización, que debe ser implementada por los desarrolladores del sistema de comunicaciones que utilicen WebRTC. El proceso de señalización es abstracto, por tanto, se puede usar cualquier mecanismo y protocolo para el intercambio.

Por señalización se asume como método para el intercambio de cuatro tipos de información:

- Mensajes de control de sesión: *Para iniciar, mantener y finalizar la comunicación y además reportar errores.*

- Configuración de Red: *Se debe poder informar acerca de la dirección IP y puertos del ordenador donde corre el navegador.*

- Capacidades multimedia: *Se debe poder informas qué codecs soporta el navegador y el orden de preferencias de los mismos.*

- Registros y localización de usuario: *Para poder ubicar a un usuario de la red WebRTC, éste previamente tuvo que haber informado su ubicación mediante un mensaje de registro en la red.*

Por lo tanto, es una aplicación web en Javascript y se puede obtener un paquete SDP con todos los parámetros necesarios para negociar una sesión de media con el otro extremo. Entonces, se debe hacer llegar al otro extremo de la comunicación este SDP.

Desde el primer navegador hasta el http server, no habría inconvenientes, existen métodos del mundo web como: *HTTP Post o AJAX.*

Con eso se hace llegar el SDP desde un peer hasta el http server, pero ahora éste debe re-transmitirse desde el host webserver al host donde corre el otro navegador.

¿Cómo se podría hacer, en un contexto web HTTP (arquitectura cliente-servidor), donde no se considera una conexión bidireccional y asíncrona entre las partes involucradas (navegador – servidor web – navegador), es decir no es el servidor http quien "le entrega" algo al navegador sin que éste ultimo haya demandado algo?

OPCIONES:

- HTTP Long polling: *El cliente hace consultas periódicas al servidor, hasta que cuando éste tiene el SDP se lo entrega. Es una técnica ineficiente, recarga el navegador y la red con peticiones innecesaria.*

- Adobe Flash: *Permite una comunicación bidireccional entre el navegador y el servidor web.*

- Websocket: *Se trata de un nuevo protocolo (RFC 6455) para navegadores, que permite establecer una conexión TCP con el servidor y a partir de allí se tiene una comunicación bidireccional en real-time entre el navegador y el servidor por donde vamos a poder enviar lo que se desee, por ejemplo embeber mensajes de señalización SIP (Session Initiation Protocol). De hecho la implementación de Websockets como transporte del protocolo SIP está en draft (RFC 7118) y ya se cuentan con implementaciones.*

Por lo tanto una de las alternativas firmes para realizar la señalización en WebRTC es la de reutilizar el protocolo SIP en un contexto web utilizando Websockets como transporte de los mensajes SIP. El cual será el método adoptado para implementar señalización en WebRTC a lo largo de la tesis.

2.2.1.5.- WebRTC y el NAT

En el proceso de establecimiento de una comunicación, el navegador va a entregar su descripción (codecs, puertos para el streaming, dirección IP) para la sesión, respecto a la dirección IP se deben considerar dos escenarios:

1.- Comunicación de peers dentro de la LAN

En este ambiente, la dirección IP seteada por el stack WebRTC del navegador en el SDP que debe ser transmitido hacia el otro extremo es naturalmente la dirección IP del host donde precisamente corre el navegador web que está negociando la comunicación, es decir; hay una relación lógica coincidente entre la IP de capa tres del modelo OSI (dirección de red) y la dirección IP del stack WebRTC, ósea capa siete (capa de aplicación). Entonces al otro peer de la comunicación le llegará el SDP y éste último podrá contestar el mensaje enviando sus capacidades multimedia a la dirección IP, ya que el stack WebRTC (capa de aplicación) del nodo destino es quien ordena responder a dicha dirección IP y por ende el stack de red envía el paquete de respuesta hacia la dirección IP indicada y como esta es una dirección LAN, es alcanzable y se puede responder.

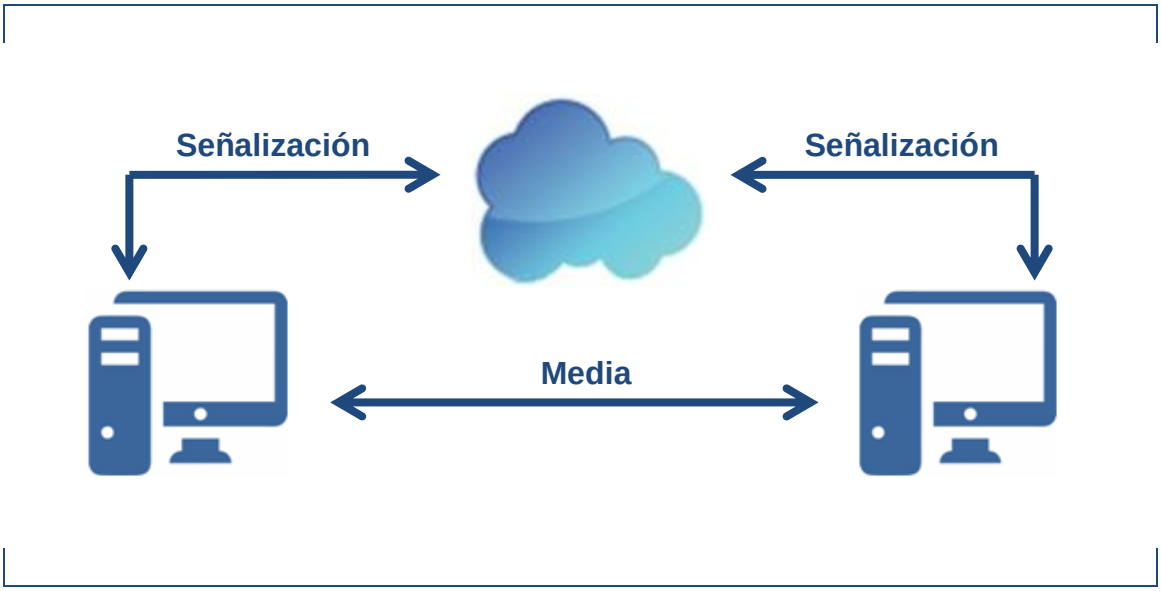


Figura 4.- Comunicación, sin NAT.

2.- Comunicación de peers en un ambiente de NAT

Bajo este contexto, la dirección IP seteada por el stack WebRTC del navegador en el SDP que debe ser transmitido hacia el otro extremo es la dirección IP pública del enrutador que finalmente enruta el paquete IP (SDP como carga útil) hacia el otro peer, en lugar de usar la dirección IP local del host en donde está corriendo, lo cual generaría una inconsistencia entre el valor "dirección IP" en capa siete (IP privada local) y el valor de capa tres (IP pública con la que el paquete sale a Internet), ya que a la aplicación destino le llegaría un SDP con el campo dirección IP con un valor de una IP privada que al momento de intentar avanzar con la negociación de la sesión buscaría contestar a dicha IP privada y obviamente cuando se intente sacar ese paquete hacia el stack de red del host, éste no podría localizar una dirección IP privada.

Quién proporciona la inteligencia para asumir un ambiente de NAT e intentar resolver dichas inconsistencias es el stack STUN/TURN/ICE que viene integrado en el estándar WebRTC. Ya vienen nativamente soportados los protocolos: STUN, TURN y el framework ICE.

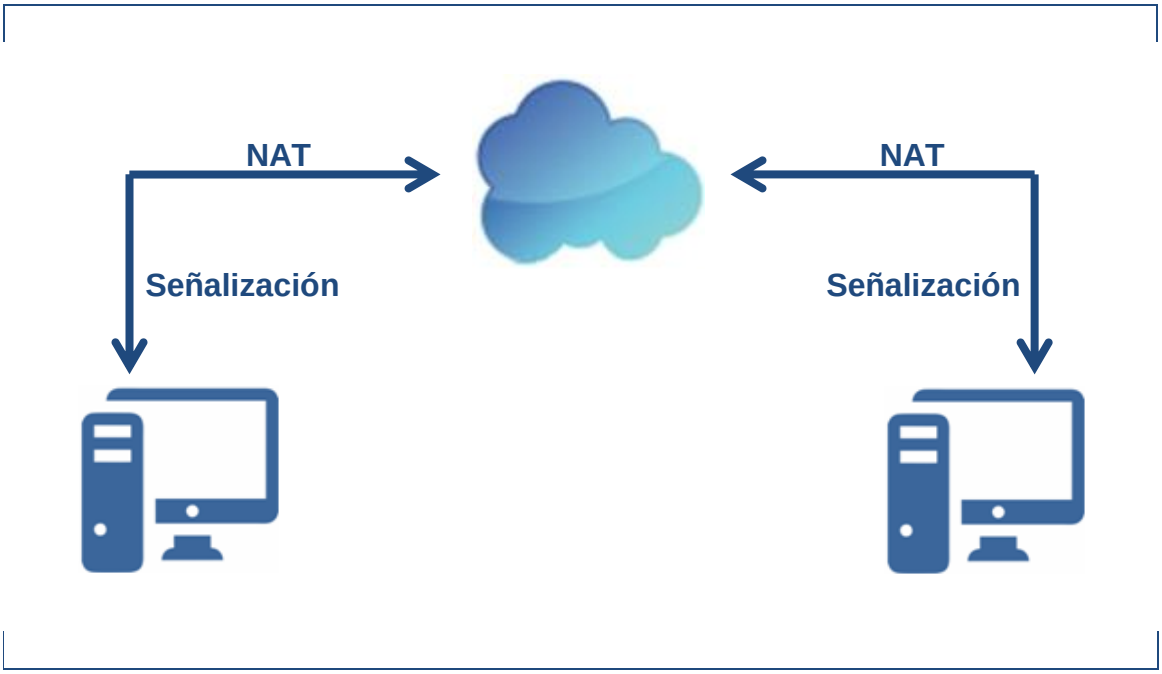


Figura 5.- Comunicación, con NAT.

2.2.1.6.- Seguridad

Siguiendo el flujo natural de una comunicación WebRTC, se van a plantear aspectos de seguridad que proporciona la tecnología en cada una de las etapas de una comunicación:

1.- Acceso a la aplicación WebRTC: *el acceso a la aplicación web puede ser utilizando el protocolo http cifrado HTTPS.*

2.- *Una vez que se accede a la aplicación, en el momento de invocar los recursos (webcam y micrófono) ya sea cuando se origina una comunicación o cuando se recibe una, en el instante de precisar la webcam y micrófono para comenzar la comunicación, WebRTC pregunta SIEMPRE al usuario si éste mismo autoriza el acceso a la webcam y micrófono.*

3.- Streaming: *WebRTC utiliza DTLS y SRTP por defecto (RFC 6347 y RFC 3711). Proporcionando así un entorno seguro para el intercambio de información.*

A continuación se describen las tecnologías sobre las cuales se asienta y/o interactúa WebRTC.

2.2.2.- La web

Cuando en un navegador web se escribe una dirección web (URL), el navegador establece una conexión HTTP con el servidor web indicado, en la cual generalmente el navegador solicita una página web (por ejemplo: http://www.iua.edu.ar/index.html), es decir un recurso del servidor web, entonces si este existe y está disponible, el servidor contesta la solicitud enviando el contenido solicitado.

Los navegadores web son las aplicaciones cliente que las computadoras utilizan para conectar con la Word Wide Web y los recursos almacenados en un servidor web. Como es habitual el servidor web es tan solo un servicio en segundo plano que implementa el protocolo HTTP y deja disponible una “página web”.

En este proceso de solicitar una página, el navegador inicia la conexión con el servidor, realiza la solicitud de página y queda esperando una respuesta del servidor, si todo es correcto el servidor aprueba la solicitud y entonces procesa la misma y allí responde al navegador enviando los datos solicitados, obviamente los datos intercambiados son ilegibles para el ser humano, quien tiene la tarea de interpretar los datos que envía el servidor web y presentarlos en una forma humanamente legible es el navegador web. Los navegadores pueden interpretar y presentar muchos tipos de datos, como texto sin formato o código HTML (el lenguaje con el que se construyen las páginas web), JavaScript, XML, etc. Sin embargo los datos soportados por un navegador pueden ser extendidos más allá de los habituales, generalmente las capacidades se extienden mediante la instalación de plug-in o add-on (complemento).

A continuación se detalla cómo es el acceso a una página web.

Se introduce la URL: **http://www.iua.edu.ar/fi/index.php**

Figura 6.- Acceso a una página web.

En primer lugar, el navegador interpreta las tres partes del URL (uniform resource locator, localizar uniforme de recursos).

http	*Es el protocolo indicado*
www.iua.edu.ar	*Es el nombre del servidor*
/fi/index.php	*Es el nombre de archivo específico solicitado*

El navegador se apoya en un servidor de nombres (DNS) para resolver la dirección IP del servidor web llamada “iua.edu.ar” y así lanzar el pedido de conexión contra el servidor HTTP que aloja la página solicitada. Utilizando el protocolo HTTP el navegador envía una solicitud GET al servidor pidiendo el archivo “index.php”. El servidor envía una respuesta HTTP y el contenido de la página index.php (contenido HTML, JavaScript, etc.), para que finalmente el navegador se encargue de descifrar la codificación recibida como dato y la presente en el formato de “página web” que conocemos en la ventana de visualización.

2.2.3.- HTML5

HTML es el lenguaje para describir la estructura de las páginas Web, donde “Hypertext” es cualquier texto presentado en un dispositivo electrónico, sea este un teléfono inteligente, tableta, o algún otro capaz de entender este tipo de contenido, en el cual este texto tiene un hipervínculo con otro texto que puede estar en la misma página Web, en otra del mismo sitio, o en una completamente diferente. El Hipertexto es tal vez lo que define la esencia del Internet, la habilidad de vincular una página Web con otra, creando así una telaraña de información. Mientras que “MarkupLanguage” toma este texto plano y con la ayuda de códigos adicionales o etiquetas, lo vuelve un texto de fácil lectura, al poder cambiar el estilo de la presentación, e incluso introducir características multimedia a las páginas Web.

World Wide Web Consortium (W3C) es la organización responsable de la creación de las especificaciones para HTML. Donde la versión más reciente, HTML5, está aún en desarrollo, pero cerca ya de completarse. Esta nueva versión de HTML intenta brindar un soporte más sólido a los entornos multimedia de la Web de hoy en día, mientras guarda su compatibilidad con versiones anteriores. Aunque HTML5 no ha sido finalizado, casi todas sus etiquetas pueden ser utilizadas con seguridad en las páginas Web de hoy.

Otro aspecto importante es que HTML5 además de utilizar las características existentes de HTML, también hace uso de las Cascading Style Sheets versión 3 (CSS3) y de JavaScript. Donde CSS3 provee presentación y JavaScript provee interactividad.

2.2.4.- JavaScripts

JavaScript es un lenguaje de programación interpretado. Se define como orientado a objetos, basado en prototipos, imperativo, débilmente tipiado y dinámico.

Se utiliza principalmente en su forma del lado del cliente (client-side), implementado como parte de un navegador web permitiendo mejoras en la interfaz de usuario y páginas web dinámicas aunque existe una forma de JavaScript del lado del servidor (Server-side JavaScript o SSJS). Su uso en aplicaciones externas a la web, por ejemplo en documentos PDF, aplicaciones de escritorio (mayoritariamente widgets) es también significativo.

Que se clasifique como “lenguaje interpretado”, implica que no es necesario compilar los programas para ejecutarlos. En otras palabras, los programas escritos con JavaScript se pueden probar directamente en cualquier navegador sin necesidad de procesos intermedios.

Cuando se dice Javascript, es usado “Del lado del cliente”, significa que cuando se ve una página que utiliza JavaScript, se descargo el código JavaScript al navegador y el navegador lo está ejecutando de acuerdo con las acciones realizadas en la página.

Todos los navegadores modernos interpretan el código JavaScript integrado en las páginas web.

2.2.5.- CCS3

CSS u Hojas de estilo en cascada es un mecanismo utilizado para agregar estilo que incluye tipo de fuente, color y diseño a los documentos Web. También se puede definir como un lenguaje de hojas con estilo que se utiliza para describir el aspecto y el formato de un documento escrito en un lenguaje de marcas, la aplicación más común son las páginas Web escritas en HTML y XHTML. CCS fue diseñado principalmente para separar el contenido de un documento, del modelo de su presentación. La separación mejora la accesibilidad de los contenidos, proporciona flexibilidad y control de las características de presentación y permite a varias páginas compartir el mismo formato, tanto para reducir la complejidad y la repetición en el contenido estructural.

Se considera que HTML contiene las estructuras, mientras que CSS da el formato al contenido estructurado.

2.2.6.- jsSIP

Se trata de una librería Javascript que implementa clases con métodos que implementan los diferentes mensajes SIP “Métodos y Respuestas”, usando una conexión persistente Websocket como transporte para dichos mensajes. Usando dicha librería se pueden construir un completo “User Agent” WebRTC (llamadas de audio y video y mensajería instantánea).

2.2.7.- sipML5

Al igual que JSSIP, sipML5 es una librería Javascript que implementa clases con métodos que implementan los diferentes mensajes SIP “Métodos y Respuestas”, usando una conexión persistente Websocket como transporte para dichos mensajes. Al usar dicha librería se pueden construir un completo “User Agent” WebRTC (llamadas de audio y video, mensajería instantánea, compartir pantalla).

2.2.8.- Gnu/Linux

Es un sistema operativo de propósitos generales cuyo nombre denota la convergencia de dos proyectos: *el núcleo Linux y las aplicaciones GNU*. Ambos proyectos se ensamblaron formando el sistema operativo que hoy se conoce y utiliza a nivel mundial.

2.2.9.- Apache, Asterisk y Kamailio

Apache: Es un servidor web HTTP de código abierto que permite poner a disposición de los clientes web (navegadores web) múltiples aplicaciones web, mediante la implementación del protocolo HTTP y el concepto de sitios virtuales.

Asterisk: Es un framework de código abierto que permite construir aplicaciones de comunicaciones basados en software. Es decir que cualquier arquitectura x86 puede convertirse en un servidor de comunicaciones mediante configuración y parametrización del framework Asterisk.

Kamailio: Es un SIP Server de código abierto.

2.3.- Protocolos

2.3.1.- El protocolo HTTP

HTTP es un protocolo de capa de aplicación, define la sintaxis y la semántica que utilizan los elementos de software de la arquitectura web (clientes, servidores, proxies) para comunicarse. Es un protocolo orientado a transacciones y sigue el esquema petición-respuesta entre un cliente y un servidor. Al cliente que efectúa la petición (un navegador web o un spider) se lo conoce como "user agent" (agente del usuario). A la información transmitida se la llama recurso y se la identifica mediante un localizador uniforme de recursos (URL). Los recursos pueden ser archivos, el resultado de la ejecución de un programa, una consulta a una base de datos, la traducción automática de un documento, etc.

El protocolo define mensajes de Solicitud/Respuesta. Dentro de las solicitudes más comunes tenemos: GET, POST y PUT

GET *es una solicitud cliente para datos. Un navegador web envía el mensaje GET para solicitar una página de un servidor web.*

POST *se utilizan para enviar mensajes que suben datos a un servidor web. Por ejemplo cuando el usuario introduce datos en un formulario incrustado en una página web, cuando el usuario selecciona "Submit", es decir envía el formulario, el navegador le pasa la información del formulario al servidor web, utilizando un mensaje POST.*

Desde el lado del servidor, cada solicitud procesada por el servidor es contestada con un mensaje de respuesta (Código de respuesta). Estos códigos de estatus están especificados por el RFC 2616, y algunos fragmentos en los estándares RFC 2518, RFC 2817, RFC 2295, RFC 2774 y RFC 4918; otros no están estandarizados, pero son comúnmente utilizados.

HTTP es un protocolo sin estado, es decir, que no guarda ninguna información sobre conexiones anteriores. El desarrollo de aplicaciones web necesita frecuentemente mantener estado. Para esto se usan las cookies, que es información que un servidor puede almacenar en el sistema cliente. Esto le permite a las aplicaciones web instituir la noción de "sesión", y también permite rastrear usuarios ya que las cookies pueden guardarse en el cliente por tiempo indeterminado.

2.3.2.- Websockets

2.3.2.1.- Introducción

Websockets es una tecnología que proporciona un canal de comunicación bidireccional sobre un único socket TCP. Es ideal para implementarse sobre arquitecturas web (navegadores y servidores web) ya que permite el intercambio de mensajes en ambos sentidos rompiendo el paradigma de de la web “cliente-servidor”, permitiendo así un intercambio bidireccional y asíncrono de mensajes entre navegadores y servidor web. Se trata de una especificación de HTML5, por lo tanto está disponible y puede ser implementada en las aplicaciones web e interpretada por navegadores. En nuestro contexto es la técnica seleccionada para transportar la señalización SIP entre los endpoints y servidores, que implementa nuestro sistema de comunicaciones.

Está designado para suceder a las tecnologías que proporcionan comunicación bidireccional usando el protocolo HTTP como capa de transporte.

2.3.2.2.- Descripción general

El protocolo consta de dos partes:

1.- *Establecer la conexión* (handshake).

2.- *La transferencia de datos.*

Para establecer una conexión WebSocket, el cliente manda una petición de negociación WS, y el servidor manda una respuesta de negociación WS.

Si la negociación fue exitosa entre el cliente y el servidor, se inicia la transferencia de datos. Por lo tanto, se logra un canal de comunicación bidireccional, donde cada parte puede, independientemente de la otra, enviar datos a voluntad.

2.3.2.3.- Ventajas

Esta tecnología permite intercambiar información entre el cliente y el servidor cuando cualquiera de las dos partes así lo requiera, sin necesidad de que el cliente esté sondeando al servidor, y con la ventaja extra de que las partes pueden enviar información al mismo tiempo por el mismo canal ya que al ser una conexión Full Duplex los mensajes no “chocan”.

Otras ventaja, Websockets es un estándar de HTML5, la API de WebSocket está siendo normalizada por el W3C, y el protocolo WebSocket, a su vez, está siendo normalizado por el IETF lo que hace que esta tecnología no tenga problemas de compatibilidad.

Por último, con la implementación de esta tecnología se reduce el tráfico innecesario en la red, gracias a que no hay paquetes de datos que solo “preguntan” al servidor si hay información para que este envíe al cliente. Eso sin contar los paquetes enviados por el cliente que terminan siendo inútiles porque el servidor no tiene información para enviar.

2.3.2.4.- Modelo de seguridad

El protocolo websocket utiliza el modelo de “origen” usado por los navegadores web, para restringir qué páginas web pueden contactar al Websocket server, cuando se utiliza el protocolo Websocket desde una página web.

Además, mediante el handshake, se asegura de que solamente un cliente websocket este en el otro extremo, y es con quien se va a establecer la conexión e intercambiar los datos. Evitando así que cualquier cliente STMP o HTTP (por ej.) envíe data al websocket server.

2.3.2.5.- Relación con el TCP y HTTP

El protocolo Websocket es un protocolo que usa TCP como protocolo de transporte, la única relación con el HTTP es que su Handshake es interpretado por servidores HTTP, más específicamente mediante una solicitud Upgrade por parte del cliente.

Por defecto el protocolo websocket espera conexiones estándar en el puerto 80 y seguras (usando el protocolo TLS) sobre el puerto 443.

2.3.3.- El protocolo SIP

2.3.3.1.- Introducción

SIP es un protocolo que aporta lo imprescindible para poder establecer una sesión (voz, video, juegos interactivos, compartición de datos) entre dos equipos de Internet, utilizando una filosofía similar a protocolos como HTTP, SMTP, etc.

SIP puede establecer sesiones entre dos participantes, entre múltiples participantes (donde todos pueden hablar y escuchar) y en modo multidifusión (donde uno habla y los otros escuchan). De la misma forma dichas sesiones pueden contener audio, video o datos. Esto permite que se pueda ejecutar cualquier tipo de aplicaciones simultáneamente a la comunicación de voz o video (por ejemplo enviar un archivo, un mensaje instantáneo o compartir el escritorio).

2.3.3.2.- Arquitectura SIP

Si bien SIP proporciona la señalización para que las sesiones multimedia puedan ocurrir, SIP no transporta la información multimedia. Por el contrario, una vez que SIP concibe el acuerdo entre las partes, el streaming multimedia es transportado por otros protocolos (habitualmente RTC/RTCP).

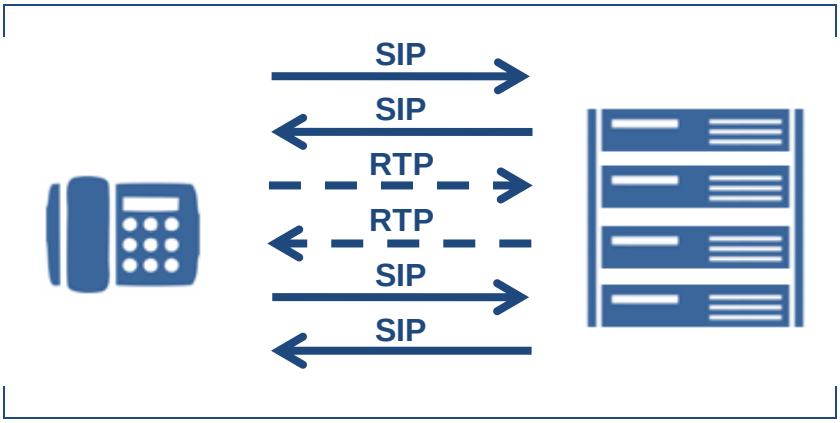


Figura 7.- Arquitectura SIP.

La arquitectura de SIP se basa en el modelo cliente-servidor en donde el cliente realiza solicitudes (requests) al servidor, quien procesa y responde (responses) dichas solicitudes aceptando, rechazando o redirigiendo dichas solicitudes. Los terminales de usuario pueden actuar tanto como clientes o como servidores.

Por ejemplo, si el terminal de usuario 201 genera una llamada al Proxy SIP en realidad se envía una solicitud al Proxy SIP (se comporta como un cliente SIP) y éste la procesa y le responde (actuando como un servidor SIP). Pero si ahora el Proxy SIP le debe notificar que otro usuario quiere establecer una llamada con él, en este escenario el Proxy SIP envía una solicitud al terminal 201 (actuando como cliente SIP) y el terminal 201 deberá procesar la solicitud y enviar una respuesta al Proxy SIP, es decir aceptar la llamada, contestar con una respuesta de ocupado o no disponible (actuando como un servidor SIP).

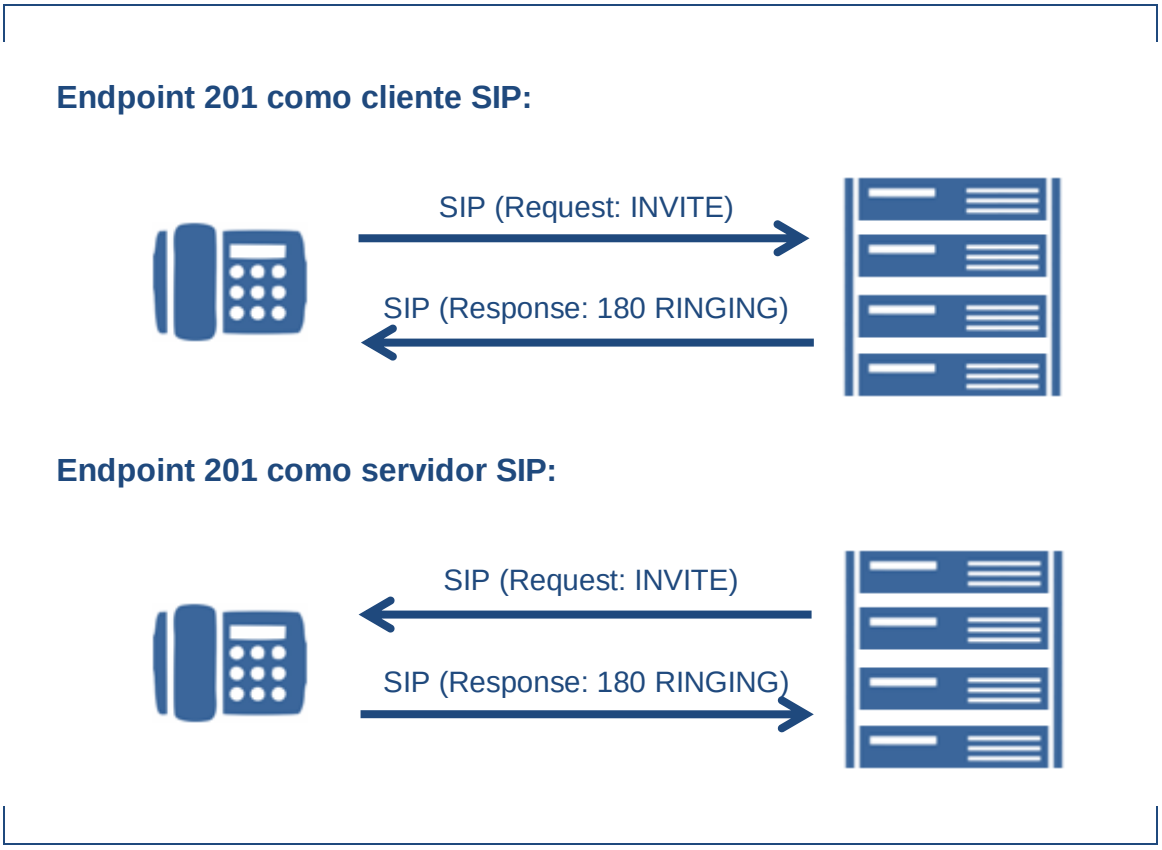


Figura 8.- Endpoint 201 como cliente SIP y servidor SIP.

Como protocolo de transporte del SIP, por defecto utiliza el UDP. Aunque también es posible transportar el SIP sobre TCP y TLS.

2.3.3.3.- Componentes de una red SIP

En una red SIP existen dos clases de dispositivos: Terminales de usuario y Servidores de red

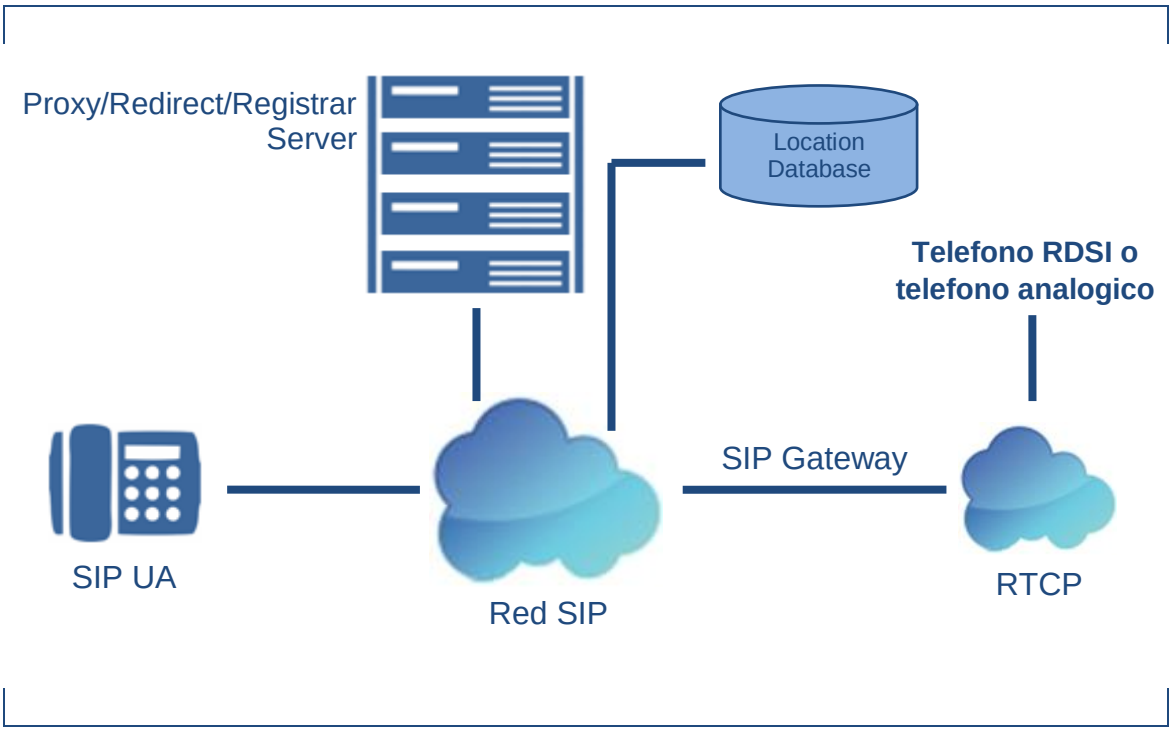


Figura 9.- Entidades de una red SIP.

.- Usuario

Dispositivos terminales de usuarios o UA (User Agents):

Como SIP es un protocolo en donde los terminales de usuario pueden comportarse como cliente SIP o servidor SIP dependiendo el contexto, éstos deben poseer dos módulos:

Agente de usuario cliente o UAC (User Agent Client): *Este módulo permite al terminal lanzar solicitudes SIP como una solicitud de registro o de nueva llamada.*

Agente de usuario servidor o UAS (user Agent Server): *Este módulo permite al terminal interpretar solicitudes SIP que le envíen y poder responder dichas solicitudes con códigos de respuesta en nombre del usuario SIP configurado en dicho terminal. Estas respuestas podrán ser aceptación, rechazo o redirecciones.*

.- Servidores

Servidor Proxy (proxy server):

Se trata de un dispositivo que recibe solicitudes de un UAC, las analiza y decide si dichas solicitudes debe enviarlas a un UAS o a otro Servidor Proxy. Habitualmente el Servidor Proxy modifica la solicitud y la envía al siguiente dispositivo en busca de entregar el mensaje SIP a quien sea el destinatario del mismo. Esta modificación implica que el mensaje SIP aparezca como generado por el Proxy SIP, por lo que quien reciba el mensaje cuando lo responda deba pasar por este Proxy SIP. Una solicitud puede atravesar varios servidores proxy hasta alcanzar el destino. El proxy server obviamente dispone de los dos módulos (UAC y UAS).

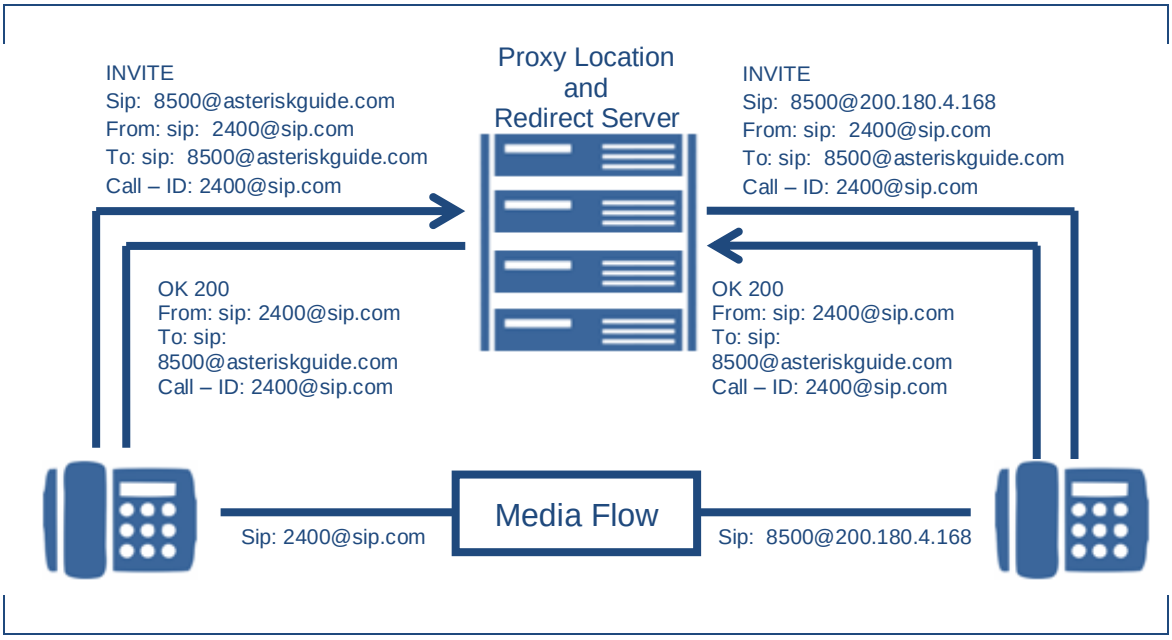


Figura 10.- Servidor Proxy.

Servidor de re direccionamiento (Redirect Server):

Se trata de un servidor quien acepta solicitudes SIP, traduce la dirección SIP de destino en una o varias direcciones de red y las devuelve al cliente. De manera contraria al Proxy Server, el Redirect Server no encamina las solicitudes SIP. En el caso de la devolución de una llamada, el Proxy Server tiene la capacidad de traducir el numero del destinatario en el mensaje SIP recibido, en un numero de reenvió de llamada y encaminar la llamada a este nuevo destino, y eso de manera transparente para el cliente de origen; para el mismo servicio, el Redirect Server devuelve el nuevo número (numero de reenvió) al cliente de origen quien se encarga de establecer una llamada hacia este nuevo destino.

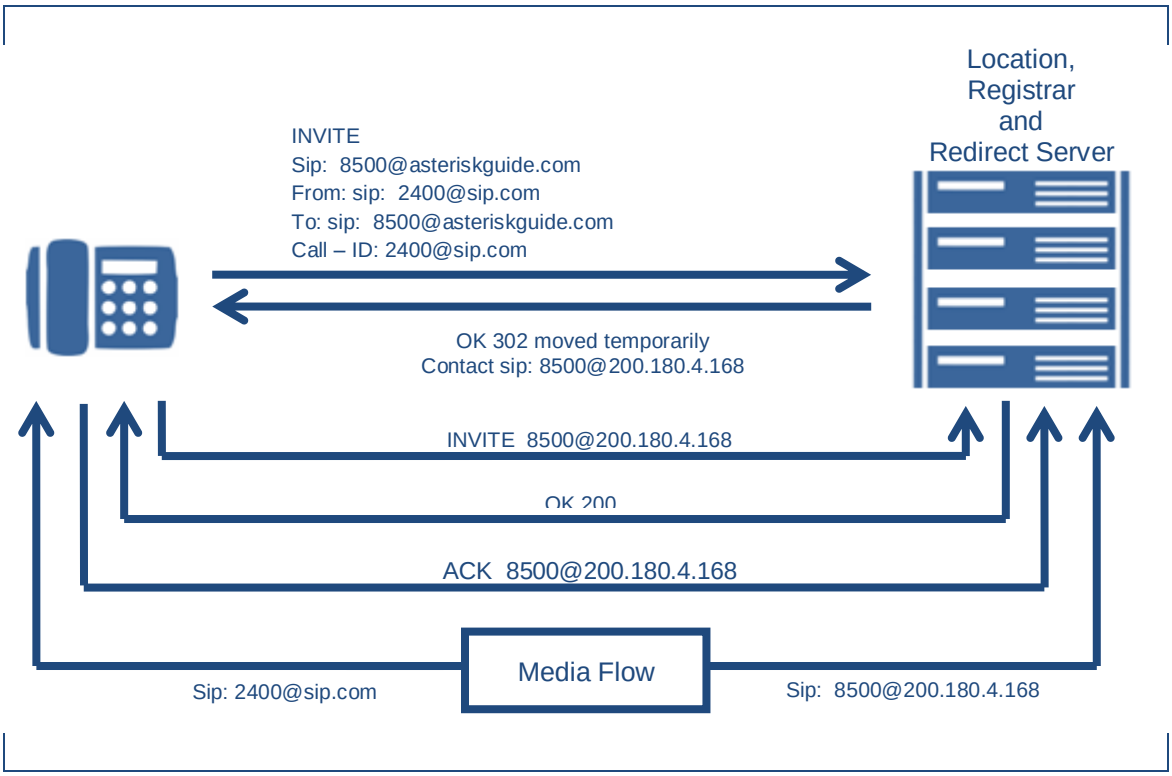


Figura 11.- Servidor de re direccionamiento.

Servidor de registro (Registrar):

Recibí solicitudes de registro de los UA y mantiene una tabla con las direcciones SIP de los usuarios y sus respectivas direcciones IP. Gracias a este servidor, se puede localizar un UA registrado al momento de tratar de iniciar una sesión con él. Una de las ventajas del SIP es la movilidad del usuario (puede realizar el registro desde el teléfono de su oficina, desde el softphone de se notebook o desde el softphone dentro de su smartphone) y el Servidor de Registro guardará su ubicación para cuando alguien intente localizarlo.

Figura 12.- Servidor de registro.

Generalmente los servicios de Proxy, Redirección y Registro se encuentran corriendo dentro de un mismo equipo.

2.3.3.4.- URI – Las direcciones SIP

El formato se basa en la estructura de direccionamiento empleada en Internet y conocida como URL (Uniform Resource Locators).

Formato de una URI SIP: SIP: **usuario@dominio**; *parámetro = valor.*

Este tipo de direccionamiento permite por ejemplo que se utilice un número telefónico tradicional en el campo usuario y en el dominio la dirección del gateway que interconecta la red SIP con la red telefónica PSTN donde se encuentra dicho número.

Ejemplos:

SIP: **usuario@dominio**

SIP: **usuario@dominio**; *transporte = TCP*

SIP: **5493516313933@190.210.6.162**

2.3.3.5.- El diálogo SIP

Se tiene dos tipos de mensajes: solicitudes (requests) y respuestas (responses). El cliente de la sesión enviara una solicitud que será respondida por el servidor.

