ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL

Facultad de Ingeniería en Electricidad y Computación

"DISEÑO DE UNA RED PARA VOZ SOBRE IP EN LA NUBE Y POSIBLE IMPLEMENTACIÓN CON HTML5"

Tesis de Grado

Previo a la obtención del Título de:

MAGISTER EN TELECOMUNICACIONES

Presentado por:

GIUSEPPE LEONARDO BLACIO ABAD

Guayaquil – Ecuador

2013

AGRADECIMIENTO

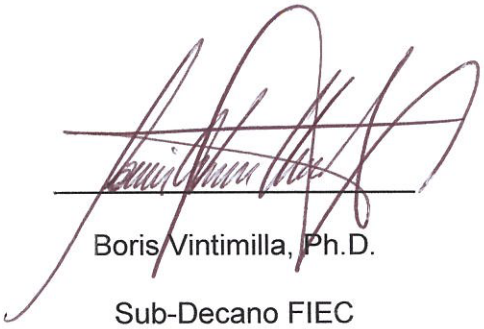
Quiero agradecer principalmente a mi Dios, quien en todo momento me ha sustentado y a quien todo le debo. A mi Madre por guiarme con todo su amor en este camino, y a mi Padre por haber sido una piedra fundamental durante el proceso de formación de mi vida. A mis compañeros de la Maestría quienes me dieron toda su amistad y apoyo. A mi director Dr. Álvaro Suárez Sarmiento por ser más que un profesor un amigo y al Dr. Boris Ramos por ayudarnos y permitirnos ser parte de este proceso. Y también quiero agradecer mucho a la ESPOL, que me abrió sus puertas de par en par desde mi primer día en 1996.

.

DEDICATORIA

Quiero dedicar este trabajo a la persona que más quiero en este mundo, a mi hija Fiorella, quien me ha dado los mayores momentos de felicidad en esta vida, y que ella pueda superar todo cuanto yo logre con la ayuda de Dios.

TRIBUNAL DE SUSTENTACIÓN



Boris Vintimilla, Ph.D.

Sub-Decano FIEC

Álvaro Suárez Sarmiento, Ph.D.

Director de Tesis

Boris Ramos, Ph.D.

Vocal Principal

DECLARACIÓN EXPRESA

"La responsabilidad del contenido de esta Tesis de Grado, me corresponde exclusivamente; y el patrimonio intelectual de la misma a la Escuela Superior Politécnica del Litoral".

(Art. 12 del Reglamento de Graduación de la ESPOL)

GIUSEPPE LEONARDO BLACIO ABAD

RESUMEN

Este trabajo de tesis realiza una guía para el diseño de redes para Voz sobre IP en la Nube, donde los usuarios no necesiten poseer equipos de conmutación directa a algún tipo de red telefónica, sino solamente una conexión a Internet para obtener todas las ventajas que esto conlleva, como son ahorro en inversiones considerables en equipos y pagos recurrentes a operadores telefónicos y obtener movilidad geográfica. Para lo cual se ha realizado un análisis de los diferentes elementos participantes de un diseño para una red de este tipo.

La segunda parte del trabajo fue realizar una investigación sobre si era posible utilizar protocolos Web como lo es *HTML5* para realizar llamadas de voz, sin la necesidad de instalar un *software* preexistente en el equipo terminal, sino realizarlas a través de un navegador Web. Para lo cual se realiza un detalle de *WebSockets* y *WebRTC* para dicho fin, y un diseño final considerando todos estos aspectos.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

GLOSARIO

API	Application Programming Interface
AS	Autonomous System
BGP	Border Gateway Protocol
CA	Call Agent
CAPEX	CAPital EXpenditures
CCS3	Cascading Style Sheets version 3
CO	Central Office’
Códec	Coder-Decoder
CSS	Cascading Style Sheets
DMZ	DeMilitarized Zone
DOM	Document Object Model
EGP	Exterior Gateway Protocol
EIGRP	Enhanced Interior Gateway Routing Protocol
GUI	Graphical User Interface
HTML	Hypertext Mark-Up Language
HTML5	Hypertext Mark-Up Language version 5
HTTP	Hypertext Transfer Protocol
IaaS	Infrastructure as a service
IAX	Inter-Asterisk eXchange
ICE	Interactive Connectivity Establishment
IETF	Internet Engineering Task Force

IGP	Interior Gateway Protocols
IP	Internet Protocol
IPSec	Internet Protocol Security
IS-IS	Intermediate System to Intermediate System
ISP	Internet Service Provider
IT	Information Technology
JS	JavaScript
LAN	Local Area Network
MCU	Multipoint Control Unit
MGCP	Media Gateway Control Protocol
NAT	Network Address Translation
OPEX	OPerational Expenditures
OSPF	Open Shortest Path First
PaaS	Platform as a service
PSTN	Public switched telephone network
PYMES	Pequeñas y Medianas Empresas
QoS	Quality of service
RAS	Registration, Admission and Status
REST	Representational State Transfer
RFC	Request for Comments
RIP	Routing Information Protocol
RTB	Red de telefonía básica
RTC	Real-time communication
RTP	Real-time Transport Protocol

SaaS	Software as a service
SCTP	Stream Control Transmission Protocol
SDP	Session Description Protocol
SIGTRAN	Signaling Transport
SIP	Session Initiation Protocol
SLA	Service Level Agreement
SRTP	Secure Real-time Transport Protocol
SS7	Signalling System No. 7
SSL	Secure Socket Layer
STUN	Session Traversal Utilities for NAT
TCP	Transmission Control Protocol
TLS	Transport Layer Security
TURN	Traversal Using Relays around NAT
UAC	User Agent Client
UAS	User Agent Server
UDP	User Datagram Protocol
UIT	Unión Internacional de Telecomunicaciones
URI	Uniform Resource Identifier
URL	Universal Resource Locator
VLAN	Virtual LAN
VoIP	Voice over Internet Protocol
VPN	Virtual private network
W3C	World Wide Web Consortium
WAN	Wide Area Network

WebRTC	Web Real Time Communication
XHTML	eXtensible HyperText Markup Language
XML	eXtensible Markup Language

INTRODUCCIÓN

La presente tesis tiene como objetivo realizar el análisis para un diseño para una red de Voz sobre Internet Protocol (*VoIP*) en la Nube, para lo cual se ha tomado como referencia a los modelos de implementación en la Nube, así como consideraciones para el diseño de redes, y los conceptos para utilizar navegadores Web en vez de *software* preexistentes para la realización de llamadas.

El capítulo 1 contiene la definición del proyecto, los objetivos, las limitaciones y el método a utilizar en la investigación.

El capítulo 2 contiene conceptos referentes a conceptos de Computación en la Nube, como son los modelos de servicios y los modelos de despliegue, así como la arquitectura general y los beneficios y limitaciones de servicios en la Nube.

El capítulo 3 contiene información acerca de *VoIP* en la Nube y se realiza una comparación frente al modelo de telefonía convencional, el modelo tradicional de *VoIP*, así como los diferentes protocolos necesarios para el funcionamiento como son los de señalización, encaminamiento y transporte. Así como las diferentes consideraciones para el diseño de las redes, como son las consideraciones para la *Red de Área Local* (*LAN*, del inglés *Local Area Network*), *Red de Área Amplia* (*WAN*, del inglés *Wide Area Network*),

Redes Privadas Virtuales (*VPN*, del inglés *Virtual Private Network*), zonas desmilitarizadas (*DMZ*, del inglés *DeMilitarized Zone*), pasarelas (*Gateways*) y se unen todos estos conceptos y se presenta el esquema general de diseño.

El capítulo 4 presenta el concepto aun en desarrollo de Lenguaje de Marcado de Hipertexto versión 5 (HTML5, del inglés, *Hypertext Mark-Up Language version 5*), así como de Interfaz de Programación de Aplicaciones (*API*, del inglés, *Application Programming Interface*) y *Javascript* los cuales son necesarios y facilitan la utilización de *HTML5*. Así como su compatibilidad con los distintos navegadores.

El capítulo 5 hace un análisis de la utilización de *HTML5* para poder realizar llamadas directamente desde navegadores Web con capacidad de procesamiento de llamadas en la Nube. Para el cual se introducen el concepto *WebSockets* y el proyecto *WebRTC* (del inglés, *Web Real Time Communication*). Después estos conceptos y los servidores Web necesarios para su servicio se complementan con el diseño general presentado en el Capítulo 3 y se presenta un esquema general para realizar llamadas desde equipos terminales e incluso desde navegadores Web.

CAPÍTULO 1

MARCO REFERENCIAL

1.1 DEFINICIÓN DEL PROYECTO

El concepto de Computación en la Nube (del inglés, *Cloud Computing*) fue utilizado por proveedores de servicio de Internet a gran escala, pero actualmente también la utilizan proveedores más pequeños.

La *VoIP* es una tecnología que permite realizar llamadas de voz utilizando una conexión a Internet, y que ha revolucionado el mundo de la telefonía, ofreciendo nuevos servicios y progresivamente a mejores precios al cliente.

Disponer de un servicio de *VoIP* en la Nube permitiría a una empresa el poder comunicarse, sin necesidad de comprar, instalar, configurar, gestionar, y mantener los equipos que realizan el procesamiento y conmutación de las llamadas telefónicas, únicamente tendría que marcar un número destino desde cualquier punto con acceso a Internet.