Si examinamos el paquete SIP, tanto las solicitudes como las respuestas están compuestas de:

Una línea de inicio*, indica si el mensaje es solicitud o respuesta.*

Una cabecera*, formada por varios campos. Dichos campos contienen información adicional a la solicitud o la respuesta en cuestión. Por ejemplo incluye el remitente y destinatario del paquete.*

El cuerpo del paquete*, SIP no define el payload ni se ocupa de lo que haya. Generalmente el payload del SIP es un paquete SDP (Session Description Protocol) que lo que hace es especificar los parámetros de la sesión que se quiere establecer. Esto ocurre en los paquetes SIP de inicio de sesión que intercambian la partes Solicitudes "INVITE" y/o Respuestas "200 OK".*

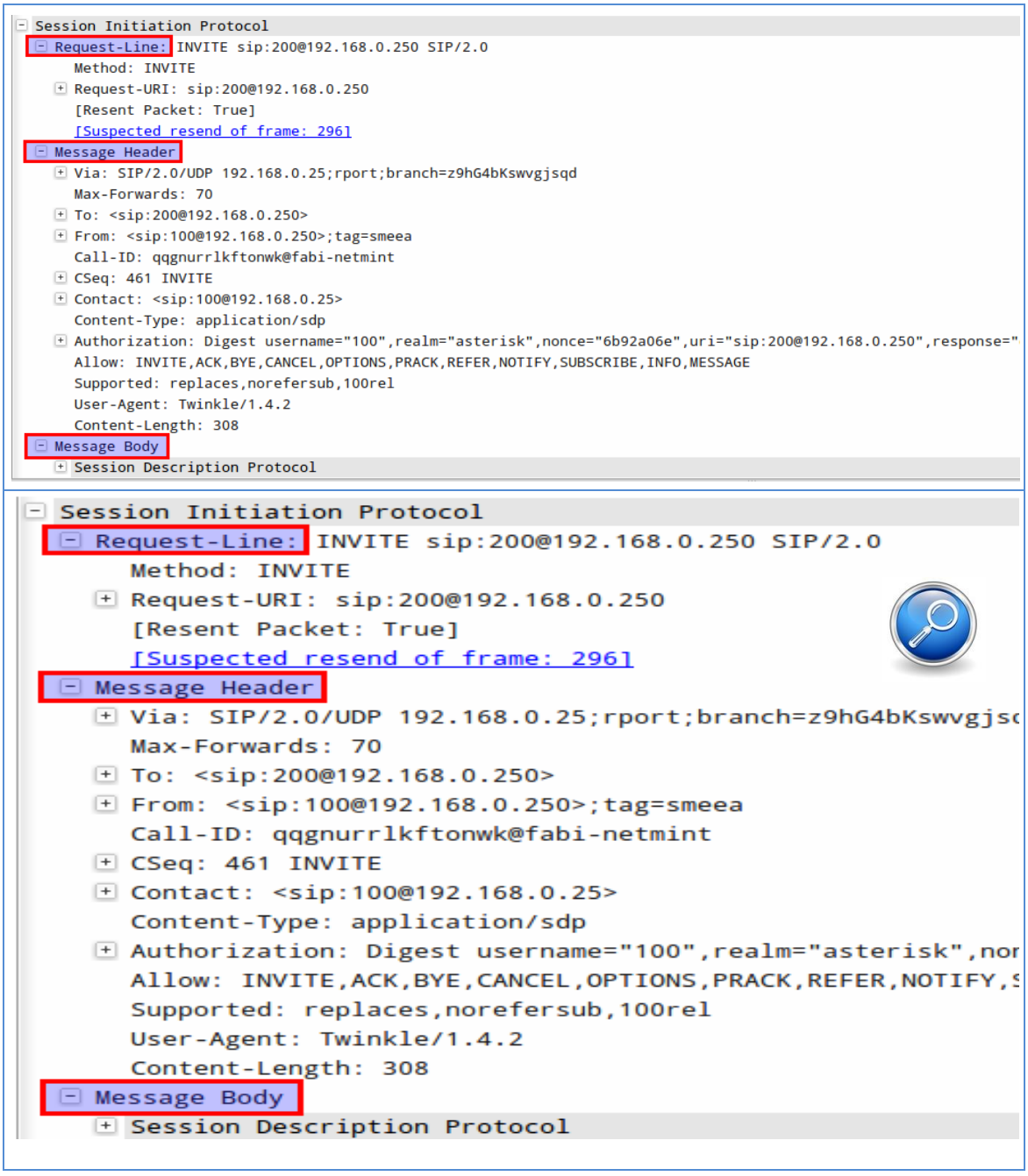


Figura 13.- Paquete SIP.

Como se menciono anteriormente, los mensajes SIP pueden ser Solicitudes (Requests) o Respuestas (Responses).

Solicitudes, en la versión 2.0 de SIP se incluyen seis tipos de métodos:

INVITE: *un dispositivo lo envía para iniciar una llamada.*

ACK: *lo envía el UAC al UAS como confirmación de la respuesta. Solo se manda cuando son códigos de respuestas frente a una solicitud INVITE.*

OPTIONS: *son mensajes que se utiliza para preguntar a un UA sobre capacidades. También usado para saber si un UA está alcanzable.*

BYE: *este mensaje se utiliza para terminar una llamada.*

CANCEL: *se utiliza para anular una solicitud de inicio de sesión INVITE que aún no se ha establecido.*

REGISTER: *se utiliza para que los UA puedan dejar registro de su localización en el Registration Server.*

Respuestas, las respuestas a las solicitudes SIP, son códigos de tres dígitos, donde el primero de ellos tiene que ver con la familia (clase de respuesta) y los otros dos con el mensaje concreto dentro de esa clase.

1XX “información”: *Indica que se ha recibido la solicitud y que se está procesando*

2XX “Aceptación”: *Indica que todo está correcto para comenzar el streaming*

3XX “Redirección”: *Indica que el usuario activó una redirección a otro UAS.*

4XX “Error de solicitud”: *Indica que el mensaje de solicitud no está bien. Por ejemplo un número que no existe o una llamada que no se puede cursar, no se está autorizado, no se soportan los códec ofrecidos, etc.*

5XX “Fallo en el servidor”: *Indica que aunque el mensaje de solicitud es válido, el servidor tiene problemas para procesar la solicitud (servidor no disponible, saturado, etc.).*

6XX “Fallo General”: *Indica que no se puede atender la solicitud por un “fallo general”.*

Salvo las respuestas 1XX, todas las demás son respuestas finales, que dan por terminada la transacción SIP.

A continuación la lista completa de códigos:

100 - Trying
180 - Ringing
181 - Call Being Forwarded
182 - Call Queued
183 - Session Progress

200 - OK
202 - Accepted

300 - Multiple Choices
301 - Moved Permanently
302 - Moved Temporarily
305 - Use Proxy
380 - Alternative Service

400 - Bad Request
401 - Unauthorized
402 - Payment Required
403 - Forbidden
404 - Not Found
405 - Method Not Allowed
406 - Not Acceptable
407 - Proxy Authentication
408 - Request Timeout
409 - Conflict
410 - Gone
411 - Length Required
413 - Request Entity Too Large
414 - Request URI Too Long
415 - Unsupported Media Type
416 - Unsupported URI Scheme
420 - Bad Extension
421 - Extension Required
423 - Interval Too Brief
480 - Temporarily Unavailable
481 - Call/Transaction Does Not Exist
482 - Loop Detected
483 - Too Many Hops
484 - Address Incomplete
485 - Ambiguous
486 - Busy Here
487 - Request Terminated
488 - Not Acceptable Here
491 - Request Pending
493 - Undecipherable

500 - Server Internal Error
501 - Not Implemented
502 - Bad Gateway
503 - Service Unavailable
504 - Server Time-Out
505 - Version Not Supported
513 - Message Too Large

600 - Busy Everywhere
603 - Declined
604 - Does Not Exist Anywhere
606 - Not Acceptable

2.3.3.6.- La Cabecera del paquete SIP (Message Header)

La cabecera está formada por campos, algunos son usados en ambos tipos de mensajes y otros solo en un tipo (Solicitudes o Respuestas).

Los campos pueden ser alterados en cada salto entre el origen y el destino (hop by hop) y otros permanecen intactos y son dirigidos al destinatario final (end to end).

Se citan algunos campos:

Call-ID: *es un código que identifica cada llamada de forma única. Nos permite por ejemplo, hacer corresponder una respuesta a una solicitud o cambiar dinámicamente los parámetros de una llamada.*

Cseq: *identifica cada solicitud, la respuesta a una solicitud lleva el mismo Cseq.*

From: *aparece en todos los mensajes de solicitud y respuesta, identificando a quien originó la llamada.*

To: *aparece en todos los mensajes de solicitud y respuesta, identificando el destinatario de la llamada.*

Via: *Registra la ruta que siguió la solicitud.*

Content-Length: *nos dice la longitud del cuerpo del mensaje SIP.*

Content-Type: *indica el contenido del cuerpo del mensaje SIP.*

Subject: *indica el asunto de la llamada.*

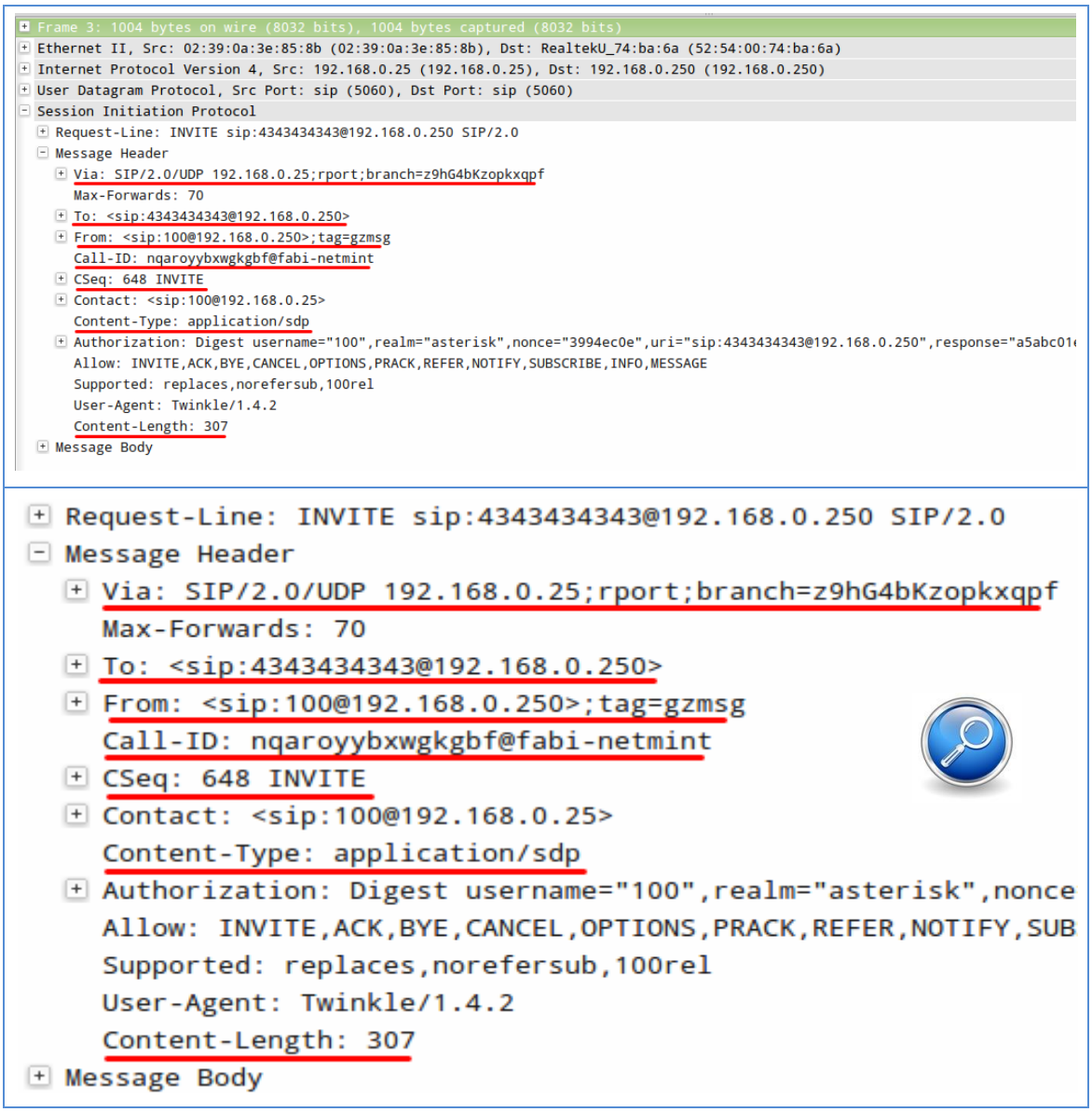


Figura 14.- Header SIP.

2.3.3.7.- El cuerpo del paquete SIP

Los paquetes SIP no están obligados a llevar una carga útil (payload) embebido. El payload puede ser información dirigida a la aplicación de usuario o información de la sesión. El contenido del paquete no es analizado por los dispositivos Proxy, sino que solamente lo interpretan los terminales.

Cuando SIP utiliza el cuerpo del paquete en el establecimiento de una sesión, este es ocupado por el protocolo SDP (Session Description Procol). Es decir un paquete SDP va embebido dentro del cuerpo del paquete SIP en algunas ocasiones.

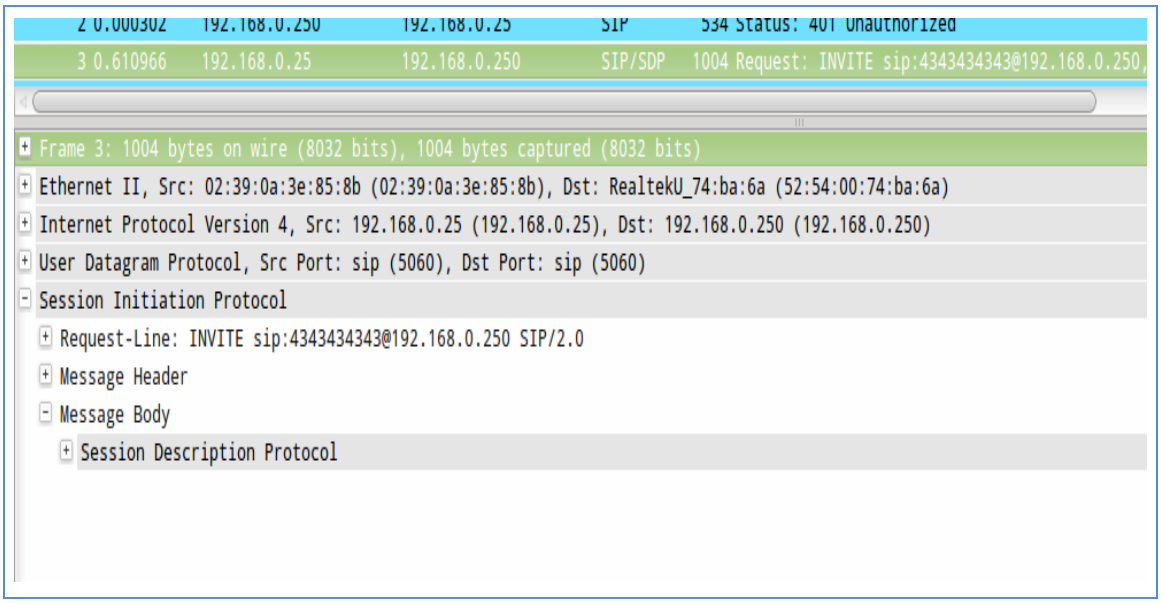


Figura 15.- Payload SIP.

SDP tiene la capacidad de describir las características de una sesión de media. Si bien SDP es muy amplio, SIP solamente utiliza una parte de las características de SDP.

SIP utiliza SDP en los inicios de sesión, es decir el terminal que origina una llamada, manda como cuerpo del paquete SIP de inicio de sesión, un paquete SDP el cual describe todas las características de “media” del terminal en cuestión (dirección ip y puerto donde espera recibir el streaming, codecs disponibles y su orden de prioridad, etc.), por otro lado la parte “llamada”, responde a dicho invitación con una respuesta SIP (200 OK), cuyo paquete lleva embebido otro paquete SDP el cual describe las características del UAS (User Agent Server) y en base a esto ambos acuerdan los parámetros para comenzar el streaming.

A continuación algunos campos del paquete SDP

		Campos del paquete SDP
v	Versión del protocolo	
o	Identifica al remitente y a la sesión. Posee 6 sub-campos	*Nombre del remitente, identificador de sesión, versión de la sesión, tipo de red, tipo de dirección (IPV4 – IPV6) y dirección de la maquina del remitente.*
m	Posee cuatro sub-campos	*Tipo de media (audio, video, aplicación, etc.), puerto (numero de puerto utilizado en el destino), protocolo de transporte utilizado (RTP) y formatos multimedia aceptados por orden de preferencia*
k	Lleva una clave de cifrado, en caso de trabajar con RTP encriptado	
a	Se utiliza para describir atributos adicionales	

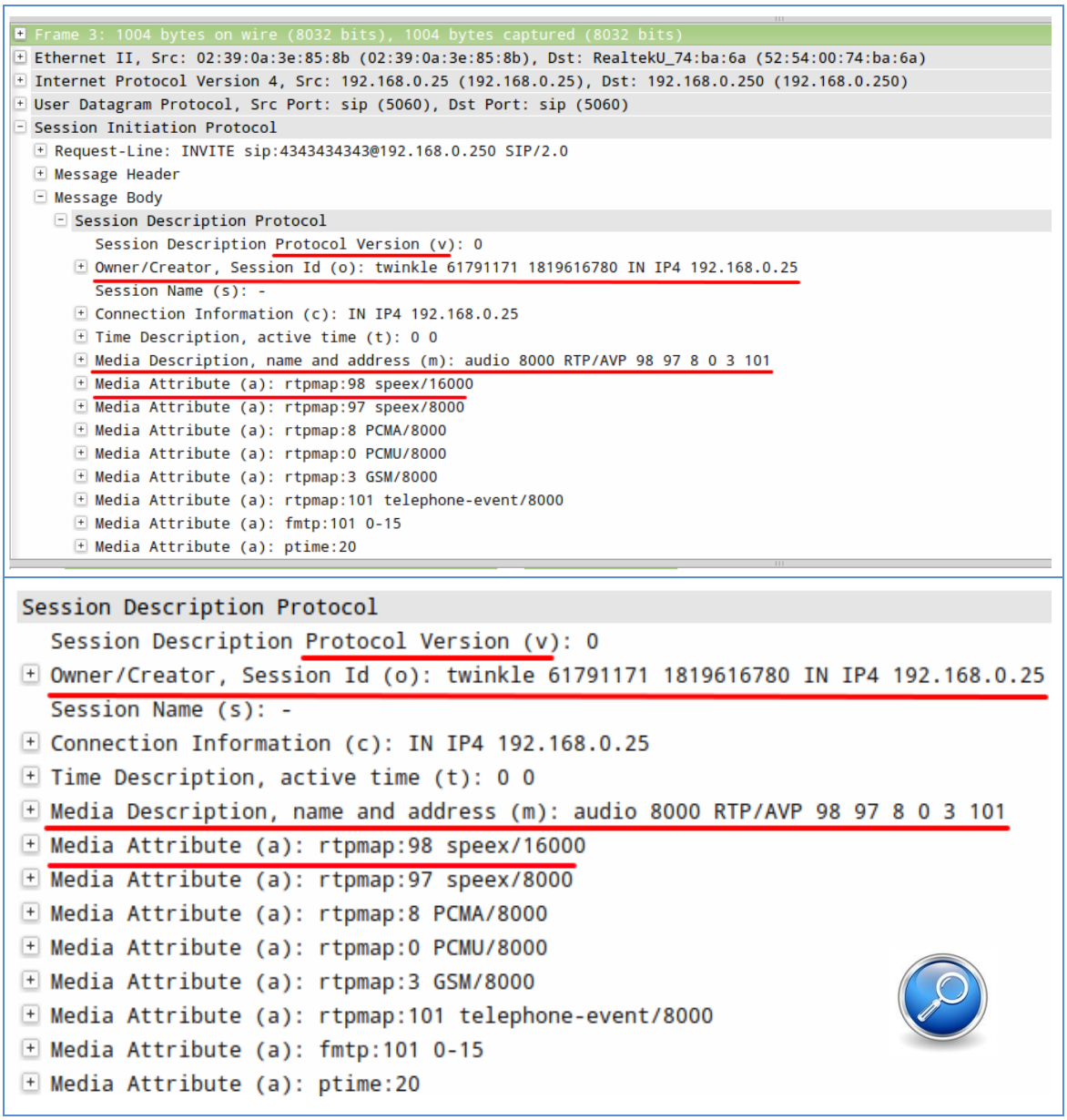


Figura 16.- Payload SIP.

Los navegadores NO implementan SIP, ellos si implementan WebRTC y websokets, pero la aplicación JS es quien implementa el intercambio de mensajes SIP usando como transporte websockets.

JSSIP, es una librería JS que permite construir endpoints SIP en una página web. Se puede hacer una analogía con jquery.

Ofrece un API para poder utilizar (generar y responder) Métodos y Respuestas del protocolo SIP.

JSSIP se encarga de:

- *Obtener el SDP de WebRTC.*
- *Enviar el INVITE sobre websockets.*
- *Procesa las respuestas.*
- *Mantiene el dialogo SIP.*

2.3.4.- El protocolo RTP

Tal como se explicó en la sección “El protocolo SIP”, RTP es quien entra en acción como responsable de transportar los paquetes de “media”, una vez que las partes de la comunicación acordaron la sesión.

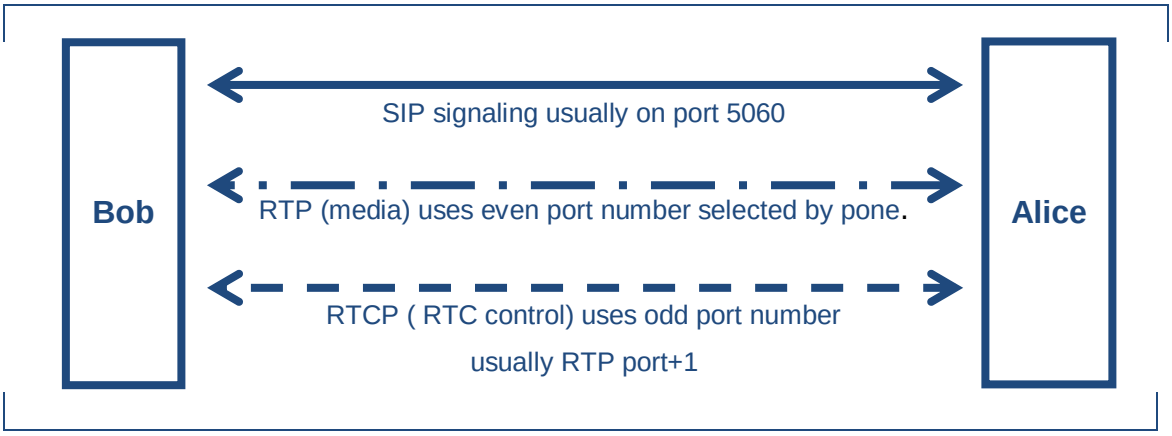


Figura 17.- El protocolo RTP.

De los dos protocolos de la capa de transporte (UDP y TCP), el adecuado para el envío de información en tiempo real es el UDP, no obstante este protocolo no garantiza que lleguen todos los paquetes, ni que los que si lo hacen lleguen en el mismo orden en que fueron enviados. Por lo que frente a esta realidad, el UDP es complementado para mejorar una comunicación en tiempo real, precisamente de esto se ocupa el protocolo conocido como RTP – Real Time Protocol.

RTP se encarga de añadirle a los paquetes UDP un número de secuencia, una marca de tiempo y una identificación del payload que transporta cada paquete UDP. Estos datos son aprovechados por el protocolo RTCP (RTP Control Protocol) quien informa a los participantes de la sesión sobre la calidad del streaming (perdidas de paquetes, desorden, jitter).

Gracias al campo “número de secuencia” de la cabecera RTP, se puede detectar la pérdida o desorden de los paquetes recibidos, mientras que con el capo “tiempo” puedo detectar el delay y la variación del dalay (jitter).

Generalmente los protocolos RTP y RTCP se utilizan conjuntamente, es decir que cuando se asigna un puerto para una sesión RTP, también se asigna uno para la sesión RTCP.

2.4.- Ws como transporte para SIP

El protocolo websocket permite establecer y mantener una conexión bidireccional entre un navegador y un servidor web y que en el marco de este TFG se utiliza Websocket como transporte para los mensajes SIP.

A continuación se describe el uso de Websockets como un mecanismo de transporte de mensajes SIP en un contexto web.

Hasta entonces SIP soportaba UDP, TCP y TLS como protocolos de transporte, pero a partir del RFC 7118 se especifica la implementación del protocolo Websockets como transporte del SIP.

2.4.1.- Introducción

Como se menciono, el protocolo websockets permite el intercambio de mensajes entre clientes y servidor sobre conexiones persistentes TCP (opcionalmente TLS). El “handshake” se realiza aprovechando la semántica e infraestructura HTTP.

Las últimas versiones de todos los navegadores web standard (google-chrome, mozilla firefox, opera) tienen integrada la capacidad de manejar conexiones websocket por especificación de la W3C. Es interesante considerar que también otras aplicaciones clientes comiencen a soportar el protocolo, sobre todo en el contexto de Smartphones o Tablets, en donde trabajar con un navegador web suele tornarse incómodo.

2.4.2.- Definiciones

SIP Websocket Client: *es una entidad SIP capaz de solicitar una conexión al Websocket Server y establecer, mantener y/o finalizar sesiones SIP utilizando la conexión Websocket para intercambiar los mensajes SIP.*

SIP Websocket Server: *es una entidad SIP capaz de escuchar solicitudes de conexión Websocket de clientes y establecer, mantener y/o finalizar sesiones SIP utilizando la conexión Websocket para intercambiar los mensajes SIP.*

Como se menciono en la sección pertinente al protocolo Websocket, este es un protocolo de transporte que trabaja por encima del TCP por el cual el cliente y el servidor pueden intercambiar mensajes bidireccionalmente.

El handshake consiste de un mensaje HTTP tipo “Upgrade” desde el cliente al servidor, donde este ultimo devuelve la solicitud con un mensaje respuesta “101”, de esta manera se establece la conexión Websocket. Durante el handshake también se acuerda entre las partes, qué protocolo de aplicación (o sub-protocolo del Websocket) van a dialogar sobre Websocket. Este protocolo de aplicación, puede ser tanto algo “custom” así como también un standard, en este documento se plantea el protocolo SIP como este protocolo de aplicación que va a ser transportado por Websocket. Entonces, lo que podemos ir afirmando es que una vez llevado a cabo el Handshake, es decir cuando el server acepta el HTTP Upgrade con su mensaje de HTTP 101, queda establecida nuestra conexión bidireccional para intercambiar mensajes Websocket (SIP sobre Websocket) y frames de control.

2.4.3.- SIP sobre Websockets

Durante el Handshake Websocket además de acordar realizar el Upgrade de HTTP a Websockets, se informa el protocolo de aplicación que se va a dialogar sobre la conexión Websocket. Esto se debe informar en el campo "Sec-WebSocket-Protocol" del Header HTTP en ambos mensajes del Handshake (El HTTP Get Upgrade que envía el cliente y el HTTP 101 que responde el server).

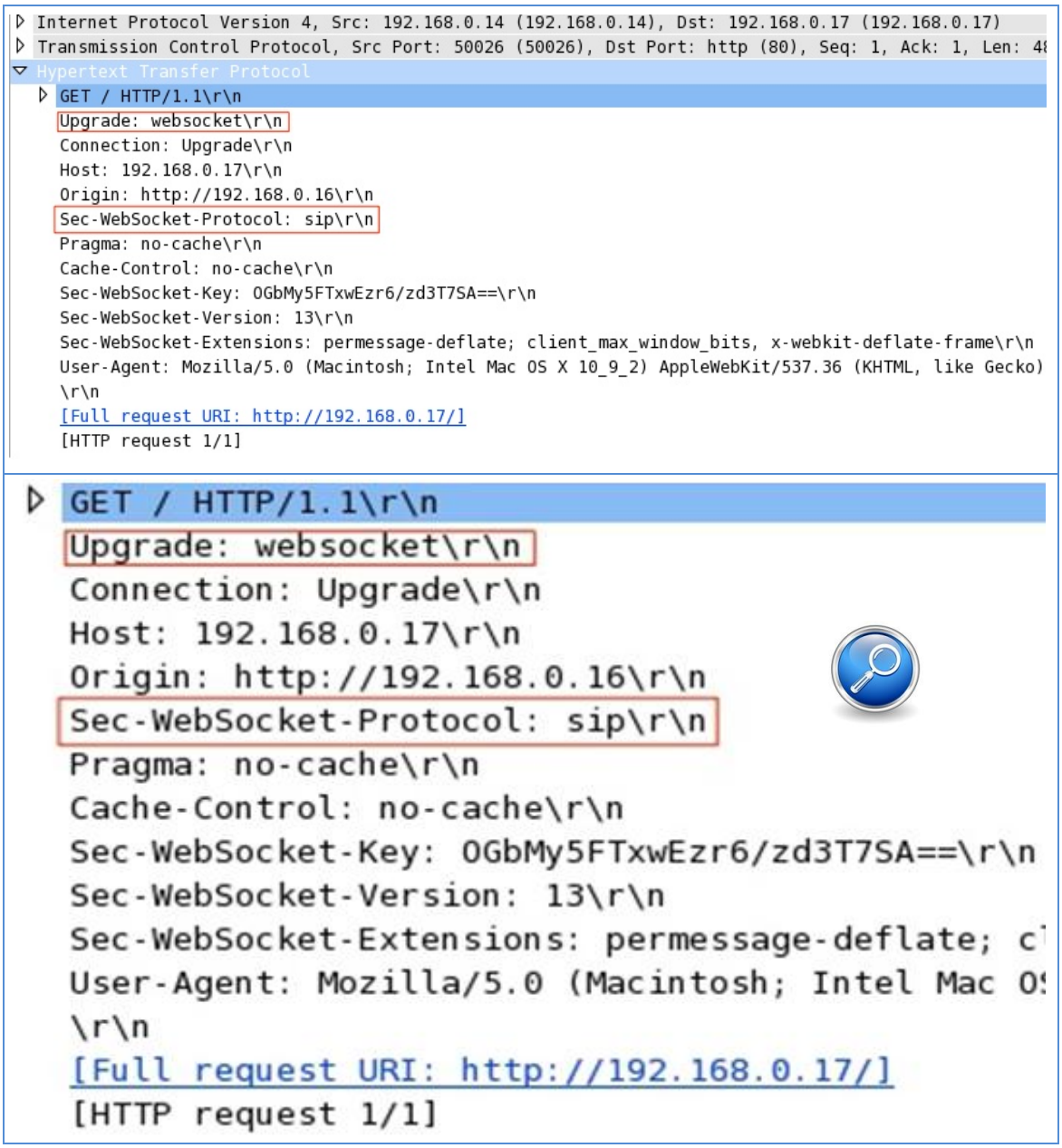


Figura 18.- HTTP Get Upgrade.

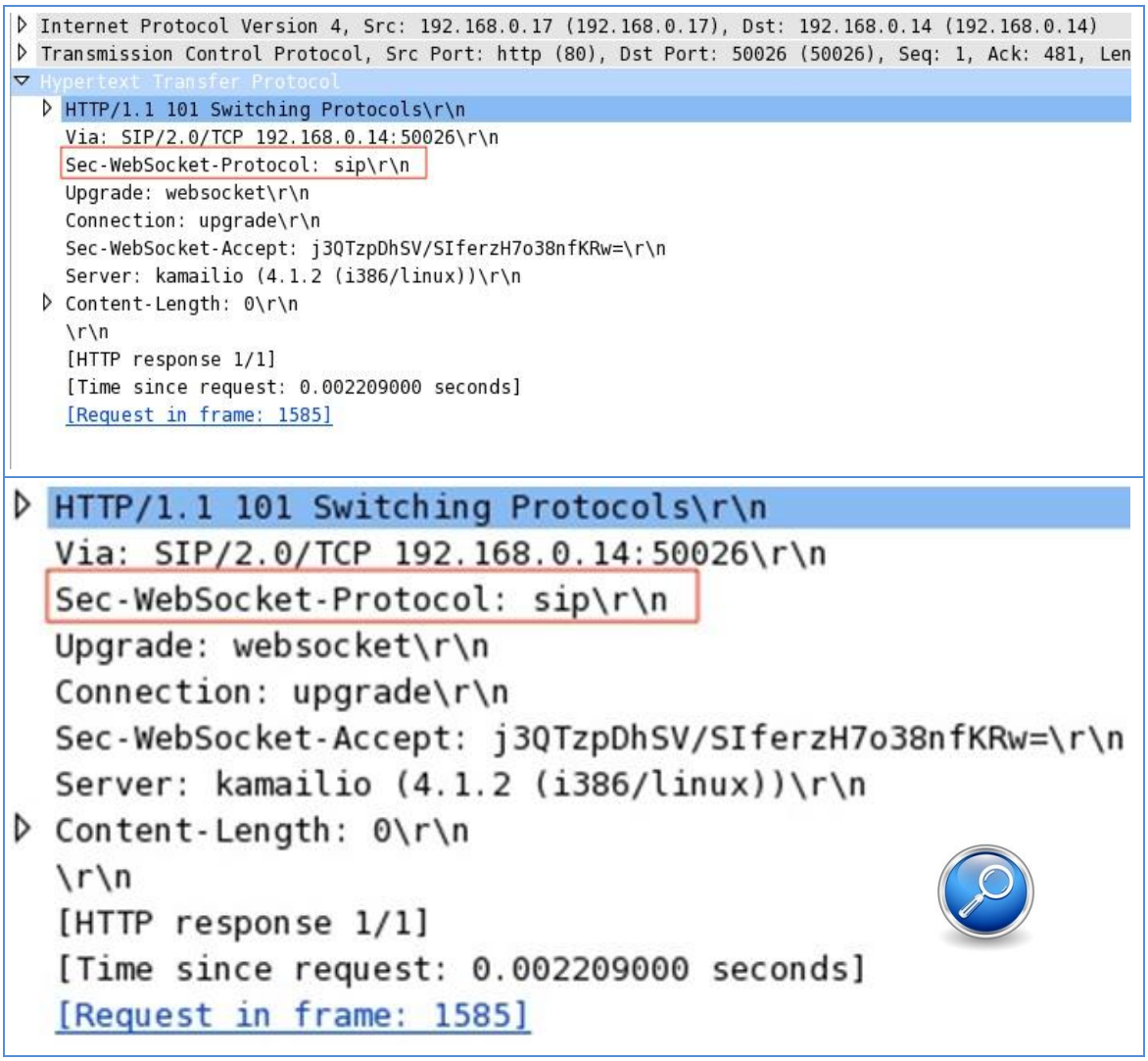


Figura 19.- HTTP 101.

A partir de dicha negociación ambas partes acuerdan usar el protocolo Websocket para intercambiar solicitudes y respuestas SIP.

Los paquetes Websocket pueden transportar frames UTF-8 o frames binarios. Por lo tanto los Clientes SIP Websocket y Servidores SIP Websocket deben soportar ambos métodos: Frames UTF-8 y Frames binarios.

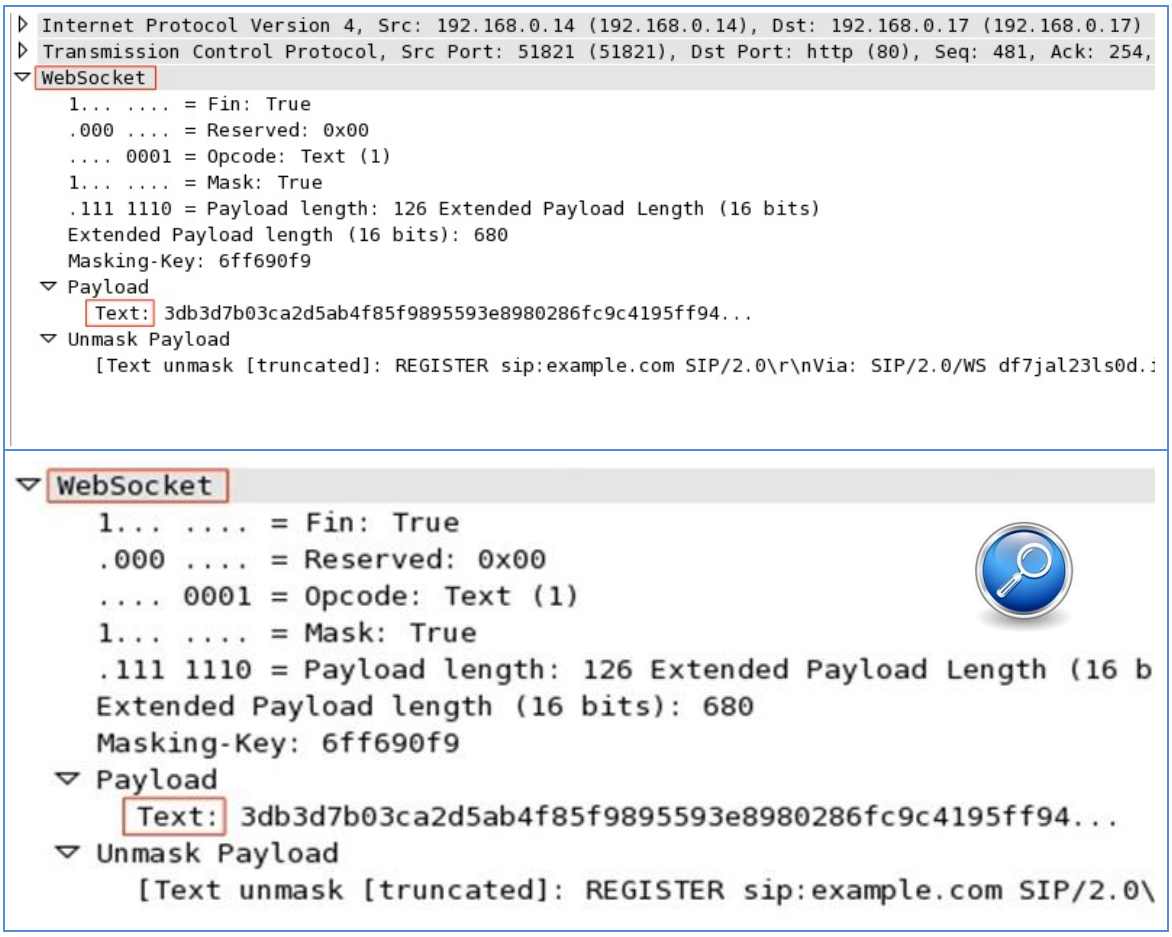


Figura 20.- Mensaje SIP como Frame UTF-8 sobre Websocket.

La conexión Websocket entre un cliente y un servidor se mantiene abierta usando un método “keep-alive” es decir que cada navegador en vía un “ping” Websocket frame. La frecuencia con la que se envía estos frames lo dispone cada navegador web.

2.5.- Requisitos de hardware y software

Para la ejecución del sistema, los requerimientos de software y hardware recomendables pueden resumirse en los siguientes puntos:

.- Hardware:

- *Notebook con webcam, micrófono y un browser google-chrome instalado.*
- *PC servidor ix86.*
- *Hard phone VoIP.*
- *Servidor proxy SIP.*
- *Servidor intranet.*
- *Central de comunicaciones.*

.- Software:

- *Sistema Operativo GNU/Linux Debian.*
- *Aplicación de servidor de código abierto Asterisk.*
- *Aplicación de servidor Proxy SIP de código abierto OVERSIP.*
- *Aplicación cliente "webphone" desarrollada en Javascript/HTML5 – que implementa SIP sobre websokets y WebRTC mediante una librería Java Script "JSSIP".*
- *Aplicaciones de análisis de protocolos: tcpdump y wireshark.*
- *Aplicación cliente SIP phone.*

CAPÍTULO 3.- DESARROLLO DE LA TECNOLOGÍA WEBRTC

Este capítulo comienza con un resumen técnico en donde se exponen dos diagramas de flujo que sintetizan los conceptos teóricos del Capítulo 2.

El siguiente apartado presenta un diagrama en bloque con el sistema propuesto y varios diagramas que detallan el funcionamiento del sistema de comunicación que se desea establecer.

Se continúa con el diseño de la nueva tecnología y la implementación de la misma, que le permitirá a los desarrolladores poder desarrollar e implementar comunicaciones en tiempo real a través del navegador web.

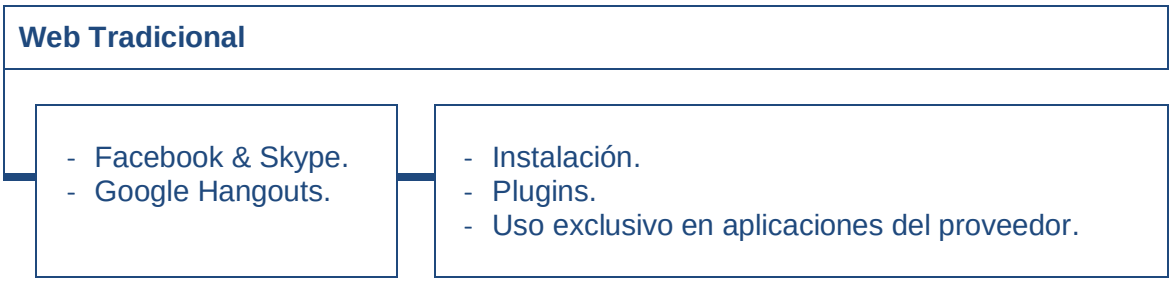
Finalmente se exponen los resultados alcanzados, las dificultades que se han presentados para poder implementar las comunicaciones en tiempo real y las actividades realizadas en el desarrollo del Informe Final de Grado.

3.1.- Resumen técnico

A partir de nuevos protocolos y APIs se logran establecer comunicaciones de "tiempo real" entre un host de la red con un navegador web con otros terminales convencionales con navegadores web. Para llevar el tiempo real al browser no hay dependencia del sistema operativo ni de plugins propietarios, el soporte es proporcionado por APIs y protocolos estándares abiertos.

A continuación, dos diagramas de flujo que resumen el capitulo 2.

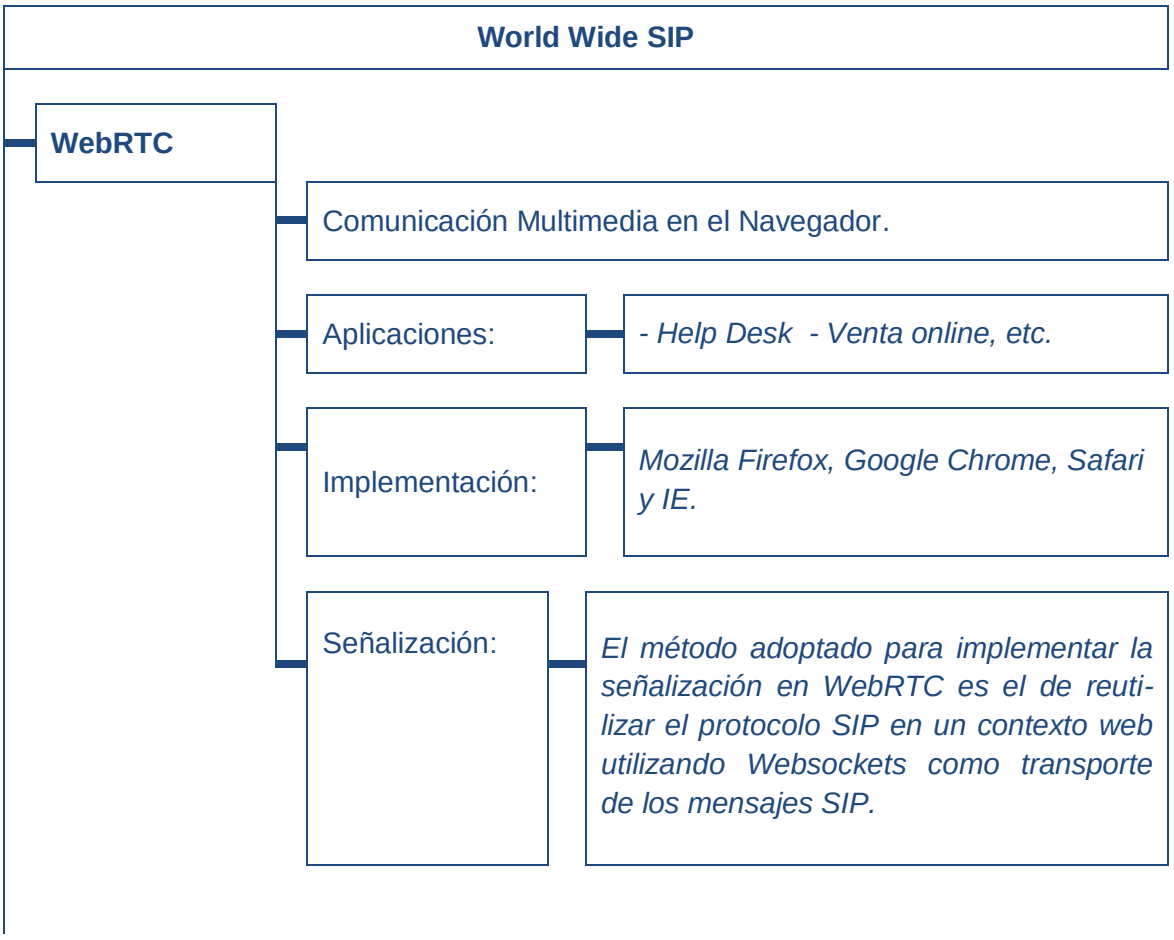
3.1.1.- Diagrama de la "Web tradicional"

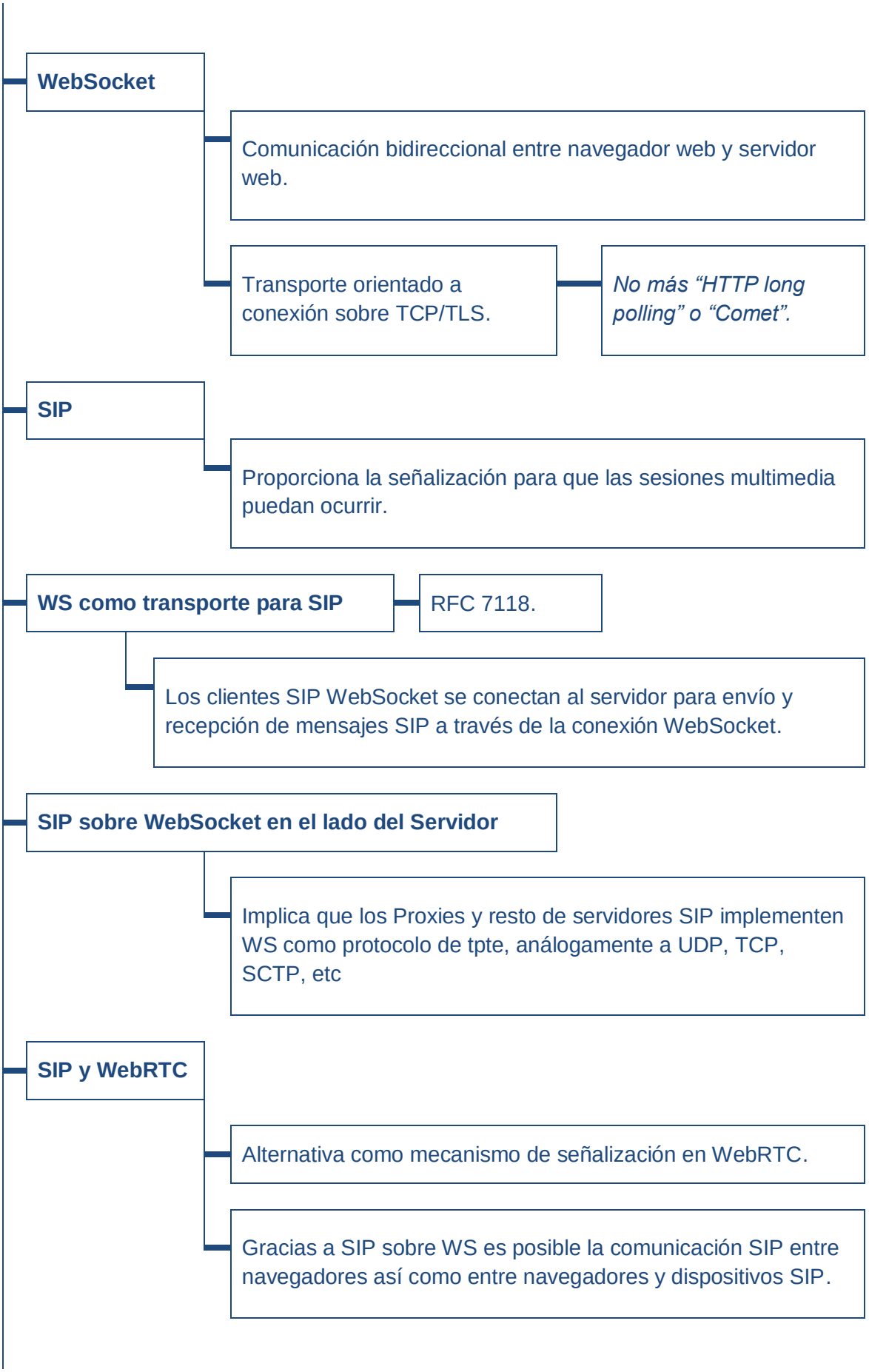


La Web actual se basa en HTTP para la comunicación, que es un protocolo de petición-respuesta. Utilizando las técnicas de Long Polling.

Todas las peticiones realizadas al servidor a través de HTTP contienen gran cantidad de información en la cabecera que es innecesario, y generan una gran sobrecarga del ancho de banda en los escenarios en tiempo real.

3.1.2.- Diagrama general de la parte teórica



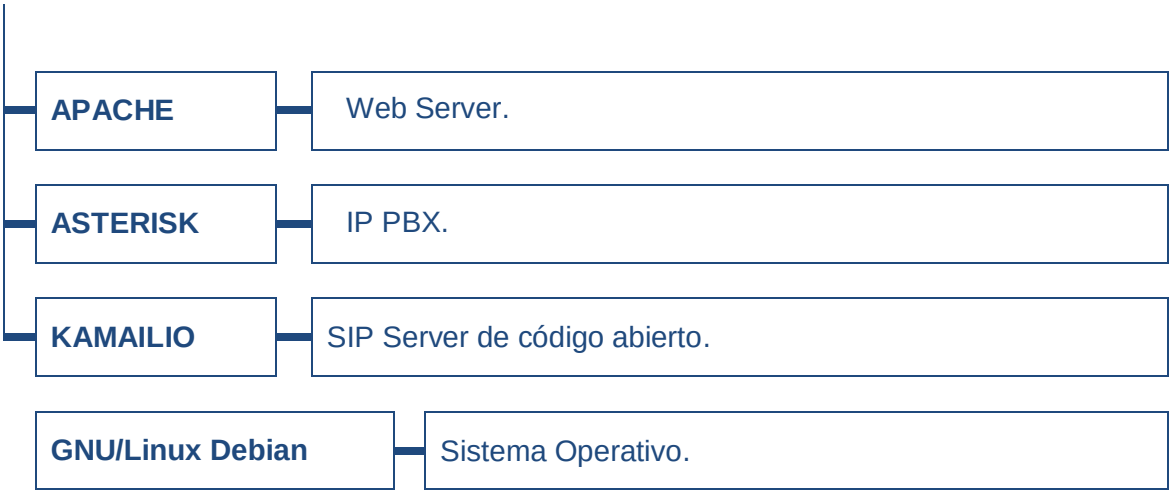

WebSocket
Comunicación bidireccional entre navegador web y servidor web.
Transporte orientado a conexión sobre TCP/TLS.
No más "HTTP long polling" o "Comet".
SIP
Proporciona la señalización para que las sesiones multimedia puedan ocurrir.
WS como transporte para SIP
RFC 7118.
Los clientes SIP WebSocket se conectan al servidor para envío y recepción de mensajes SIP a través de la conexión WebSocket.
SIP sobre WebSocket en el lado del Servidor
Implica que los Proxies y resto de servidores SIP implementen WS como protocolo de tpte, análogamente a UDP, TCP, SCTP, etc
SIP y WebRTC
Alternativa como mecanismo de señalización en WebRTC.
Gracias a SIP sobre WS es posible la comunicación SIP entre navegadores así como entre navegadores y dispositivos SIP.

El protocolo RTP

- Responsable de transportar los paquetes de “media”, una vez que las partes de la comunicación acordaron la sesión.

JsSIP

- API para crear clientes SIP.
- Funcionalidades:
  - *Llamadas de a/v.*
  - *Registro SIP.*
  - *Mensajería SIP.*
- Maneja el stack WebRTC del navegador a través del API WebRTC:
  - *Acceso a dispositivos multimedia.*
  - *Gestión de audio/vídeo.*
- Obtiene el SDP generado por el stack WebRTC y se lo envía al remoto usando SIP.
- Funcionamiento:
  - *Configuración.*
  - *Conexión WS.*
  - *Registro SIP.*
  - *Recepción de mensajes SIP.*
  - *Transacciones, diálogos...*
  - *Diseño basado en callbacks.*
  - *Acciones (iniciar llamada, mensajería...).*
- API.


APACHE
Web Server.
ASTERISK
IP PBX.
KAMAILIO
SIP Server de código abierto.
GNU/Linux Debian
Sistema Operativo.

3.1.3.- Diagrama en bloques del sistema propuesto

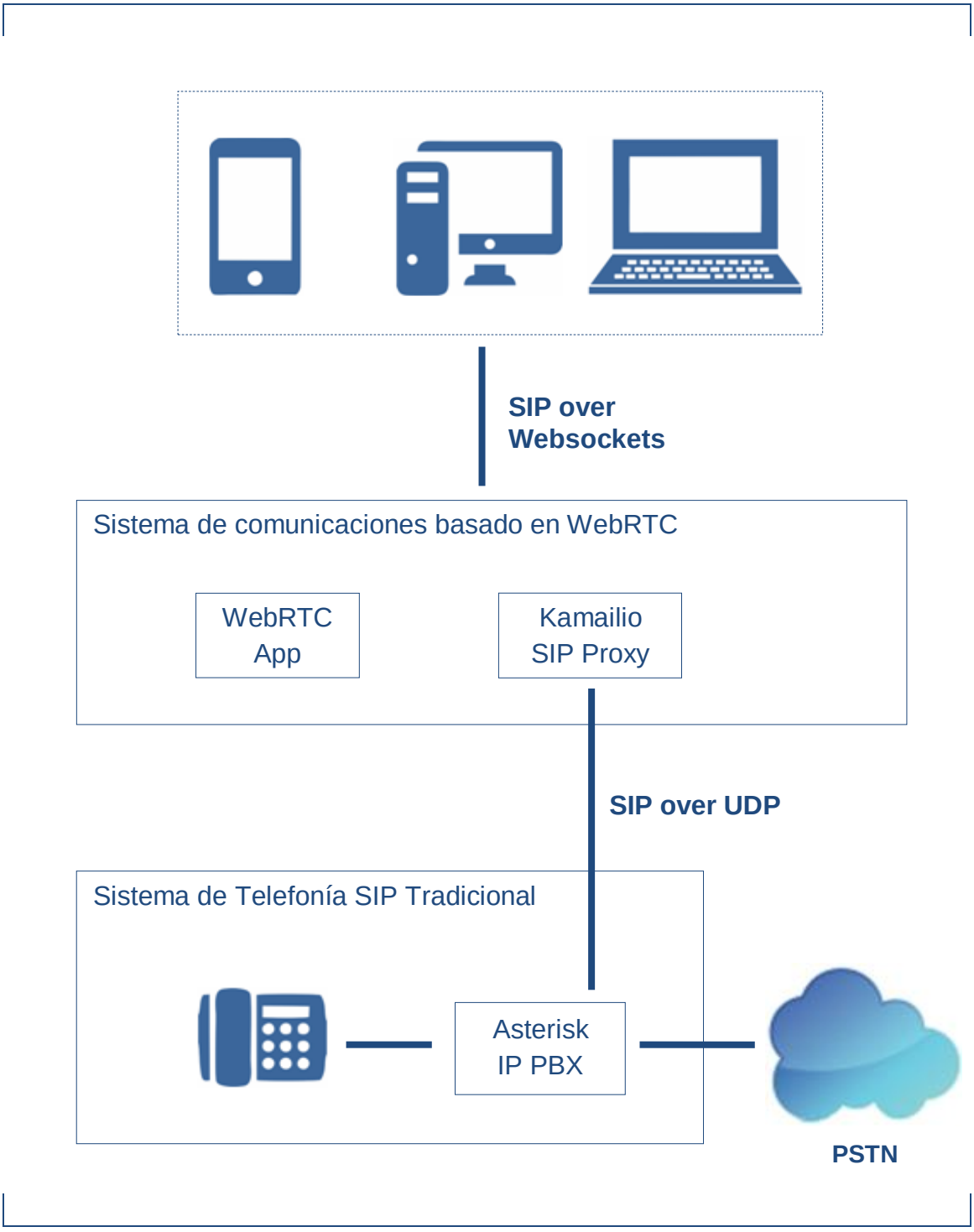


Figura 21.- Sistema propuesto.

3.1.4.- Funcionamiento del sistema propuesto

ETAPA 1: *Desde diferentes dispositivos con google chrome browser, se ingresa a la aplicación WebRTC. En dicha aplicación se realiza un login para registrarse como SIP peer de la red.*

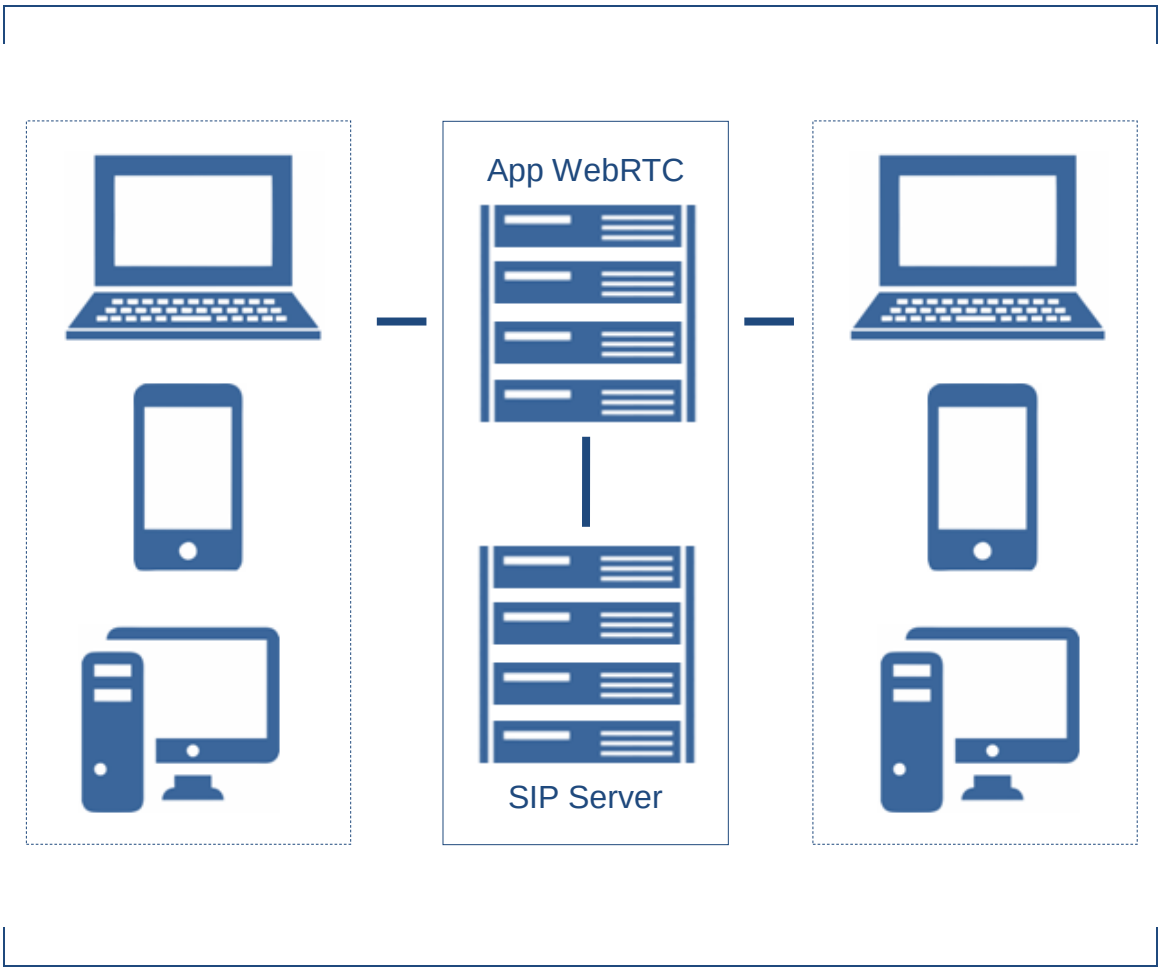


Figura 22.- Registro de usuarios con google chrome browser.

ETAPA 2: *Una vez que los usuarios están registrados, se pueden comunicar entre sí. Cualquiera puede iniciar una sesión de audio llamada, video llamada o intercambiar mensajes instantáneos desde el navegador google chrome.*

Comunicación entre dos peers WebRTC (Llamadas de voz, video, chat).

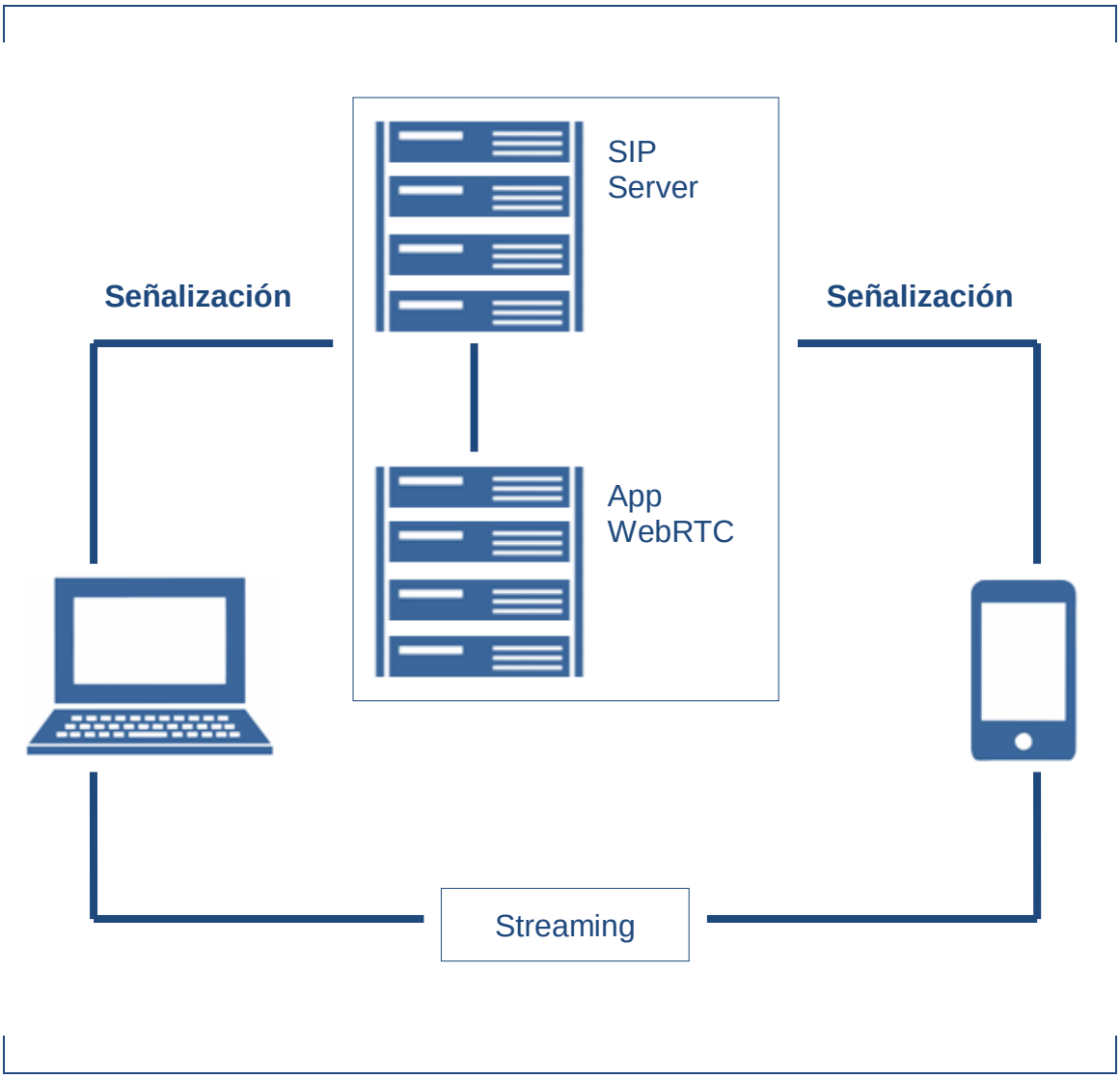


Figura 23.- Comunicación entre dos peers WebRTC.

ETAPA 3: *Cualquier usuario de la aplicación WebRTC puede inicia o responder llamadas contra usuarios de la IP PBX Asterisk.*

Comunicación entre peer WebRTC e IP PBX SIP UDP.

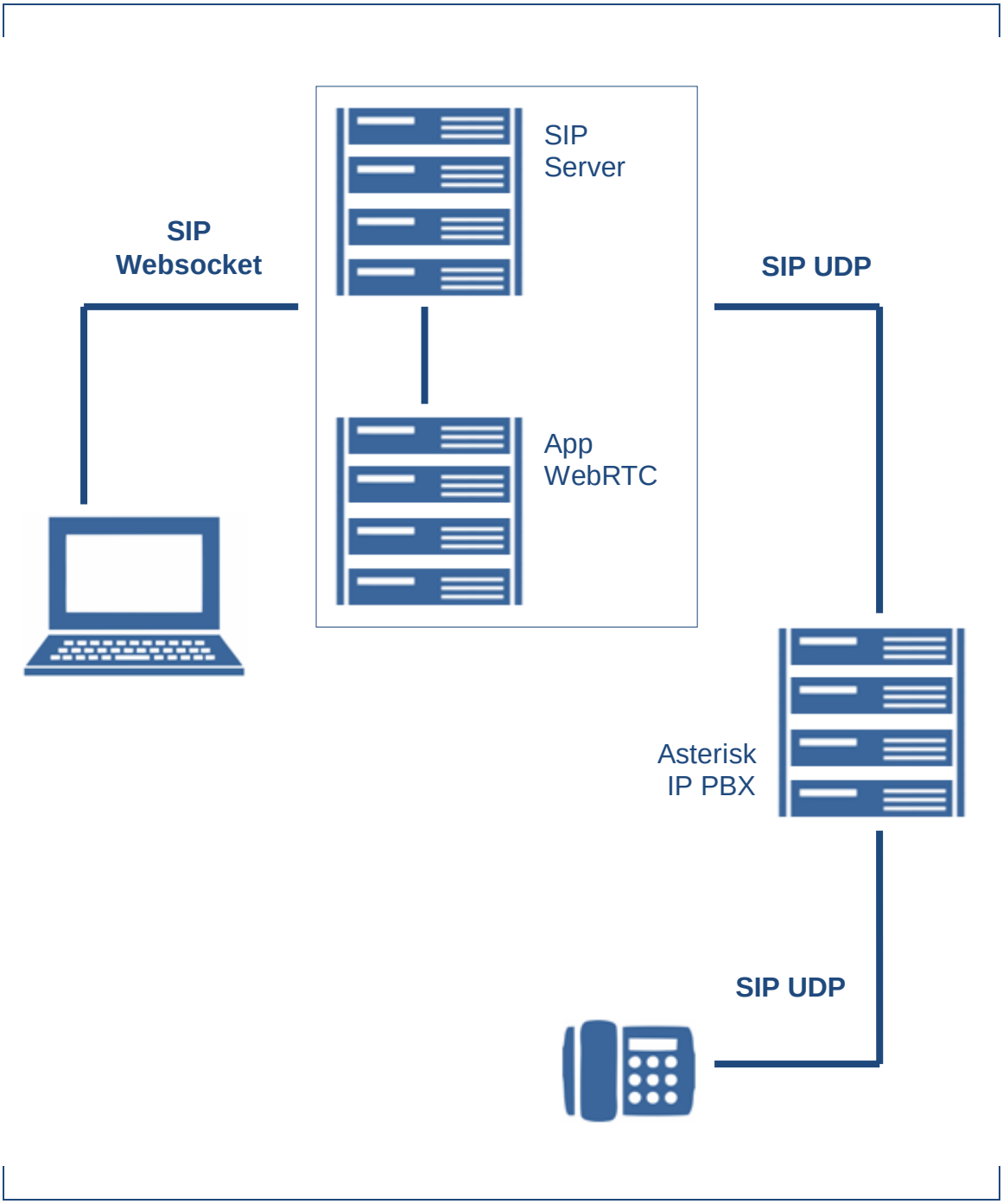


Figura 24.- Comunicación entre peer WebRTC e IP PBX SIP UDP.

ETAPA 4: *Como la IP PBX tiene troncales de salida a la PSTN, los usuarios del sistema de comunicaciones WebRTC pueden dialogar con abonados PSTN.*

Comunicaciones entre peer WebRTC y abonados PSTN.

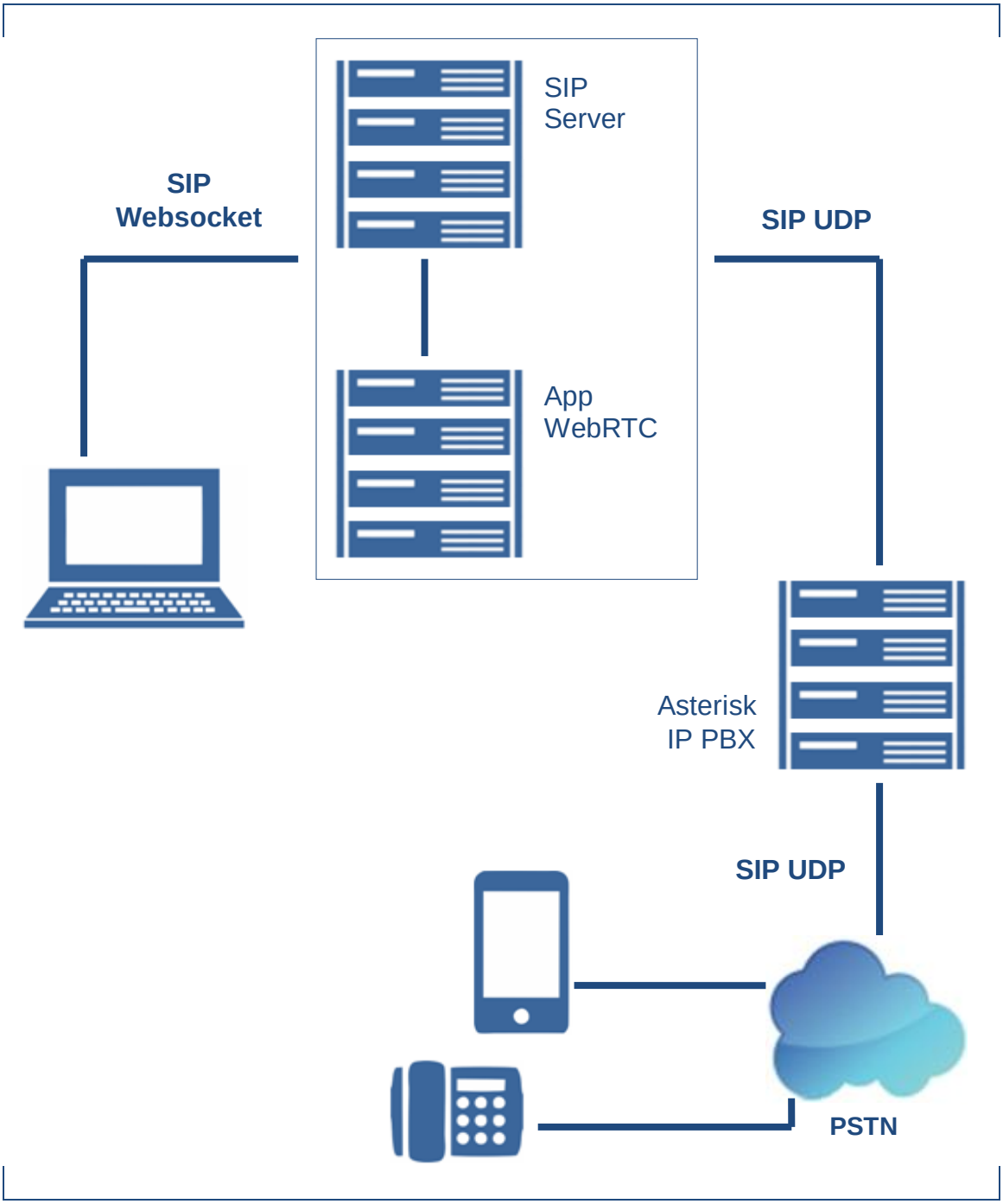


Figura 25.- Comunicaciones entre peer WebRTC y abonados PSTN.

3.2.- Diseño de la tecnología WebRTC

En este apartado se describe detalladamente los procedimientos que se han empleado para el desarrollo de las comunicaciones en tiempo real a través del navegador web, de manera que las empresas y personas que deseen desarrollar esta nueva tecnología la puedan implementar.

Las actividades que se realizan, son las siguientes:

- Se plantea el escenario de comunicación.
- Se presentan los componentes de Software y Hardware
- Se describe el funcionamiento de cada servidor y hots, dentro de la arquitectura de comunicación.
- Instalación/configuración cada uno de los servidores:
    - *Instalación del Web Server Host.*
    - *Instalación SIP Router Kamailio.*
    - *Instalación Asterisk IP PBX.*
- Instalación/configuración de las plataformas de comunicaciones:
    - *WebRTC con Asterisk.*
    - *WebRTC con Kamailio.*
    - *WebRTC en Jitsi Web Meet (Anexo 7.8.3 - Pagina 172).*

- Pruebas:

  Se explica y analiza con fundamentos técnicos cada una de las pruebas.

  - *Pruebas en la plataforma Asterisk:*

    - *Registro de endpoints SIP WebRTC en Asterisk.*
    - *Registro de endpoint SIP UDP en Asterisk.*
    - *Llamada de audio entre dos endpoints SIP WebRTC.*
    - *Llamada de audio entre un endpoint SIP WebRTC y un abonado PSTN.*
    - *Llamada desde un abonado PSTN a un endpoint SIP WebRTC.*
    - *Llamada desde un endpoint SIP WebRTC activando el debug Javascript del browser.*

  - *Pruebas en la plataforma Kamailio:*

    - *Registrar los endpoints WebRTC.*
    - *Pruebas de llamadas entre los dos endpoints SIP WebRTC.*
    - *Comprobar la mensajería instantánea vía SIP WebRTC.*
    - *Se Realizan pruebas de llamadas y análisis de los paquetes SIP tanto a nivel navegador web (activando la consola de Javascript) como en el servidor SIP Proxy Kamailio (Utilizando el sniffer ngrep).*

3.2.1.- Escenarios de comunicación

El escenario RTC que se plantea, establece comunicaciones entre peer WebRTC y abonados PSTN.

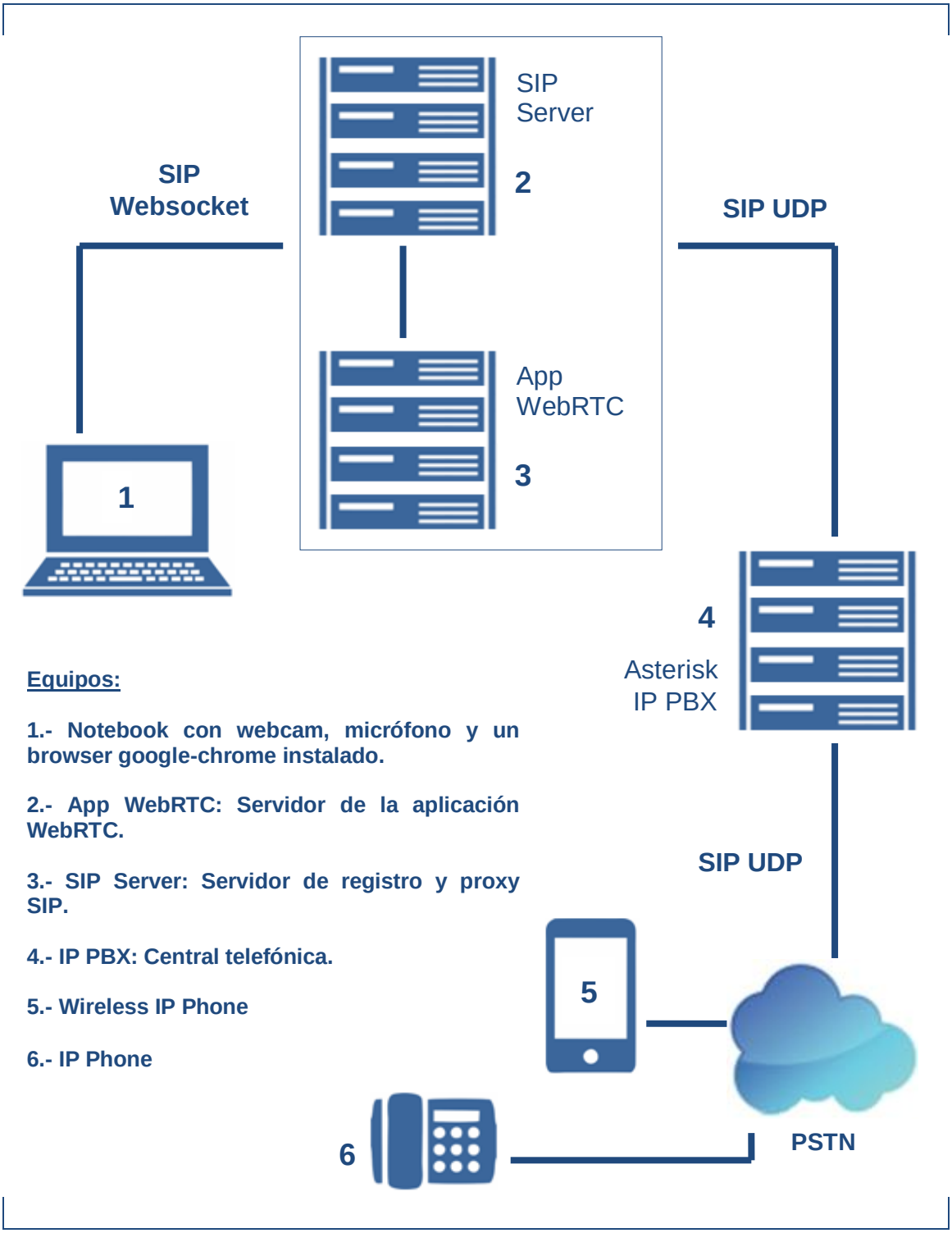


Figura 26.- Escenario de comunicación.

3.2.2.- Componentes de Software y Hardware

Cliente	
Navegador Web:	*Google Crome.*
Lenguaje:	*HTML5, JavaScript, CSS3.*
Librería:	*JsSIP.*
Framework:	*WebRTC.*
Sistema Operativo Clientes:	*Gnu/Linux.*
Servidores	**Aplicaciones**
Web Server:	*Apache.*
SIP Router:	*Kamailio.*
IP PBX:	*Asterisk.*
Sistema Operativo de los Servidores (Distribución Linux):	*Debian Whezy*
Hardware	**Equipo**
Equipo de Red:	Switch.
Clientes:	Laptops, Netbooks.
Servidores o Servidor:	PC o Laptop.

3.2.3.- Descripción de cada componente

El esquema propuesto se compone de tres host GNU/Linux debian wheezy. En cada uno de ellos, corren como servicio diferentes aplicaciones de manera tal que uno de los host se comporte como servidor de la aplicación WebRTC, otro como un Router SIP y Gateway SIP UDP – SIP Websockets y finalmente una central IP PBX capaz de manejar SIP sobre UDP y soportar conexiones a la PSTN tradicional.

A continuación se realiza una descripción de la funcionalidad que ofrece cada host dentro de la arquitectura planteada:

Web Server: *el web server cumple la función de servir la aplicación WebRTC de manera tal que los usuarios puedan acceder a la misma y comenzar a utilizarla a través de sus navegadores web.*

Sobre el GNU/Linux Debian host corre como aplicación el proceso Apache. Dicho proceso dota al host de la capacidad de entender las peticiones HTTP y servir contenido web.

SIP Router: *el SIP Router cumple una doble función, por un lado es el servidor que proporciona todo el soporte SIP (Registro y localización de usuarios y SIP Proxy) para los usuarios del sistema WebRTC, por lo tanto soporta y entiende todas las transacciones entre los usuarios WebRTC en SIP sobre Websockets. Por otro lado este servidor tiene la capacidad de soportar SIP sobre UDP, SIP sobre TCP y SIP sobre TLS, por lo que es un Gateway que permite a los usuarios WebRTC interactuar con la IP PBX Asterisk mediante SIP sobre UDP.*

Sobre el GNU/Linux Debian host corre como aplicación el proceso Kamailio. Dicho proceso dota al host de la capacidad de actuar como un SIP Server capaz de soportar el SIP sobre diferentes transportes.

IP PBX: *es la central telefónica. Cuenta con capacidad de conmutación de usuarios cuyos enpoints dialogan SIP sobre UDP, pero también en nuestro esquema cuenta con conexión a la PSTN tradicional de manera tal que los usuarios WebRTC puedan interactuar con abonados PSTN.*

Sobre el GNU/Linux Debian host corre como aplicación el proceso Asterisk. Dicho proceso dota al host de la capacidad de actuar como una completa IP PBX capaz de conmutar canales entre enpoints SIP sobre UDP, así como también proveer una interfaz de comunicación con abonados de la PSTN tradicional.

3.2.4.- Instalación/configuración de los servidores

Antes de comenzar el deploy de cada uno de los servidores que conforman la arquitectura propuesta, se definen las direcciones IP de cada uno:

```
IP PBX Server            192.168.0.15
SIP Server               192.168.0.16
Web Server               192.168.0.17
```

A continuación, la instalación de los servidores.

- Web Server Host.
- SIP Router Kamailio.
- Asterisk IP PBX.

.- Instalación del Web Server Host

1.- Se instala: *GNU/Linux Debian.*

2.- Con el sistema operativo instalado, se procede con la configuración de los parámetros de red para dejar al mismo disponible como host y con acceso a internet para la descarga de los paquetes necesarios. Luego se editar el archivo */etc/network/interfaces* y queda de la siguiente manera:

```
# The loopback network interface
auto lo
iface lo inet loopback

# The primary network interface
allow-hotplug eth0
iface eth0 inet static
address 192.168.0.15
netmask 255.255.255.0
gateway 192.168.0.1
```

3.- Se reinicia los servicios de red y se procede con el acceso remoto (SSH) para proceder con el deploy del servidor.

4.- Se instala el web server apache y el lenguaje de programación php.

```
sudo apt-get install apache2 php5 php5-cli
```

5.- Se instalación del código fuente de la aplicación WebRTC. La aplicación se obtiene como un archivo comprimido *.tar.gz*, a partir de obtener el mismo se procede a aplicarlo sobre el servidor web que implementa nuestra aplicación WebRTC.

6.- Se crea un directorio en donde queda nuestra aplicación.

```
mkdir/var/www/tesis
```

7.- Se carga el archivo comprimido (aplicación) al servidor, más precisamente sobre el directorio */tmp.*

8.- Luego en el servidor, sobre el directorio se realiza la descompresión de la aplicación en el directorio creado en el punto anterior.

```
cd /tmp/
tar xzvf /tmp/tesis.tgz -C /var/www/tesis/
```

9.- Se cambian los permisos del directorio que contiene la aplicación WebRTC para que puedan ser ejecutados por usuario correspondiente al servicio web Apache.

```
chown www-data.www-data -R /var/www/tesis/
```

A continuación se procede con una pequeña de comprobación para verificar si el procedimiento se impactó correctamente. Para ello, abrimos un navegador web desde una PC cliente e introducimos la siguiente URL:

http://192.168.0.17/tesis/call.htm

Si todo resultó como debía, entonces debemos visualizar lo siguiente en nuestro navegador web:

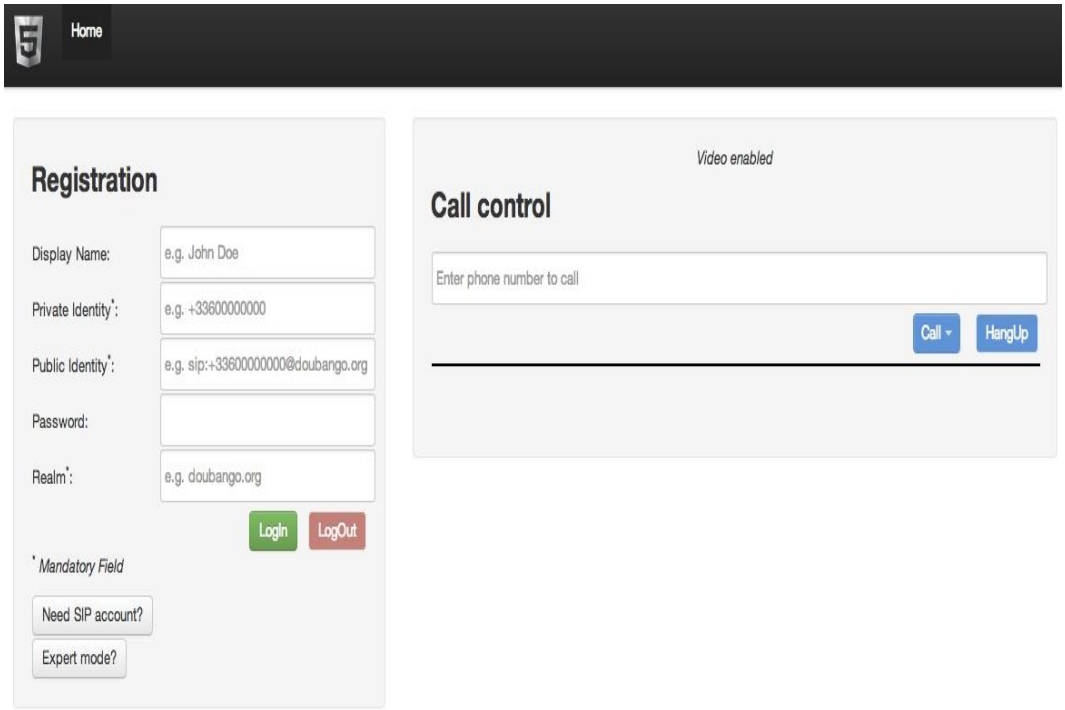


Figura 27.- Navegador web.

.- Instalación SIP Router Kamailio

1.- Se instala: *GNU/Linux Debian Wheezy.*

2.- Una vez instalado el sistema operativo, se procede con la configuración de los parámetros de red para dejar al mismo disponible como host y con acceso a internet para la descarga de los paquetes necesarios. Entonces se editar el archivo */etc/network/interfaces* y queda como:

```
# The loopback network interface
auto lo
iface lo inet loopback

# The primary network interface
allow-hotplug eth0
iface eth0 inet static
address 192.168.0.16
netmask 255.255.255.0
```

3.- Se reinicia los servicios de red y se procede con el acceso remoto (SSH) para proceder con el deploy del servidor.