La *VoIP* en la Nube más que un servicio suministrado por la Web a centros de datos cercanos o remotos, es una solución que puede ayudar rápidamente a las organizaciones a optimizar recursos. Está demostrado que las empresas consideran que esta tecnología está en el núcleo de la forma en que operan sus negocios puesto que las comunicaciones de bajo coste económico son un elemento esencial de su modelo de negocio.

Las empresas enfrentan dificultades en relación a la complejidad de satisfacer las necesidades tecnológicas de manera rápida, con el mínimo costo y con el menor riesgo para sus inversiones. En este sentido, la *VoIP* en la Nube ofrece una amplia gama de servicios a una variedad de clientes, desde hogares, *Pequeñas y Medianas Empresas* (PYMES) o clientes empresariales, lo que les permite transformar sus comunicaciones gracias la Nube y las interfaces Web.

Como en toda solución deben existir recomendaciones a considerar, algunas de las cuales se abarcaran en el presente trabajo: medidas básicas de diseño de la infraestructura, el despliegue lógico del servicio, los tipos de tecnologías a usar, la escalabilidad del servicio. De especial interés es el estudio de las nuevas interfaces de telefonía basadas en la Web. Por ello, se estudia la posibilidad de implementar *VoIP* en dispositivos finales sin necesidad de un *software* pre-existente sino accediendo a ellas directamente desde un navegador Web.

1.2 IMPORTANCIA Y JUSTIFICACIÓN

El análisis de Predicciones de Movilidad para el 2012, del Yankee Group [1] señala que los servicios de Nube personal están llegando a una fase de elevado crecimiento. Y todo esto porque la Nube ofrece muchas ventajas como ahorros de costos, flexibilidad, mayor seguridad en contra de fallos o pérdidas, etc.

Por otro lado, los servicios en la Nube tienen detractores que alegan su poca seguridad, así como depender exclusivamente de un tercero para los servicios y necesitar de una conexión de Internet de banda ancha.

Nosotros pensamos que además un inconveniente desde el punto de vista de la usabilidad del servicio, es el hecho que el usuario deba tener una aplicación pre-instalada para poder utilizar los servicios de *VoIP* en la Nube,

independiente del protocolo de señalización. Si tenemos en cuenta que el mismo Yankee Group señala que en el 2012 las empresas optarán por *HTML5* sobre la codificación nativa para aplicaciones móviles en el mediano y largo plazo. Es por ello que se plantea estudiar la proyección empresarial de un servicio de *VoIP* en la Nube en el que el usuario no instala ninguna aplicación en su dispositivo y sólo utiliza un navegador Web compatible con la tecnología *HTML5*. De esta manera, las comunicaciones se llevan a cabo directamente desde un navegador sin necesidad de instalar una aplicación (generalmente de escritorio) en el terminal del usuario final [2].

Hasta donde nosotros conocemos, la tecnología de *VoIP* en la Nube utilizando *HTML5* no existe todavía, por lo cual al final del presente trabajo teórico se podrá decir si esto es factible. En caso que fuera posible indudablemente supondría una solución de incuestionable valor tanto técnico como comercial.

1.3 OBJETIVOS

El objetivo principal de esta tesis es esquematizar un diseño de *VoIP* en la Nube y estudiar la viabilidad técnica del uso de *HTML5* para trasiego de datos entre el Cliente y la Nube.

Los objetivos específicos son:

1. Esquematizar la infraestructura y requisitos básicos para una red escalable que ofrezca servicios *VoIP* en la Nube.
2. Mostrar sus fortalezas y debilidades de los sistemas de *VoIP* en la Nube frente a los sistemas tradicionales de *VoIP*.
3. Exponer la posibilidad de utilizar el protocolo *HTML5* para uso de *VoIP* en la Nube.

1.4 ALCANCES Y LIMITACIONES

Al culminar el presente trabajo de investigación se espera obtener los siguientes resultados:

1. Poner de manifiesto las ventajas que presenta un diseño de *VoIP* en la Nube y mostrar los elementos de red presentes.
2. Evidenciar la posibilidad de utilizar *HTML5* para reemplazar a los protocolos de señalización actuales en la interfaz del usuario.
3. Un documento en el que se recojan las tareas realizadas y las conclusiones principales del estudio realizado.

Subrayar que una parte del documento final serviría como base de la evidencia que es posible usar *HTML5*, para los propósitos expuestos en cuyo caso serviría como base a posteriores implantaciones. Pero también podría servir como base para evidenciar las dificultades o imposibilidades técnicas que demuestren que no es posible usar *HTML5*. En cuyo caso serviría como base para que se analicen alternativas que se podrían también atisbar en el documento inicialmente. En cualquier caso como resultado de valor añadido se podría obtener publicaciones técnicas en revistas de tirada mundial.

1.5 METODOLOGÍA

En el presente trabajo para la solución del problema planteado la metodología de la investigación es de tipo bibliográfico porque se realiza la consulta de libros, tesis, folletos, revistas, boletines y cualquier otro tipo de información escrita que se considere importante y necesaria para realizar la investigación.

Analizado el estado del arte bibliográfico se realiza una propuesta de viabilidad técnica del uso de *HTML5* a modo de consultoría científico-técnica. Para ello se usará un modelo de proyectado en el que se analiza el estado del arte y en paralelo posibles diseños de la interfaz Cliente-Nube mediante *HTML5*. Este esquema paralelo se itera secuencialmente hasta alcanzar una

solución posible o la evidencia de que no es posible llevar a cabo un diseño eficaz de dicha interfaz.

CAPÍTULO 2

COMPUTACIÓN EN LA NUBE

2.1 DEFINICIÓN DE COMPUTACIÓN EN LA NUBE

La Computación en la Nube se puede definir como el más alto nivel de servicios computacionales ofrecidos por un tercero, disponibles para su uso cuando sea necesario, y que puede ser escalable de forma dinámica en respuesta a las necesidades cambiantes [3]. La Computación en la Nube representa un punto de partida para el desarrollo, operación y administración de los sistemas de Tecnología de la Información (del inglés, *IT*: *Information Technology*). Desde el punto de vista económico, no sólo la adopción de Computación en la Nube tiene el potencial de proporcionar beneficios

económicos enormes, sino que proporciona también mucha mayor flexibilidad.

La Computación en la Nube se refiere a la ejecución de las aplicaciones y los servicios en servidores de Internet (utilizando servidores distribuidos) y cuyo acceso se hace por medio de protocolos y lenguajes estándares, normalmente *Protocolo de Transferencia de Hipertexto* (*HTTP*, del inglés, *Hypertext Transfer Protocol*) y *HTML*. Se distingue por la idea de que los recursos son virtuales y sin límites y que los detalles de los sistemas físicos en los que ejecutan el *software* solicitado son abstractos para el usuario [4]. Es por ello que la Computación en la Nube representa un verdadero cambio de paradigma en la forma en que se despliegan los sistemas.

2.2 CARACTERÍSTICAS ESENCIALES DE COMPUTACION EN LA NUBE

Se puede resumir los principios fundamentales de la Computación en la Nube de la siguiente manera (Figura 2.1):

1. Recursos informáticos compartidos disponibles para cualquier usuario suscriptor.
2. Recursos informáticos virtualizados para aprovechar al máximo la utilización del *hardware*.

3. Escalabilidad según las necesidades.
4. Facturación de recursos solo cuando son utilizados o el pago de una tarifa ilimitada.

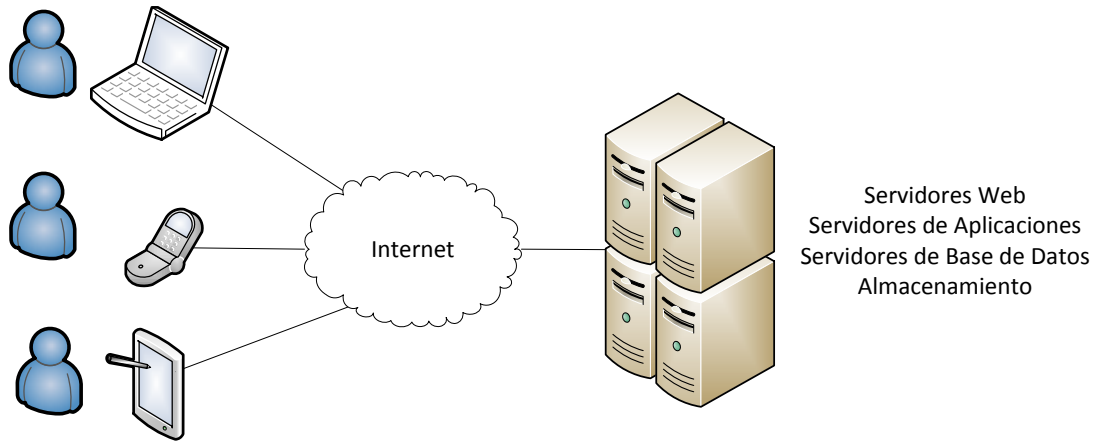


Figura 2.1. Esquema básico de servicios en la Nube. [5]

2.3 MODELOS DE SERVICIO EN LA NUBE

El término servicio en un modelo de Computación en la Nube se refiere a ser capaz de utilizar componentes reusables a través de Internet. Esto significa que las empresas no harían inversiones en costosos equipos de computación ni *software* de escritorio tradicional. Por lo tanto prácticamente elimina las barreras de entrada al mercado a pequeñas empresas. Así mismo es un modelo de servicios que permite una alta escalabilidad, permite que los recursos sean compartidos por muchos usuarios y proporciona una

independencia del *hardware* de los dispositivos de computación y comunicación.