4.- Se instala la aplicación *GIT* para poder obtener el código fuente del SIP Server Kamailio.

```
apt-get install git-core
```

5.- Se instalan dependencias y herramientas para compilar el código fuente de Kamailio.

```
apt-get install gcc flex bison libmysqlclient-dev
make
libssl-dev libcurl4-openssl-dev libxml2-dev
libpcre3-dev
libunistring-dev
```

6.- Se procede con la descarga del código fuente (última versión estable).

```
mkdir -p /usr/local/src/kamailio-4.1
cd /usr/local/src/
git clone --depth 1 --no-single-branch git://git.sip-
router.org/kamailio kamailio-4.1
cd kamailio-4.1
git checkout -b 4.1 origin/4.1
make cfg
```

7.- Ahora se procede con la compilación, para agregar soporte de SIP sobre Websockets se debe configurar un parámetro que indica que se debe compilar el módulo que brinda el soporte de SIP sobre Websockets. Para ello, se debe editar el archivo *modules.lst* y dejar el parámetro include_modules con los siguientes valores:

```
include_modules=db_mysql dialplan websocket xhttp
msrp sl
```

8.- Se pasa a correr la compilación y la posterior instalación de los módulos generados.

```
make PREFIX="/usr/local/kamailio-devel"
make all
make install
```

9.- Se copia el script que maneja (inicia, detiene, reinicia y recarga la configuración) del servicio Kamailio SIP Serverm, en el directorio de Debian correspondiente.

```
cp /usr/local/src/kamailio-4.1/pkg/kamailio/
deb/wheezy/
kamailio.init/etc/init.d/kamailio
```

10.- Se dan permisos de ejecución al script que maneja el servicio.

```
chmod 755 /etc/init.d/kamailio
```

11.- Dentro de dicho script, se deben editar los parámetros *$DAEMON* y *$CFGFILE*, que indican los path absolutos hacia el archivo ejecutable que inicia el servicio y el archivo de configuración que parametriza el servicio Kamailio SIP Server, respectivamente.

Para ello, mediante nuestro editor de texto, abrimos el archivo */etc/init.d/kamailio* y dejamos los siguientes parámetros:

```
DAEMON=/usr/local/kamailio-devel/sbin/kamailio
CFGFILE=/usr/local/kamailio-devel/etc/kamailio/
kamailio.cfg
```

12.- Ahora se debe copiar y configurar el archivo *kamailio.default* dentro del directorio */etc/default/.* Dicho archivo es leído también a la hora de iniciar el servicio.

```
cp /usr/local/src/kamailio-4.1/pkg/kamailio/deb/wheezy
/kamailio.default /etc/default/kamailio
```

13.- Se debe crear el directorio donde se creará el archivo de *PID* (Process Identificator) de Debian.

```
mkdir -p /var/run/kamailio
```

14.- Se crea el usuario y grupo de Debian con el que correrá el servicio Kamailio.

```
adduser --quiet --system --group --disabled-password –
shell/bin/false --gecos "Kamailio" --home/var/run/
kamailio
```

15.- A continuación se debe proceder con la configuración del servidor para que opere según lo planteado en nuestra arquitectura y descripción de la operación. Para ello, debemos aplicar las siguientes sentencias de configuración sobre el archivo */usr/local/etc/kamailio/kamailio.cfg.*

Primero vamos a realizar una copia del archivo original.

```
cp /usr/local/etc/kamailio/kamailio.cfg/usr/local/etc/
kamailio/kamailio.cfg.backup
```

Ahora si se procede con la edición del archivo, de manera tal que se impacten las sentencias de configuración. Este archivo se muestra en el capítulo 7.- (Anexos 7.8.1 - pagina 150). *Archivo de configuración kamailio.cfg.*

16.- Ahora estamos en condiciones de comprobar si todo funciona como se espera, para ello procedemos a Iniciar el servicio Kamailio.

```
/etc/init.d/kamailio start
```

Si todo fue correcto, debemos visualizar la siguiente salida correspondiente al comando de inicio:

```
[....] Starting Kamailio SIP server: kamailio:loading
modules
under /usr/local/lib/kamailio/modules/
Listening on
             udp: 192.168.0.16:5060
             tcp: 192.168.0.16:5060
             tcp: 192.168.0.16:80
Aliases:

. ok
```

Esto nos indica que el servicio arrancó y que además se abrieron los puertos 5060 y 80 sobre la dirección IP de nuestro host. En otras palabras, nuestro servidor deja “escuchando” los sockets correspondientes para que nuestros endpoints puedan comenzar a intercambiar mensajes SIP.

Podemos además realizar un listado de puertos abiertos en nuestro Debian y deberíamos visualizar los puertos 5060 TCP y UDP y además el puerto 80 TCP en modo “LISTEN”.

Para ello usamos el comando netstat, la salida del mismo debería coincidir con las filas resaltadas en la siguiente figura:

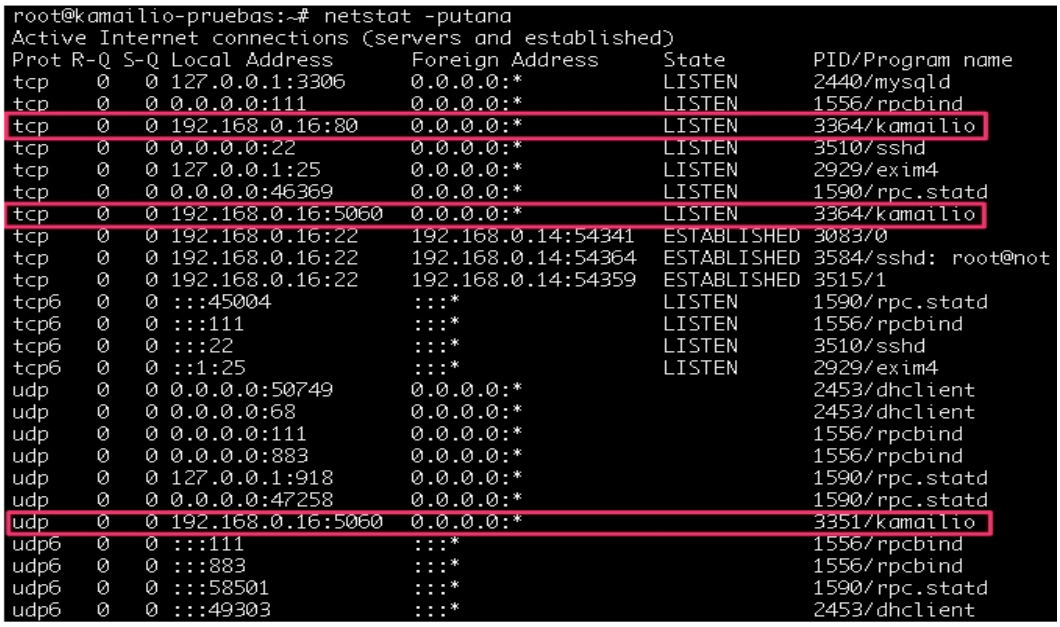


Figura 28.- Verificación Instalación SIP Router Kamailio.

.- Instalación Asterisk IP PBX

1.- Se instala: *GNU/Linux Debian Wheezy.*

2.- Una vez instalado el sistema operativo, se procede con la configuración de los parámetros de red para dejar al mismo disponible como host y con acceso a internet para la descarga de los paquetes necesarios. Entonces vamos a editar el archivo */etc/network/interfaces* y lo dejamos con el siguiente aspecto:

```
# The loopback network interface
auto lo
iface lo inet loopback

# The primary network interface
allow-hotplug eth0
iface eth0 inet static
address 192.168.0.17
netmask 255.255.255.0
gateway 192.168.0.1
```

3.- Se reinicia los servicios de red y procedemos con el acceso remoto (SSH) para proceder con el deploy del servidor.

4.- Se instala el paquete subversión, el cual implementa la utilidad para descargar el código fuente de la última versión estable de *Asterisk PBX* desde el repositorio oficial en Internet.

```
apt-get install subversion –y
```

5.- Descargamos el código fuente de Asteris PBX:

```
cd /usr/local/src/
svn co http://svn.asterisk.org/svn/asterisk/branches/
11/asterisk-11
```

6.- Se instalan dependencias y herramientas para poder compilar las fuentes de Asterisk.

```
sudo apt-get install build-essential libncurses5-dev
libxml2-dev libsqlite3-dev uuid uuid-dev libjansson-
dev –y
```

Cabeceras del Kernel de GNU/Linux:

```
sudo apt-get install linux-headers-`uname -r`
```

Librerías para implementar SRTP y SIP TLS:

```
apt-get install libssl-dev libsrtp0-dev
```

7.- Se procede a descargar las fuentes de Asterisk, compilar e instalar todo.

```
cd /usr/src
svn co http://svn.asterisk.org/svn/asterisk/branches/
11/ asterisk
cd /usr/src/asterisk/contrib/scripts
./install_prereq install
cd /usr/src/asterisk
./configure
make menuconfig
make && make install
```

8.- Al igual que cuando se realizó el deploy de la configuración del SIP Server Kamailio, se procede con la explicación de la configuración de Asterisk PBX sin entrar en detalles finos del framework Asterisk en sí. Por el contrario, se plantea seguir paso a paso las instrucciones citadas a continuación.

Se procede con la edición del archivo /etc/asterisk/sip.conf. Con el editor de texto disponible, se abre el archivo y en el final del mismo se agregan las siguientes líneas de código.

```
[proveedor]
type=friend
disallow=all
allow=alaw
allow=ulaw
allow=gsm
host=101.elastix.org.ar
fromuser=TRKCLA-01158764490
username=TRKCLA-01158764490
secret=password
qualify=yes
insecure=invite
deny=0.0.0.0/0.0.0.0
permit=50.116.20.247/255.255.255.255
context=from-pstn

[kamailio]
type=friend
context=from-kamailio
qualify=yes
insecure=invite
allow=all
encryption=yes
avpf=yes
host=192.168.0.18

[internos](!)
type=friend
disallow=all
allow=alaw
host=dynamic
qualify=yes
context=from-internos
callcounter=yes
allowsubscribe=yes
subscribecontext=states


notifyringing=yes
notifyhold=yes

[201](internos)
secret=1234

[202](internos)
secret=1234
```

A grandes rasgos, lo que se acaba de hacer es crear cuatro entidades SIP (UDP), de las cuales dos corresponden a endpoints que usaremos para las pruebas, otra entidad corresponde al troncal SIP contra el SIP Server Kamailio para poder enviar y recibir comunicaciones con el SIP Server (usuarios WebRTC) y la última se trata un troncal SIP contra un proveedor de telefonía IP quien ofrece el servicio de terminación hacia abonados PSTN.

9.- A continuación se procede con la creación del plan de discado que realice el tratamiento de las pruebas a realizar en la siguiente etapa, es decir que pueda enrutar las comunicaciones en todos los sentidos:

- *Usuarios WebRTC hacia usuarios del IP PBX Asterisk.*
- *Usuarios del IP PBX Asterisk hacia usuarios WebRTC.*
- *Usuarios del IP PBX Asterisk hacia abonados PSTN.*
- *Usuarios WebRTC hacia abonados PSTN.*
- *Abonados PSTN hacia usuarios del IP PBX Asterisk y/o usuarios WebRTC.*

Para ello debemos editar el archivo */etc/asterisk/extensions.conf.*

Se abre el archivo y al final del mismo se agregan las siguientes líneas de código:

```
[from-internos]
exten => _2XX,1,NoOp(llamada al interno ${EXTEN})
same => n,Dial(SIP/${EXTEN})
same => n,Hangup()

exten => _0[1-3]XXXXXXXXX.,1,NoOp(llamada al
número PSTN ${EXTEN})
same => n,Dial(SIP/proveedor/${EXTEN})
same => n,Hangup()

[from-pstn]
exten => _X.,1,NoOp(llama desde la PSTN)
same => n,Dial(SIP/kamailio/fabian)
```

10.- Una vez realizados los cambios, se procede a recargar la configuración de Asterisk PBX para que surtan efecto los cambios.

```
Asterisk -rx 'module reload’
```

3.2.4.- Instalación/configuración de las plataformas de comunicaciones

A continuación se procede con la instalación/configuración de tres plataformas de comunicaciones que implementan WebRTC como tecnología.

.- WebRTC con Asterisk

El esquema propuesto se compone de dos host GNU/Linux Debian 7. En cada uno de ellos, corren como servicio diferentes aplicaciones de manera tal que un host se comporte como servidor web de la aplicación WebRTC y el otro como una central IP PBX capaz de conmutar llamadas SIP sobre Websockets, SIP sobre UDP y soportar conexiones a la PSTN tradicional.

El primer escenario de uso de la tecnología WebRTC es en el framework de código abierto, Asterisk. Es habitual tornar Asterisk como la IP PBX corporativa de una institución, organización o empresa. A continuación se listan los pasos correspondientes para instalar y configurar una solución WebRTC basada en Asterisk y que a su vez conviva con diferentes endpoints SIP-UDP, así como también abonados PSTN, ya que Asterisk puede dialogar con casi cualquier sistema de telefonía, incluyendo a los típicos accesos a la PSTN (E1, líneas analógicas, líneas SIP, etc.).

Por lo tanto el esquema a implementar es el siguiente:

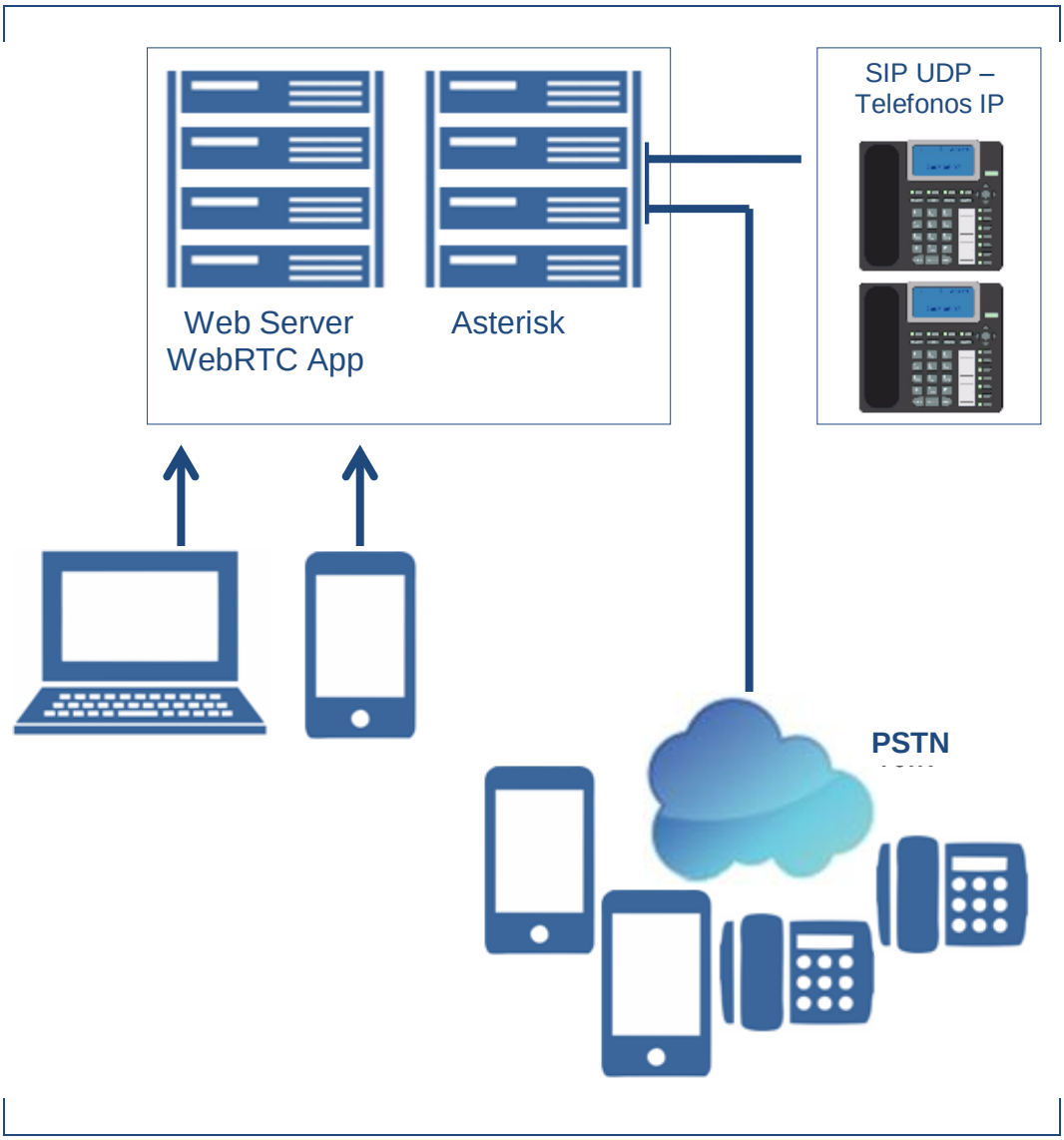


Figura 29.- Implementando WebRTC en Asterisk.

Antes de comenzar el deploy de cada uno de los servidores que conforman la arquitectura propuesta, se definen las direcciones IP de cada uno:

```
Asterisk Server          192.168.66.22
Web Server               192.168.66.23
```

Pasos para instalar y configurar WebRTC con Asterisk:

1.- Instalación GNU/Linux Debian 7 para el servidor Web (Pagina 73).

2.- Instalación GNU/Linux Debian 7 para el servidor Asterisk (Pagina 80).

3.- Configuraciones de red GNU/Linux Debian 7 para el servidor Web.

4.- Configuraciones de red GNU/Linux Debian 7 para el servidor Asterisk.

5.- Instalación de la aplicación WebRTC en el servidor Web.

Ingresar con un cliente ssh al servidor web e ingresar los siguientes comandos:

```
#aptitude install apache2 php5 php5-cli
#mkdir /var/www/tesis
#cd /tmp/
#svn checkout http://sipml5.googlecode.com/svn/trunk/
sipml5-read-only
#mv sipml5-read-only /var/www/tesis
#chown www-data.www-data -R /var/www/tesis
```

6.- Instalación de Asterisk.

Ingresar con un cliente ssh al servidor Asterisk e ingresar los siguientes comandos:

```
#aptitude install subversion
#cd /usr/src
#svn co http://svn.asterisk.org/svn/asterisk/branches/
11/ asterisk
#aptitude install install build-essential libncurses5-dev
 libxml2-dev libsqlite3-dev libssl-dev libsrtp0-dev
uuid-dev
#cd /usr/src/asterisk/contrib/scripts
#./install_prereq install
#ldconfig -v
#cd /usr/src/asterisk/
#./configure --with-crypto --with-ssl --with-
srtp=/usr/local/lib
#make && make install
#make samples
#make config
#reboot
```

7.- Generación de claves y certificados.

```
cd /usr/src/asterisk/contrib/scripts./
ast_tls_cert -C asterisk-
webrtc.example.com -O
"Tesis IUA" -d/etc/asterisk/keys
```

8.- Configuración y creación de enpoints SIP WebRTC y SIP UDP.

Se debe ingresar al archivo /etc/asterisk/sip.conf y a continuación agregar las siguientes líneas al final del mismo:

```
; Registración SIP en el proveedor de terminación PSTN
register => usuario:password@XXX.XXX.XXX.XXX:YYYY

; Proveedor de terminación PSTN
[proveedor]
type=friend
disallow=all
allow=gsm
host=XXX.XXX.XXX.XXX
port=YYYY
fromuser=usuario
defaultuser=usuario
secret=password
insecure=invite
context=from-pstn
qualify=yes

; Enpoint SIP-UDP
[201]
type=friend
host=dynamic
secret=AAA123456
disallow=all
allow=alaw
allow=g722
context=from-internal
```

```
; Template Endpoints SIP-WebRTC
[webrtc](!)
type=friend
transport=ws
videosupport=yes
avpf=yes
force_avp=yes
icesupport=yes
encryption=yes
disallow=all
allow=alaw
allow=h264
host=dynamic
secret=123456
context=from-internal
dtlsenable=yes
dtlsverify=fingerprint
dtlscertfile=/etc/asterisk/keys/asterisk.pem
dtlscafile=/etc/asterisk/keys/ca.crt
dtlssetup=actpass

[fulano](webrtc)
[mengano](webrtc)
```

9.- Configuración del Web Server.

Se debe abrir el archivo */etc/asterisk/http.conf* y debajo de la línea "[general]" agregar las siguientes líneas:

```
enabled=yes
bindaddr=0.0.0.0
bindport=8088
```

10.- Configuración de un plan de discado que permita conmutar los diferentes tipos de comunicaciones.

Se debe abrir el archivo */etc/asterisk/extensions.conf* y al final del archivo, agregar las siguientes líneas:

```
[from-internal]
exten => _.,1,Dial(SIP/${EXTEN})
same => n,Hangup()

exten => 1234,1,Playback(demo-congrats)
same => n,Hangup()

exten => _XXXXXXX,1,Dial(SIP/proveedor/${EXTEN})
exten => _XXXXXXXXX,1,Dial(SIP/proveedor/${EXTEN})

[from-pstn]
exten => _X.,1,Dial(SIP/fulano&SIP/mengano)
ssame => n,Hangup()
```

Al finalizar la edición de todos los archivos involucrados, se debe reiniciar la configuración de Astesisk:

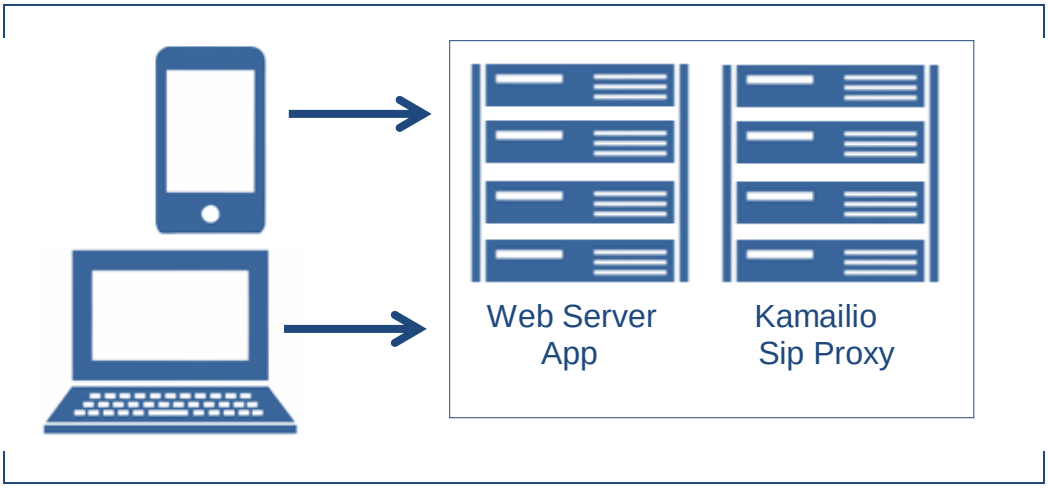
```
#asterisk -rvvvv
CLI>module reload
```

.- WebRTC con Kamailio

Kamailio es un SIP Proxy. Es decir, un ruteador de paquetes SIP.

A diferencia de Asterisk, esta plataforma no intercede en el intercambio de la media (audio, video, mensajes, compartición de pantalla) entre las partes de una comunicación, siendo su única tarea el análisis de las cabeceras SIP para poder rutear los mensajes que le llegan, hacia los usuario u otros proxies SIP. Para poder ubicar usuarios, Kamailio además actúa como un Register Server.

Al igual que en la implementación anterior (WebRTC Asterisk), se continúa utilizando el Web Server (192.168.66.23) que comparte la aplicación WebRTC. Solo que ahora, el registro de usuarios se llevará a cabo en Kamailio (192.168.66.21).



. **Figura 30.-** Implementando WebRTC en Kamailio

Por lo tanto, a la arquitectura actual, simplemente se añadirá un nuevo GNU/Linux Debian 7 en el cual se procederá con la instalación del Proxy SIP Kamailio.

Pasos para instalar y configurar WebRTC con Kamailio:

1.- Instalación GNU/Linux Debian 7 para el servidor Kamailio (Pagina 75).

2.- Configuraciones de red GNU/Linux Debian 7 para el servidor Kamailio.

3.- Instalación del Proxy SIP Kamailio en GNU/Linux Debian 7.

Ingresar con un cliente ssh al servidor Kamailio e ingresar los siguientes comandos:

```
#apt-key adv --recv-keys –keyserver
keyserver.ubuntu.com
0xfb40d3e6508ea4c8
```

4.- Importar clave de los repositorios de Kamailio para debian:

```
#apt-key adv --recv-keys --keyserver
keyserver.ubuntu.com
0xfb40d3e6508ea4c8
```

5.- Crear archivo en */apt/sources.list.d/kamailio.list* con el siguiente contenido:

```
deb http://deb.kamailio.org/kamailio wheezy main
deb-src http://deb.kamailio.org/kamailio wheezy main
```

6.- Actualizar paquetes:

```
#aptitude update
```

7.- Instalar paquetes:

```
#aptitude install kamailio kamailio-mysql-modules
kamailio-websocket-
modules mysql-server
```

8.- Configurar base de datos de usuarios.

Se debe editar el archivo */etc/kamailio/kamctl*, es decir desconectar las líneas:

```
DBENGINE=MYSQL
DBHOST=localhost
DBNAME=kamailio
DBRWUSER="kamailio"
DBRWPW="kamailiorw"
DBROUSER="kamailioro"
DBROPW="kamailioro"
DBROOTUSER="root"
```

9.- Luego lanzar el comando para crear la base de datos y las tablas pertinentes (Ingresar “y” a todas las preguntas):

```
kamdbctl create
```

10.- Configuración principal de Kamailio.

Editar el archivo de configuración principal */etc/kamailio/kamailio.cfg,* copiar el contenido.

Este archivo se muestra en el capítulo 7- (Anexos 7.8.2 - pagina 165).

11.- Reiniciar el servicio para que se carguen los cambios a memoria.

```
#service kamailio restart
```

Si todo está correcto, el comando debería arrojar la siguiente salida:

```
[ ok ] Stopping Kamailio SIP Server: kamailio:.
[....] Starting Kamailio SIP Server: kamailio:
loading modules under config
 path: /usr/lib/i386-linux-gnu/kamailio/modules/
Listening on
             udp: 192.168.66.21:5060
             tcp: 192.168.66.21:5060
             tcp: 192.168.66.21:80
Aliases:

. ok
```

Donde se puede observar cómo se indican los sockets que atienden las peticiones SIP UDP, SIP TCP y SIP Websocket.

12.- Alta de dominio y usuarios.

```
#kamctl domain add example.com
#kamctl domain reload
#kamctl add fulano@example.com 123456
#kamctl add mengano@example.com 123456
```

3.2.5.- Pruebas de comunicaciones Real-Time

Ahora con todos los cambios impactados en memoria, se proceden con las pruebas.

3.2.5.1.- Pruebas en la plataforma Asterisk

.- Registro de endpoints SIP WebRTC en Asterisk.

En este punto se procederá con la registración de usuarios SIP desde endpoints SIP WebRTC, es decir navegadores Web. Además será muy importante el análisis y la lectura que se realice de esta acción bastante trivial. El análisis deberá contemplar los conceptos descriptos durante este trabajo, principalmente la acción de “Handshake” entre el cliente y el servidor SIP Websockets. A continuación se procede con la prueba.

Desde un navegador web chrome o firefox, se accede a la siguiente URL:

http://192.168.66.23/tesis/call.htm?svn=230

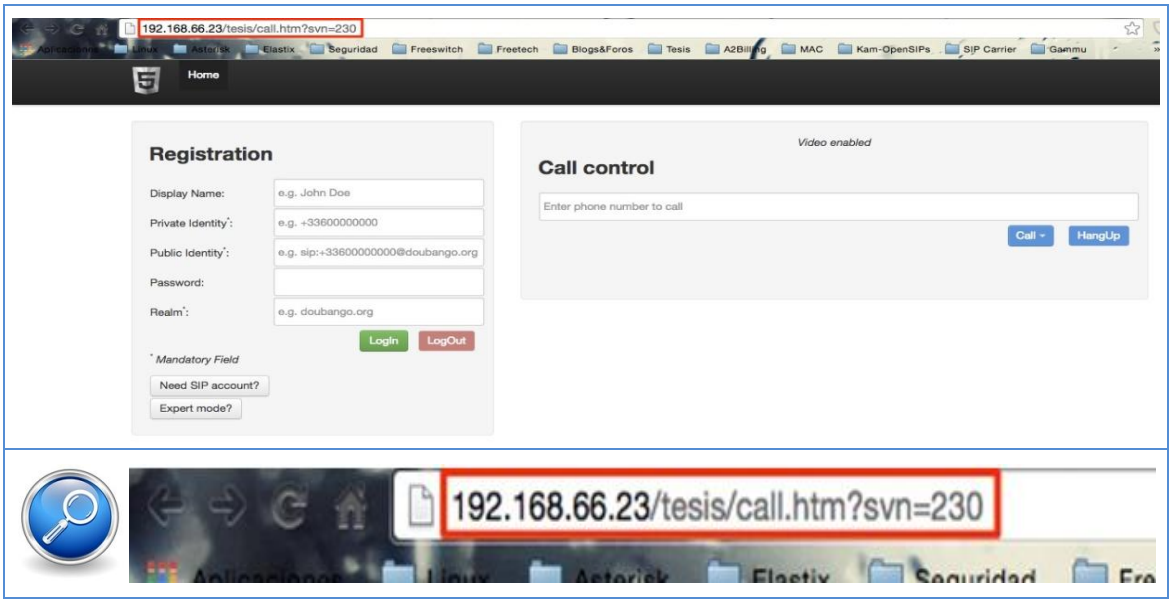


Figura 31.- Pantalla con la URL de acceso.

Se debe visualizar un sitio web similar a la siguiente figura:

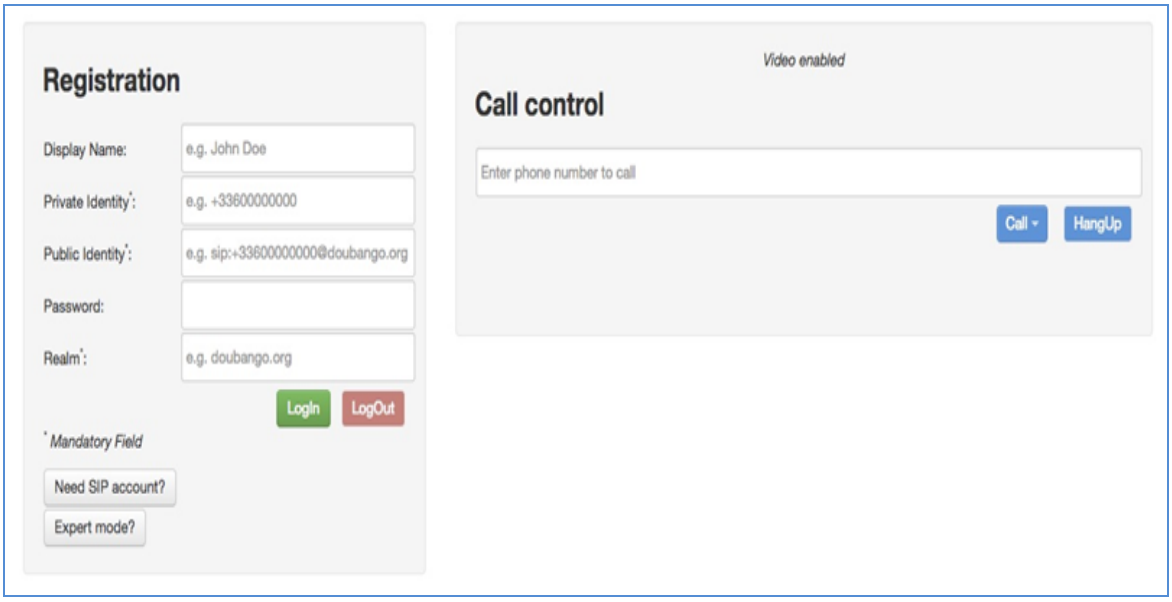


Figura 32.- Pantalla web de aplicación WebRTC.

Se trata de la aplicación WebRTC que permite registrar un endpoint SIP WebRTC de manera tal que se pueda recibir llamadas desde otros enpints de Asterisk y realizar llamadas hacia estos.

Se deben editar algunos parámetros de configuración, para indicar a la aplicación la dirección IP del servidor Asterisk, usuario y contraseña para realizar la registración SIP.

Para ello, se debe hacer click en el botón: Expert Mode?

Se abre una nueva pestaña en el navegador, con la siguiente vista:

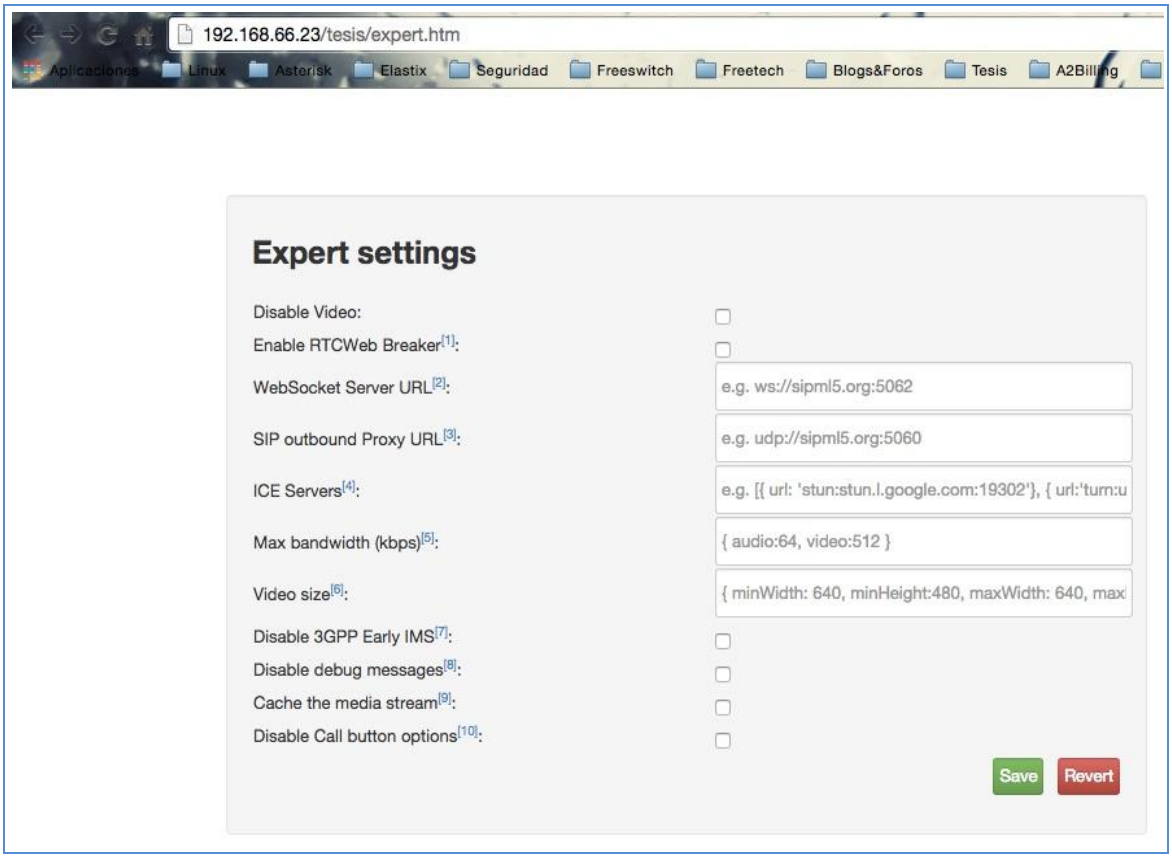


Figura 33.- Pantalla Expert settings.

Esta pantalla, se modifica y queda de la siguiente forma:

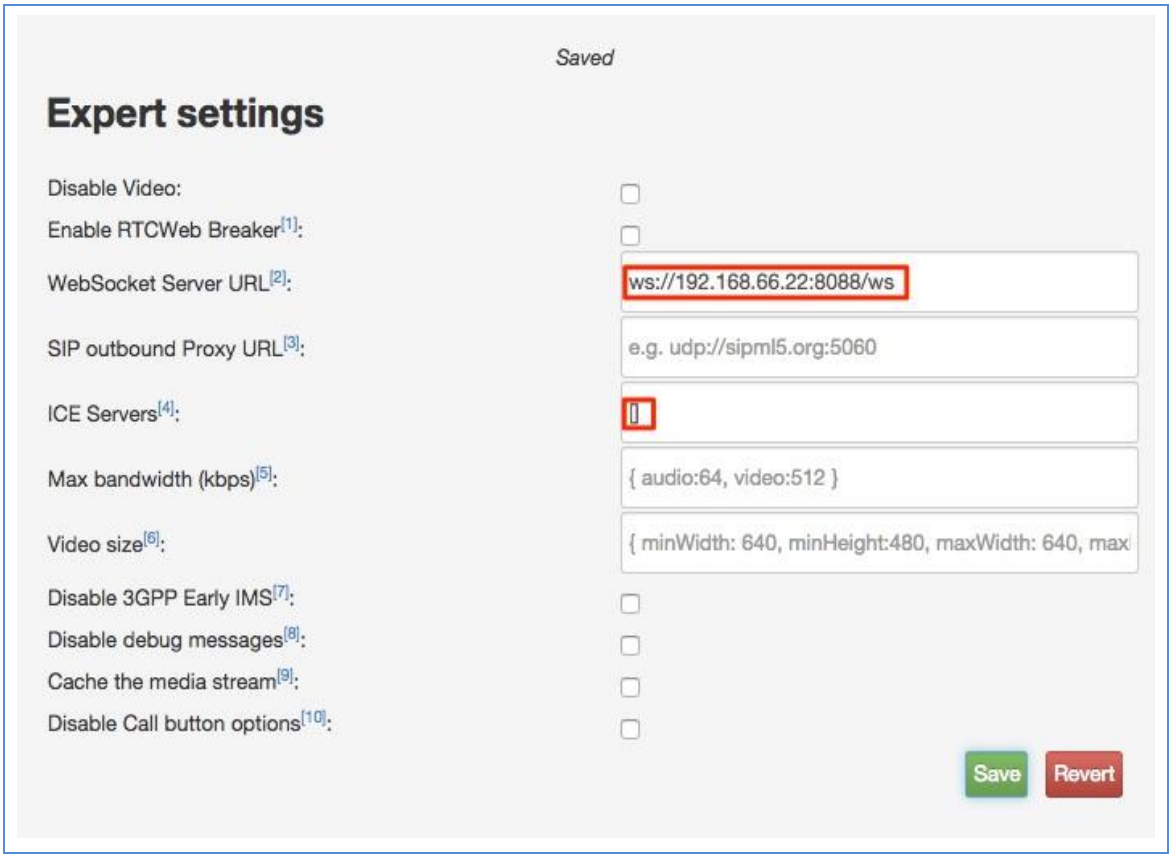


Figura 34.- Pantalla Expert settings modificada.

A continuación se hace click en el botón guardar y se vuelve a la pestaña principal de la aplicación.

En la pantalla principal, se completan los campos de acuerdo a la siguiente figura:

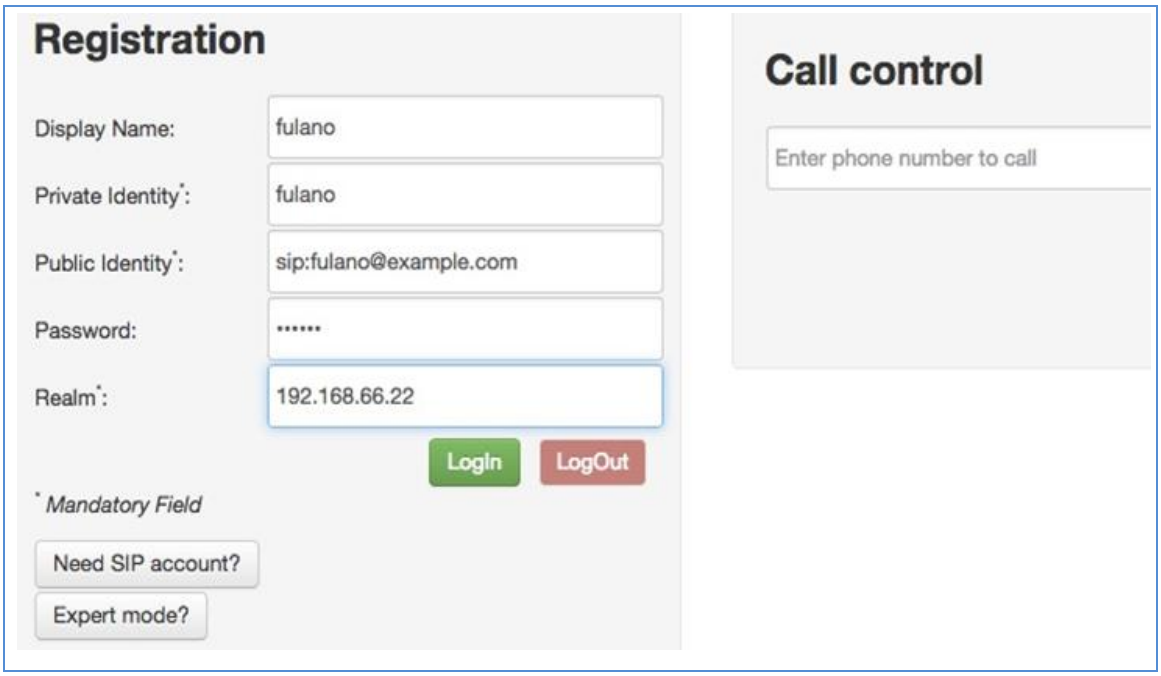


Figura 35.- Pantalla principal modificada.

Donde el campo “password”, es el parámetro “secret” en la definición del endpoint SIP WebRTC dentro del archivo sip.conf de Asterisk.

Una vez completados los parámetros, se procede con el registro (se hace click en el botón Login).

Si todo fue correcto, debe aparecer una leyenda que indica que se ha registrado correctamente.

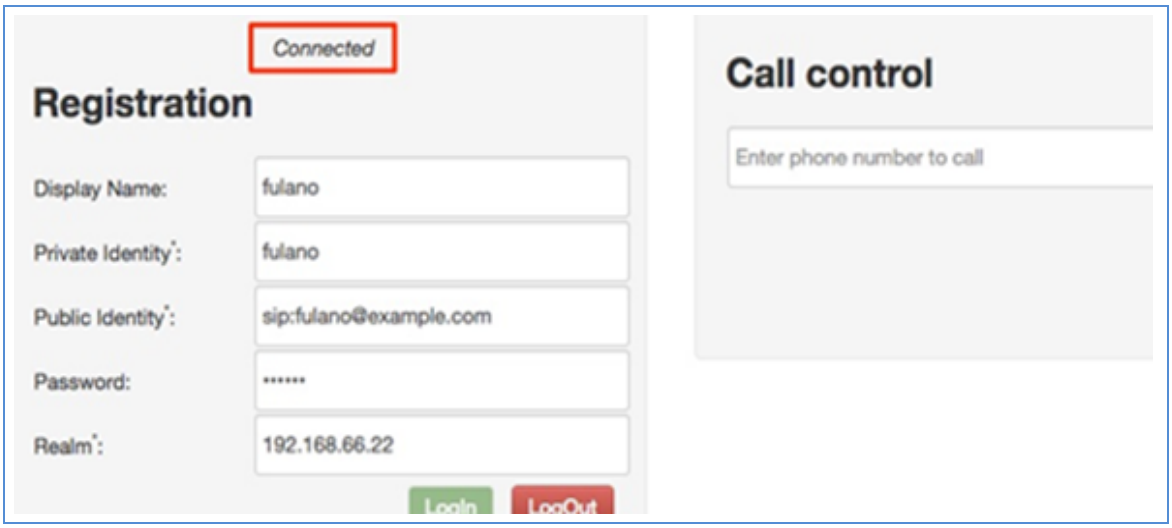


Figura 36.- Endepoint WebRTC Conectado.

Para el registro de otro endpoint WebRTC, se debe realizar los mismos pasos y se usa el usuario remanente (usuario: fulano, password: 123456), este se configura desde otro navegar web en otro host (smartphone, tablet o sistema x86).

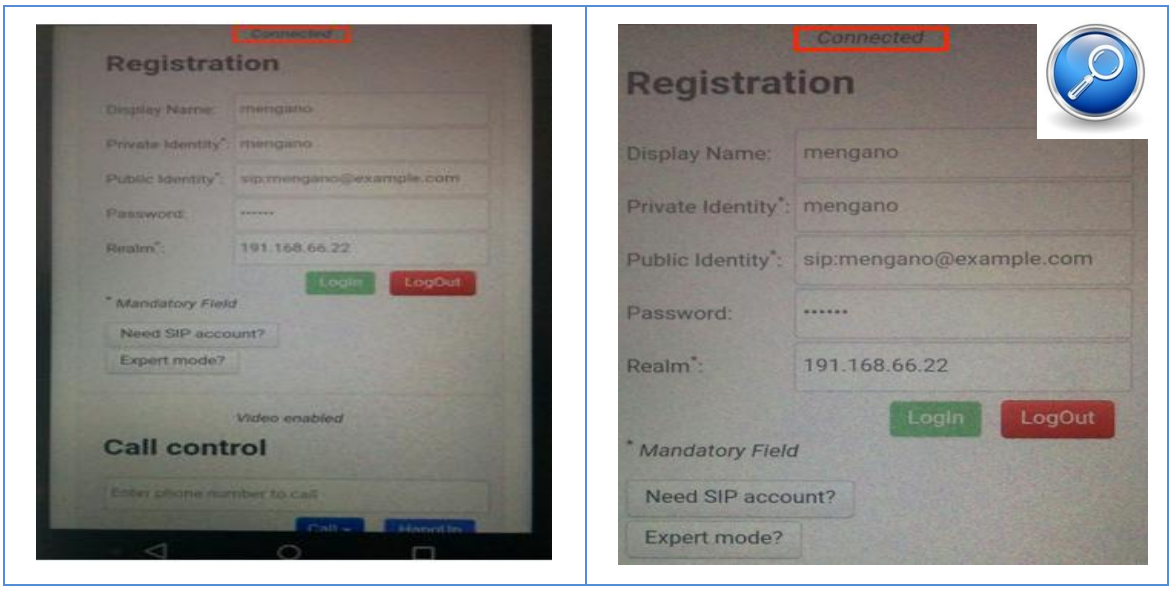


Figura 37.- Aplicación abierta en un smartphone Motorola Moto X.

Además, se debe comprobar el concepto descripto en la sección SIP sobre Websocket, en la cual se afirmaba que durante el handshake de conexión, se sucedían unos pasos que ahora se proceden a comprobar. Para ello, se realiza una captura de paquetes en el servidor Asterisk, utilizando el sniffer de protocolos “tcpdump”.

En el server Asterisk:

```
tcpdump -s0 -nn -U -w websocekt.pcap tcp port 8088
```

Con este comando se lanza la captura de paquetes.

Ahora se procede con el registro del próximo endpoint SIP WebRTC.

Una vez realizado el registro, se procede a cortar la ejecución de la aplicación “tcpdump” en el servidor Asterisk.

La misma genera un archivo con extensión “.pcap”, lo cual permite abrirlo con la aplicación de escritorio Wireshark, lo cual facilitará la visualización de la captura almacenada previamente.

El paso siguiente es traer la captura desde el servidor hacia una PC y abrir dicho archivo con la aplicación Wireshark.

Si se analiza la captura, se puede verificar lo siguiente:

Durante Handshake Websocket además de acordar realizar el Upgrade de HTTP a Websockets, se informa el protocolo de aplicación que se va a dialogar sobre la conexión Websocket. Esto se debe informar en el campo “Sec-WebSocket-Protocol” del Header HTTP en ambos mensajes del Handshake (El HTTP Get Upgrade que envía el cliente y el HTTP 101 que responde el server)

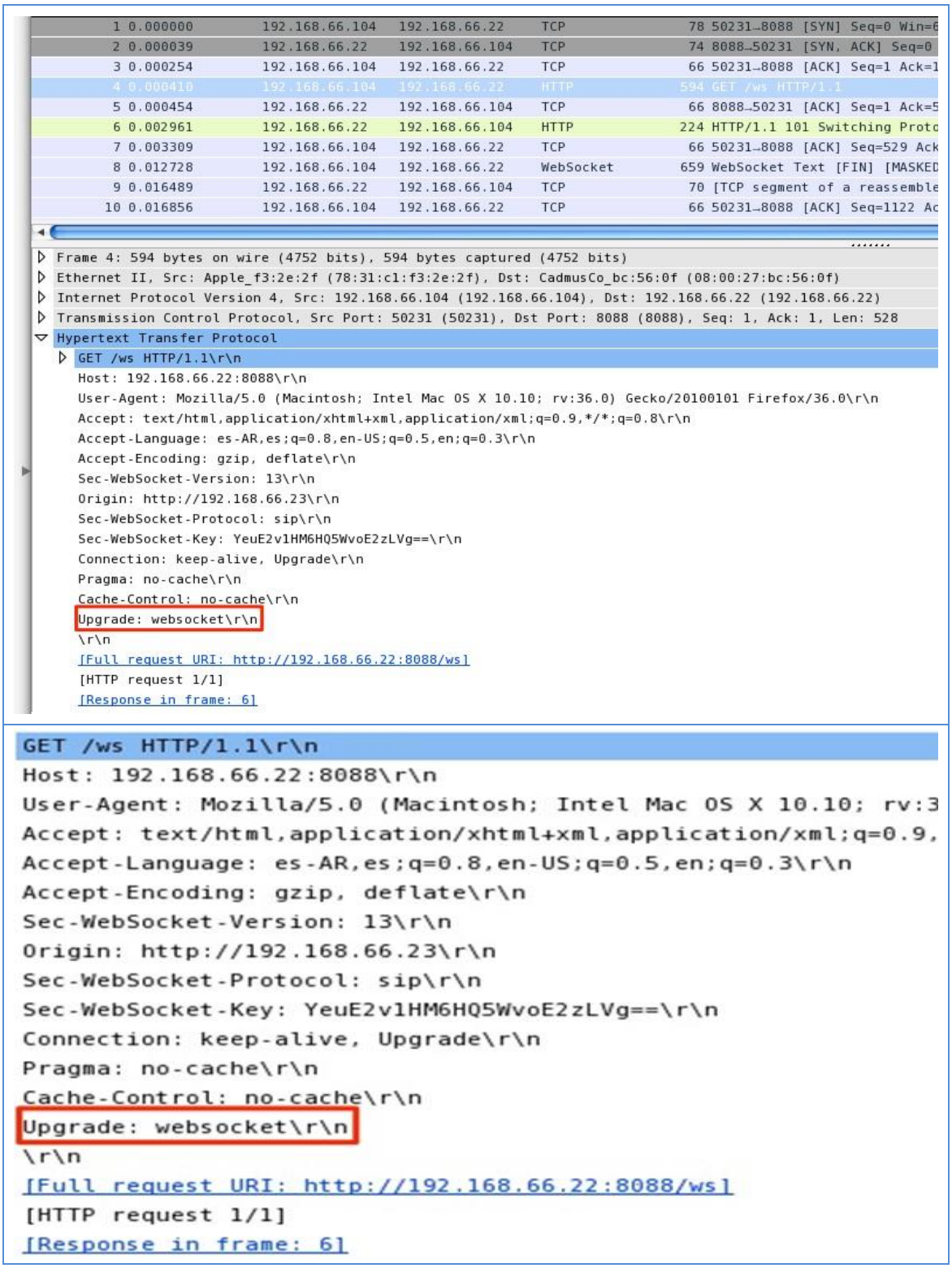


Figura 38.- Establecimiento de conexión websocket entre aplicación y servidor WebRTC (Solicitud).

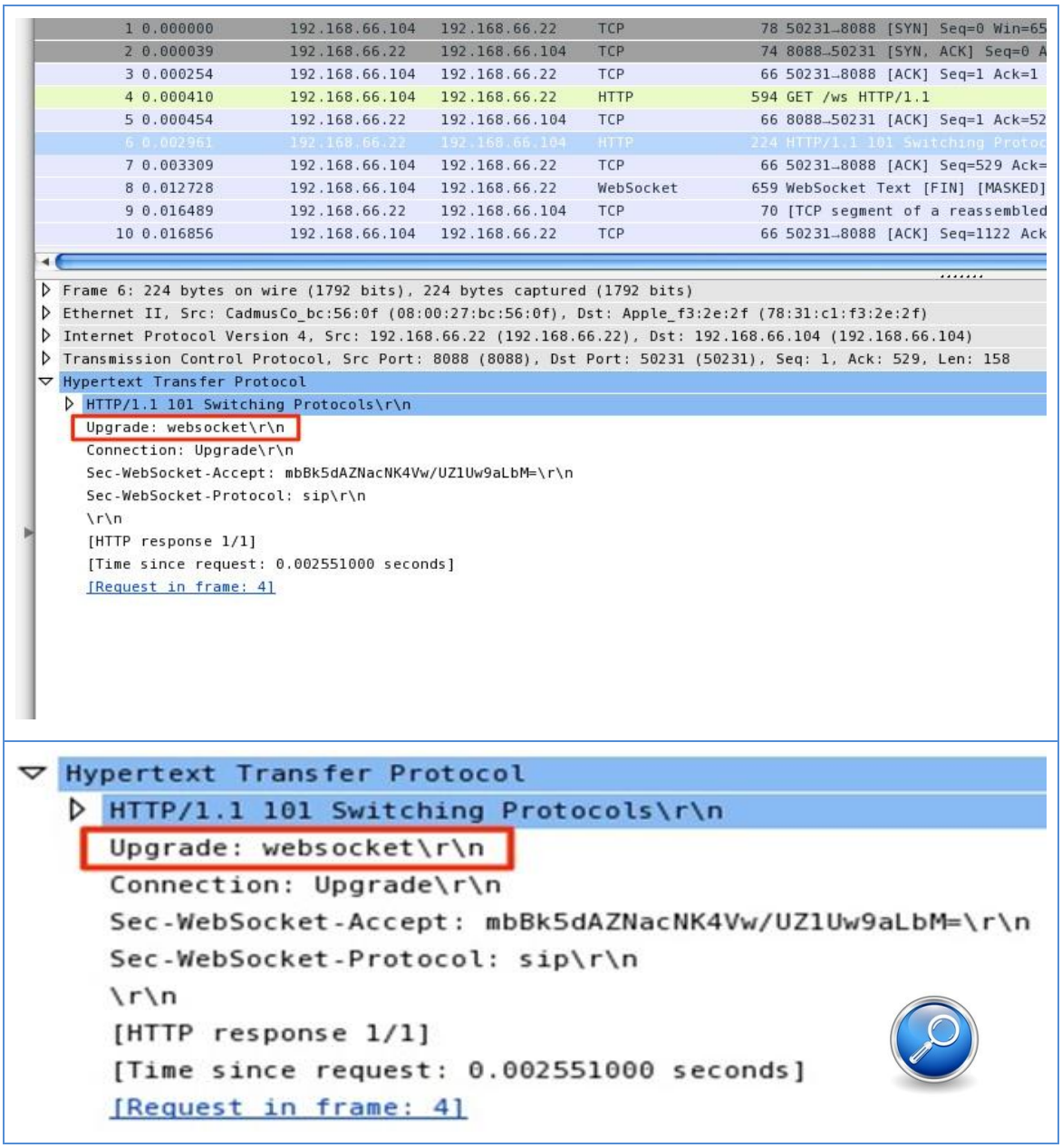


Figura 39.- Establecimiento de conexión websocket entre aplicación y servidor WebRTC (Respuesta).

A partir de dicha negociación ambas partes acuerdan usar el protocolo Websocket para intercambiar solicitudes y respuestas SIP.

Los paquetes Websocket pueden transportar frames UTF-8 o frames binarios. Por lo tanto los Clientes SIP Websocket y Servidores SIP Websocket deben soportar ambos métodos: Frames UTF-8 y Frames binarios.

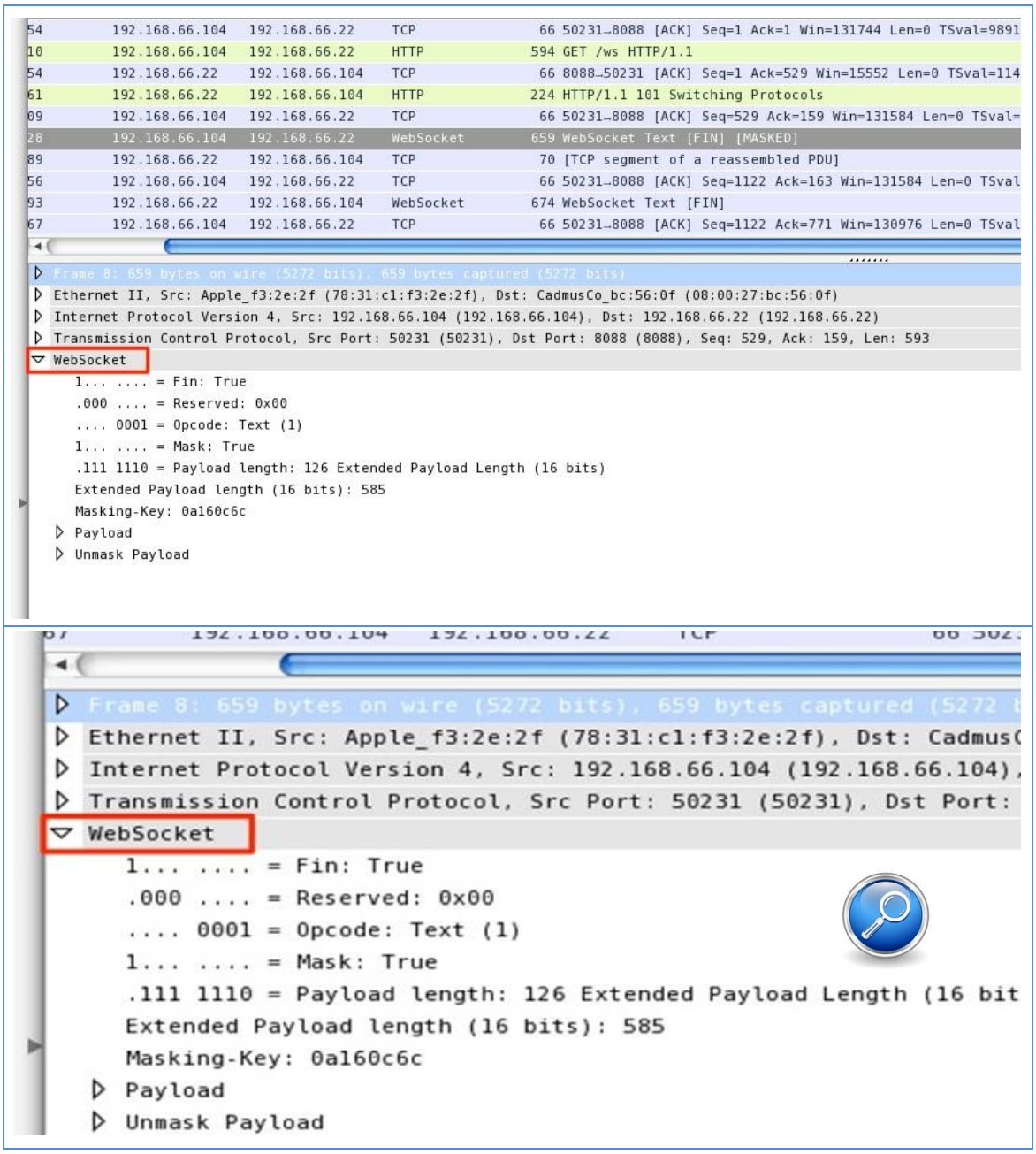


Figura 40.- Websocket establecido entre Cliente y Servidor WebRTC.

Si se deja unos minutos corriendo la nueva captura en el servidor Asterisk, se va a poder comprobar otros conceptos. El hecho de que la conexión Websocket entre el cliente y el servidor se mantiene abierta usando un método “keep-alive” es decir que cada navegador en vía un “ping” Websocket frame. La frecuencia con la que se envía estos frames lo dispone cada navegador web.

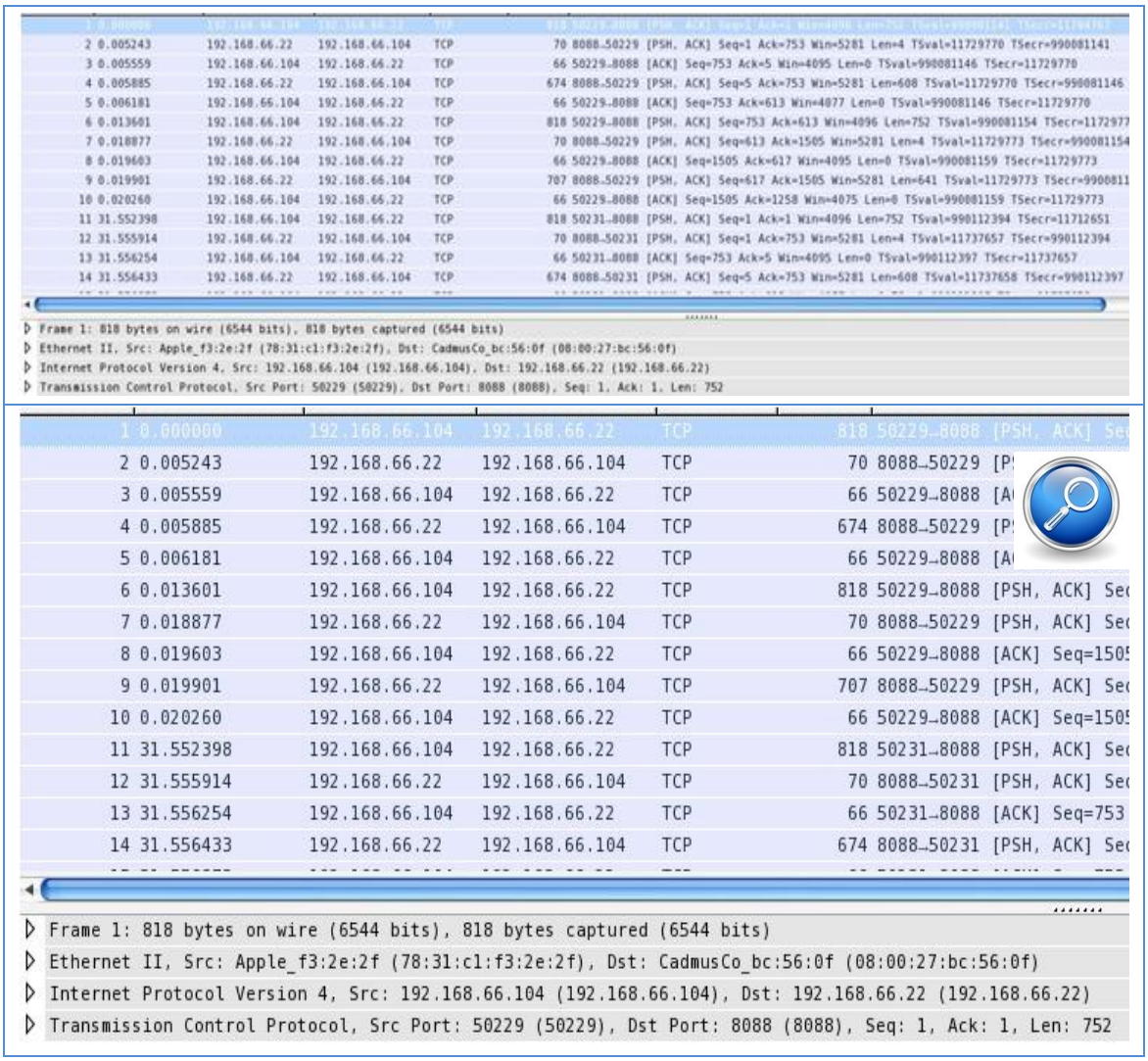


Figura 41.- Técnica de “Keep Alive” para mantener la conexión TCP Websocket establecida.

.- Registro de endpoint SIP UDP en Asterisk.

Como endpoint SIP UDP, se plantea el softphone Zoiper, el mismo se puede obtener de manera gratuita en el sitio:

http://www.zoiper.com/en/voip-softphone/download/zoiper-classic

Una vez descargado, se debe instalar y ejecutar.

Figura 42.- Aplicación Softphone con soporte de protocolo SIP (UDP - TCP).

Para configurar la cuenta SIP UDP, se debe hacer click en el ícono “herramienta” y se despliega una ventana donde se pueden indicar los parámetros de configuración SIP, similar a la siguiente figura:

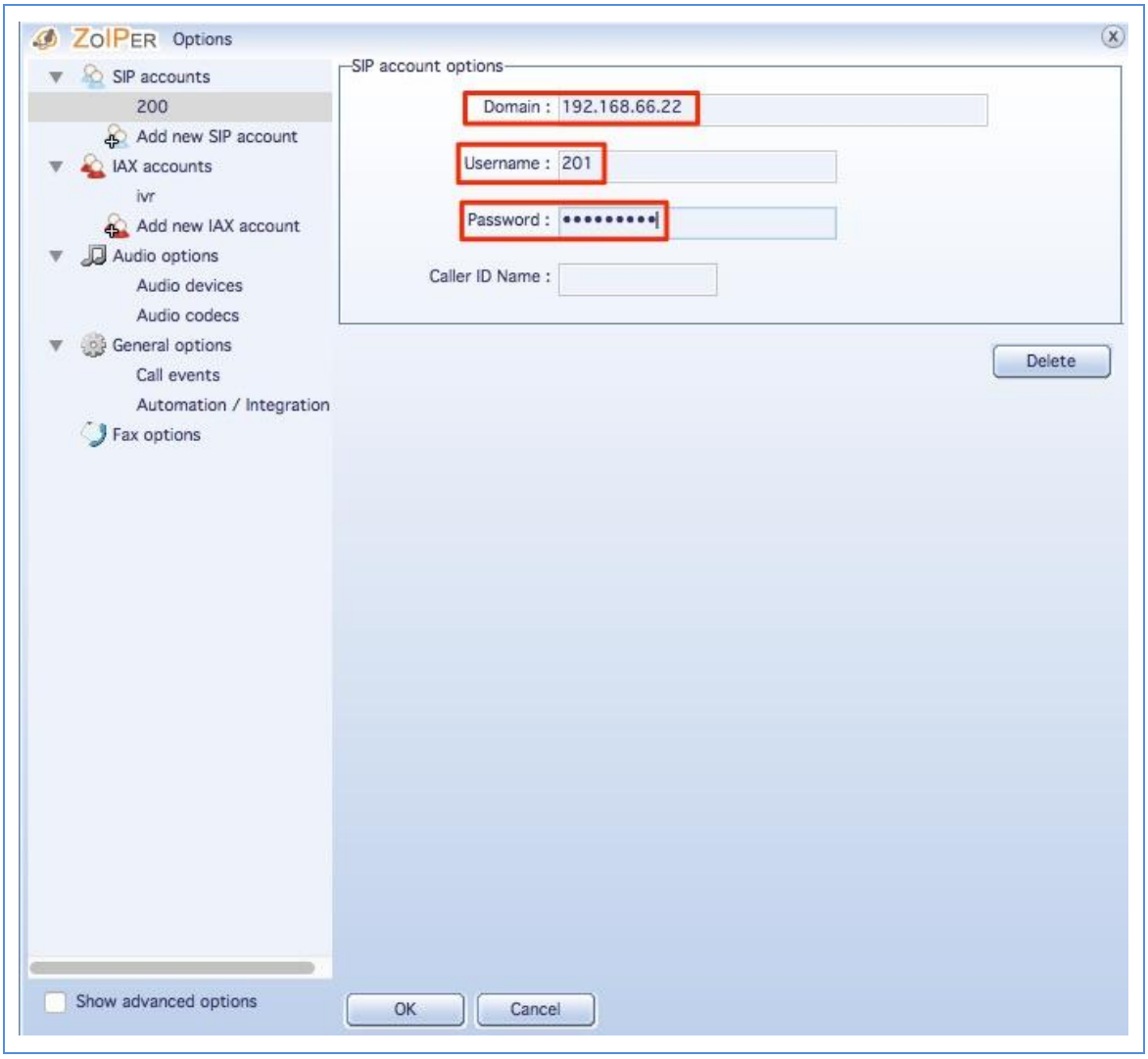


Figura 43.- Configuración de usuario SIP-UDP para registrar en el servidor SIP.

Donde nuevamente los datos ingresados, deben ser los creados en Asterisk para el endpoint SIP UDP.

Se guardan los cambios dando click en el botón “OK” y se vuelve a la ventana principal de la aplicación. Una vez en esta, se debe hacer click en el botón “Register” y si todo fue bien, debe aparecer la leyenda “Registered”, indicando que el endpoint está listo para usar.

Figura 44.- Usuario SIP-UDP registrado exitosamente en el Servidor SIP.

.- Llamada de audio entre dos endpoints SIP WebRTC

Se realiza la primera prueba de llamada. Se toma un navegador web y se "disca" al usuario del otro navegador.

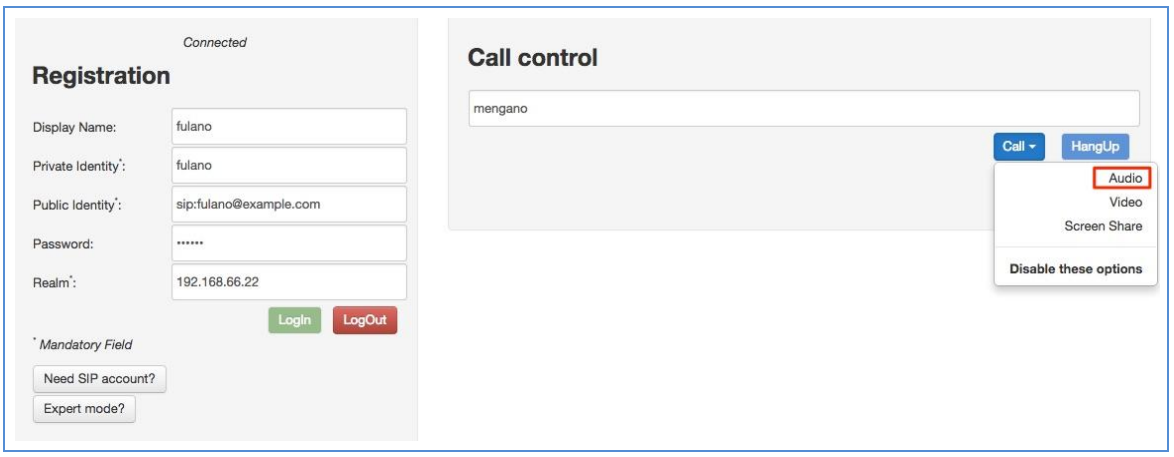


Figura 45.- Test de llamada entre dos usuarios SIP-WebRTC P-I.

Si todo va bien, en el otro navegador debe comenzar el “ring” y si se acepta la llamada, ambos endpoints quedan comunicados.

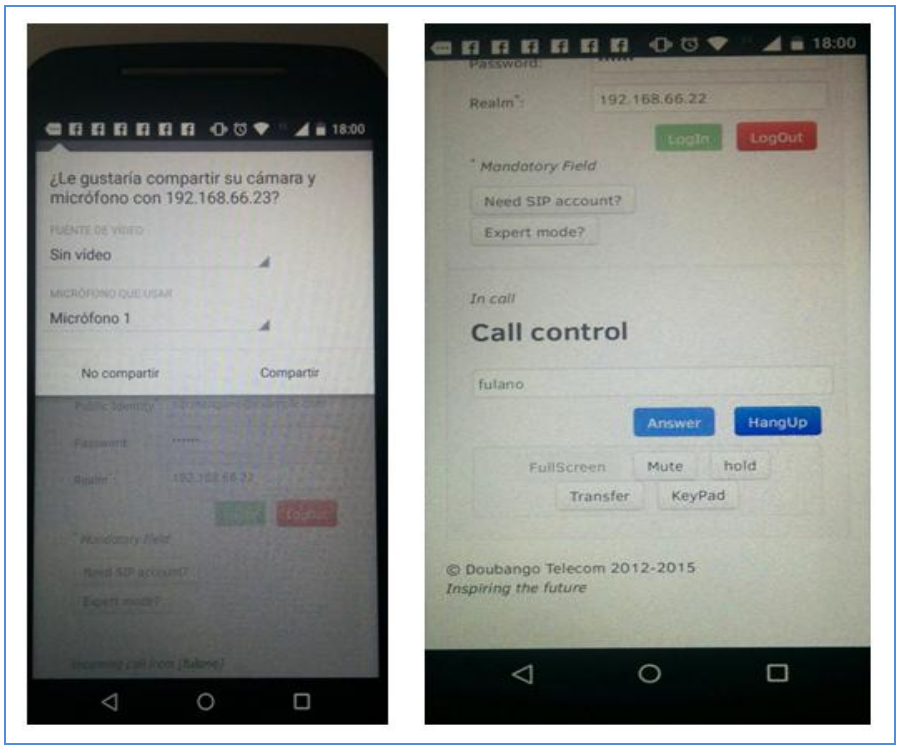


Figura 46.- Test de llamada entre dos usuarios
SIP-WebRTC P-II

NOTA: *se puede repetir la prueba, poniendo atención que tal como se describió en módulos anteriores, tanto en el establecimiento de la comunicación por parte del endpoint que realiza la llamada, como en la aceptación de la misma por parte del endpoint que acepta la llamada, siempre la el navegador pregunta si se desea permitir el acceso al micrófono y webcam por parte de la aplicación WebRTC.*

.- Llamada de audio entre un endpoint SIP WebRTC y otro SIP UDP

Para realizar esta prueba de llamada, simplemente se debe tomar un navegador web y “discar” al número del endpoint Zoiper (201) y viceversa.

.- Llamada de audio entre un endpoint SIP WebRTC y un abonado PSTN

Bajo este escenario se plantea que el usuario del endpoint SIP WebRTC pueda discar un número de algún abonado PSTN, de manera tal que se pueda comprobar la interacción.

NOTA: *Para esta prueba se debe configurar un SIP Trunk con terminación a la PSTN.*

.- Llamada desde un abonado PSTN a un endpoint SIP WebRTC

En esta prueba, se podrá discar el número de abonado correspondiente a la central Asterisk y debido al plan de discado que afectan las llamadas entrantes desde el troncal SIP del proveedor con terminación PSTN, las mismas se encaminan hacia un ring en simultáneo de los dos endpoints SIP WebRTC (fulano y mengano).

NOTA: *Para esta prueba se debe configurar un SIP Trunk con terminación a la PSTN.*

.- Llamada desde un endpoint SIP WebRTC activando el debug Javascript del browser.

Para la última prueba, se intenta reflejar toda la teoría planteada acerca de SIP y SIP sobre Websockets usando librerías que implementa SIP mediante código Java Script.

Para ello, se debe activar la consola de debug del navegador web donde esté corriendo la aplicación WebRTC. Para este ejemplo, se va a realizar con Firefox.

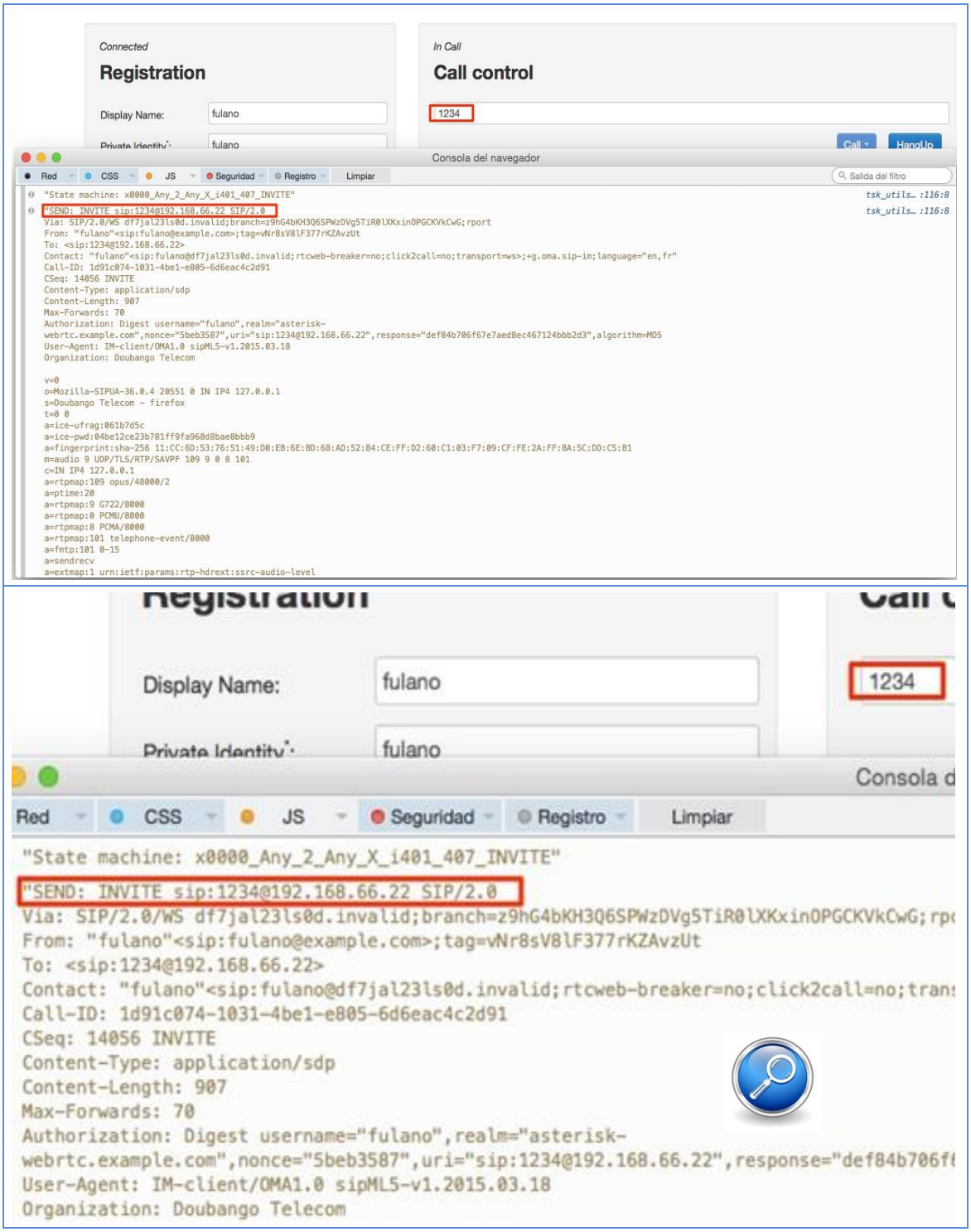


Figura 47.- Trafico de llamada desde un navegador “Mozilla” utilizando la herramienta de debug para Java Script.

.- Conclusión

La implementación de WebRTC en Asterisk plantea un escenario estable para llamadas de audio. No es el caso de las comunicaciones de video-llamadas, compartición de escritorio o mensajería instantánea.

Con respecto a las comunicaciones de video llamadas, el impedimento surge del hecho de que Asterisk al día de la fecha no incluye el códec VP8 por cuestiones de patentamiento. Al tratarse de un B2BUA o Back to Back User Agent, no se permite el diálogo directo entre los navegadores, siendo obligatorio que tanto la señalización como la media pase por el servidor Asterisk.

La mensajería instantánea y compartición de escritorio, no vienen implementados a nivel SIP en Asterisk, por lo que es imposible que funcione.

No obstante, Asterisk siempre es una herramienta bastante ajustada a los estándares, muy implementada alrededor del mundo con IPPBX a nivel corporativo y con una curva de aprendizaje accesible, lo cual lo hace muy interesante a la hora de tener en cuenta el uso de WebRTC.

3.2.5.2.- Pruebas en la plataforma Kamailio

.- Registrar los endpoints WebRTC

Desde un navegador web chrome o firefox, se accede a la siguiente URL:

http://192.168.66.23/tesis/call.htm?svn=230

Se deben seguir los pasos ya listados en el escenario anterior, solo que hay que modificar la dirección IP y puerto del servidor Websocket, quedando de la siguiente manera.

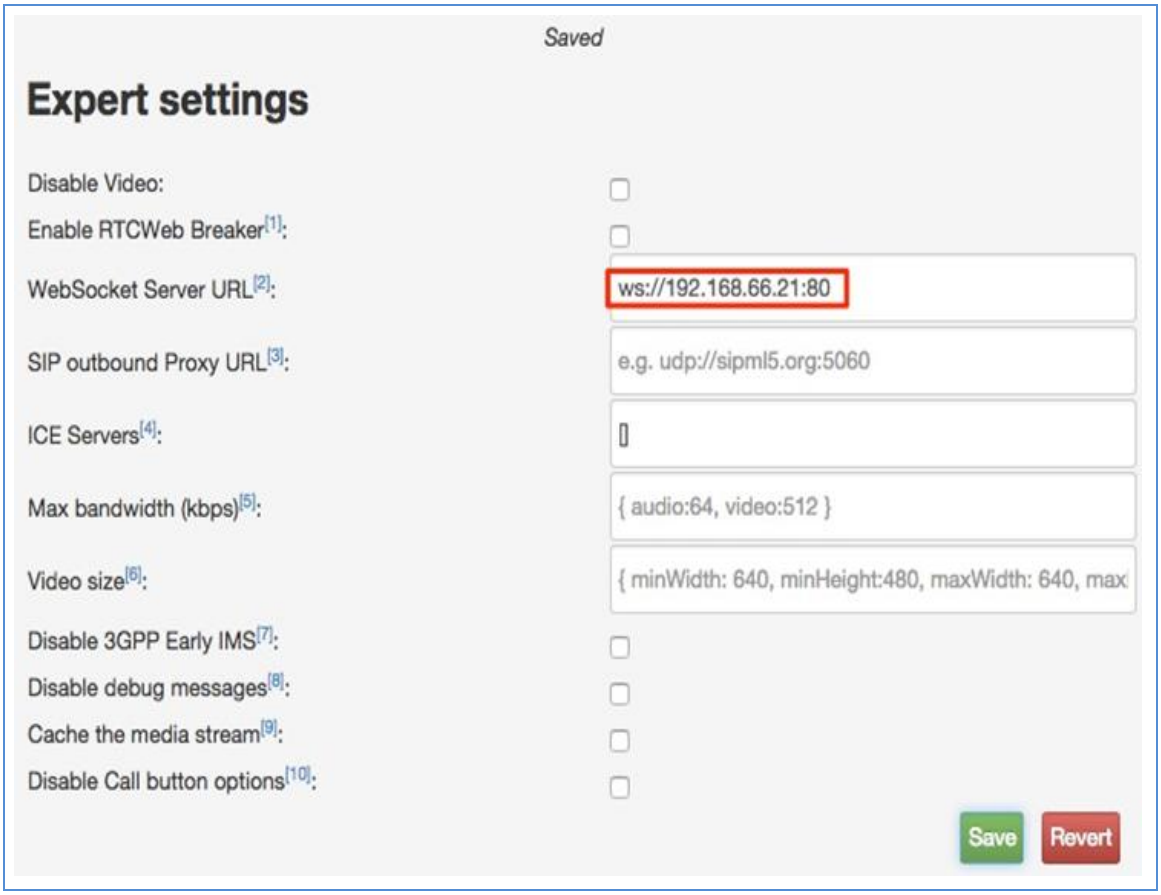


Figura 48.- Parámetros de configuración para registrar un cliente WebRTC en el servidor WebRTC Kamailio.

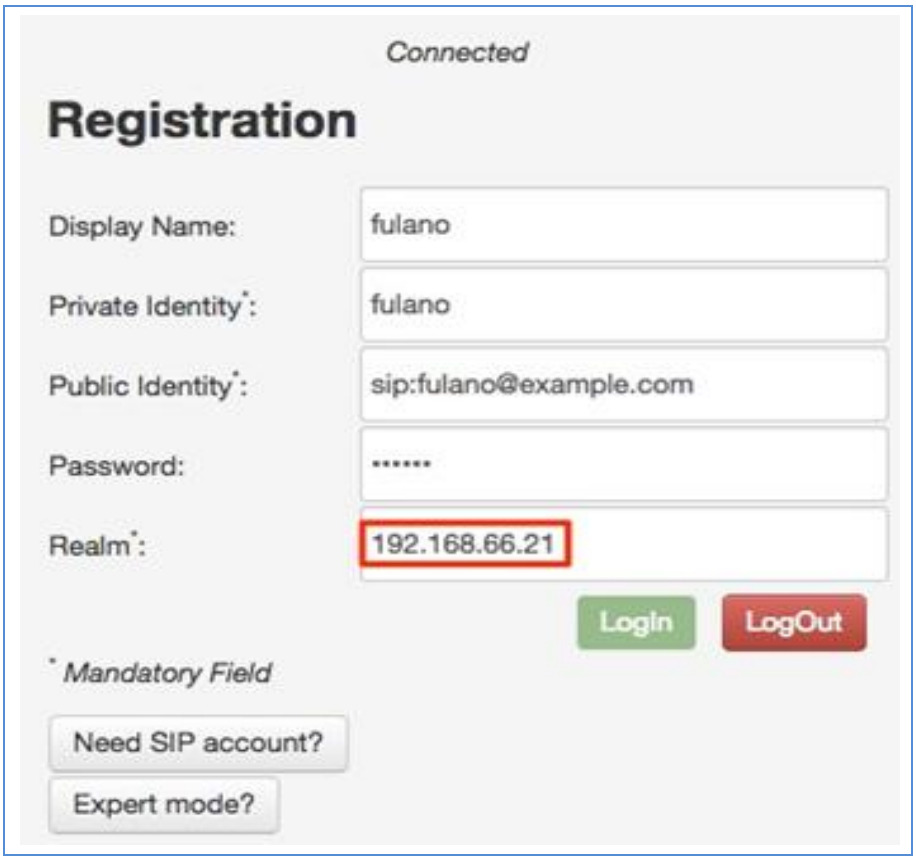


Figura 49.- Cliente WebRTC correctamente registrado en el servidor WebRTC Kamailio.

.- Pruebas de llamadas entre los dos endpoints SIP WebRTC

Se repite la prueba realizada en el escenario Asterisk WebRTC, solo que ahora se pone énfasis en la comunicación de video-llamada, que fue algo que no se pudo realizar en el escenario anterior, por limitaciones de Asterisk.

.- Comprobar la mensajería instantánea vía SIP WebRTC

Para avanzar sobre esta prueba, se debe proceder con la ejecución de una aplicación WebRTC que incluye el tratamiento de Mensajes Instantáneos. Para ello, se accede al sitio web http://tryit.jssip.net/

Esta aplicación es un ejemplo de WebRTC softphone que comparte la gente del proyecto mencionado en capítulos anteriores JSSIP (jssip.net).

De manera similar a lo realizado en la aplicación WebRTC utilizada hasta ahora, se deben completar los parámetros de autenticación y registro de los usuarios en el servidor Kamailio.