A continuación se detallan los distintos modelos para servicios en la Nube: *Software como Servicio* (*SaaS*), *Plataforma como servicio (PaaS)* e *Infraestructura como servicio (IaaS)* [5].

2.3.1 Software como Servicio

SaaS es un modelo de distribución de *software* donde una empresa proveedora ofrece el mantenimiento, soporte y operación que usa el cliente durante el tiempo que contrate el servicio. El proveedor del servicio mantiene la información del cliente en sus sistemas y provee los recursos necesarios para explotar esa información, pero el usuario no tiene control sobre las mismas [5], como se aprecia en la Figura 2.2.

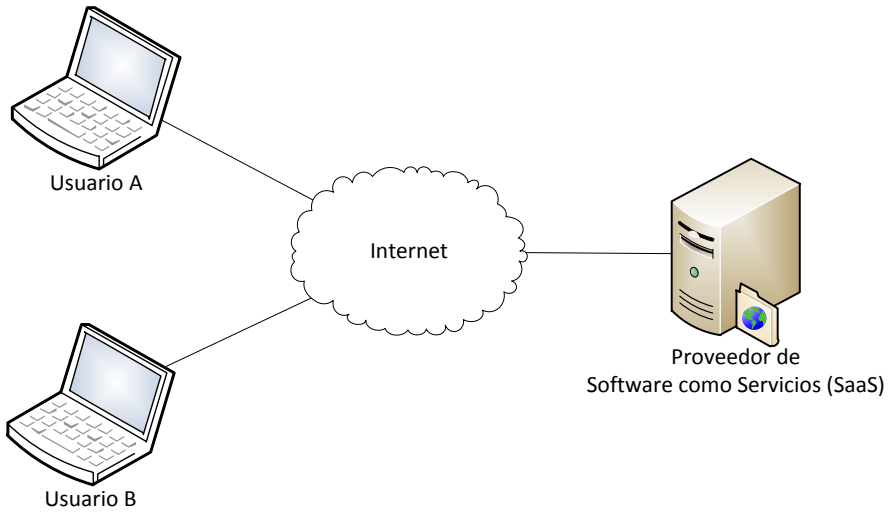


Figura 2.2. Esquema de *Software como Servicio (SaaS)* en la Nube. [5]

Se debe considerar que el esquema de costos en este servicio puede ser del tipo bajo demanda o una tarifa fija por la utilización del servicio. Para las empresas proveedoras de estos servicios, *SaaS* tiene el atractivo de ofrecer una mayor protección de su propiedad intelectual.

Hay muchos tipos de aplicaciones que se prestan para el modelo *SaaS*. Generalmente los programas que realizan una tarea simple, sin mucha necesidad de interactuar con otros sistemas los hace candidatos ideales. Los clientes que no están inclinados a llevar a cabo el desarrollo de *software*, pero que necesitan de potentes aplicaciones también pueden beneficiarse de *SaaS*. Algunas de estas aplicaciones incluyen servicios de atención al cliente, videoconferencia, gestión de servicios *IT*, aplicaciones específicas, entre otros.

Las aplicaciones *SaaS* fueron diseñadas específicamente para utilizar herramientas Web, como los navegadores (del inglés, *browser*), esto lo convierte en Web nativo [6]. *SaaS* se ideó para suministrar *software* desde una ubicación central, y accedido por los usuarios desde cualquier sitio con conexión a Internet, así como ofrecer capacidad multiusuario.

2.3.2 Plataforma como Servicio

PaaS proporciona todos los recursos necesarios para crear aplicaciones y servicios, completamente desde Internet, sin tener que descargar o instalar un *software* adicional. Los servicios *PaaS* incluye el diseño de aplicaciones, desarrollo, prueba, despliegue y alojamiento. Otros servicios incluyen la colaboración en equipo, la integración de servicios Web, integración de bases de datos, seguridad, escalabilidad y almacenamiento [5].

El usuario no tiene control sobre la plataforma ni sobre las infraestructuras pero si sobre sus aplicaciones, como se muestra en la Figura 2.3.

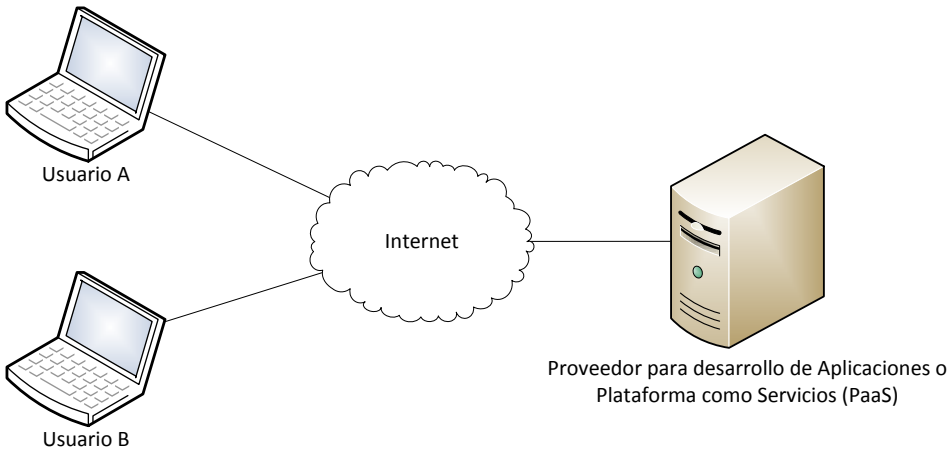


Figura 2.3. Esquema de *Plataforma como Servicio (PaaS)* en la Nube. [5]

Una debilidad de *PaaS* es la falta de interoperabilidad y la portabilidad entre los proveedores. Esto es, si se crea una aplicación con un proveedor de la Nube y decide trasladarse a otro proveedor, puede que no sea capaz de hacerlo. Además, si el proveedor se retira del negocio, las aplicaciones y los datos se perderían.

2.3.3 Infraestructura como Servicio

Mediante *IaaS* el cliente contrata únicamente las infraestructuras tecnológicas (capacidad de procesamiento, de almacenamiento y/o de comunicaciones) [4]. Sobre dicha *IaaS* el cliente aloja sus aplicaciones y plataformas; sobre estas últimas tendría el control pero no sobre las infraestructuras totales del proveedor.

Además, la infraestructura puede ser dinámicamente escalable, basada en los recursos que la aplicación necesita. Múltiples usuarios pueden estar compartiendo recursos en el equipo al mismo tiempo [5].

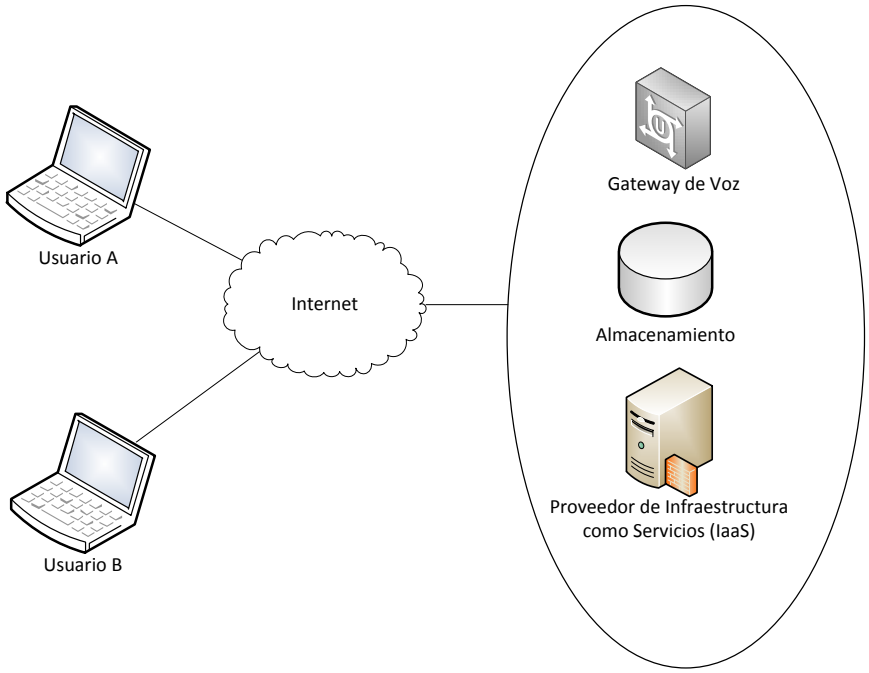


Figura 2.4. Esquema de *Infraestructura como Servicio* (*IaaS*) en la Nube. [5]

2.4 MODELOS DE DESPLIEGUE EN LA NUBE

Según la National Institute of Standards and Technology de EE.UU. dependiendo del tipo de despliegue en la Nube, la misma puede tener recursos informáticos limitados privados, o puede tener acceso a grandes cantidades de recursos de acceso remoto [7]. Los siguientes modelos de

implementación presentan cómo los clientes pueden controlar sus recursos, su escalabilidad, costo y la disponibilidad de recursos:

1. Nube Pública
2. Nube Privada
3. Nube Hibrida

Estos modelos, comparten aspectos fundamentales de Computación en la Nube entre los que se destacan que utilizan dispositivos conectados a Internet, proveen escalamiento dinámico para los recursos virtualizados y generalmente los usuarios no tienen control sobre la tecnología utilizada.

2.4.1 Nube Pública

La Nube Pública es un esquema de implementación de Computación en la Nube, generalmente abierta para el uso del público en general y esta infraestructura es propiedad de una organización que ofrece servicios en la Nube [6], como se aprecia en la Figura 2.5.

Por lo general, la Nube es operada y administrada en un centro de datos de propiedad de un proveedor de servicios que aloja varios clientes y utiliza el aprovisionamiento dinámico. La implementación de una plataforma de servicios escalable y el pago de servicios bajo demanda, son elementos atractivos en una Nube Pública, así como las ventajas de compartir

infraestructura de *hardware* y de *software*, y por ende el desarrollo, el mantenimiento y las actualizaciones. Desde el punto de vista económico, utilizar una Nube Pública puede proporcionar ahorros de costos casi de inmediato, por la eliminación de la carga del mantenimiento de la infraestructura de *IT*.

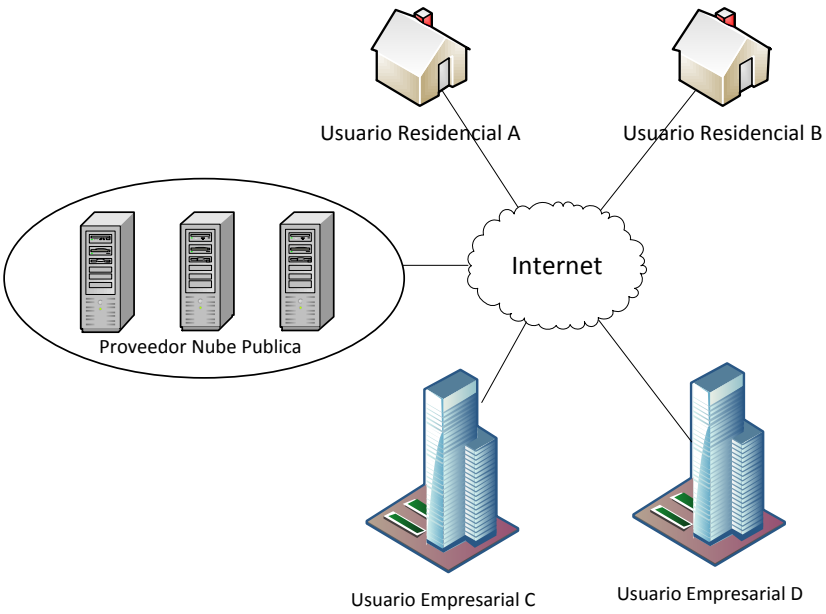


Figura 2.5. Esquema de un proveedor de servicios de Nube Pública. [6]

2.4.2 Nube Privada

Con el uso de la virtualización, algunas empresas están construyendo Computación en la Nube, en ambientes destinados a ser utilizados solamente por sus empleados o socios designados convirtiendo estos en

espacios privados [6]. El nivel de desarrollo de las Nubes Privadas ha llegado a un nivel elevado puesto que sistemas operativos abiertos como Ubuntu ofrecen la posibilidad de instalar este tipo de servicios.

Las Nubes Privadas pueden ofrecer los beneficios de una Nube Pública, al tiempo que permite a la organización mantener un mayor control sobre sus datos y procesos.

Por lo tanto se puede describir una Nube Privada como una infraestructura en la Nube operada solamente por una organización, gestionado por la organización o de un tercero y existente ya sea en las instalaciones o fuera de ella, aunque la Nube Privada normalmente se aloja en los límites de la organización propietaria, como se aprecia en la Figura 2.6.

Difieren principalmente de las Nubes Públicas en el que la infraestructura asociada a la Nube no es compartida con alguna otra empresa o entidad.

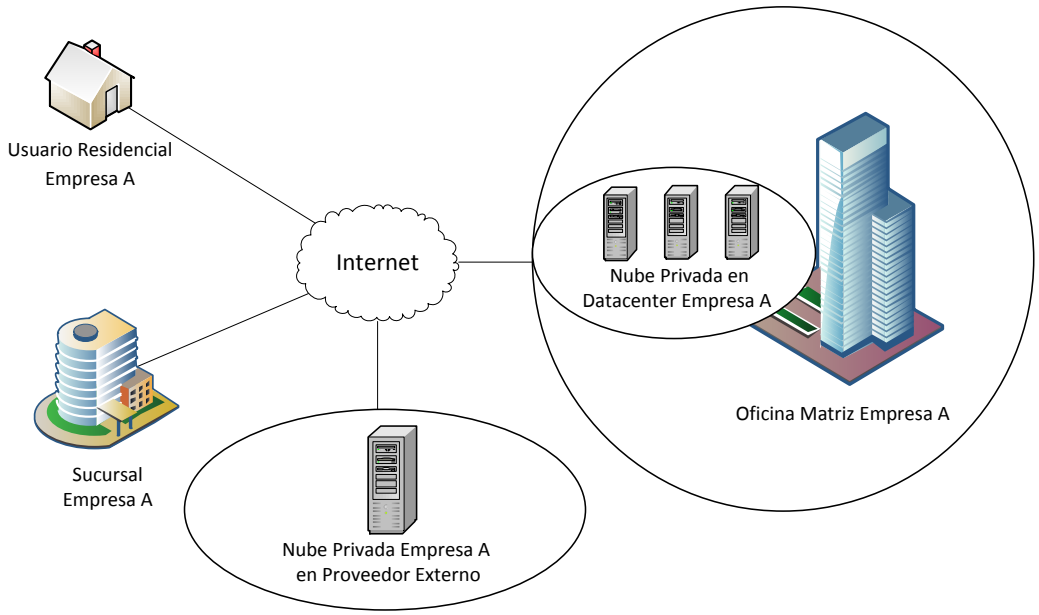


Figura 2.6. Esquema de una empresa que ofrece servicios de Nube Privada. [6]

2.4.3 Nube Hibrida

Las Nubes Híbridas combinan los modelos de Nubes Públicas y Privadas [4]. Aunque las Nubes retienen sus identidades únicas, sin embargo, funcionan como una unidad (la visión del usuario es integral y sólo “ve” una Nube). Una Nube Híbrida puede ofrecer un acceso estandarizado o propietario a los datos y aplicaciones, así como portabilidad de las aplicaciones. Es por ello que el usuario es propietario de una parte de la infraestructura pero comparte otras, aunque de una manera controlada [6], como se muestra en la Figura 2.7.

La Nube Híbrida puede ser un buen paso intermedio antes de pasar la mayor parte de las aplicaciones a la Nube Pública, ya que es algo menos

arriesgado. Por lo tanto, sería interesante pasar algunas aplicaciones no críticas y útiles para la Nube Pública y en el momento que se esté más cómodo, mover las que sean necesarias o restantes [6]. Una Nube Híbrida tiene la ventaja de una inversión inicial más moderada y a la vez contar con *SaaS*, *PaaS* o *IaaS* bajo demanda.

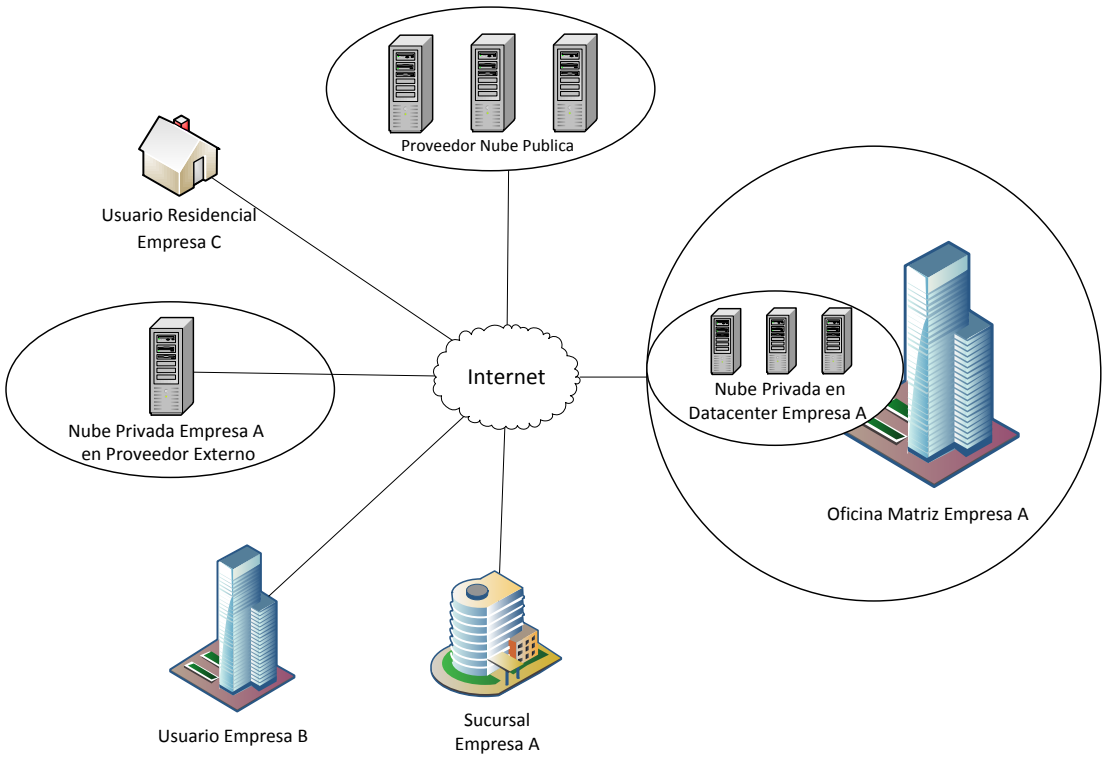


Figura 2.7. Esquema de una empresa que ofrece servicios de Nube Privada y utiliza servicios de Nube Publica. [6]

2.5 ARQUITECTURA GENERAL DE COMPUTACIÓN EN LA NUBE

La arquitectura de la Computación en la Nube consta de una colección de servidores que son accesibles vía internet generalmente [8]. Por tanto los usuarios se conectan a la Nube por medio de sus computadores personales o dispositivos portátiles. Para los usuarios, la Nube es vista como una aplicación particular, un dispositivo o un documento, por lo tanto el *hardware* para ellos es invisible, normalmente se esquematiza como una nube, como se observa en la Figura 2.8.