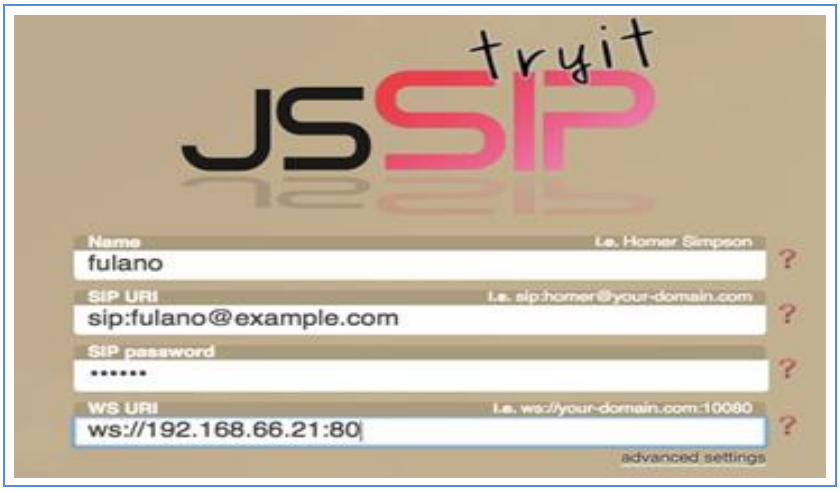


Figura 50.- Parámetros de configuración para registrar un cliente WebRTC en el servidor WebRTC Kamailio.

Si todo fue correcto, se visualiza el registro del usuario en la aplicación.

Figura 51.- Cliente WebRTC correctamente registrado en el servidor WebRTC Kamailio.

Se inicia una conversación vía Mensajería Instantánea.

Figura 52.- Test de mensajería instantánea entre dos usuarios WebRTC registrados en el servidor WebRTC Kamailio.

A continuación se realiza una conexión de video llamada, de la misma manera que se realizo con la aplicación WebRTC anterior.

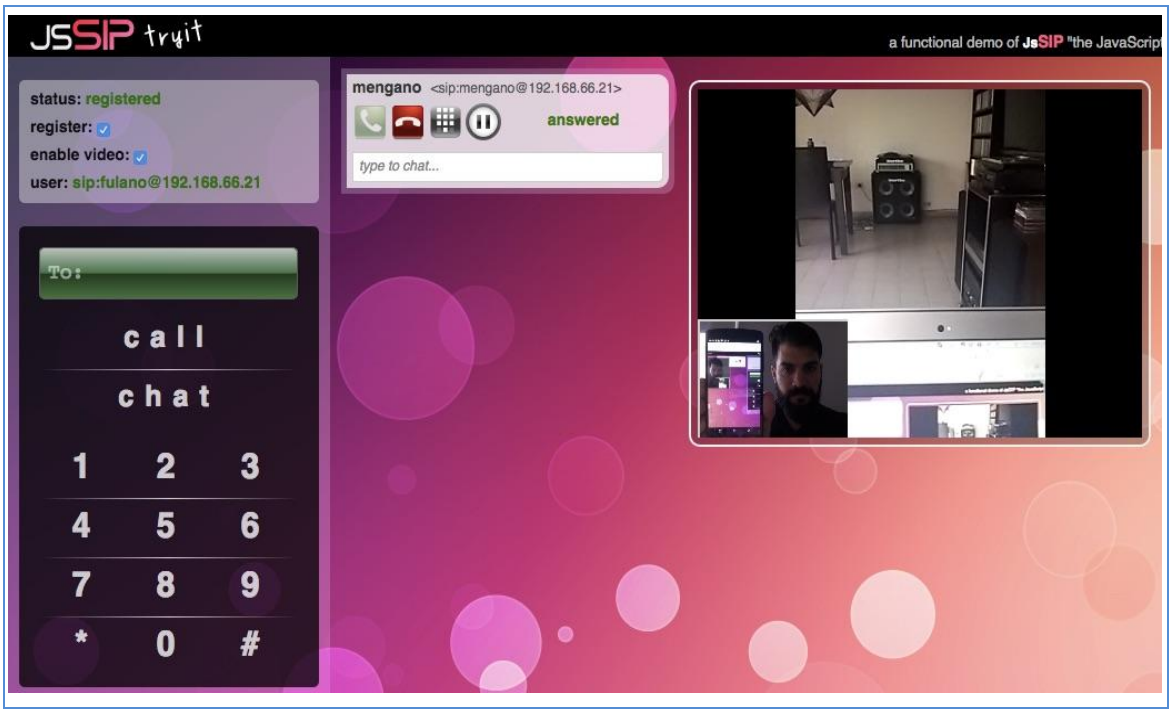


Figura 53.- Video llamada realizada entre dos clientes WebRTC registrados en el servidor WebRTC Kamailio.

.- Pruebas de llamadas y análisis de los paquetes SIP tanto a nivel navegador web (activando la consola de Javascript) como en el servidor SIP Proxy Kamailio (Utilizando el sniffer ngrep)

Para realizar esta prueba, se instala el sniffer ngrep en el GNU/Linux Kamailio.

```
#aptitude install ngrep -y
```

Una vez instalado, se procede a ejecutar una captura de paquetes SIP Websockets (TCP 80).

```
#ngrep -W byline -d eth0 port 80
```

Se envía un mensaje instantáneo desde un browser a otro.

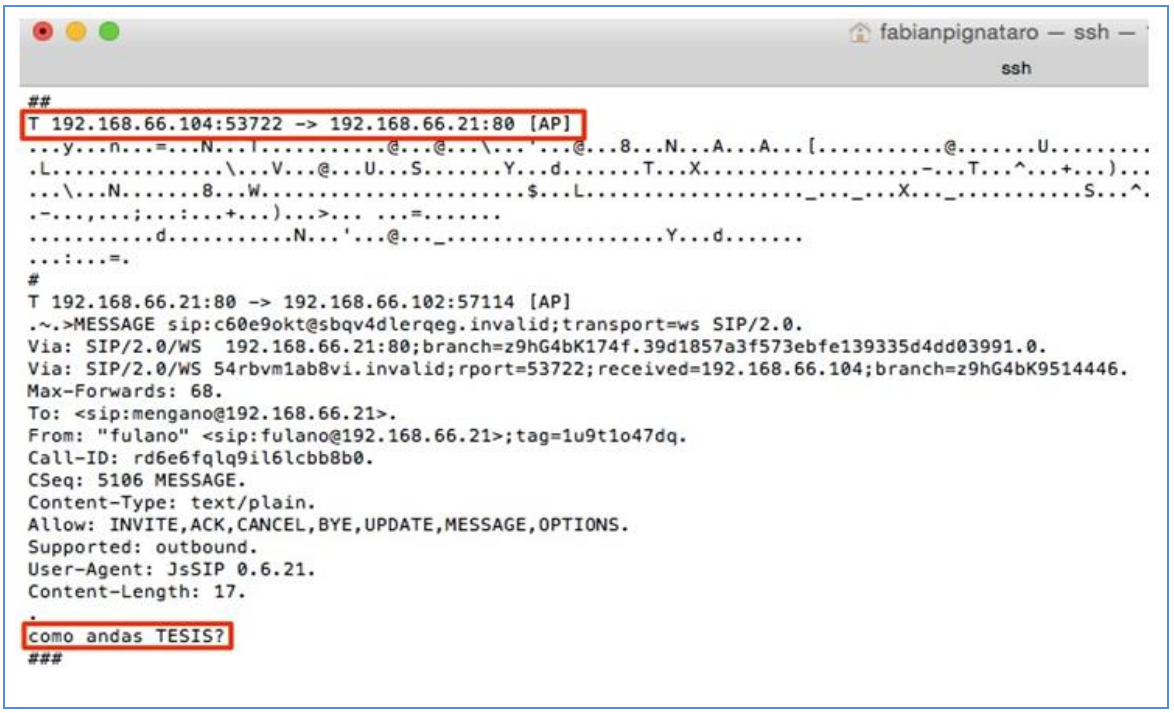


Figura 54.- Mensaje desde el browser al Proxy SIP Kamailio.

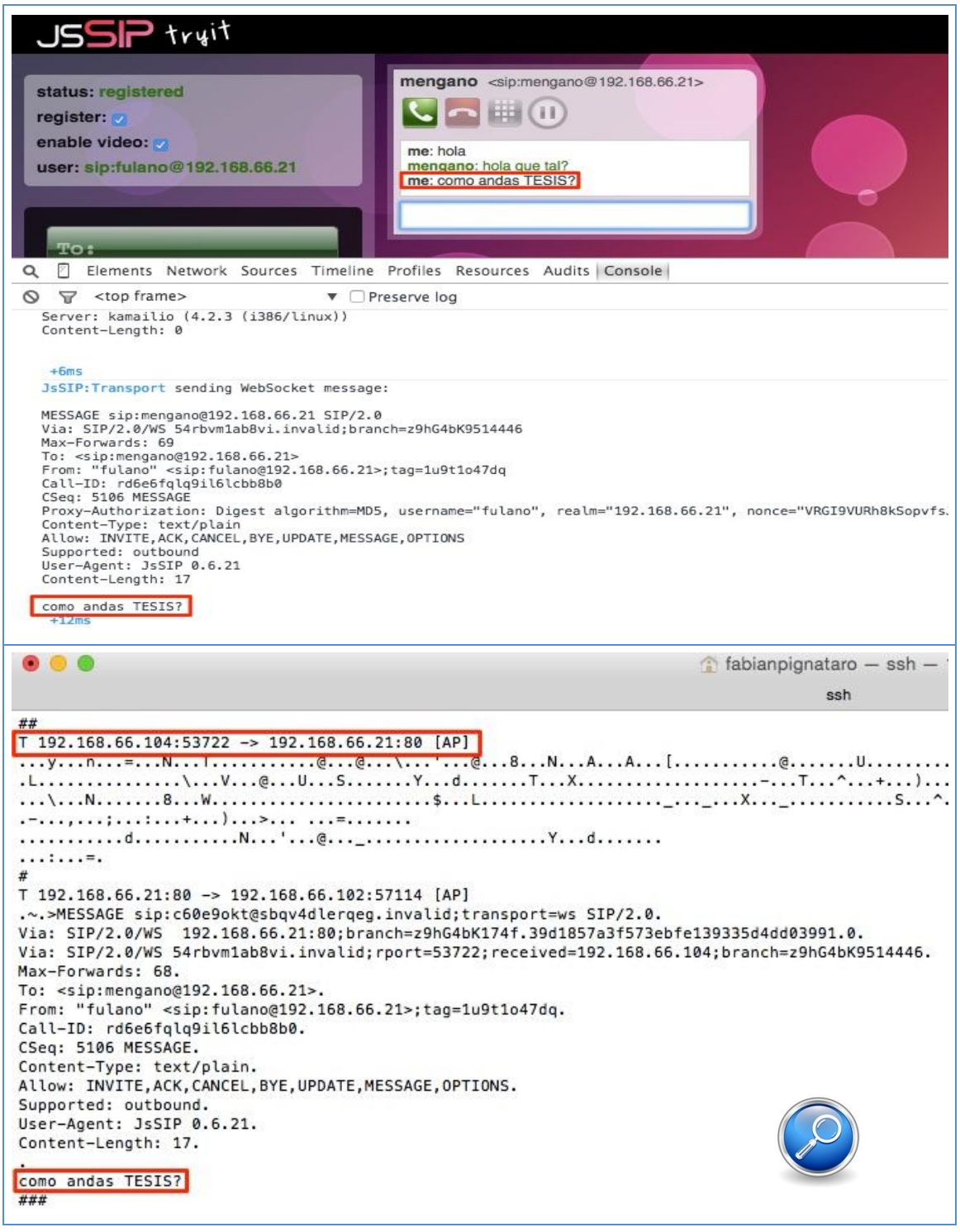


Figura 55.- Análisis de un paquete Websocket que transporta un mensaje instantáneo (respuesta), a partir de lanzar la captura de paquetes en la consola de comandos del host Proxy SIP Kamailio.

Es importante notar como fluyen los mensajes SIP desde el browser hacia el Proxy SIP.

.- Conclusión

Las pruebas que se realizan en Kamailio SIP Proxy fueron satisfactorias en un 100% ya que se pudieron comprobar las funcionalidades como Mensajería Instantánea y Video Llamadas, además de las llamadas de audio. El hecho de que Kamailio funcione como SIP Proxy, sin intervenir en la “media”, dejando que los browsers intercambien la media entre ambos, permite que las comunicaciones de video llamadas funcionen a la perfección. Por otro lado, como Kamailio implementa la mensajería instantánea sobre SIP este servicio también resulta un éxito.

No obstante, Kamailio resulta ser una plataforma con una curva de aprendizaje con una pendiente muy pronunciada, lo cual lo hace muy complicada de implementar en un 100% de sus funcionalidades. Como por ejemplo, trabajar en la interacción entre los usuarios SIP WebRTC y SIP UDP, SIP TLS o SIP TCP, así como también la interacción con abonados PSTN. Si bien, con bastante configuración y estudio de la plataforma, todo esto se puede lograr perfectamente, el nivel escapa al alcance de este trabajo.

3.3.- RESULTADOS

A continuación, se detallan los resultados alcanzados desde dos puntos de vista, uno con relación a los objetivos planteados y el otro, desde una perspectiva técnica.

Por lo tanto, en concordancia con los objetivos planteas, se puede decir que se han realizado las siguientes acciones:

- *Se cumplió con el objetivo general de establecer comunicaciones en tiempo real a través de una aplicación web.*

- *Se ha estudiado y aplicado la tecnología WebRTC y los nuevos protocolos de comunicación que permiten implementar comunicaciones unificadas en un navegador web.*

- *Se ha implementado una arquitectura cliente-servidor que utiliza la tecnología WebRTC y SIP sobre WebSocket.*

- *Se ha Implementado una arquitectura hibrida conformada por terminales convencionales de voz, video, mensajería y tecnología WebRTC.*

- *Se definió y documento esta nueva tecnología.*

Y finalmente, desde el punto de vista técnico, se destacan los siguientes resultados:

- *Se ha desarrollado una nueva tecnología de comunicación.*

- *Se ha demostrado con éxito el funcionamiento de la tecnología planteada.*

- *Se ha definido como implementar las comunicaciones Real-Time para cualquier desarrollador que desee aplicarla. Con explicaciones paso a poso y definiendo cada sentencia.*

3.4.- DIFICULTADES QUE SE HAN PRESENTADO

Al ser una tecnología que aún está en proceso de estandarización, los principales inconvenientes apuntan a los cambios introducidos por los navegadores a medida que van lanzando nuevas actualizaciones.

Desde que comenzó el proceso de escritura de este trabajo final, diferentes versiones de Firefox y Chrome han presentado diferentes problemas principalmente en el dialogo con el framework Asterisk.

Un ejemplo de esto fue la actualización de Google Chrome en la cual a partir de la versión 35, se deja de soportar SDES para centrarse en DTLS como único método de para proporcionar privacidad en el intercambio de claves durante el proceso de establecimiento de una comunicación.

En el transcurso del trabajo se probaron diferentes alternativas para lograr la correcta comunicación entre versiones de Firefox y Chrome con Asterisk.

Alguna de ellas fue:

Congelar el versionado: *es decir, se buscó una combinación de versiones de Google Chrome, Firefox y Asterisk de manera tal que todo funcione. No obstante esto implicó quedar retrasado en lo que respecta a las nuevas versiones que los componentes fueron ofreciendo al mercado.*

Utilizar un componente Proxy; *se trata de un módulo que lanzó al mercado la empresa responsable de la librería WebRTC SIPML5 y básicamente oficia de traductor de tecnologías. Es capaz de dialogar WebRTC con los navegadores y SIP UDP convencional con Asterisk, de esta manera se garantiza el funcionamiento de WebRTC y Asterisk manejando las últimas versiones disponibles en el mercado.*

3.5.- ACTIVIDADES REALIZADAS

Las actividades realizadas durante el desarrollo de la tesis, se han desarrollado de forma general en dos etapas, que a su vez se dividen en varias fases.

Primera etapa.

Fase 1.- Carencias detectadas y necesidades planteadas:

Partiendo de los conocimientos adquiridos en el programa académico de la carrera de Ingeniería en Telecomunicaciones, dan inicio, por un lado, al dilucidar y detectar ciertas carencias que se ha creído importante abordar y, por otro lado, al desarrollo de una nueva tecnología que mejora, en este caso, los sistemas de comunicaciones vía web.

- Carencias detectadas:

1º *Se dispone de tecnologías con un completo servicio de comunicaciones (telefonía, fax, video conferencias, mensajería instantánea, integración con ERP/CRM, etc.), sin embargo del lado del cliente que consume dichos servicios, se afirma que son aplicaciones duras corriendo en un sistema operativo o dispositivos hardware.*

2º *Tecnologías como la Voz sobre IP, Video conferencias y/o Mensajería, aun no han logrado convertirse en un estándar tan abierto, son aplicaciones que requieren de instalación y que son dependientes del sistema operativo usado.*

- Necesidades planteadas:

Se plantea la necesidad del desarrollo de una nueva tecnología de comunicación que sea eficiente, simple y libre.

Fase 2.- Investigación inicial:

Lectura y análisis de las RFCs y de los protocolos involucrados para el desarrollo de la nueva tecnologia de comunicación.

Estudio de las tecnologías y técnicas ligadas a comunicaciones Real-Time. Además, se ha prestado especial atención a los trabajos relacionados con esta alternativa (tesis doctorales y proyectos de fin de carrera).

Fase 3: Objetivos generales: *con las fases anteriores cumplidas, se plantean los objetivos generales:*

- *Aplicar tecnologías RTC.*
- *Combinar protocolos para establecer comunicacines en tiempo real.*
- *Establecer una comunicación Real-Time desde un navegador web.*

Segunda etapa (desarrollo del informe).

Fase 1: Objetivos generales y especificos: *Se definen los puntos a analizar y las características a determinar con cada objetivo planteado. Se suman a los objetivos generales de la primera etapa, los siguientes objetivos específicos:*

- *Unificar y optimizar las comunicaciones.*
- *Fomentar, impulsar y documentar esta nueva tecnología.*
- *Ampliar las posibilidades de interacción entre usuarios de internet y los sistemas de comunicaciones convencionales.*

Fase 2: Estudio técnico: *Se definen las tecnologías, técnicas, estándares, protocolos, procedimientos y equipos necesarios.*

Fase 3: Diseño de sistema propuesto: *Se presenta la arquitectura del sistema de comunicación que se desea implementar.*

Fase 4: Escenarios de comunicación: *Muestra una comunicación entre dos peers WebRTC.*

Fase 5: Diseño de la tecnología WebRTC: *Se exponen los procesos y sentencias de configuaracion que se deben seguir para el desarrollo de esta nueva tecnología para poder implementarla.*

Fase 6: Pruebas: *Se indican las acciones a ejecutar y se detallan los procedimientos y los pasos de ejecución a seguir en cada una de ellas, que demuentra el correcto funcionamiento de la tecnología.*

Fase 7: Resumen de resultados: *Se usa la información reflejada en la fase de prueba para realizar un análisis de los datos extraídos y realizar la unificación de resultados. Los resultados van encaminados a cubrir los objetivos planteados.*

Fase 8: Dificultades: *A pesar de los resultados satisfactorios que se han obtenido, se plantean las dificultades que surgieron en el desarrollo del TFG.*

Fase 9: Planteamiento de las conclusiones: *En base a los resultados obtenidos se plantean las conclusiones.*

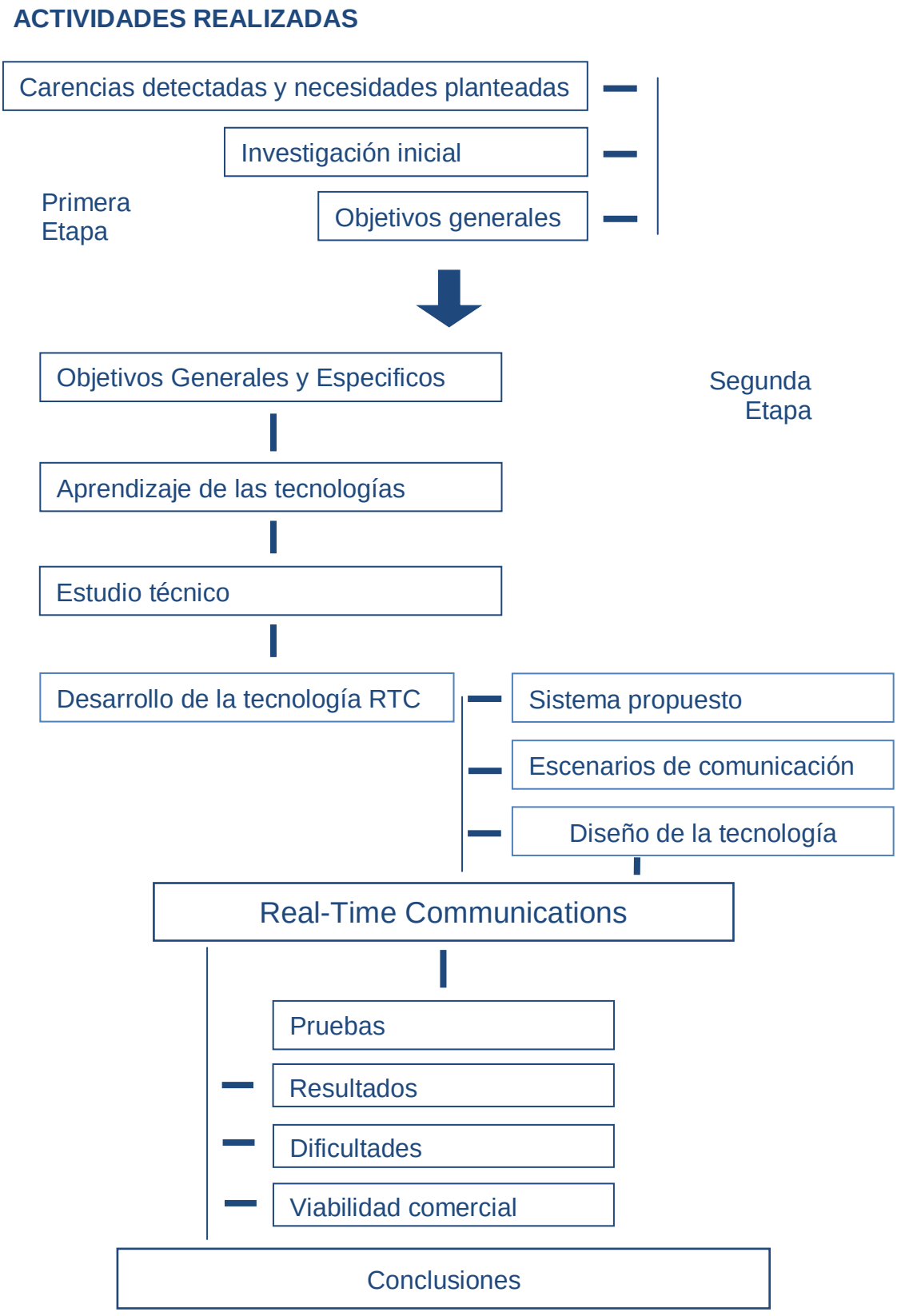


Figura 56.- Actividades realizadas.

CAPITULO 4.- VIABILIDAD COMERCIAL

A continuación, se hace un análisis de viabilidad comercial, en donde se aborda en consideraciones técnicas y económicas la nueva tecnología desarrollada. Y en base a estas consideraciones se realiza una conclusión que determina si es viable el desarrollo de la implementación de la comunicación en tiempo real desde el punto de vista comercial.

.- Consideraciones técnicas:

En cuanto a las consideraciones técnicas se evalúa desde un punto de vista técnico los siguientes aspectos:

- Las técnicas y las tecnologías empleadas: *Básicamente consta de una API y un conjunto de protocolos que hacen posible que navegadores web "de serie" puedan soportar comunicaciones de voz, video, mensajería instantánea, compartición de pantalla y transferencia de archivos, entre otros servicios de forma nativa.*

- Los equipos implementados: *Se pueden desarrollar sistemas de comunicaciones utilizando tan solo los navegadores web como "endpoints" de la comunicación. Es decir que solo se necesitan equipos con un browser web instalado (PC, notebook, tablet o smartphone).*

- Los procesos desarrollados: *Todas las herramientas utilizadas son de naturaleza "Open Source" logrando que los costos de inversión sean nulos en cuanto a licencias de software.*

.- Consideraciones económicas:

En cuanto a las consideraciones económicas se evaluarán desde un punto de vista económico:

- La inversión requerida:

1º *Costo por horas de los desarrolladores, para la dedicación especifica en el desarrollo de la nueva alternativa aquí presentada.*

2º *Costo de equipos para poder establecer las comunicaciones via web (PC, Notebook, Wireless IP Phone, etc.).*

- La utilización de los equipos: *Cualquier empresa o persona con un navegador Web y un micrófono puede hacer llamadas a cualquier otro dispositivo con un navegador Web y un micrófono. Si ambas partes tienen una cámara de video, la llamada también puede incluir video.*

- Los tiempos de configuración y desarrollo: *Los desarrolladores deben hacer que la nueva tecnología esté disponible para su uso y que sea segura. De este modo, tienen que incorporar: SRTP, protocolos de señalización de Gateway como SIP, y protocolos de seguridad como STUN e ICE.*

- Costos de mantenimientos: *No requiere costos de mantenimiento, ya que esta tecnología, no va a ser perpetrado en un futuro por culpa de las licencias y patentes.*

Conclusión:

- Se puede concluir que las características técnicas de esta nueva tecnología tienen la particularidad de plantear una tecnología que implemente el soporte de múltiples servicios en un mismo cliente y establecer comunicaciones en tiempo real, con un costo nulo en licencias de software y equipamiento adicionales.

- Las consideraciones económicas arrojan que los costos son generales y apuntan hacia los equipos, no hacia la tecnología WebRTC implementada.

¿Es Viable el Desarrollo e Implementación de esta Tecnología?	SI ✓

CAPITULO 5.- CONCLUSIONES

Esta tesis se ha desarrollado en torno a una nueva tecnología que permite las comunicaciones en tiempo real en el navegador web mediante el uso de SIP sobre Websocket para señalizar y WebRTC para proporcionar soporte de media.

Los aportes a destacar en este trabajo final son las siguientes:

- *El primer aporte y más fundamental esta relacionado con el hecho de haber implementado y utilizado una tecnología aún en etapa de desarrollo y estandarización. Es decir que se ha logrado trasladar desde plano terórico al práctico todos los conceptos implicados alrededor de la tecnología WebRTC, arrojando resultados óptimos por el hecho de percibir que todo lo investigado y aprendido se mantuvo alineado a la hora del despligue de las configuraciones en el entorno de laboratorio. En definitiva se pudo cumplir con los objetivos planteados, llegando a resultados positivos.*

- *A través de los diferentes escenarios propuestos en laboratorio, con diferentes plataformas que utilizan WebRTC y SIP Websockets se han podido establecer, mantener y finalizar comunicaciones (llamadas telefónicas, video conferencias y mensajería instantánea) en los navegadores web.*

- *Se ha comprobado la viabilidad técnica de las alternativas planteadas y se han desarrollado metodologías de trabajo para llevarlas a cabo. Para ello se han descripto detalladamente los equipos necesarios y los procedimientos a llevar a cabo para lograr los resultados comentados.*

- *Se ha aportado tanta documentación teórica conceptual así como también pautas de trabajo que le facilitan a los desarrolladores e integradores de tecnología, comprender el alcance y los fundamentos de la tecnología WebRTC tanto en el plano teórico como también en el práctico.*

- *Se ha demostrado mediante las pruebas que WebSocket es un protocolo sólido y maduro para transportar mensajes "SIP".*

- *También se ha demostrado el potencial a la hora de optimizar las comunicaciones empresariales. Mediante la eliminación de la dependencia de los proveedores de telefonía, aportando los conceptos de movidilad y nomadismo del "número telefónico" así como también demostrando como se puede implementar la comunicación directamente un cliente visitando la web de la empresa y un miembro de atención al cliente, ambos a través de un navegador web.*

- *Se ha demostrado que WebRTC puede enriquecer la experiencia del cliente y aumentar la productividad de los empleados al tiempo que elimina el costo y complejidad de tener que mantener puntos finales sobre cada puesto de trabajo. El teléfono o aplicación softphone, ahora es una página web, es decir un único sobre el cual se manejan conceptos como funcionalidades, actualizaciones, etc.*

Recomendaciones:

A partir del desarrollo de este trabajo podemos decir que se hace notorio el hecho de que la tecnología aún se encuentre en estado de desarrollo del estándar por parte de la W3C. Esto implica que aún los fabricantes de los diferentes navegadores web, así como también los fabricantes de los servidores de comunicaciones o framework de comunicaciones no terminan de ajustarse al estándar, de manera tal que no se logra una compatibilidad del 100% a la hora de intentar la comunicación entre los diferentes componentes de una arquitectura WebRTC. Por lo tanto, si se desea desarrollar algún sistema de comunicaciones basado en WebRTC es importante tener en cuenta que es muy probable que no pueda ser utilizado desde cualquier navegador web e inclusive en la medida que salen nuevas actualizaciones de cada navegador web, puede que el sistema vaya quedando inútil si no se van ajustando algunas cuestiones relacionadas a las APIs WebRTC.

Percepciones:

En referencia al preámbulo de esta tesis, donde se citó la frase *"El software se está comiendo el mundo"*, se puede afirmar que frente al desarrollo y avance de la tecnología WebRTC se esperan cambios de paradigmas en las tres aristas que conforman el triángulo; Usuarios - Empresas - Proveedores de Telefonía/Internet (Telcos), que se explica a continuación:

- Usuarios: *La tecnología WebRTC llegó al mercado para satisfacer la demanda de los nuevos usuarios, que han adoptado a internet como la forma de comunicación principal y tal como se comentó durante el trabajo, para la mayoría de las personas Internet se centra en el uso Navegador web, por lo tanto WebRTC como tecnología otorga a los innovadores tecnológicos las bases sobre las cuales pueden innovar el mercado de las comunicaciones planteando el navegador web, como base de las comunicaciones de teléfono, video, conferencia, mensajería, etc. de manera tal que también las mismas converjan hacia el verdadero concepto de Comunicaciones Unificadas.*

- Empresas: *WebRTC permite a las empresas brindar diversos medios de comunicación para un mismo punto de origen; el Navegador Web. Por lo tanto desde una mera página web una empresa podrá atender las demandas de comunicaciones de sus clientes usando el medio predilecto de los nuevos clientes; Internet.*

- Proveedores de Telefonía: *WebRTC plantea un impulso importante para masificar la comunicación en tiempo real, es decir llamadas de voz y video, ya que la comunicación mediante mensajería ya es el presente. A este ritmo se plantea una extinción de las compañías Telco tal cual las conocemos hoy en día, es decir la compañía telco que factura las comunicaciones por distancia y tiempo en favor de devenir en compañías de Internet móvil y domiciliario que vendan planes de datos.*

CAPITULO 6.- BIBLIOGRAFÍA

Este capítulo contiene bibliografía que se ha consultado:

[1] A. B. Johnston y D. C. Burnett, WebRTC: APIs and RTCWEB Protocols of HTML5 Real-Time Web, CreateSpace, 2012.

[2] V. Wang, F. Salim y P. Moskovits, The Definitive Guide to HTML5 WebSocket, Apress, 2013.

[3] A. B. Jhonston, SIP: Understanding the Session Initiation Protocol, 2009.

[4] WebRTC Cookbook - Andriil Sergiienko - ISBN 978-1-78328-445-0

[5] WebRTC Integrator's Guide - Altanai - ISBN 978-1-78398-126-7

Ademas, se han accedido a las siguientes páginas web, para la realización de la tesis:

[1] GOOGLE, “WebRTC”, 27 de Julio de 2012:
http://www.webrtc.org/

[2] S. Dutton, “HTML5 ROCKS”, 21 de Febrero de 2014:
http://www.html5rocks.com/en/tutorials/webrtc/basics/

[3] KAAZING, “websocket”:
http://www.websocket.org/

[4] J. Rosenberg, R. Mahy, P. Matthews, and D. Wing. Session Traversal Utilities for NAT (STUN):
http://tools.ietf.org/html/rfc5389, 2008.

[5] J. Rosenberg. Interactive Connectivity Establishment (ICE): *http://tools.ietf.org/html/rfc5245*, 2010.

[6] J. Rosenberg, R. Mahy, and P. Matthews. Traversal Using Relays around NAT(TURN): *http://tools.ietf.org/html/rfc5766*, 2010.

[7] JM. Valin, K. Vos, and T. Terriberry. Definition of the Opus Audio Códec: *http://tools.ietf.org/html/rfc6716*, 2012.

[8] R. Stewart. Stream Control Transmission Protocol: *http://tools.ietf.org/html/rfc4960*, 2007.

[9] R. Jesup, S. Loreto, and M. Tuexen. RTCWeb Datagram Connection: *http://tools.ietf.org/html/draft-ietf-rtcweb-data-channel*, 2012.

[10] *http://www.ramonmillan.com/tutoriales/webrtc.php#sthash.NdYpN0RZ.dpuf*

[11] JSSIP - *http://jssip.net/SipML5 - http://sipml5.org/*

CAPITULO 7.- ANEXOS

7.1.- Lista de figuras

Fig. 1.- Áreas de conocimiento 7
Fig. 2.- Arquitectura de WebRTC 20
Fig. 3.- Funcionamiento de WebRTC 25
Fig. 4.- Comunicación, sin NAT 28
Fig. 5.- Comunicación, con NAT 29
Fig. 6.- Acceso a una página web 32
Fig. 7.- Arquitectura SIP 41
Fig. 8.- Endpoint 201 como cliente SIP y servidor SIP 42
Fig. 9.- Entidades de una red SIP 43
Fig. 10.- Servidor Proxy 45
Fig. 11.- Servidor de re direccionamiento 46
Fig. 12.- Servidor de registro 47
Fig. 13.- Paquete SIP 49
Fig. 14.- Header SIP 53
Fig. 15.- Payload SIP 54
Fig. 16.- Payload SIP 56
Fig. 17.- El protocolo RTP 57
Fig. 18.- HTTP Get Upgrade 60
Fig. 19.- HTTP 101 61
Fig. 20.- Mensaje SIP como Frame UTF-8 sobre Websocket 62
Fig. 21.- Sistema propuesto 70
Fig. 22.- Registro de usuarios con google chrome browser 71
Fig. 23.- Comunicación entre dos peers WebRTC 72
Fig. 24.- Comunicación entre peer WebRTC e IP PBX SIP UDP 73
Fig. 25.- Comunicaciones entre peer WebRTC y abonados PSTN 74
Fig. 26.- Escenario de comunicación 77
Fig. 27.- Navegador web 83
Fig. 28.- Verificación Instalación SIP Router Kamailio 89

Fig. 29.- Implementando WebRTC en Asterisk 96
Fig. 30.- Implementando WebRTC en Kamailio 101
Fig. 31.- Pantalla con la URL de acceso 105
Fig. 32.- Pantalla web de aplicación WebRTC 106
Fig. 33.- Pantalla Expert settings 107
Fig. 34.- Pantalla Expert settings modificada 108
Fig. 35.- Pantalla principal modificada 109
Fig. 36.- Endepoint WebRTC Conectado 110
Fig. 37.- Aplicación abierta en un smartphone Motorola Moto X 110
Fig. 38.- Establecimiento de conexión websocket entre aplicación y servidor WebRTC (Solicitud) 112
Fig. 39.- Establecimiento de conexión websocket entre aplicación y servidor WebRTC (Respuesta) 113
Fig. 40.- Websocket establecido entre Cliente y Servidor WebRTC 114
Fig. 41.- Técnica de “Keep Alive” para mantener la conexión TCP Websocket establecida 115
Fig. 42.- Aplicación Softphone con soporte de protocolo SIP (UDP - TCP) 116
Fig. 43.- Configuración de usuario SIP-UDP para registrar en el servidor SIP 117
Fig. 44.- Usuario SIP-UDP registrado exitosamente en el Servidor SIP 118
Fig. 45.- Test de llamada entre dos usuarios SIP-WebRTC P-I 119
Fig. 46.- Test de llamada entre dos usuarios SIP-WebRTC P-II 120
Fig. 47.- Trafico de llamada desde un navegador “Mozilla” utilizando la herramienta de debug para Java Script 122
Fig. 48.- Parámetros de configuración para registrar un cliente WebRTC en el servidor WebRTC Kamailio 124
Fig. 49.- Cliente WebRTC correctamente registrado en el servidor WebRTC Kamailio 125
Fig. 50.- Parámetros de configuración para registrar un cliente WebRTC en el servidor WebRTC Kamailio 126
Fig. 51.- Cliente WebRTC correctamente registrado en el servidor WebRTC Kamailio 126

Fig. 52.- Test de mensajería instantánea entre dos usuarios WebRTC registrados en el servidor WebRTC Kamailio 127
Fig. 53.- Video llamada realizada entre dos clientes WebRTC registrados en el servidor WebRTC Kamailio 128
Fig. 54.- Mensaje desde el browser al Proxy SIP Kamailio 129
Fig. 55.- Análisis de un paquete Websocket que transporta un mensaje instantáneo (respuesta), a partir de lanzar la captura de paquetes en la consola de comandos del host Proxy SIP Kamailio 130
Fig. 56.- Actividades realizadas 137
Fig. 57.- Resumen de la tesis 160
Fig. 58.- Resumen del Capítulo 1 161
Fig. 59.- Arquitectura global de WebRTC 162
Fig. 60.- Evolución de la comunicación vía Web 163
Fig. 61.- Modelo del navegador 164
Fig. 62.- Resumen capitulo 2 166
Fig. 63.- Análisis de un paquete Websocket que transporta un mensaje instantáneo, a partir de lanzar la captura de paquetes en la consola de comandos del host Proxy SIP Kamailio 192
Fig. 64.- Pantalla de inicio post-boot de la maquina desde la imagen de instalación de GNU/Linux Debian 193
Fig. 65.- Pantalla de selección: Idioma y País 194
Fig. 66.- Pantalla de configuración del teclado 195
Fig. 67.- Pantalla de configuración de nombre de host 195
Fig. 68.- Ingreso de clave del usuario root de Linux 196
Fig. 69.- Ingreso de nombre de usuario normal de Linux a ser creado 196
Fig. 70.- Ingreso de clave para el usuario normal 197
Fig. 71.- Pasos a realizar para particionar el disco de la maquina 198
Fig. 72.- Instalación de paquetes básicos 198
Fig. 73.- Selección de repositorios de software 199
Fig. 74.- Ingreso de parámetros de Proxy Web (en caso de existir uno en la red) 200

Fig. 75.- Permitir o Denegar la encuesta sobre uso de paquetes 200
Fig. 76.- Pantalla de selección de programas a instalar 201
Fig. 77.- Pantalla de selección para instalar el cargador de arranque 201
Fig. 78.- Pantalla de aceptación de fin de instalación y reinicio de la maquina 202
Fig. 79.- Pantalla de selección de sistema operativo y/o kernel a cargar 202
Fig. 80.- Proceso de carga finalizado, sistema listo para utilizar 203
Fig. 81.- Sesión iniciada como usuario root 203

7.3.- Abreviaturas y acrónimos

Letra	Acrónimo	Significado
a -	**AEC**	*Acoustic Echo Canceler*
	API	*Application Programming Interface*
b -	**BGP**	*Border Gateway Protocol*
c -	**CO**	*Central Office'*
	Códec	*Coder-Decoder*
	CSPDN	*Circuit Switched Public Data Network*
	CSS	*Cascading Style Sheets*
d -	**DTLS**	*Datagram Transport Layer Security*
h -	**HTML**	*Hyper Text Markup Language*
	HTML5	*fifth revision of the HTML standarrd*
	HTTP	*Hypertext Transfer Protocol*
i -	**ICE**	*Information Connection Establishment*
	ICT	*Information and Communications Technology*
	IETF	*Internet Engineering Task Force*
	iLBC	*Internet Low Bitrate Códec*
	iSAC	*Internet Speech Audio Códec*
	ISDN	*Integrated Services Digital Network*
	IP	*Internet Protocol*
	IPv4	*Internet Protocol Version 4*
	IPv6	*Internet Protocol Version 6*
	IT	*Information Technology*
j -	**JS**	*JavaScript*
l -	**LAN**	*Local Area Network*
m -	**MCU**	*Multipoint Control Unit*
	MGCP	*Media Gateway Control Protocol*
n -	**NAT**	*Network Address Translation*
	NR	*Noise Reduction*

	PLMN	*Public Land Mobile Network*
p -	**PSTN**	*Public switched Telephone Network*
q -	**QoS**	*Quality of Service*
r -	**RFC**	*Request For Comment*
	RTC	*Real-Time Communication*
	RTCP	*Real-time Transport Control Protocol*
	RTP	*Real-time Transport Protocol*
s -	**SDP**	*Session Description Protocol*
	SIP	*Session Initiation Protocol*
	SRTP	*Secure Real-time Transport Protocol*
	SS7	*Signalling System No. 7*
	SSL	*Secure Socket Layer*
	STUN	*Sesión Traversal Utilities for NAT*
t -	**TCP**	*Transmission Control Protocol*
	TLS	*Transport Layer Security*
	TURN	*Traversal Using Relays around NAT*
u -	**UDP**	*User Datagram Protocol*
	URI	*Uniform Resource Identifier*
	URL	*Universal Resource Locator*
v -	**VLAN**	*Virtual LAN*
	VoIP	*Voice over Internet Protocol*
	VPN	*Virtual private network*
w -	**W3C**	*World Wide Web Consortium*
	WAN	*Wide Area Network*
	WebRTC	*Web Real Time Communication*
	WWW	*World Wide Web*
x -	**XHML**	*eXtensible HyperText Markup Language*
	XML	*eXtensible Markup Language*

7.4.- Definiciones

Adobe RTMFP: *El protocolo de flujo multimedia en tiempo real (RTMFP) es un protocolo de comunicación de Adobe que permite la comunicación entre pares de navegadores web entre varias instancias del cliente Adobe® Flash® Player y aplicaciones creadas con el framework Adobe AIR® para la distribución de comunicaciones sofisticadas en directo y en tiempo real.*

Arquitectura Hibrida: *Arquitectura de comunicaciones donde convergen tecnologías IP y de conmutación de circuitos.*

Cloud Computing: *Se trata de un nuevo paradigma dentro de la informática basado acceder a servicios informáticos a través de internet. Servicios como el almacenamiento de datos, programas de ofimática, sistemas de gestión, central telefónica y tantos más, son bajo este paradigma ofrecidos desde servidores en Internet.*

Cloud Híbrida: *Una "Cloud híbrida" es un tipo de modelo de implementación del "Cloud Computing" donde algunos recursos de TI se proporcionan a través de servidores locales (en la red interna de la compañía) y otros recursos son provistos por proveedores de servicios de terceros en Internet (Cloud Computing).*

Códec: *Códec es la abreviatura de codificador-decodificador. Describe una especificación desarrollada en software, hardware o una combinación de ambos, capaz de transformar un archivo con un flujo de datos (stream) o una señal.*

Codecs de A/V

Propietarios: *Codecs de audio y/o video propietarios, son codecs que implican adquirir una licencia de uso para poder utilizarlos. Además, los mismos suelen estar protegidos por una patente.*

Comunicaciones

Unificadas: *El término Comunicaciones unificadas es utilizado comúnmente por los proveedores de tecnologías de la información para designar la integración de "los servicios de telefonía, mensajería unificada (la misma bandeja de entrada para correo electrónico, correo de voz y fax), mensajería instantánea corporativa, video conferencias y estado de disponibilidad del usuario (presencia) en una sola e innovadora experiencia para los colaboradores y para el personal que administra y da mantenimiento a la infraestructura".*

Endpoints: *Dispositivo físico (teléfono IP) o aplicación software capaz de iniciar, modificar, mantener y finalizar una comunicación telefónica usando la re IP.*

Forma Nativa: *Hace referencia que es posible desarrollar aplicaciones web en las cuales los navegadores que accedan a éstas, pueden establecer comunicaciones, sin la necesidad de instalar ningún paquete de software (plugin) externo en el sistema donde corre el navegador web como aplicación.*

Open Source: *Código abierto es la expresión con la que se conoce al software distribuido y desarrollado bajo licencias que garantizan el acceso al código fuente que implementa dicho software.*

Framework: *En el desarrollo de software, un framework es una estructura conceptual y tecnológica de soporte definido, normalmente con artefactos o módulos de software concretos, que puede servir de base para la organización y desarrollo de software. Típicamente, puede incluir soporte de programas, librerías, y un lenguaje interpretado, entre otras herramientas, para así ayudar a desarrollar y unir los diferentes componentes de un proyecto.*

Google Hangouts: *Servicio de mensajería instantánea y video llamadas basado en WebRTC brindado por la compañía Google Inc. Para usuarios que posean una cuenta de Gmail o Google Plus.*

Help Desk: *La mesa de ayuda de una compañía es el primer nivel de soporte técnico informático. Suele ser contactado por teléfono, mensaje instantáneo o a través de un sistema de tickets.*

Home Work: *El teletrabajo o trabajo desde la casa es una tendencia actual desarrollada a partir del incremento de la calidad y cantidad de conexiones a Internet. Esto permite que el personal de una compañía pueda realizar las tareas laborales desde su domicilio.*

Virtual Offices: *El servicio de oficina virtual es una aplicación del teletrabajo o home work, ya que se basa en empresas que brindan el servicio de telefonía (numeración local) y secretaria para tomar recados a empresas que no pueden mantener dicha estructura.*

Integración Con ERP/CRM: *Integrar el servicio de telefonía con los sistemas de información de la empresa. De esta manera ambos sistemas pueden interactuar de manera tal que al ingresar una llamada al teléfono de un miembro de la empresa, se despliegue un pop-up con los datos de quien está llamando o también es muy común poder discar un contacto a partir de un "click" sobre la ficha de una persona en el sistema de gestión.*

Nomadismo: *Desde el punto de vista de este trabajo, el nomadismo a plica al hecho de poder disponer del número de interno de la centralita telefónica a partir de un login desde una aplicación.*

NAT Transversal: *NAT trasversal es un término aplicado a las técnicas que establecen y mantienen conexiones en redes utilizando los protocolos TCP/IP o UDP que atraviesan (NAT) gateways. Dichas técnicas de NAT trasversal suelen ser requeridas por aplicaciones cliente-cliente, especialmente las peer-to-peer y Voip.*

P2P (Peer To Peer): *Una red peer-to-peer, red de pares, red entre iguales o red entre pares (P2P, por sus siglas en inglés) es una red de computadoras en la que todos o algunos aspectos funcionan sin clientes ni servidores fijos, sino una serie de nodos que se comportan como iguales entre sí.*

Plugins Flash De Adobe: *Adobe Flash Player es un plugin que permite a los navegadores web mostrar "contenido Flash" en páginas web. Flash se usa a menudo para las animaciones, videos, juegos y aplicaciones multimedia.*

Protocolos: *En informática y telecomunicación, un protocolo de comunicaciones es un sistema de reglas que permiten que dos o más entidades de un sistema de comunicación se comuniquen entre ellas para transmitir información por medio de cualquier tipo de variación de una magnitud física. Se trata de las reglas o el estándar que define la sintaxis, semántica y sincronización de la comunicación, así como también los posibles métodos de recuperación de errores.*

Servicios De Forma Nativa: *De forma nativa, me refiero a alguna cosa que funciona sin tener que modificar nada. Por ejemplo hacer uso de una aplicación web y que no te pida que instales ningún "plugin" o complemento.*

Stack De Protocolos: *Pila de protocolos.*

7.5.- Resumen general de la tesis

Se resume de manera grafica lo que se ha desarrollado en la Tesis.