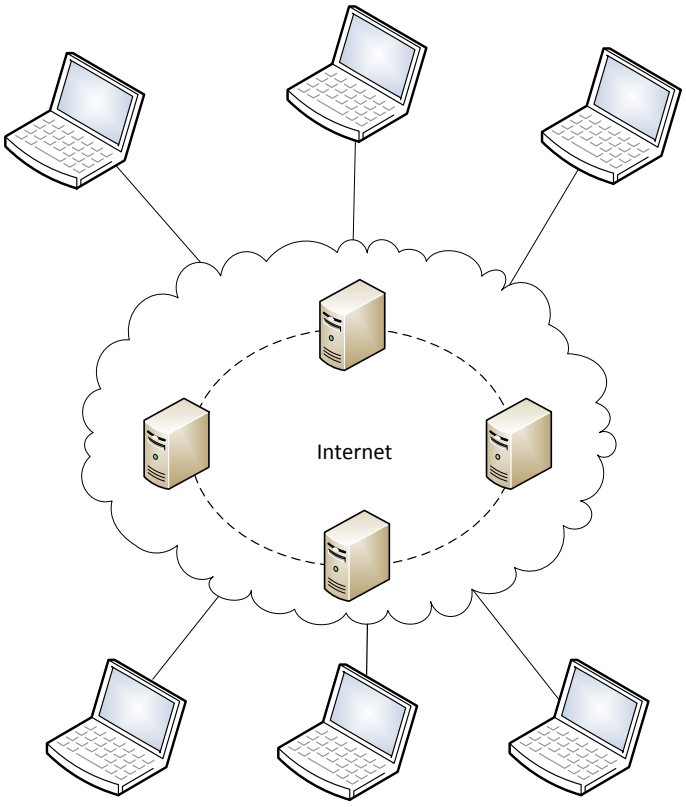


Figura 2.8. La Nube desde el punto de vista del Usuario. [8]

En la Figura 2.9 la arquitectura en la Nube consta además de una Interfaz para el usuario, donde selecciona una tarea o un servicio. La petición del usuario, entonces se pasa a la Gestión del Sistema, que encuentra los recursos correctos y luego llama al sistema de Servicios de Aprovisionamiento adecuados. Estos servicios buscan los recursos necesarios en la Nube, y después se crea o se abre el servicio solicitado desde los Servidores en la Nube.

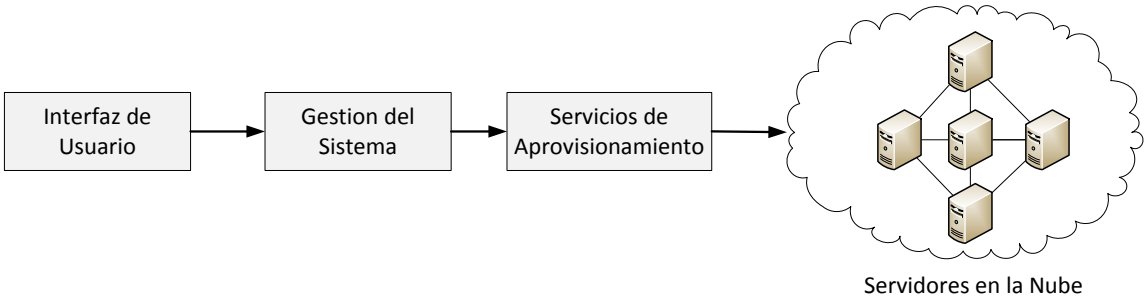


Figura 2.9. Arquitectura esquematizada de un sistema de Computación en la Nube. [8]

Por lo tanto en un sentido estructural, una solución de Computación en la Nube se compone de varios elementos: Clientes, Centro de Datos, y Servidores Distribuidos [5], como se muestra en la Figura 2.10.

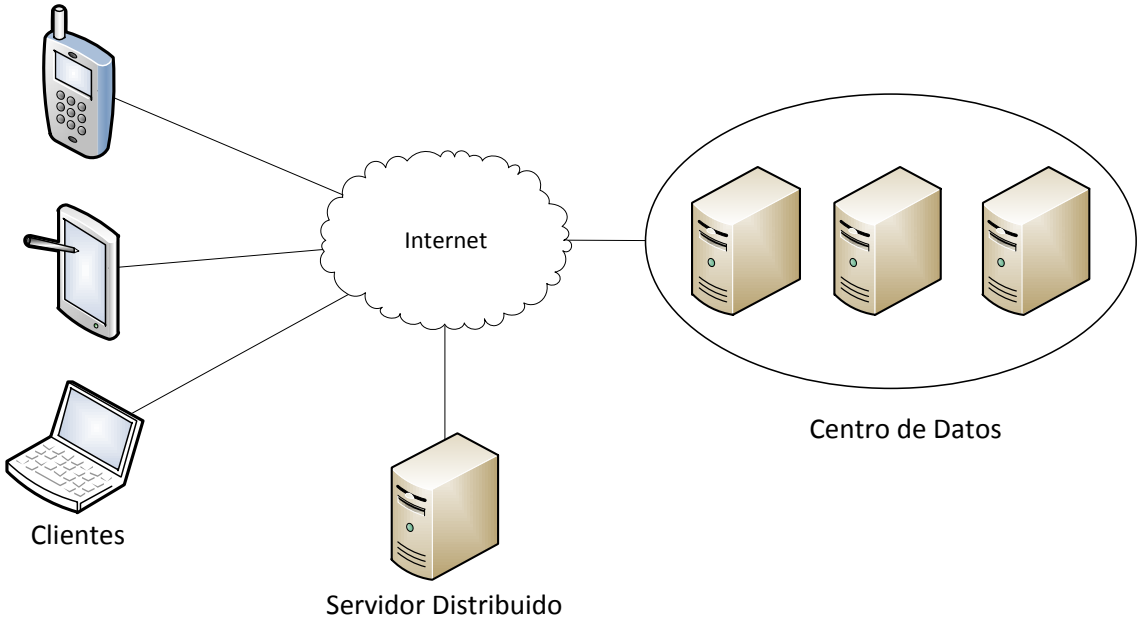


Figura 2.10. Componentes del servicio de Computación en la Nube. [5]

El escenario típico de una empresa es que los Clientes son usuarios de Computación en la Nube, los elementos para conectarse se encuentran normalmente en una *LAN*. Ellos son, típicamente computadores de escritorio, computadores portátiles, tabletas, teléfonos móviles inteligentes.

El Centro de Datos es una colección de servidores, en el que se alojan las aplicaciones a las que se suscribe el usuario. Un aspecto a considerar es que existe una tendencia creciente en el mundo de *IT* a la virtualización de servidores [8], es decir que el *software* puede ser instalado permitiendo que se utilicen varias instancias de servidores. De esta manera se puede tener

algunos servidores virtuales que se ejecutan en un solo servidor físico, como se muestra en la Figura 2.11.

Figura 2.11. Virtualización de Servidores. [5]

Los Servidores Distribuidos nacen de la idea, de que no todos los servidores están en la misma sede, sino dispersos geográficamente. Pero para el usuario es como si todos estuvieran juntos, lo cual crea una gran ventaja, sobre todo para previsión de desastres lo cual permite a los servicios estar siempre operativos.

2.6 BENEFICIOS Y LIMITACIONES DE COMPUTACION BASADA EN LA NUBE

Entre los beneficios de la utilización de Computación en la Nube podemos considerar:

1. Trae menos costos para los usuarios, dado que no necesitan de computadores con gran capacidad de procesamiento y almacenamiento, para poder ejecutar aplicaciones desde la Nube, y más bien utilizar esta capacidad libre en sus propias aplicaciones y/o almacenamiento lo que conlleva en mayor velocidad para el usuario.
2. La reducción de costos para el usuario dado que no necesita invertir y mantener grandes cantidades de servidores, sino los mínimos necesarios para que brinden la conectividad hacia la Nube. El valor añadido de esto es el de ahorro del consumo de energía de los servidores.
3. Los usuarios no tienen la difícil decisión de utilizar *software* obsoleto o altos costos de actualización, sino que como las aplicaciones se utilizan a través de interfaces Web principalmente, las actualizaciones ocurren la próxima vez que el usuario realiza un “Inicio de Sesión” en la Nube.
4. Los usuarios disponen de capacidad de almacenamiento ilimitada, mayor interoperabilidad de equipos de diferentes marcas y/o sistemas

operativos; aumento de la disponibilidad de la información, ya que la misma no se encuentra en el disco duro del equipo, sino que reposa en servidores que seguramente cuentan con los más altos estándares de respaldos y duplicaciones de información.

5. El acceso ubicuo a la información y/o aplicaciones desde cualquier lugar que cuente con una conexión a Internet
6. Mayor capacidad de procesamiento (dado que ésta no depende solo del procesador del equipo sino que posee la capacidad de procesamiento de toda la Nube) y esto da la posibilidad de realizar "súper-cálculos".

Entre las limitaciones o desventajas, se puede considerar:

1. Para acceder a aplicaciones en la Nube, se requiere una conexión a Internet constante y de banda ancha, las cuales no están siempre disponibles en todos los lugares. Por ello también al tener que hacer descargas constantes, se podría tener la impresión que es más lento que trabajar con las aplicaciones o documentos almacenados directamente en el equipo.
2. Los servicios adicionales o especializados que dan las aplicaciones en la red pueden ser más básicos que las aplicaciones instaladas directamente en los equipos, como por ejemplo Microsoft Word ofrece mayores servicios que Google Drive.

3. La disponibilidad del servicio, la seguridad y confidencialidad de la información, las cuales son discutidas en la sección 2.6.1 son muy cuestionadas.

2.6.1 Disponibilidad del Servicio

Este aspecto es importante para la relación entre los elementos de procesamiento y los datos en los que se operan. Dado que la mayoría de equipos con servicios en la Nube almacenan datos en la Nube, y no en los discos locales, se necesita de un tiempo de comunicación para que los datos sean llevados a un servidor o viceversa [9]. Es así que los datos físicos cumplen con una simple ecuación, la cual indica el tiempo de transmisión de la información.

$$tiempo = \frac{bytes*8}{tasa\ de\ bits} \qquad (2.1)$$

De la ecuación se puede concluir que si la cantidad de información proporcionalmente es mucho mayor a la tasa de bits disponible, el tiempo para contar con la aplicación o los datos disponibles en el equipo sería tan alto, que las aplicaciones en la Nube, no serían recomendables para este caso.

Por otro lado los usuarios a pesar de no tener control sobre la infraestructura de la Nube, es necesario para ellos asegurar la calidad, disponibilidad,

confiabilidad y rendimiento de estos recursos cuando los usuarios han migrado sus aplicaciones o archivos a la Nube [10]. En otras palabras, es vital para los usuarios obtener garantías de los proveedores en el servicio de entrega. Normalmente, estos se proporcionan a través de Acuerdos de Niveles de Servicio (del inglés SLA: *Service Level Agreement*) negociados entre los proveedores y usuarios.

La dependencia de un tercero es otro aspecto de las limitaciones del servicio, porque si el lugar donde estas almacenados los datos en la Nube sufre de algún problema o desastre, simplemente no se podría acceder a los datos, lo cual limita la independencia del usuario. Es por ello que el proveedor de la Nube, debe ofrecer esquemas de redundancia interna y geográfica, para aumentar la disponibilidad de sus servicios.

2.6.2 Seguridad y Confidencialidad de la Información

Los datos son considerados tan o más importantes que el resto de los activos de una empresa o de un usuario, y por lo tanto deben estar muy protegidos. Es fácil argumentar que una alta seguridad es necesaria para proteger los datos por la forma en que un intruso puede potencialmente llegar a ellos desde cualquier lugar en Internet. Algunos pasos son [6]:

1. Cifrar los datos en reposo, de manera que si algún intruso es capaz de penetrar la seguridad del proveedor de la Nube, o si un error de

configuración hace que los datos accesibles a personas no autorizadas, los datos no puedan ser interpretados.

2. Cifrar los datos en tránsito, para que los datos que pasan por infraestructuras públicas no puedan ser observados por otros en el medio.
3. Requerir fuerte autenticación entre los componentes de aplicaciones para que los datos transmitidos únicamente sean conocidos por las partes.

CAPÍTULO 3

VoIP EN LA NUBE

3.1 INTRODUCCIÓN

La *VoIP* en la Nube se refiere a la tecnología de procesamiento y señalización de llamadas que están basadas en familias de protocolos estándares que proporcionan servicios de comunicación de voz. Estos estándares han sido definidos principalmente por la *Internet Engineering Task Force* (*IETF*) y la *Unión Internacional de Telecomunicaciones* (*UIT*), para proveer servicios interoperables de comunicación entre redes telefónicas clásicas (como por ejemplo la *Red de Telefonía Básica* (*RTB*)) e Internet [11]. Entre los protocolos de señalización, establecimiento, control y

gestión de las sesiones de voz que se usan están: H.323, *SIP*, *MGCP*, IAX, entre otros.

La *VoIP* en la Nube es un servicio de *VoIP* tradicional que incluye un nuevo conjunto de protocolos basados en la Web que persigue ayudar a sus usuarios con mayor flexibilidad y facilidad de implementación, pudiendo en ciertos casos a ser más rentables y/o eficientes. La idea básica es que el usuario final únicamente marca un número telefónico para conversar con su homólogo distante. Se persigue que este proceso de marcado sea lo más eficaz y rápido posible. De esta manera el usuario únicamente dispondría de los terminales de comunicación necesarios para realizar las llamadas y no dispondría de infraestructuras típicas que conlleva la *VoIP*. Esto para algunas empresas puede suponer un ahorro en costes y otras ventajas como las analizadas en el capítulo 2.

En el siguiente apartado presentamos una evolución desde la telefonía clásica hasta una visión general de la *VoIP* en la Nube, partiendo de supuestos realistas del estado actual de la telefonía.

3.2 FORTALEZAS Y DEBILIDADES FRENTE AL MODELO CLÁSICO DE VOIP Y AL DE TELEFONIA TRADICIONAL

En el modelo tradicional, los teléfonos se conectan a las oficinas centrales (*CO*, del inglés, *Central Office*) de las compañías telefónicas, sobre una línea de 2 hilos de cobre [12]. La corriente eléctrica para operar los teléfonos les llega desde una batería en el *CO*, es así que el teléfono no necesita energía adicional. Esto fue así desde sus inicios porque los hogares u oficinas no contaban con electricidad propia. Para la operación de los teléfonos, es necesario que la electricidad fluya en un lazo de extremo a extremo donde están conectados ambos teléfonos. En el modelo más básico de telefonía convencional la porción de la conexión entre el usuario y el *CO* se llama bucle de abonado. Luego de la aparición de las conexiones digitales y sumada al tipo de conmutación por circuitos automática, donde la latencia y la cabecera es fija, lo que brinda una capacidad o canal dedicado por llamada, como se aprecia en la Figura 3.1, lo cual previene la interferencia entre los usuarios, dando como resultado una de las mayores ventajas inherentes de este tipo de tecnología, mejor calidad de la voz recibida.

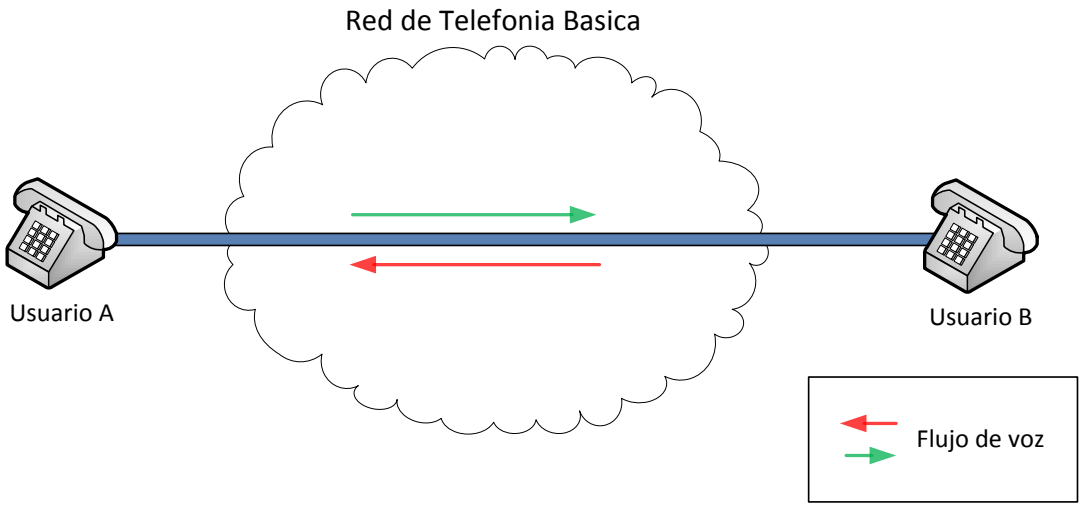


Figura 3.1. Canal dedicado para llamadas en conmutación por circuitos de *RTB*

La *VoIP* es un estándar posterior en las comunicaciones de voz que ofrece mayores funcionalidades que los sistemas de telefonía tradicional, como son una mejor gestión y control de las llamadas.