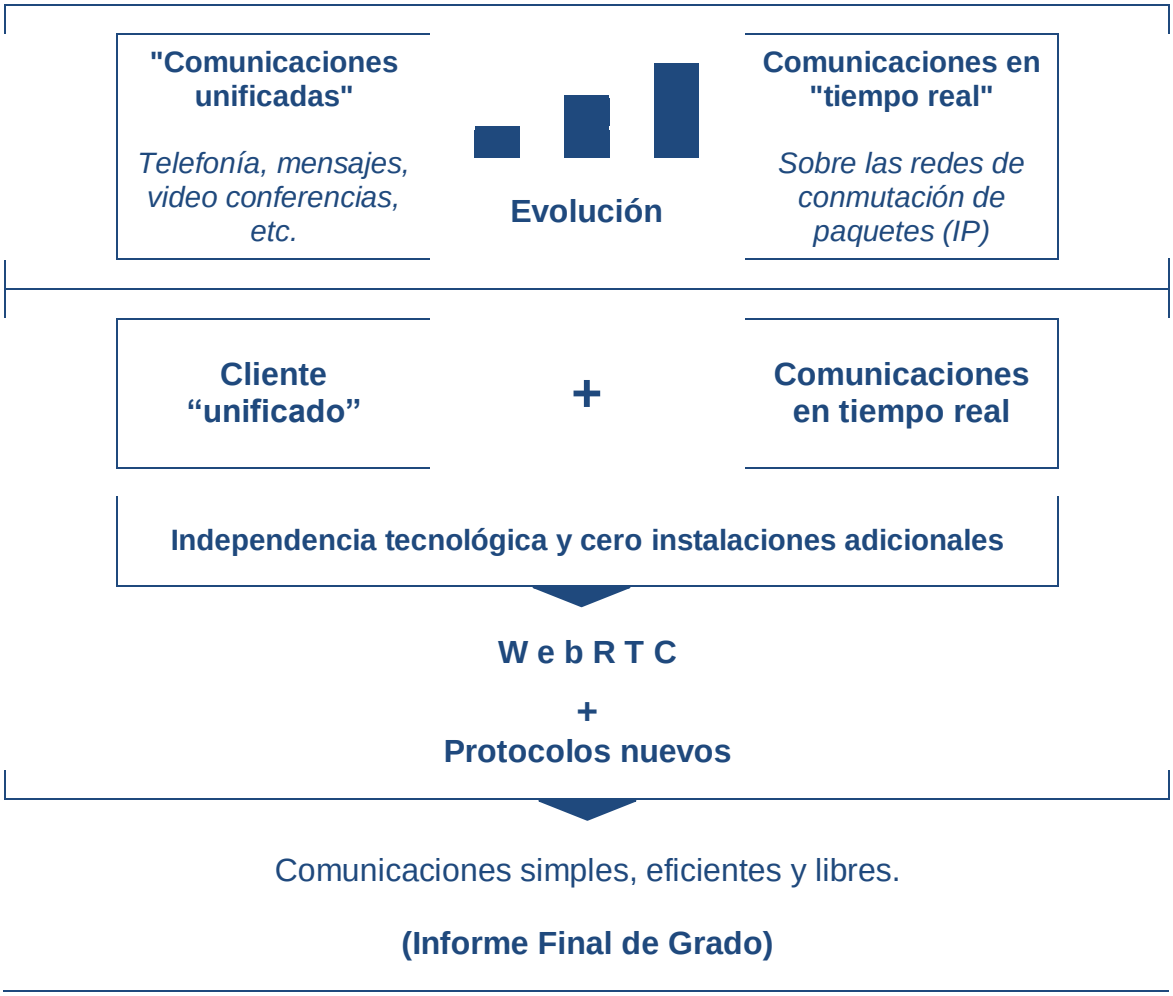


Figura 57.- Resumen de la tesis.

7.6.- Anexos del Capítulo 1

7.6.1.- Resumen del Capitulo

Se resume con un grafico los objetivos generales y específicos, los beneficios y destinatarios.

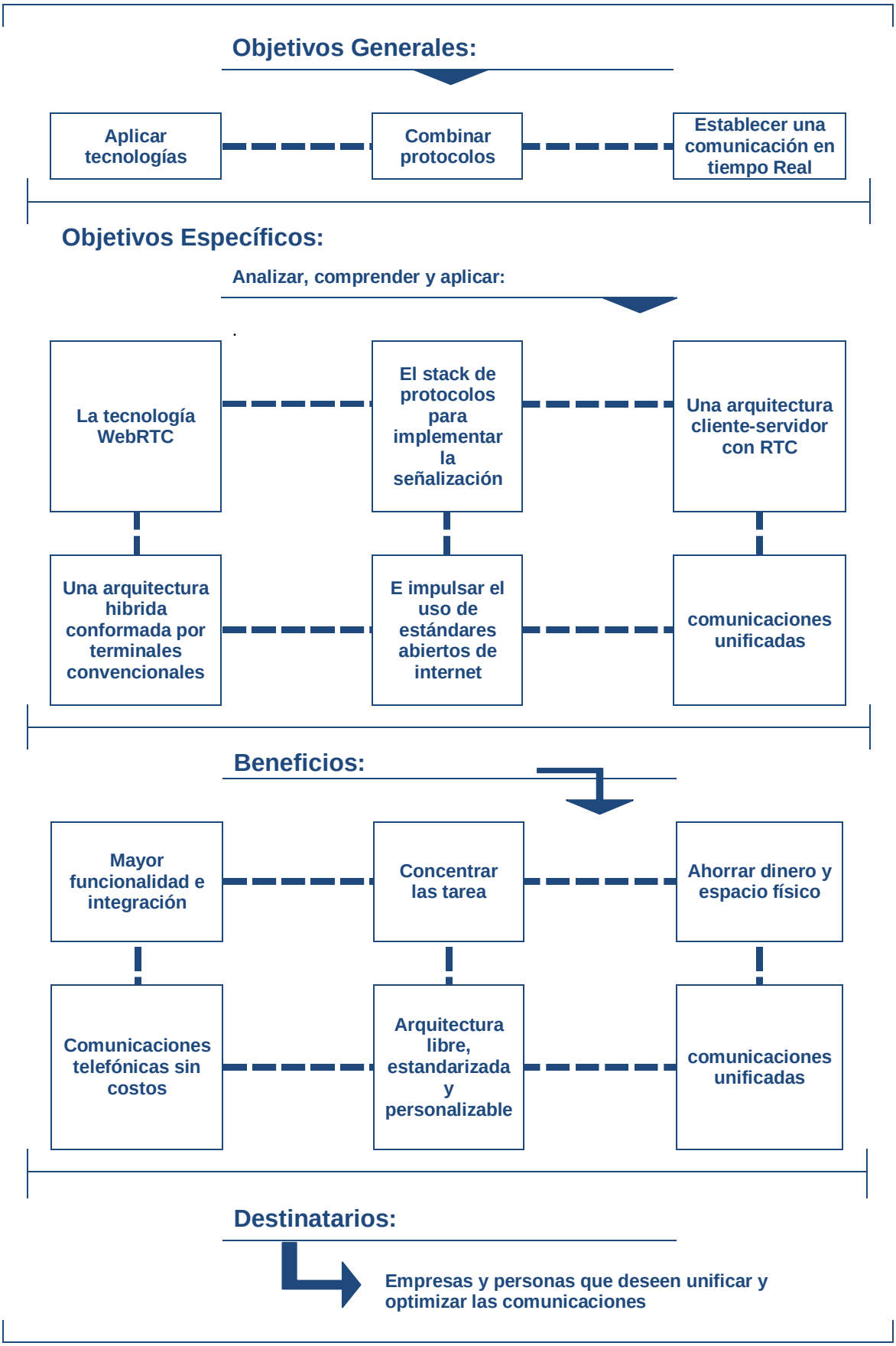


Figura 58.- Resumen del Capítulo 1.

7.7.- Anexos del Capítulo 2

7.7.1.- Arquitectura global de WebRTC

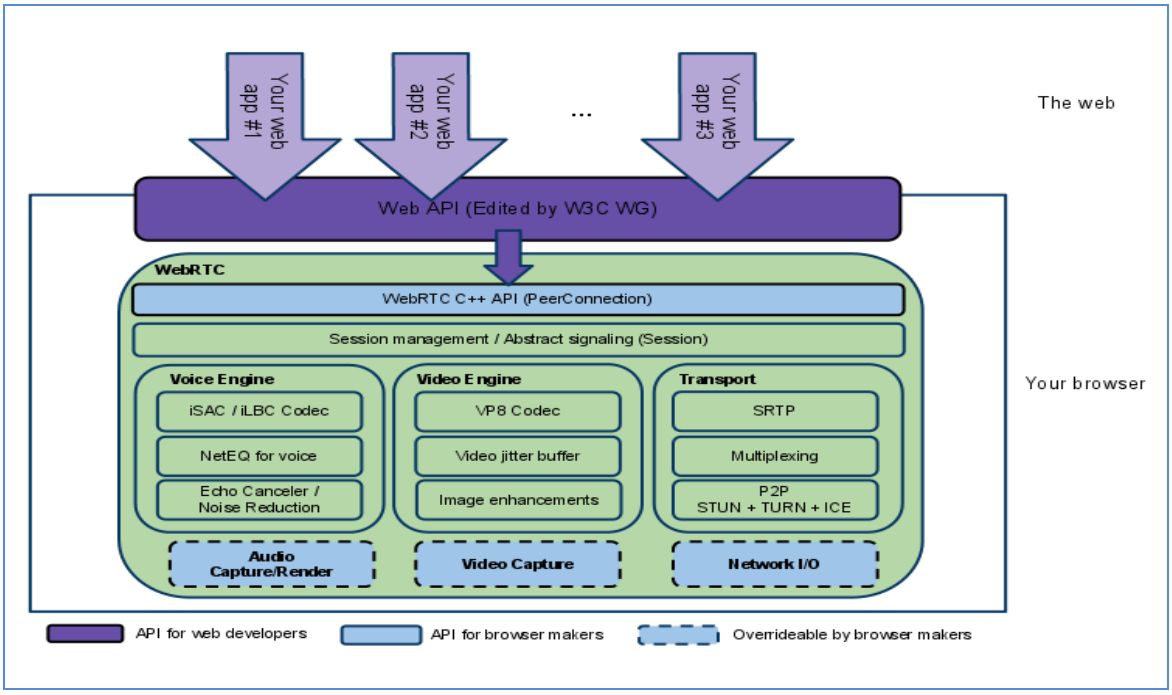


Figura 59.- Arquitectura global de WebRTC.

http://www.webrtc.org/architecture

7.7.2.- Evolución de la comunicación vía Web

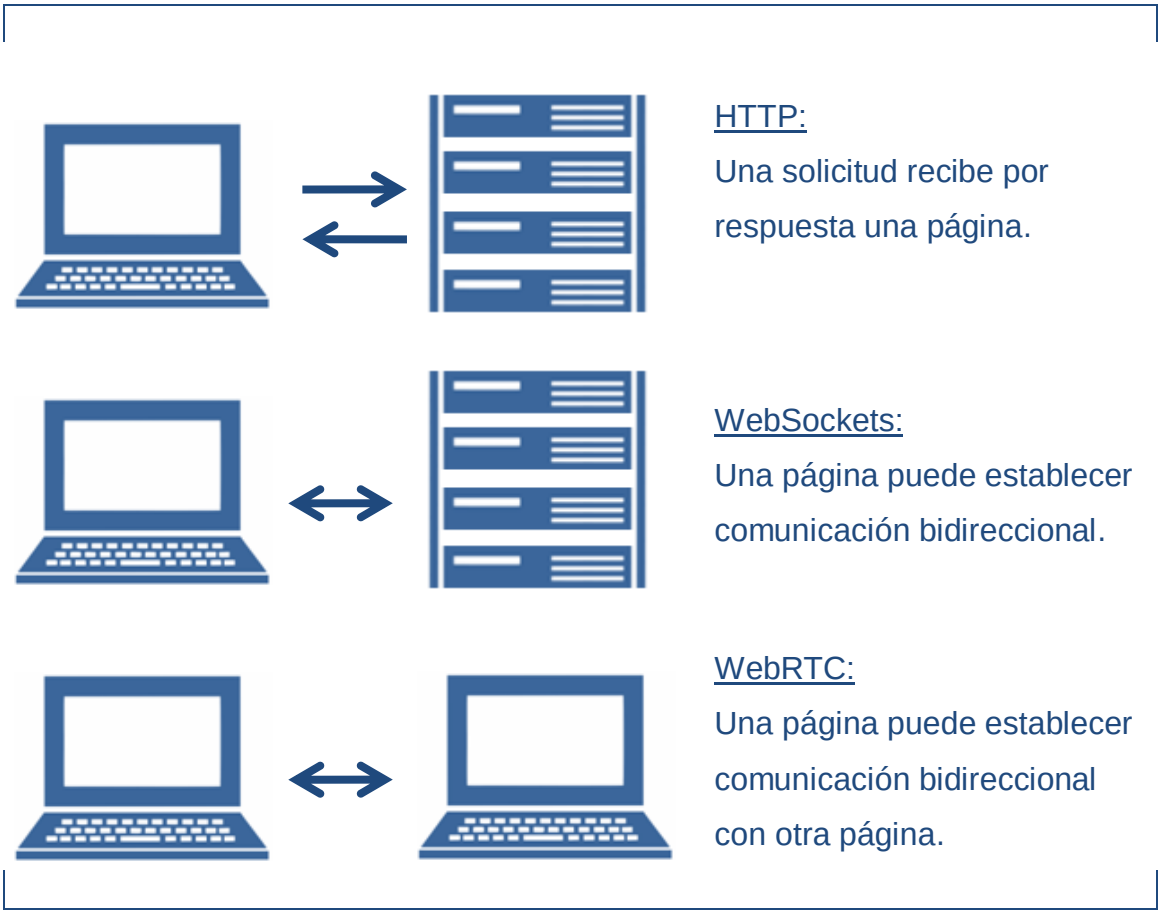


Figura 60.- Evolución de la comunicación vía Web.

7.7.3.- Modelo del navegador

WebRTC interactúa con las aplicaciones en servidores Web utilizando API estándares, y se comunica con el sistema operativo utilizando el navegador.

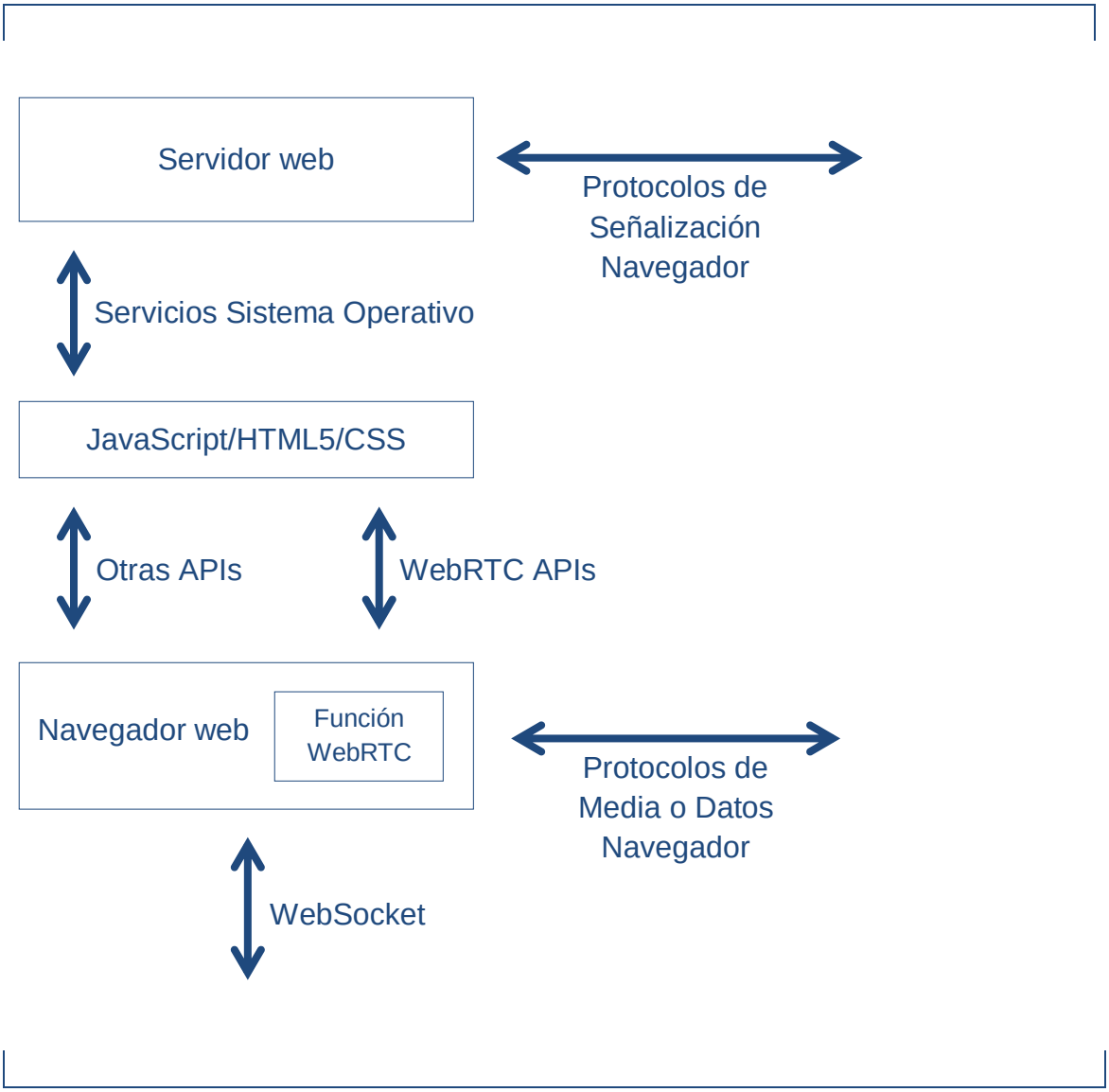


Figura 61.- Modelo del navegador.

7.7.4.- Javascripts

Ejemplo del código JavaScript integrado en las páginas web.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
Transitional//EN""http://www.w3.org/TR/xhtml1/DTD/xhtml1
-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=iso
-8859-1" />
<title>Ejemplo de codigo JavaScript</title>
<script type="text/javascript">
  alert("Trabajo de Grado – Pignataro Ciocatto");
</script>
</head>
<body>
<p>Ingenieria en Telecomunicaciones.</p>
</body>
</html>
```

7.7.5.- Resumen Capítulo 2

El siguiente resumen describe las tecnologías, protocolos, hardware y software utilizados para el desarrollo del informe final de grado.

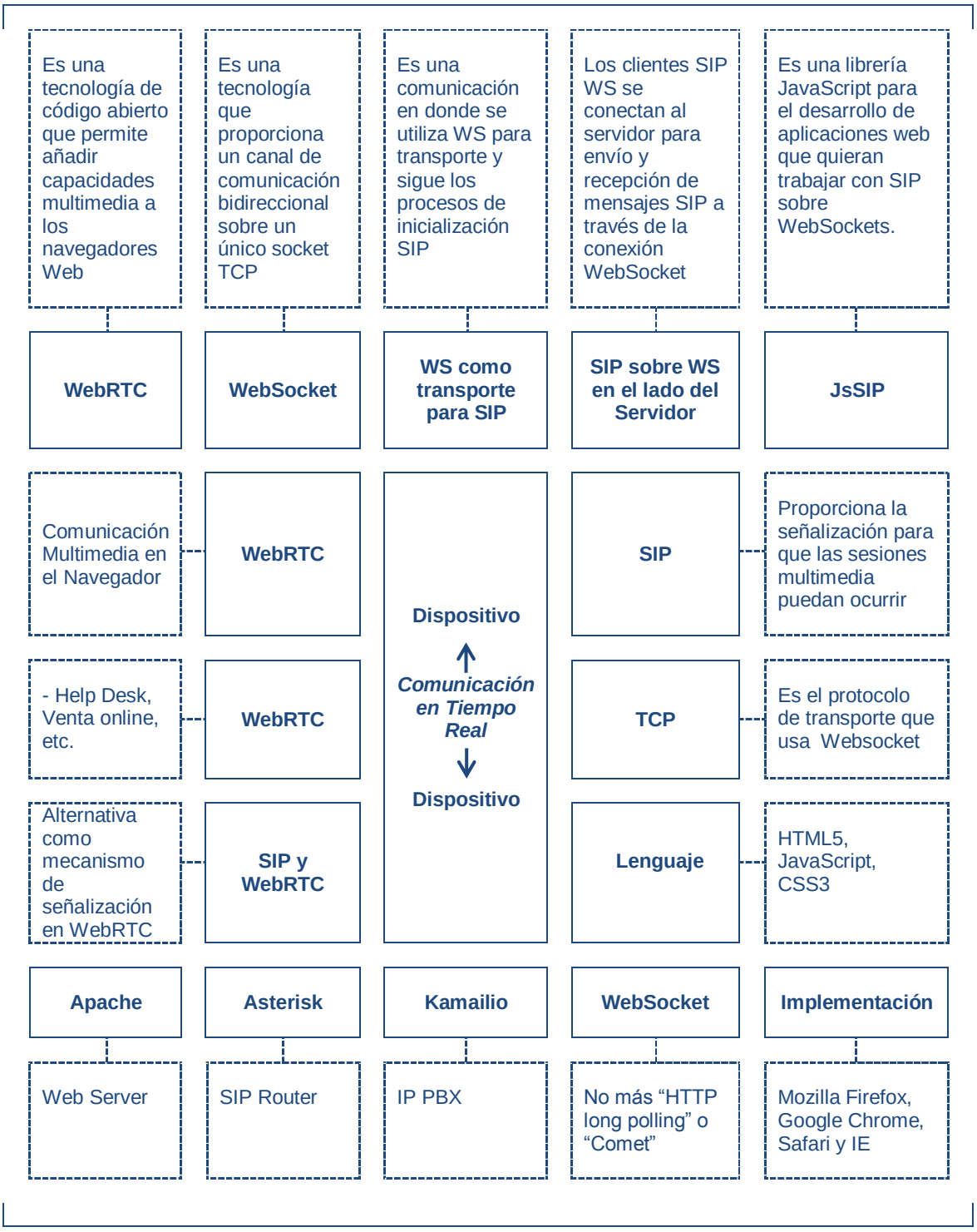


Figura 62.- Resumen capitulo 2.

7.8.- Anexos del Capítulo 3

7.8.1.- El archivo kamailio.cfg

El siguiente archivo de configuración se utiliza para la **instalación SIP Router Kamailio**:

```
#!KAMAILIO
#
# Simple/sample kamailio.cfg for running a proxy/registrar
with TLS and
# WebSockets support.

#!substdef
"!DBURL!mysql://root:Freetech123@localhost/kamailio!g"
#!substdef "!MY_IP_ADDR!192.168.0.16!g"
#!substdef "!MY_WS_PORT!80!g"
#!substdef "!MY_WSS_PORT!443!g"
#!substdef "!MY_WS_ADDR!tcp:MY_IP_ADDR:MY_WS_PORT!g"
#!substdef "!MY_WSS_ADDR!tls:MY_IP_ADDR:MY_WSS_PORT!g"

##!define LOCAL_TEST_RUN
##!define WITH_TLS
#!define WITH_WEBSOCKETS
##!define TESTBED_MODE

#!define WITH_PSTN

####### Global Parameters #########

fork=yes
children=4

#alias="example.com"

#!ifdef WITH_TLS
enable_tls=1
#!endif

listen=MY_IP_ADDR
#!ifdef WITH_WEBSOCKETS
listen=MY_WS_ADDR
#!endif
#!ifdef WITH_TLS
listen=MY_WSS_ADDR
#!endif
tcp_connection_lifetime=3604
tcp_accept_no_cl=yes
tcp_rd_buf_size=16384
```

```
#!ifdef TESTBED_MODE
  debug=5
  log_stderror=yes
#!else
  debug=2
  log_stderror=no
#!endif
#debug=2

#!ifdef WITH_PSTN
# PSTN GW Routing
#
# - pstn.gw_ip: valid IP or hostname as string value,
example:
# pstn.gw_ip = "10.0.0.101" desc "My PSTN GW Address"
#
# - by default is empty to avoid misrouting
pstn.gw_ip = "192.168.0.16" desc "PSTN GW Address"
pstn.gw_port = "5060" desc "PSTN GW Port"
#!endif

#!ifdef WITH_VOICEMAIL
# VoiceMail Routing on offline, busy or no answer
#
# - by default Voicemail server IP is empty to avoid
misrouting
voicemail.srv_ip = "192.168.0.16" desc "VoiceMail IP
Address"
voicemail.srv_port = "5060" desc "VoiceMail Port"
#!endif


# set paths to location of modules (to sources or
installation folders)
#!ifdef WITH_SRCPATH
mpath="modules_k:modules"
#!else
mpath="/usr/local/lib/kamailio/modules/"
#!endif

loadmodule "db_mysql.so"
loadmodule "tm.so"
loadmodule "sl.so"
loadmodule "rr.so"
loadmodule "pv.so"
loadmodule "maxfwd.so"
loadmodule "usrloc.so"
loadmodule "registrar.so"
loadmodule "textops.so"
loadmodule "siputils.so"
loadmodule "xlog.so"
loadmodule "sanity.so"
loadmodule "ctl.so"
```

```
loadmodule "auth.so"
loadmodule "auth_db.so"
loadmodule "kex.so"
loadmodule "mi_rpc.so"
#!ifdef WITH_TLS
loadmodule "tls.so"
#!endif
#!ifdef WITH_WEBSOCKETS
loadmodule "xhttp.so"
loadmodule "websocket.so"
loadmodule "nathelper.so"
#!endif

# ----------------- setting module-specific parameters -----
---------

# ----- tm params -----
# auto-discard branches from previous serial forking leg
modparam("tm", "failure_reply_mode", 3)
# default retransmission timeout: 30sec
modparam("tm", "fr_timer", 30000)
# default invite retransmission timeout after 1xx: 120sec
modparam("tm", "fr_inv_timer", 120000)

# ----- rr params -----
# add value to ;lr param to cope with most of the UAs
modparam("rr", "enable_full_lr", 1)
# do not append from tag to the RR (no need for this script)
modparam("rr", "append_fromtag", 0)

# ----- registrar params -----
modparam("registrar", "method_filtering", 1)
modparam("registrar", "max_expires", 3600)
modparam("registrar", "gruu_enabled", 0)

# ----- usrloc params -----
modparam("usrloc", "db_url", "DBURL")
modparam("usrloc", "db_mode", 0)

# ----- auth_db params -----
modparam("auth_db", "db_url", "DBURL")
modparam("auth_db", "calculate_ha1", yes)
modparam("auth_db", "password_column", "password")
modparam("auth_db", "load_credentials", "")

#!ifdef WITH_TLS
# ----- tls params -----
modparam("tls", "config",
"/usr/local/kamailio/etc/kamailio/tls.cfg")
#!endif
```

```
#!ifdef WITH_WEBSOCKETS
# ----- nathelper params -----
modparam("nathelper|registrar", "received_avp",
"$avp(RECEIVED)")
# Note: leaving NAT pings turned off here as nathelper is
_only_ being used for
#       WebSocket connections.  NAT pings are not needed
as WebSockets have
#       their own keep-alives.
#!endif

####### Routing Logic ########

# Main SIP request routing logic
# - processing of any incoming SIP request starts with this
route
# - note: this is the same as route { ... }
request_route {

   # per request initial checks
   route(REQINIT);

#!ifdef WITH_WEBSOCKETS
   if (nat_uac_test(64)) {
         # Do NAT traversal stuff for requests from a
WebSocket
         # connection - even if it is not behind a NAT!
         # This won't be needed in the future if Kamailio
and the
         # WebSocket client support Outbound and Path.
         force_rport();
         if (is_method("REGISTER"))
               fix_nated_register();
         else {
               if (!add_contact_alias()) {
                     xlog("L_ERR", "Error aliasing contact
<$ct>\n");
                     sl_send_reply("400", "Bad Request");
                     exit;
               }
         }
   }




#!endif

   # handle requests within SIP dialogs
   route(WITHINDLG);

   ### only initial requests (no To tag)
```

```
    # CANCEL processing
    if (is_method("CANCEL")) {
          if (t_check_trans())
               t_relay();
          exit;
    }

    t_check_trans();

    # authentication
    route(AUTH);

    # record routing for dialog forming requests (in case
they
are routed)
    # - remove preloaded route headers
    remove_hf("Route");
    if (is_method("INVITE"))
          record_route();

    # handle registrations
    route(REGISTRAR);

    if ($rU==$null) {
          # request with no Username in RURI
          sl_send_reply("484","Address Incomplete");
          exit;
    }
    # dispatch destinations to PSTN
    route(PSTN);

    # user location service
    route(LOCATION);

    route(RELAY);

}

route[RELAY] {
    if (!t_relay()) {
          sl_reply_error();
    }
    exit;
}

# Per SIP request initial checks
route[REQINIT] {
    if (!mf_process_maxfwd_header("10")) {
          sl_send_reply("483","Too Many Hops");
          exit;
    }
```

```
if(!sanity_check("1511", "7")) {
        xlog("Malformed SIP message from $si:$sp\n");
        exit;
    }
}


# Handle requests within SIP dialogs
route[WITHINDLG] {
    if (has_totag()) {
        # sequential request withing a dialog should
        # take the path determined by record-routing
        if (loose_route()) {

#!ifdef WITH_WEBSOCKETS
            if ($du == "") {
                if (!handle_ruri_alias()) {
                    xlog("L_ERR", "Bad alias <$ru>\n");
                    sl_send_reply("400", "Bad
Request");
                    exit;
                }
            }
#!endif
            route(RELAY);
        } else {
            if ( is_method("ACK") ) {
                if ( t_check_trans() ) {
                    # no loose-route, but stateful ACK;
                    # must be an ACK after a 487
                    # or e.g. 404 from upstream server
                    t_relay();
                    exit;
                } else {
                    # ACK without matching
transaction...
                    # ignore and discard
                    exit;
                }
            }
            sl_send_reply("404","Not here");
        }
        exit;
    }
}

# Handle SIP registrations
route[REGISTRAR] {
    if (is_method("REGISTER")) {
        if (!save("location"))
            sl_reply_error();
```

```
        exit;
    }
}

# USER location service
route[LOCATION] {
    if (!lookup("location")) {
        $var(rc) = $rc;
        t_newtran();
        switch ($var(rc)) {
            case -1:
            case -3:
                send_reply("404", "Not Found");
                exit;
            case -2:
                send_reply("405", "Method Not Allowed");
                exit;
        }
    }
}




# Authentication route
route[AUTH] {
    if (is_method("REGISTER") || from_uri==myself) {
        # authenticate requests
        if (!auth_check("$fd", "subscriber", "1")) {
            auth_challenge("$fd", "0");
            exit;


        }
        # user authenticated - remove auth header
        if(!is_method("REGISTER"))
            consume_credentials();
    }
    # if caller is not local subscriber, then check if it
calls
    # a local destination, otherwise deny, not an open relay
here
    if (from_uri!=myself && uri!=myself) {
        sl_send_reply("403","Not relaying");
        exit;
    }
}

#!ifdef WITH_WEBSOCKETS
onreply_route {
```

```
if (nat_uac_test(64)) {
        # Do NAT traversal stuff for replies to a
WebSocket connection
        # - even if it is not behind a NAT!
        # This won't be needed in the future if Kamailio
and the
        # WebSocket client support Outbound and Path.
        add_contact_alias();
    }
}

event_route[xhttp:request] {
    set_reply_close();
    set_reply_no_connect();

    if ($Rp != MY_WS_PORT && $Rp != MY_WSS_PORT) {
        xlog("L_WARN", "HTTP request received on $Rp\n");
        xhttp_reply("403", "Forbidden", "", "");
        exit;
    }

    xlog("L_DBG", "HTTP Request Received\n");

    if ($hdr(Upgrade)=~"websocket"
            && $hdr(Connection)=~"Upgrade"
            && $rm=~"GET") {
        xlog("L_DBG", "WebSocket\n");
        xlog("L_DBG", " Host: $hdr(Host)\n");
        xlog("L_DBG", " Origin: $hdr(Origin)\n");

        if ($hdr(Host) == $null || !is_myself($hdr(Host)))
{
            xlog("L_WARN", "Bad host $hdr(Host)\n");
            xhttp_reply("403", "Forbidden", "", "");
            exit;
        }

        # Optional... validate Origin
        # Optional... perform HTTP authentication

        # ws_handle_handshake() exits (no further
configuration file
        # processing of the request) when complete.
        if (ws_handle_handshake())
        {


# Optional... cache some information abou the
            # successful connection
            exit;
        }
    }
```

```
    xhttp_reply("404", "Not found", "", "");
}

event_route[websocket:closed] {
    xlog("L_INFO", "WebSocket connection from $si:$sp has
closed\n");
}
#!endif

# PSTN GW routing
route[PSTN] {
#!ifdef WITH_PSTN
    # check if PSTN GW IP is defined
    if (strempty($sel(cfg_get.pstn.gw_ip))) {
        xlog("SCRIPT: PSTN rotuing enabled but pstn.gw_ip
not defined\n");
        return;
    }

    # route to PSTN dialed numbers starting with '+' or '00'
    #     (international format)
    # - update the condition to match your dialing rules for
PSTN routing
    if(!($rU=~"^(\+|00)[1-9][0-9]{3,20}$"))
        return;

    # only local users allowed to call
    if(from_uri!=myself) {
        sl_send_reply("403", "Not Allowed");
        exit;
    }

    if (strempty($sel(cfg_get.pstn.gw_port))) {
        $ru = "sip:" + $rU + "@" +
$sel(cfg_get.pstn.gw_ip);
    } else {
        $ru = "sip:" + $rU + "@" + $sel(cfg_get.pstn.gw_ip)
+ ":"
                        + $sel(cfg_get.pstn.gw_port);
    }

    route(RELAY);
    exit;
#!endif

    return;
}
```

A continuación se hace una descripción de cada sección del archivo de configuración kamailio.cfg:

La configuración general del servicio se trabaja en este archivo. El archivo es complejo y su explicación escapa al alcance de este trabajo final de grado. Sin embargo se puede decir a grandes rasgos que el mismo contiene 5 secciones y una sintaxis parecida al lenguaje de programación C, pero aplicado a procesar solicitudes SIP y manejar respuestas SIP.

1.- La primera sección tiene que ver con las constantes a configurar para usar en “tiempo de ejecución”.

```
#!KAMAILIO
#
# Simple/sample kamailio.cfg for running
a proxy/registrar with
TLS and
# WebSockets support.

#!substdef "!DBURL!mysql:
//root:Freetech123@localhost/kamailio!g"
#!substdef "!MY_IP_ADDR!192.168.0.16!g"
#!substdef "!MY_WS_PORT!80!g"
#!substdef "!MY_WSS_PORT!443!g"
#!substdef "!MY_WS_ADDR!tcp:MY_IP_ADDR:MY_WS_PORT!g"
#!substdef "!MY_WSS_ADDR!tls:MY_IP_ADDR:MY_WSS_PORT!g"

##!define LOCAL_TEST_RUN
##!define WITH_TLS
#!define WITH_WEBSOCKETS
##!define TESTBED_MODE

#!define WITH_PSTN
```

Sin entrar en tantos detalles, se puede observar a grandes rasgos que las sentencias definen constantes, como por ejemplo “MY_IPADDR” contiene la dirección IP del host SIP Server.

2.- La segunda sección del archivo de configuración contiene el setting de los parámetros globales, los mismos tienen como finalidad controlar el comportamiento del servicio SIP Server.

```
####### Global Parameters #########

fork=yes
children=4

#alias="example.com"

#!ifdef WITH_TLS
enable_tls=1
#!endif

listen=MY_IP_ADDR
#!ifdef WITH_WEBSOCKETS
listen=MY_WS_ADDR
#!endif
#!ifdef WITH_TLS
listen=MY_WSS_ADDR
#!endif

tcp_connection_lifetime=3604
tcp_accept_no_cl=yes
tcp_rd_buf_size=16384

#!ifdef TESTBED_MODE
  debug=5
  log_stderror=yes
#!else
  debug=2
  log_stderror=no
#!endif

#debug=2

#!ifdef WITH_PSTN
# PSTN GW Routing
#
# - pstn.gw_ip: valid IP or hostname as string
value, example:
# pstn.gw_ip = "10.0.0.101" desc "My PSTN GW Address"
#
# - by default is empty to avoid misrouting
pstn.gw_ip = "192.168.0.16" desc "PSTN GW Address"
pstn.gw_port = "5060" desc "PSTN GW Port"
#!endif
```

```
#!ifdef WITH_VOICEMAIL
# VoiceMail Routing on offline, busy or no answer
#
# - by default Voicemail server IP is empty to
avoid misrouting
voicemail.srv_ip = "192.168.0.16" desc "VoiceMail
IP Address"
voicemail.srv_port = "5060" desc "VoiceMail Port"
#!endif
# set paths to location of modules (to sources
or installation folders)
#!ifdef WITH_SRCPATH
mpath="modules_k:modules"
#!else
mpath="/usr/local/lib/kamailio/modules/"
#!endif
```

Cuestiones como indicarle al proceso sobre qué dirección IP abrir los puertos listen=MY_IP_ADDR o en qué directorios del sistema operativo buscar los módulos mpath="/usr/local/lib/kamailio/modules/" que proporcionan las funcionalidades del sistema. Son parámetros que se configuran en dicha sección.

3.- Luego se presenta la sección en la cual se indica al proceso qué módulos cargar (disponibles en el directorio comentado en el punto anterior).

```
loadmodule "db_mysql.so"
loadmodule "tm.so"
loadmodule "sl.so"
loadmodule "rr.so"
loadmodule "pv.so"
loadmodule "maxfwd.so"
loadmodule "usrloc.so"
loadmodule "registrar.so"
loadmodule "textops.so"
loadmodule "siputils.so"
loadmodule "xlog.so"
loadmodule "sanity.so"
loadmodule "ctl.so"
loadmodule "auth.so"
loadmodule "auth_db.so"
loadmodule "kex.so"
loadmodule "mi_rpc.so"
#!ifdef WITH_TLS
loadmodule "tls.so"
#!endif
#!ifdef WITH_WEBSOCKETS
loadmodule "xhttp.so"
loadmodule "websocket.so"
loadmodule "nathelper.so"
#!endif
```

La sentencia loadmodule "websocket.so", está indicando que se debe cargar el módulo “websocket.so” quien proporciona al Proxy el soporte necesario para manejar SIP usando Websockets como protocolo de transporte.

4.- En la sección siguiente, se parametrizan los módulos cargados.