Para centrar el tema de la *VoIP* en la Nube, en primer lugar planteamos un conjunto de escenarios diferentes posibles de uso de la *VoIP* desde el punto de vista del usuario final:

1. *Hogar*. Estas implantaciones típicamente utilizan un *softphone* para registrarse en un servidor de *VoIP* alojado en Internet y utilizar sus servicios (proveedor de servicios de *VoIP*). El tipo de conmutación que se usaría en este caso, generalmente, es conmutación por paquetes de Internet siempre que se llame a otro *softphone* o teléfono

IP (Protocolo de Internet, del inglés, *Internet Protocol*), considerando también que si se llama a un teléfono clásico la llamada puede cruzar por la red telefónica apropiada. En particular pueden existir hogares en los que se utilice el teléfono clásico para realizar llamadas de *VoIP*: en ese caso normalmente están conectados mediante algún adaptador del teléfono a los encaminadores que permiten al hogar disponer de una conexión a Internet. En casos muy concretos pueden existir hogares que dispongan de un servidor de *VoIP*.

2. *Usuarios en movilidad*. Normalmente los usuarios que disponen de teléfonos móviles inteligentes hacen uso de la telefonía *IP* (las llamadas sólo cruzan Internet) o bien hacen llamadas *VoIP*. En este último caso el servidor de *VoIP* está en las instalaciones de un proveedor de servicios de *VoIP*.
3. *Empresarial*. Los equipos de conmutación o procesamiento generalmente se encuentran en las instalaciones de los usuarios, y están conectados directamente con los teléfonos *IP* a través de *LAN*. Las sedes remotas pueden conectarse directamente a través de Internet, utilizando en este caso conmutación por paquetes, por lo cual los paquetes no necesariamente poseen un camino especifico entre los actores de llamada (llamante y llamado), como se muestra en la Figura 3.2, lo cual tiene ventajas como es la conmutación por fallo (del

inglés, *failover*) si las líneas de comunicación principales fallan, pero presenta desventajas como el *jitter* y retraso indeterminado.

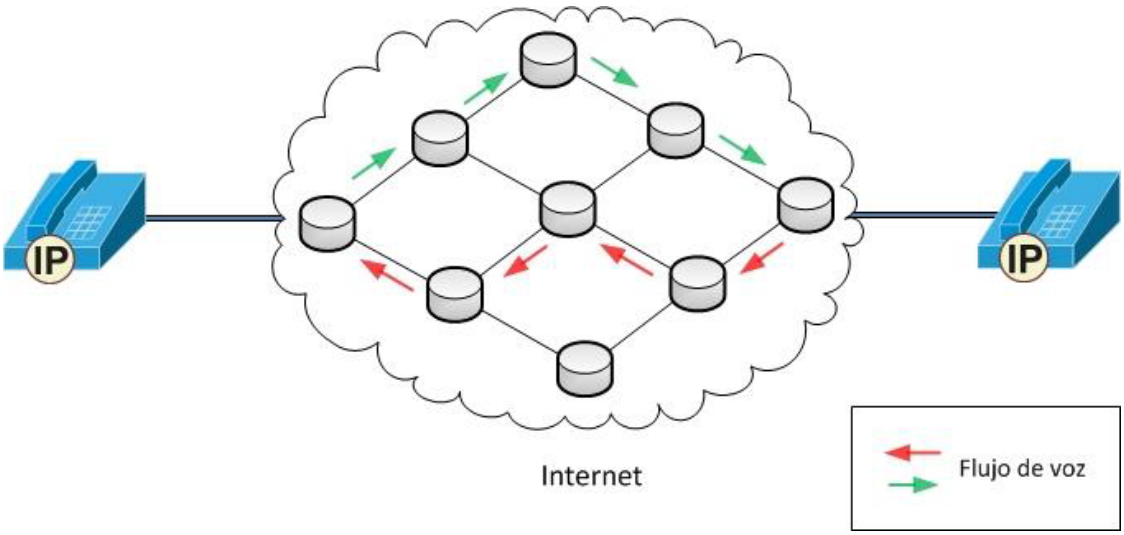


Figura 3.2. Los paquetes de voz pueden tomar caminos diferentes en la conmutación por paquetes

En un escenario empresarial, otra característica importante de las implementaciones de *VoIP* es que requieren de inversiones económicas considerables en infraestructura o *CAPEX* (Inversiones en bienes de capitales, del inglés, *CAPital EXpenditures*), así como los gastos recurrentes de servicios de telecomunicaciones y de personal con experiencia para administrarlas y mantenerlas o *OPEX* (Gastos operacionales, del inglés *OPerational EXpenditures)*, por lo cual una migración desde la telefonía convencional a *VoIP* necesita estudios de viabilidad óptimos.

Un aspecto también a considerar es que sin importar si la telefonía es convencional o *VoIP*, o si la conmutación es por circuitos o paquetes, siempre van a existir dos planos necesarios e independientes entre sí, el de señalización y el del transporte de la voz. Sobre todo en el plano de señalización se debe mencionar que hoy en día, es difícil encontrar redes que utilicen un solo tipo de tecnología entre los extremos de los actores de las llamadas; esta señalización depende del tipo de red, por la que está siendo transportada.

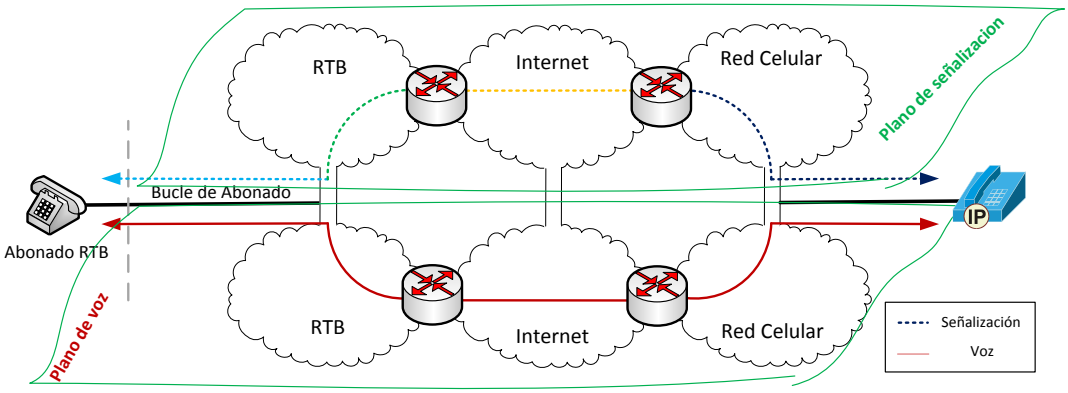


Figura 3.3. Esquema general de interconexión de redes y planos de señalización y voz

Por ejemplo en la Figura 3.3 se muestra un abonado a la *RTB* y su señalización de acceso a la red sería del tipo para bucle de abonado, una vez en la *RTB* cambiaría a SS7, después podría haber una conexión a Internet donde la señalización se convertiría en *SIGTRAN* (Señalización de

Transporte, del inglés, *Signaling Transport*), a continuación, podría haber una red celular para acceso para el otro actor de la llamada, convirtiéndose la señalización al tipo de red donde es transportada y terminar en una *LAN*, donde migraría a señalización *IP*.

Mientras que la voz viaja por los diferentes tipos de redes, adaptándose al transporte según su necesidad, pudiendo comunicarse en un plano completamente diferente al de la señalización.

En el presente estudio, sin pérdida de generalidad, no se va a considerar el cambio de señalización entre el paso por diferentes tipos de redes, sino la conexión entre el llamante y el llamado por un solo tipo de red, o a dos redes en forma paralela. Entonces en relación a lo expuesto en estas consideraciones, se presentan dos modelos para procesamientos de llamadas en las redes para *VoIP*, los cuales son el modelo distribuido y centralizado [13], este modelo último sirve como base para un modelo de procesamiento de llamadas en la Nube.

3.2.1 Modelo distribuido para procesamiento de llamadas

Actualmente, en la práctica existen configuraciones de *VoIP* en el escenario de empresas en las que se dispone de varios equipos de procesamiento de la voz en distintas ubicaciones (sedes). En el modelo distribuido las sedes se interconectan por medio de Internet principalmente, y el proceso de

llamadas se realiza en cada sede de manera independiente, es decir cuentan con equipos, aplicaciones y demás recursos necesarios que les permitan completar las llamadas, por tanto deberán contar con conexión a la *RTB* para la comunicación con los sistemas telefónicos convencionales.