```
# ------- setting module-specific parameters -----------

# ----- tm params -----
# auto-discard branches from previous serial forking leg
modparam("tm", "failure_reply_mode", 3)
# default retransmission timeout: 30sec
modparam("tm", "fr_timer", 30000)
# default invite retransmission timeout after 1xx: 120sec
modparam("tm", "fr_inv_timer", 120000)

# ----- rr params -----
# add value to ;lr param to cope with most of the UAs
modparam("rr", "enable_full_lr", 1)
# do not append from tag to the RR (no need for
this script)
modparam("rr", "append_fromtag", 0)

# ----- registrar params -----
modparam("registrar", "method_filtering", 1)
modparam("registrar", "max_expires", 3600)
modparam("registrar", "gruu_enabled", 0)

# ----- usrloc params -----
modparam("usrloc", "db_url", "DBURL")
modparam("usrloc", "db_mode", 0)

# ----- auth_db params -----
modparam("auth_db", "db_url", "DBURL")
modparam("auth_db", "calculate_ha1", yes)
modparam("auth_db", "password_column", "password")
modparam("auth_db", "load_credentials", "")

#!ifdef WITH_TLS
# ----- tls params -----
modparam("tls", "config",
"/usr/local/kamailio/etc/kamailio/tls.cfg")
#!endif

#!ifdef WITH_WEBSOCKETS
# ----- nathelper params -----
modparam("nathelper|registrar", "received_avp",
"$avp(RECEIVED)")
# Note: leaving NAT pings turned off here as nathelper is
_only_ being used for
#       WebSocket connections.  NAT pings are not needed
as WebSockets have
#       their own keep-alives.
#!endif
```

Cuestiones como qué método a usar para almacenar temporalmente la ubicación de cada usuario (local, MySQL, LDAP, etc) modparam ("usrloc", "db_url", "DBURL") o modparam ("usrloc", "db_mode", 0). Tiempos de expiración de un registro modparam ("registrar", "max_expires", 3600), son sentencias de esta sección.

Este archivo es el corazón de nuestro componente SIP Server, ya que la gran parte del archivo está dedicada a definir las reglas de enrutamiento de los mensajes SIP que llegan al Server, ya sean SIP sobre UDP o SIP sobre Websockets.

5.- Finalmente, el resto del archivo indica cómo realizar el manejo de dichos mensajes para poder establecer las sesiones SIP entre los endpoints.

A continuación se procede con un pequeño análisis para ayudar a entender la lógica de enrutamiento del SIP Server. La sección corresponde al encaminamiento de llamadas desde el SIP Server (192.168.0.16) hacia el Asterisk IPPBX (192.168.0.15), que permite comunicar llamadas desde usuarios WebRTC hacia usuarios de la IPPBX tradicional.

```
01 # PSTN GW routing
02 route[PSTN] {
03 #!ifdef WITH_PSTN
04 # check if PSTN GW IP is defined
05 if (strempty($sel(cfg_get.pstn.gw_ip))) {
06 xlog("SCRIPT: PSTN rotuing enabled but pstn.gw_ip not defined\n");
07 return;
08    }
09 # route to PSTN dialed numbers starting with '+' or '00'
10 #     (international format)
```

```
11 # - update the condition to match your dialing rules for PSTN
routing
12 if(!($rU=~"^(\+|00)[1-9][0-9]{3,20}$"))
13 return;
14 # only local users allowed to call
15 if(from_uri!=myself) {
16 sl_send_reply("403", "Not Allowed");
17 exit;
18 }
19 if (strempty($sel(cfg_get.pstn.gw_port))) {
20 $ru = "sip:" + $rU + "@" + $sel(cfg_get.pstn.gw_ip);
21 } else {
22 $ru = "sip:" + $rU + "@" + $sel(cfg_get.pstn.gw_ip) + ":"
23 + $sel(cfg_get.pstn.gw_port);
24 }
25 route(RELAY);
26 exit;
27 #!endif
```

En el fragmento citado, se observa como:

- *Línea “02” se declara la “ruta” llamada “PSTN”*

- *Línea “03” se comprueba si la constante “WITH_PSTN” fue declarada, para luego entrar en la condición verdadera y procesar la llamada o terminar la ejecución de la ruta PSTN.*

- *Línea “05” se comprueba si se a seteado la dirección IP del dispositivo a enviarle las llamadas.*

- *Línea “12” evalúa si viene un “00” como prefijo y un número de 3 dígitos de largo posteriormente.*

Los líneas posteriores, se encargan de otras validaciones para finalmente enviar la una llamada hacia el IPPBX Server y así lograr una comunicación entre ambos tipos de usuarios.

7.8.2.- El archivo /etc/kamailio/kamailio.cfg

El siguiente archivo se utiliza para instalar y configurar WebRTC con Kamailio:

```
#!KAMAILIO
#
# Simple/sample kamailio.cfg for running a proxy/registrar with
TLS and
# WebSockets support.

#!substdef
"!DBURL!mysql://root:Freetech123@localhost/kamailio!g"
#!substdef "!MY_IP_ADDR!192.168.66.21!g"
#!substdef "!MY_WS_PORT!80!g"
#!substdef "!MY_WSS_PORT!443!g"
#!substdef "!MY_WS_ADDR!tcp:MY_IP_ADDR:MY_WS_PORT!g"
#!substdef "!MY_WSS_ADDR!tls:MY_IP_ADDR:MY_WSS_PORT!g"

##!define LOCAL_TEST_RUN
##!define WITH_TLS
#!define WITH_WEBSOCKETS


####### Global Parameters #########

fork=yes
children=4

#alias="example.com"

#!ifdef WITH_TLS
enable_tls=1
#!endif

listen=MY_IP_ADDR
#!ifdef WITH_WEBSOCKETS
listen=MY_WS_ADDR
#!ifdef WITH_TLS
listen=MY_WSS_ADDR
#!endif
#!endif

tcp_connection_lifetime=3604
tcp_accept_no_cl=yes
tcp_rd_buf_size=16384

debug=2
```

```
# set paths to location of modules (to sources or installation
folders)
#!ifdef WITH_SRCPATH
mpath="modules_k:modules"
#!else
mpath="/usr/lib/i386-linux-gnu/kamailio/modules/"
#!endif


loadmodule "mi_fifo.so"
loadmodule "db_mysql.so"
loadmodule "tm.so"
loadmodule "sl.so"
loadmodule "rr.so"
loadmodule "pv.so"
loadmodule "maxfwd.so"
loadmodule "usrloc.so"
loadmodule "registrar.so"
loadmodule "textops.so"
loadmodule "siputils.so"
loadmodule "xlog.so"
loadmodule "sanity.so"
loadmodule "ctl.so"
loadmodule "auth.so"
loadmodule "auth_db.so"
loadmodule "kex.so"
loadmodule "mi_rpc.so"
#!ifdef WITH_TLS
loadmodule "tls.so"
#!endif
#!ifdef WITH_WEBSOCKETS
loadmodule "xhttp.so"
loadmodule "websocket.so"
loadmodule "nathelper.so"
#!endif

# ----------------- setting module-specific parameters ---------
------

# ----- tm params -----
# auto-discard branches from previous serial forking leg
modparam("tm", "failure_reply_mode", 3)
# default retransmission timeout: 30sec
modparam("tm", "fr_timer", 30000)
# default invite retransmission timeout after 1xx: 120sec
modparam("tm", "fr_inv_timer", 120000)

# ----- rr params -----
# add value to ;lr param to cope with most of the UAs
modparam("rr", "enable_full_lr", 1)
# do not append from tag to the RR (no need for this script)
modparam("rr", "append_fromtag", 0)
```

```
# ----- registrar params -----
modparam("registrar", "method_filtering", 1)
modparam("registrar", "max_expires", 3600)
modparam("registrar", "gruu_enabled", 0)

# ----- usrloc params -----
modparam("usrloc", "db_url", "DBURL")
modparam("usrloc", "db_mode", 0)

# ----- auth_db params -----
modparam("auth_db", "db_url", "DBURL")
modparam("auth_db", "calculate_ha1", yes)
modparam("auth_db", "password_column", "password")
modparam("auth_db", "load_credentials", "")

#!ifdef WITH_TLS

# ----- tls params -----
modparam("tls", "config",
"/usr/local/kamailio/etc/kamailio/tls.cfg")
#!endif

#!ifdef WITH_WEBSOCKETS
# ----- nathelper params -----
modparam("nathelper|registrar", "received_avp",
"$avp(RECEIVED)")
# Note: leaving NAT pings turned off here as nathelper is _only_
being used for
#       WebSocket connections.  NAT pings are not needed as
WebSockets have
#       their own keep-alives.
#!endif

modparam("mi_fifo", "fifo_name", "/tmp/kamailio_fifo")

####### Routing Logic ########

# Main SIP request routing logic
# - processing of any incoming SIP request starts with this
route
# - note: this is the same as route { ... }
request_route {

    # per request initial checks
    route(REQINIT);

#!ifdef WITH_WEBSOCKETS
    if (nat_uac_test(64)) {
         # Do NAT traversal stuff for requests from a WebSocket
         # connection - even if it is not behind a NAT!
         # This won't be needed in the future if Kamailio
         and the
```

```
# WebSocket client support Outbound and Path.
          force_rport();
          if (is_method("REGISTER"))
               fix_nated_register();
          else {
               if (!add_contact_alias()) {
                    xlog("L_ERR", "Error aliasing contact
<$ct>\n");
                    sl_send_reply("400", "Bad Request");
                    exit;
               }
          }
     }
#!endif

     # handle requests within SIP dialogs
     route(WITHINDLG);

     ### only initial requests (no To tag)

     # CANCEL processing
     if (is_method("CANCEL")) {
          if (t_check_trans())
               t_relay();
          exit;
     }


     t_check_trans();

     # authentication
     route(AUTH);

     # record routing for dialog forming requests (in case they
are routed)
     # - remove preloaded route headers
     remove_hf("Route");
     if (is_method("INVITE"))
          record_route();

     # handle registrations
     route(REGISTRAR);

     if ($rU==$null) {
          # request with no Username in RURI
          sl_send_reply("484","Address Incomplete");
          exit;
     }

     # user location service
     route(LOCATION);
```

```
    route(RELAY);
}


route[RELAY] {
    if (!t_relay()) {
        sl_reply_error();
    }
    exit;
}

# Per SIP request initial checks
route[REQINIT] {
    if (!mf_process_maxfwd_header("10")) {
        sl_send_reply("483","Too Many Hops");
        exit;
    }

    if(!sanity_check("1511", "7")) {
        xlog("Malformed SIP message from $si:$sp\n");
        exit;
    }
}

# Handle requests within SIP dialogs
route[WITHINDLG] {
    if (has_totag()) {
        # sequential request withing a dialog should
        # take the path determined by record-routing
        if (loose_route()) {
#!ifdef WITH_WEBSOCKETS
            if ($du == "") {
                if (!handle_ruri_alias()) {
                    xlog("L_ERR", "Bad alias <$ru>\n");
                    sl_send_reply("400", "Bad Request");
                    exit;
                }
            }
#!endif
            route(RELAY);
        } else {
            if ( is_method("ACK") ) {
                if ( t_check_trans() ) {
                    # no loose-route, but stateful ACK;
                    # must be an ACK after a 487
                    # or e.g. 404 from upstream server
                    t_relay();
                    exit;
                } else {
                    # ACK without matching transaction...
                    # ignore and discard
```

```
                        exit;
                    }
                }
                sl_send_reply("404","Not here");
            }
            exit;
        }
}

# Handle SIP registrations
route[REGISTRAR] {
    if (is_method("REGISTER")) {
        if (!save("location"))
            sl_reply_error();

        exit;
    }
}


# USER location service
route[LOCATION] {
    if (!lookup("location")) {
        $var(rc) = $rc;
        t_newtran();
        switch ($var(rc)) {
            case -1:
            case -3:
                send_reply("404", "Not Found");
                exit;
            case -2:
                send_reply("405", "Method Not Allowed");
                exit;
        }
    }
}

# Authentication route
route[AUTH] {
    if (is_method("REGISTER") || from_uri==myself) {
        # authenticate requests
        if (!auth_check("$fd", "subscriber", "1")) {
            auth_challenge("$fd", "0");
            exit;
        }
        # user authenticated - remove auth header
        if(!is_method("REGISTER"))
            consume_credentials();
    }
    # if caller is not local subscriber, then check if it calls
    # a local destination, otherwise deny, not an open relay
here
```

```
    if (from_uri!=myself && uri!=myself) {
         sl_send_reply("403","Not relaying");
         exit;
    }
}

#!ifdef WITH_WEBSOCKETS
onreply_route {
    if (nat_uac_test(64)) {
         # Do NAT traversal stuff for replies to a WebSocket
connection
         # - even if it is not behind a NAT!
         # This won't be needed in the future if Kamailio and
the
         # WebSocket client support Outbound and Path.
         add_contact_alias();
    }
}

event_route[xhttp:request] {
    set_reply_close();
    set_reply_no_connect();

    if ($Rp != MY_WS_PORT && $Rp != MY_WSS_PORT) {
         xlog("L_WARN", "HTTP request received on $Rp\n");
         xhttp_reply("403", "Forbidden", "", "");
         exit;
    }

    xlog("L_DBG", "HTTP Request Received\n");

    if ($hdr(Upgrade)=~"websocket"
              && $hdr(Connection)=~"Upgrade"
              && $rm=~"GET") {
         xlog("L_DBG", "WebSocket\n");
         xlog("L_DBG", " Host: $hdr(Host)\n");



        xlog("L_DBG", " Origin: $hdr(Origin)\n");
         if ($hdr(Host) == $null || !is_myself($hdr(Host))) {
              xlog("L_WARN", "Bad host $hdr(Host)\n");
              xhttp_reply("403", "Forbidden", "", "");
              exit;
         }

         # Optional... validate Origin
         # Optional... perform HTTP authentication

         # ws_handle_handshake() exits (no further
configuration file
```

```
          # processing of the request) when complete.
          if (ws_handle_handshake())
          {
               # Optional... cache some information abou the
               # successful connection
               exit;
          }
     }

     xhttp_reply("404", "Not found", "", "");
}

event_route[websocket:closed] {
     xlog("L_INFO", "WebSocket connection from $si:$sp has
closed\n");
}
#!endif
```

7.8.3.- Configuración de la plataforma WebRTC en Jitsi Web Meet

A continuación se plantea la aplicación directa de WebRTC como tecnología implementada en un servidor de Comunicaciones Unificadas (Video Conferencia, Mensajería Instantánea y Compartición de Pantalla).

Jitsi Webmeet es un proyecto Open Source, que mediante el uso de WebRTC como tecnología libre, implementa un completo servidor de conferencias.

A continuación se procede con su instalación y respectivas pruebas.

1.- Instalación

Se agregan los repositorios Debian para descargar e instalar la plataforma

```
echo 'deb http://download.jitsi.org/nightly/
deb unstable/' >> /etc/apt/sources.list
wget -qO - https://download.jitsi.org/
nightly/deb/unstable/archive.key | apt-
key add -
```

Se actualiza la lista de paquetes disponibles y se procede con la instalación.

```
apt-get update
apt-get -y install jitsi-meet
```

En medio de la instalación, se solicita ingresar el hostname o dirección IP, por lo tanto se debe ingresar la dirección IP del host (192.168.66.24).

.- Pruebas de WebRTC en Jitsi Web Meet

Se debe acceder desde un navegador web Chrome a la dirección IP del nuevo server: **https://192.168.66.24**

Se debe visualizar una interfaz intuitiva, en la cual se da la opción de crear un salón de conferencia y comenzar a invitar participantes.

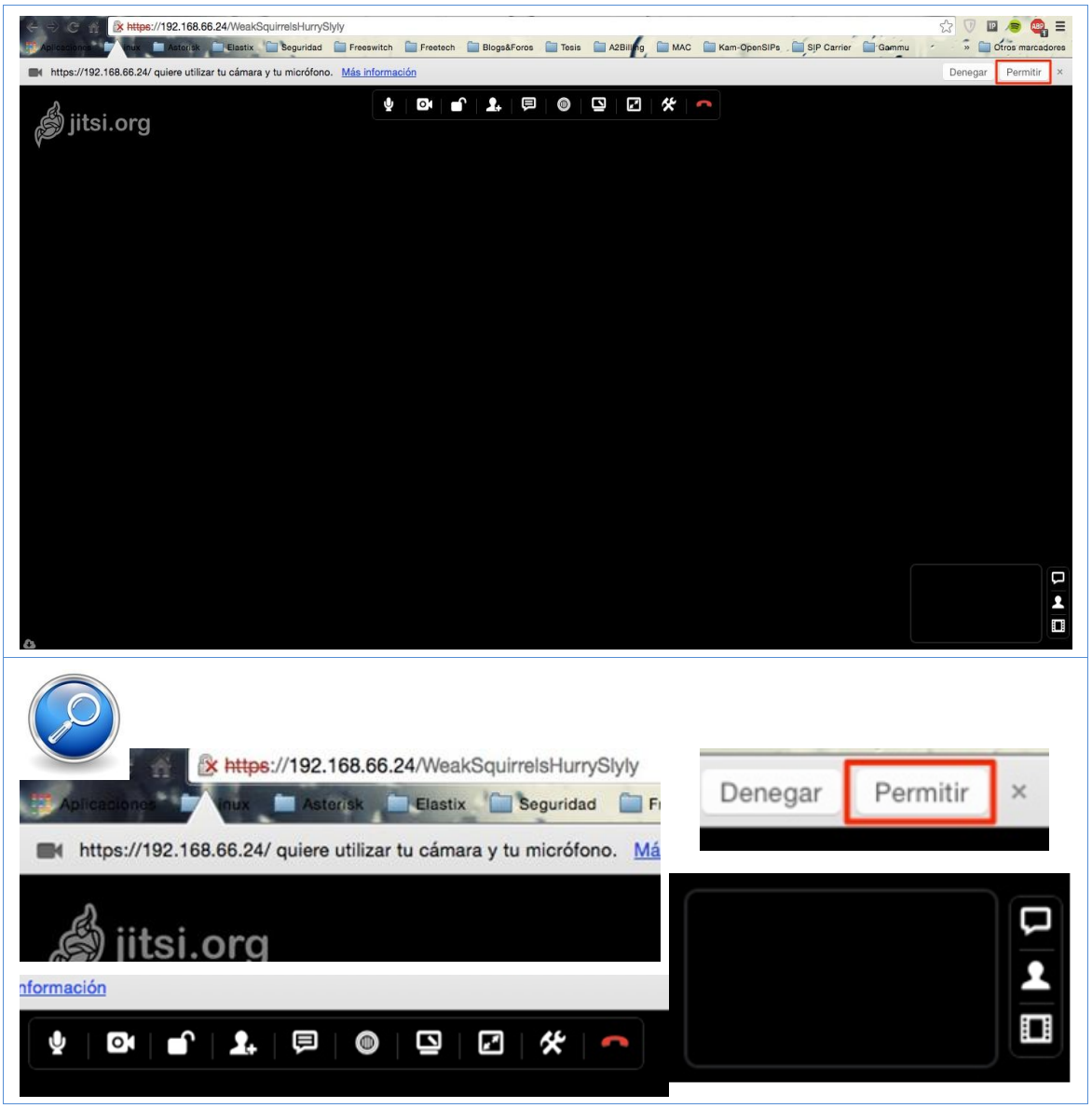


Figura 63.- Análisis de un paquete Websocket que transporta un mensaje instantáneo, a partir de lanzar la captura de paquetes en la consola de comandos del host Proxy SIP Kamailio.

7.9.- Instalación del servidor Linux

La primer pantalla nuestra cuando el sistema arranca el instalador, es el menú de opciones. Se debe seleccionar la primera opción e ingresar Enter, para comenzar la instalación.

Figura 64.- Pantalla de inicio post-boot de la maquina desde la imagen de instalación de GNU/Linux Debian.

En el menú siguiente, se debe seleccionar el idioma y el próximo paso el idioma:

Figura 65.- Pantalla de selección: Idioma y País.

A continuación se selecciona el tipo de teclado:

Figura 66.- Pantalla de configuración del teclado.

En este paso se debe nombrar al host. Como será identificado por los demás Host de la red.

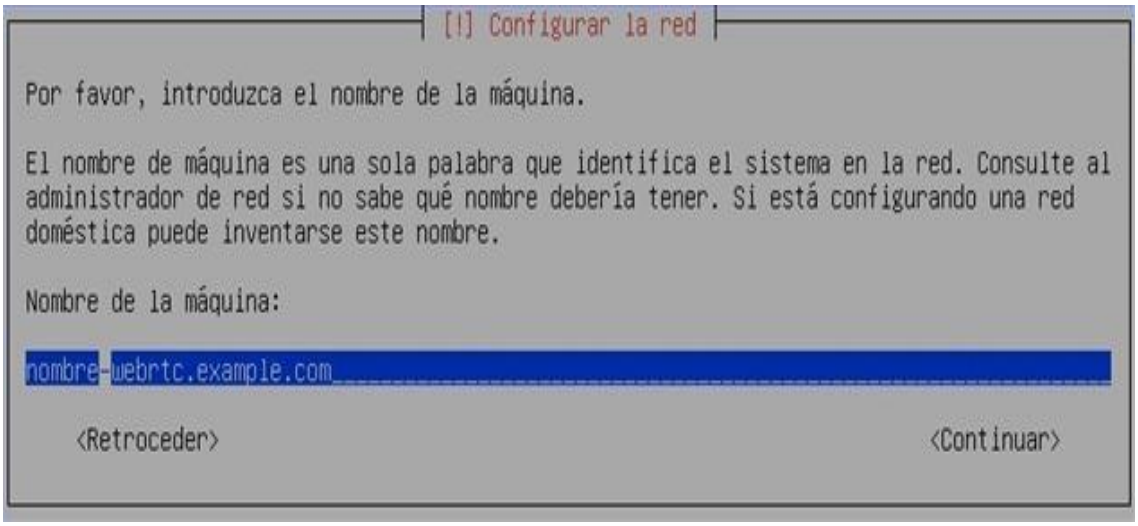


Figura 67.- Pantalla de configuración de nombre de host.

Aquí se debe seleccionar una contraseña para el super usuario root y luego reconfirmarla.

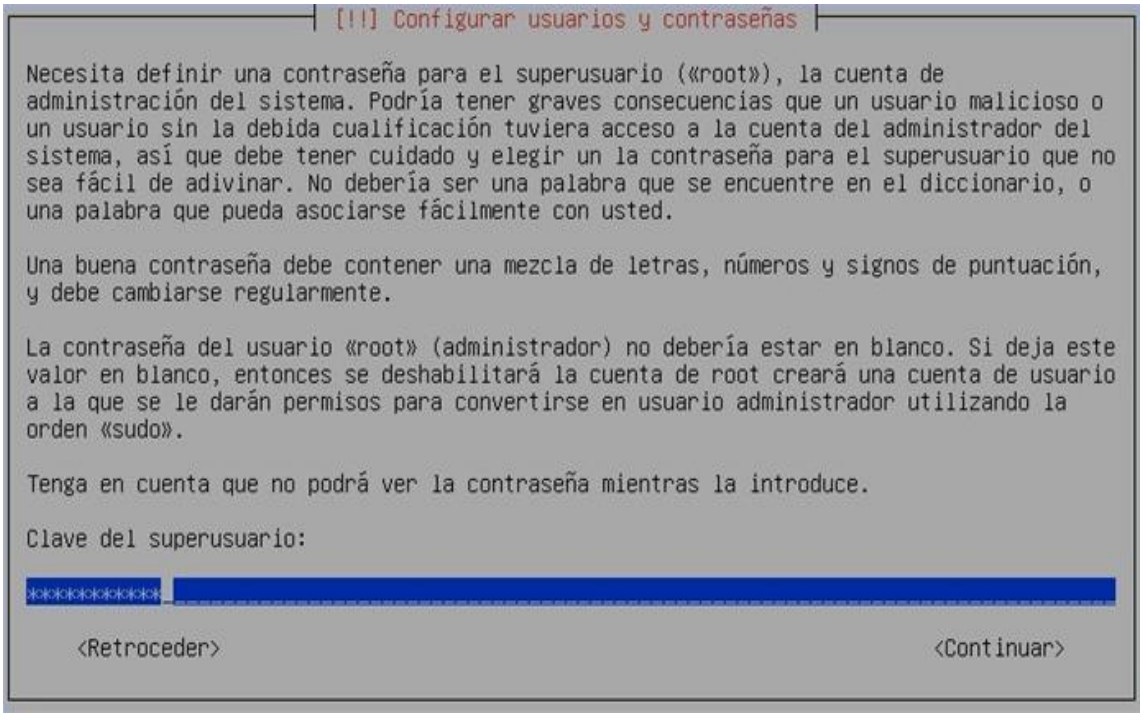


Figura 68.- Ingreso de clave del usuario root de Linux.

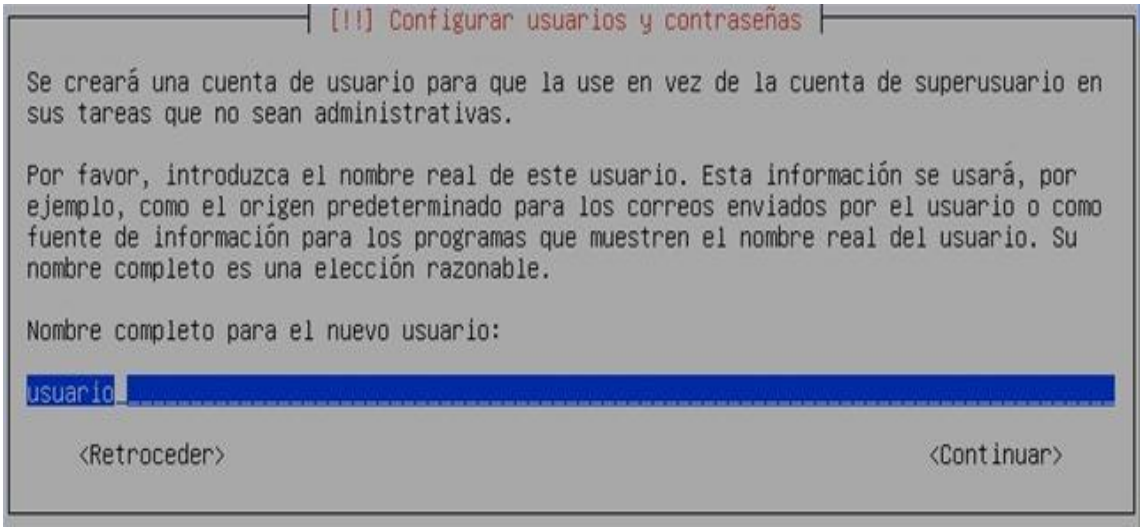


Figura 69.- Ingreso de nombre de usuario normal de Linux a ser creado.

En el siguiente paso, se debe proceder con la creación de un usuario estándar del sistema y posteriormente ingresar la contraseña para dicho usuario.

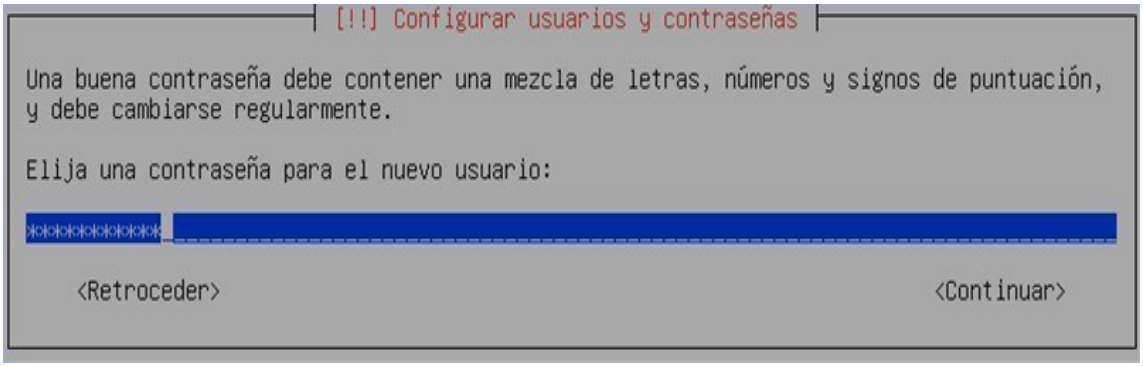
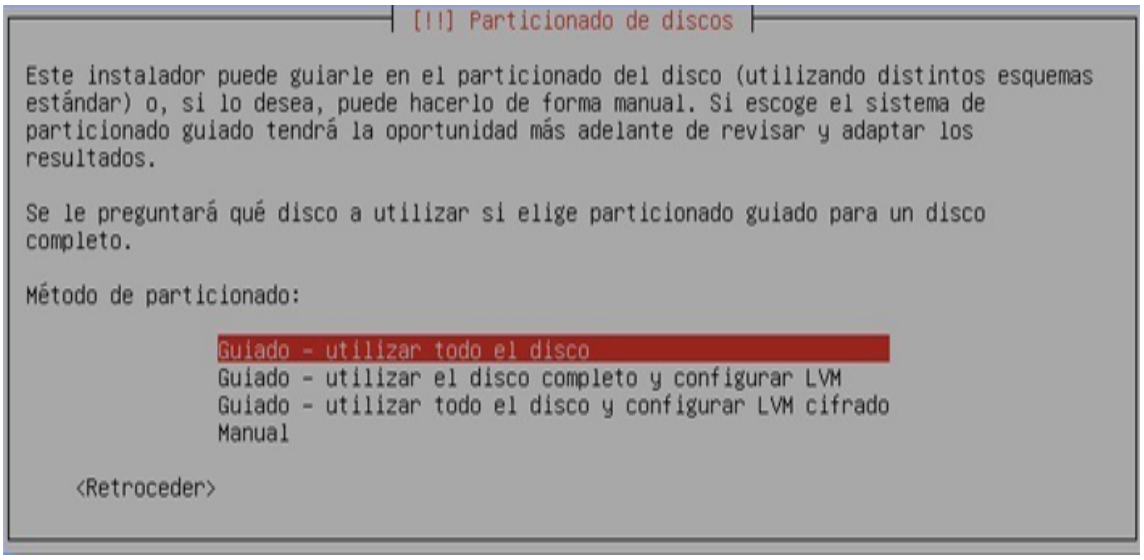


Figura 70.- Ingreso de clave para el usuario normal.

A continuación se debe seleccionar la opción para participar el disco duro del sistema. Método del participando y dispositivo en si a particionar.



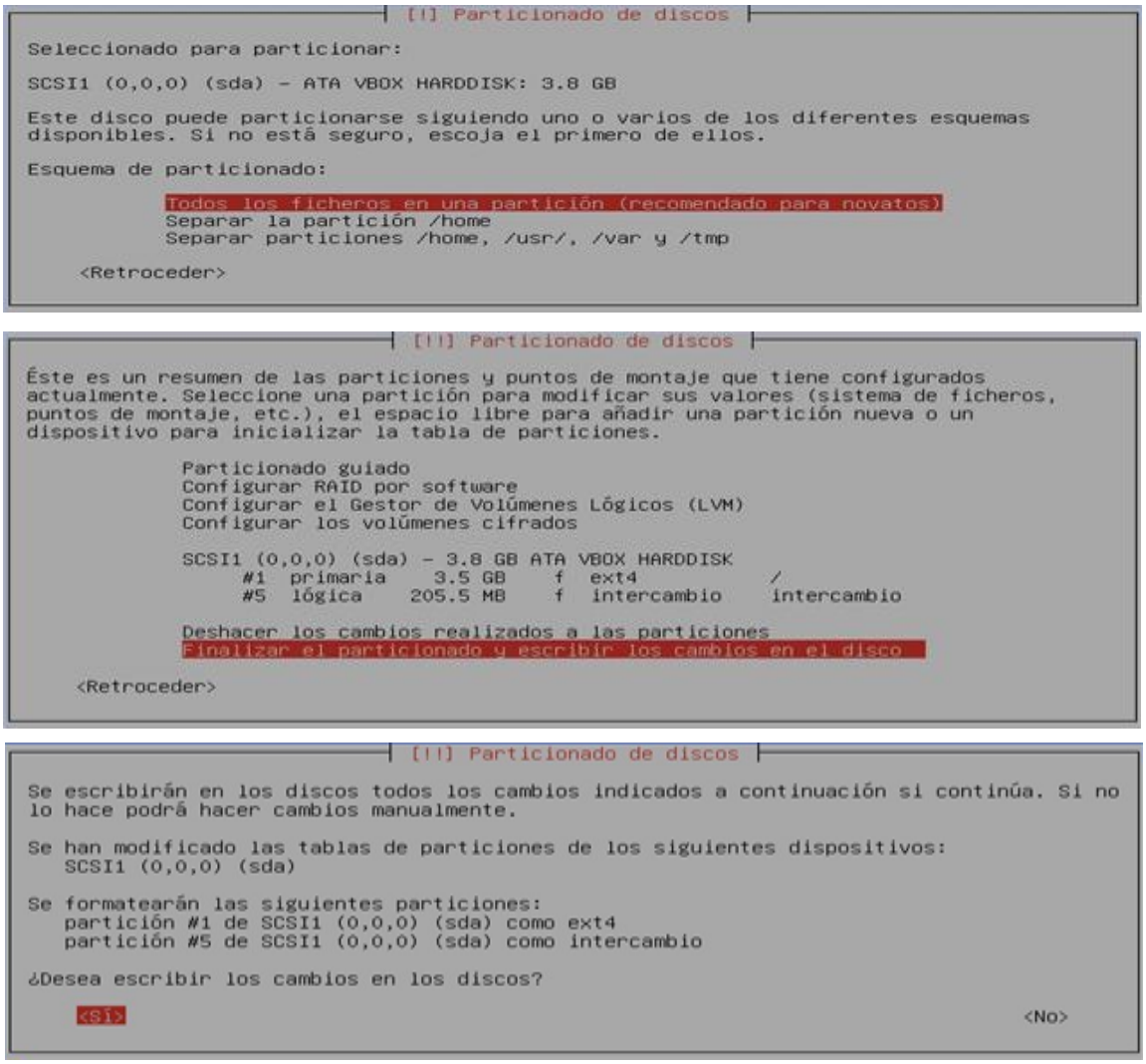


Figura 71.- Pasos a realizar para particionar el disco de la maquina.

El sistema comienza a instalar algunos paquetes para poder proceder con la instalación.

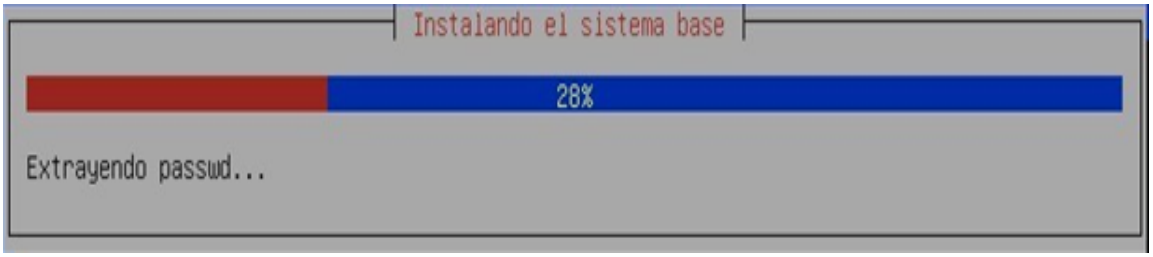


Figura 72.- Instalación de paquetes básicos.

En este menú, se debe seleccionar el país donde estamos, ya que tiene que ver con los servidores de red que serán utilizados para instalar nuevos paquetes.

Figura 73.- Selección de repositorios de software

Si no existe un proxy web en la red, se debe dejar en blanco.

Figura 74.- Ingreso de parámetros de Proxy Web

(en caso de existir uno en la red)

Esta opción simplemente indica si se desea participar con el proyecto debían, enviando información sobre el uso de paquetes que se instalan.

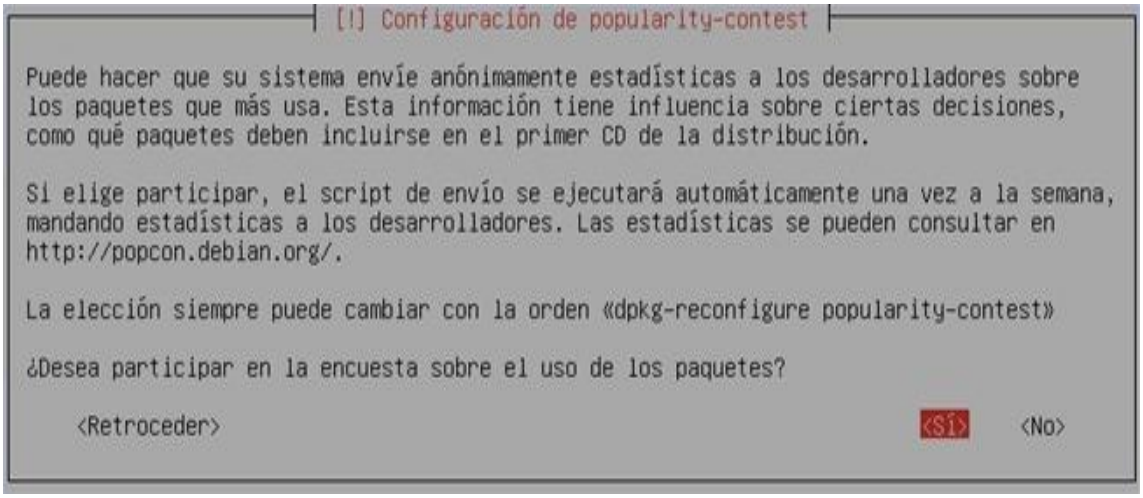


Figura 75.- Permitir o Denegar la encuesta sobre uso de paquetes.

En este punto se selecciona la distribución de paquetes deseada. Para nuestro caso, simplemente seleccionamos “SSH Server” y “Utilidades estándar del sistema”.

Figura 76.- Pantalla de selección de programas a instalar.

Se indica la instalación del “Arrancador” de Linux GRUB.

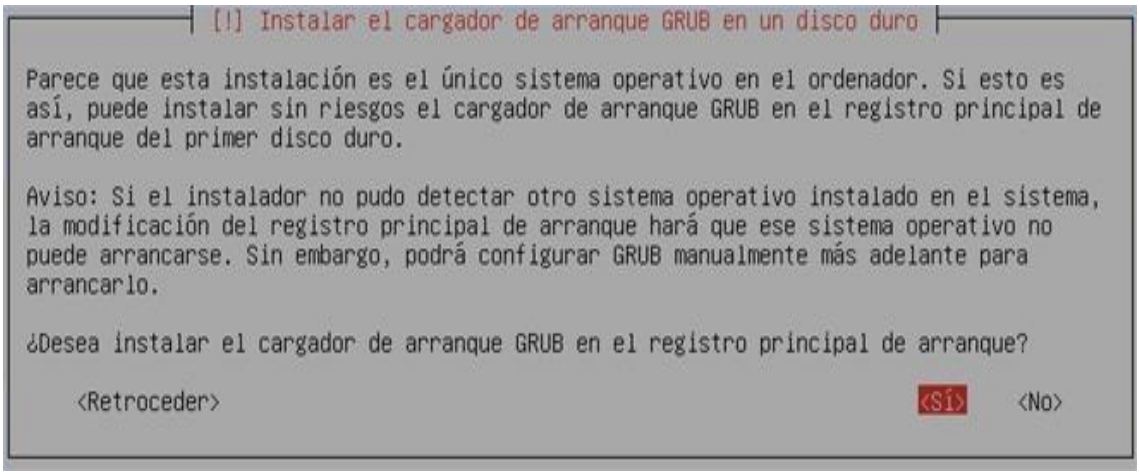


Figura 77.- Pantalla de selección para instalar el cargador de arranque.

Finalmente, se selecciona “continuar” para que se proceda con el primer arranque del recientemente instalado GNU/Linux Debian.

Figura 78.- Pantalla de aceptación de fin de instalación y reinicio de la maquina.

Se visualiza el arrancador de Debian.

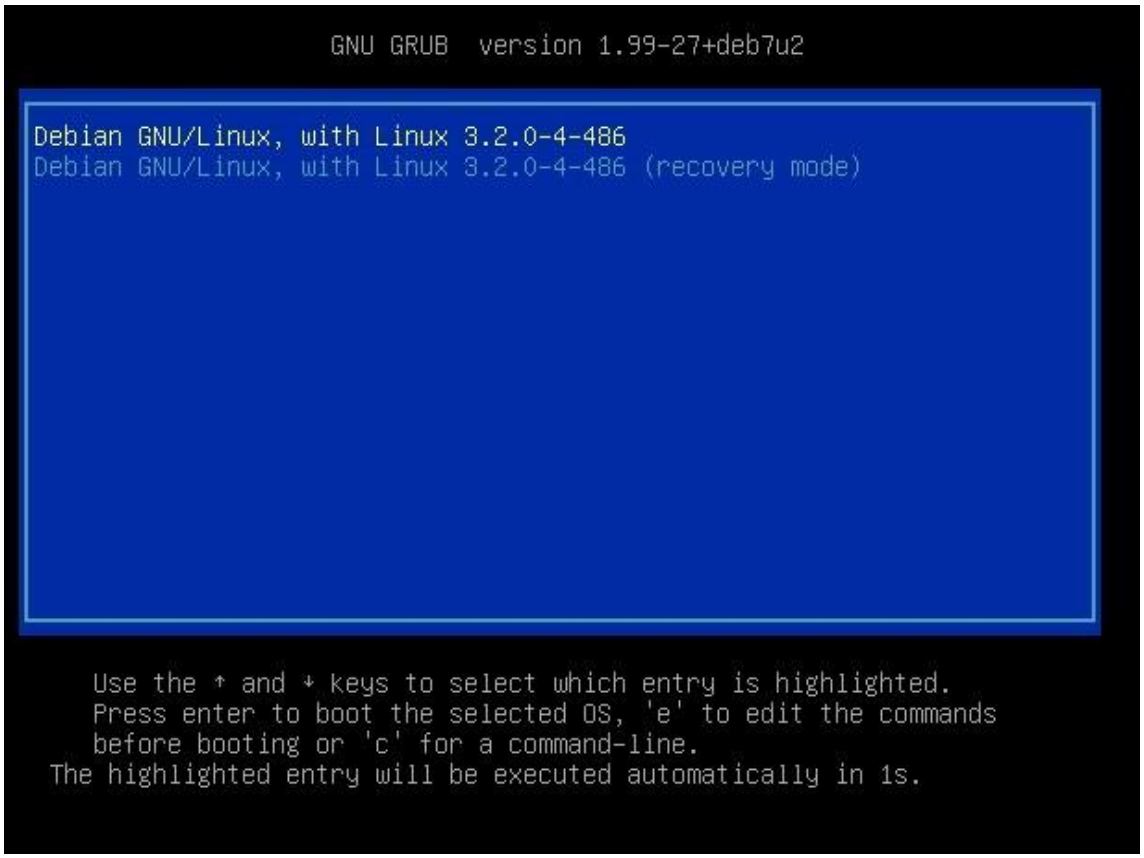


Figura 79.- Pantalla de selección de sistema operativo y/o kernel a cargar.

Luego del proceso de arranque, el sistema arrojará la pantalla de Login.

Se procede con el login del usuario root.

Figura 80.- Proceso de carga finalizado, sistema listo para utilizar.

Ya se posee el sistema GNU/Linux Debian listo para trabajar.

Figura 81.- Sesión iniciada como usuario root.

Una vez instalado el sistema operativo, se procede con la configuración de los parámetros de red para dejar al mismo disponible como host y con acceso a internet para la descarga de los paquetes necesarios. Entonces vamos a editar el archivo */etc/network/interfaces* y lo dejamos con el siguiente aspecto:

```
# The loopback network interface auto lo
iface lo inet loopback

# The primary network interface allow-hotplug eth0
iface eth0 inet static
address XXX.XXX.XXX.XXX netmask YYY.YYY.YYY.YYY
gateway ZZZ.ZZZ.ZZZ.ZZZ
```

Donde XXX.XXX.XXX.XXX, YYY.YYY.YYY.YYY y ZZZ.ZZZ.ZZZ.ZZZ debe ser reemplazado con los valores apropiados para su instalación.

Finalmente se guardan los cambios sobre el archivo y se procede con un reinicio del sistema.