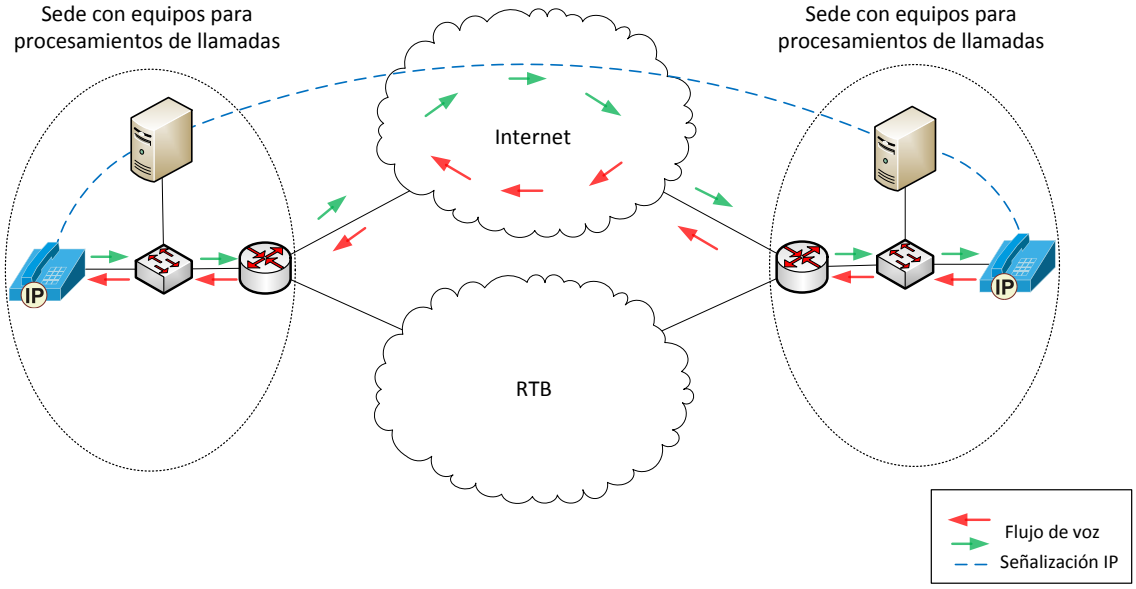


Figura 3.4. Modelo distribuido para procesamiento de llamadas

En la Figura 3.4 se muestra, que en cada sede existe señalización *IP* que depende del tipo de protocolo utilizado para *VoIP*, ilustrado por medio de líneas intermitentes, y también el flujo de voz que en Internet, se lo muestra con líneas independientes de color rojo y verde, representan la voz encapsulada en paquetes *IP* y que pueden tomar caminos distintos. En la

misma Figura 3.4 por facilidad se indica que la señalización, existente entre los equipos para procesamiento de llamadas en ambas sedes, tiene camino directo sin pasar por los encaminadores. Aunque en la realidad estos paquetes con señalización, toman el camino de un paquete normal *IP*. En las figuras siguientes no se considera el camino estricto de los paquetes de señalización *IP* sino solamente se grafican los equipos que intervienen directamente para poder realizar la comunicación. En este mismo modelo podemos comunicar dos sedes diferentes por medio de la conmutación por fallo a través de la *RTB* (si es necesario el caso), la cual tiene su propio sistema de señalización, que es independiente y diferente al utilizado en Internet. Además, el flujo de voz en la *RTB* lo representamos mediante un flujo constante (con líneas verde y rojas) que simbolizan al circuito dedicado temporal característico de este tipo de red, tal como muestra en la Figura 3.5.

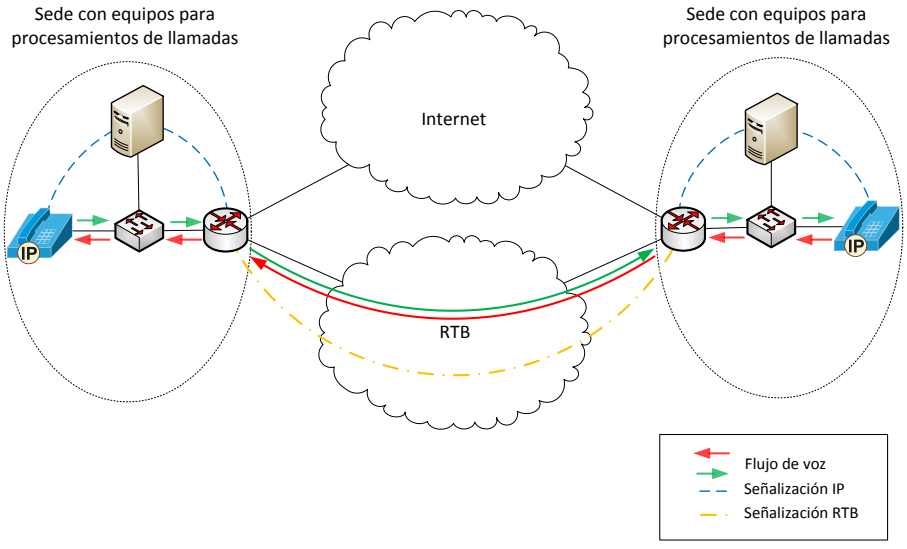


Figura 3.5. Conmutación por fallo en el modelo distribuido de procesamiento de llamadas

Nótese que mientras en la Figura 3.4 la señalización *IP* se hacen dentro de las sedes y a lo largo y ancho de Internet, en la Figura 3.5 la señalización *IP* sólo se lleva a cabo dentro de las sedes.

Otra de las ventajas de *VoIP* y sin importar que utilicemos un modelo centralizado o distribuido, es poder realizar y recibir llamadas sin importar su ubicación geográfica, por lo tanto se pueden realizar llamadas de larga distancia a costo de una llamada local, y es esta flexibilidad una de la razones por lo que *VoIP* ha tenido tanto éxito a nivel mundial [13], por poder utilizar ubicaciones *IP* más cercanas a nuestro destino, para que el tramo en la *RTB* sea el mínimo posible. A este proceso se lo conoce como "*Toll-by-pass*", pero para su aplicación habría que revisar las leyes regulatorias en cada país. Un ejemplo se muestra en la Figura 3.6.

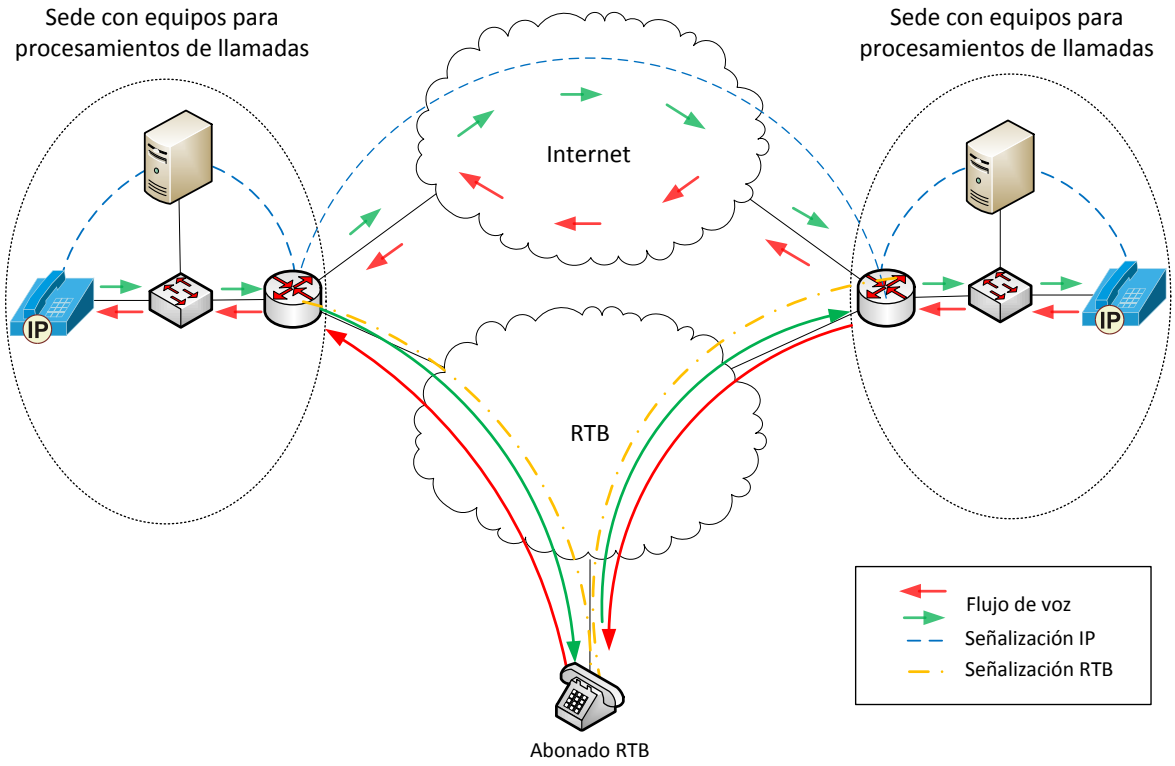


Figura 3.6. Capacidad de comunicaciones en cualquier dirección en modelo distribuido

3.2.2 Modelo centralizado para procesamiento de llamadas

En un escenario donde un usuario final disponga del equipamiento de procesamiento de *VoIP* en su hogar, o un usuario en movilidad (el equipamiento de procesamiento de *VoIP* está en el operador de telefonía móvil o en una empresa contratada por ella) o una empresa que disponga de equipos en una única sede surge el modelo centralizado. Este modelo consiste en una oficina o sede central con capacidad de procesamiento de llamadas y que utiliza Internet para transportar tráfico de voz y señalización a

sedes que dependen de ella para poder comunicarse [13], como se muestra en la Figura 3.7. Nótese la diferencia fundamental entre este modelo y el distribuido observando las Figuras 3.4 y 3.7: en la primera ambas sedes disponen de equipamiento para procesar las llamadas, mientras que en la segunda sólo la sede de la parte izquierda de la figura es la que posee equipamiento para procesar las llamadas. Esto implica que en el modelo centralizado siempre que se requiera procesar una llamada se debe acudir a la sede que dispone de equipamiento para procesar la llamada.

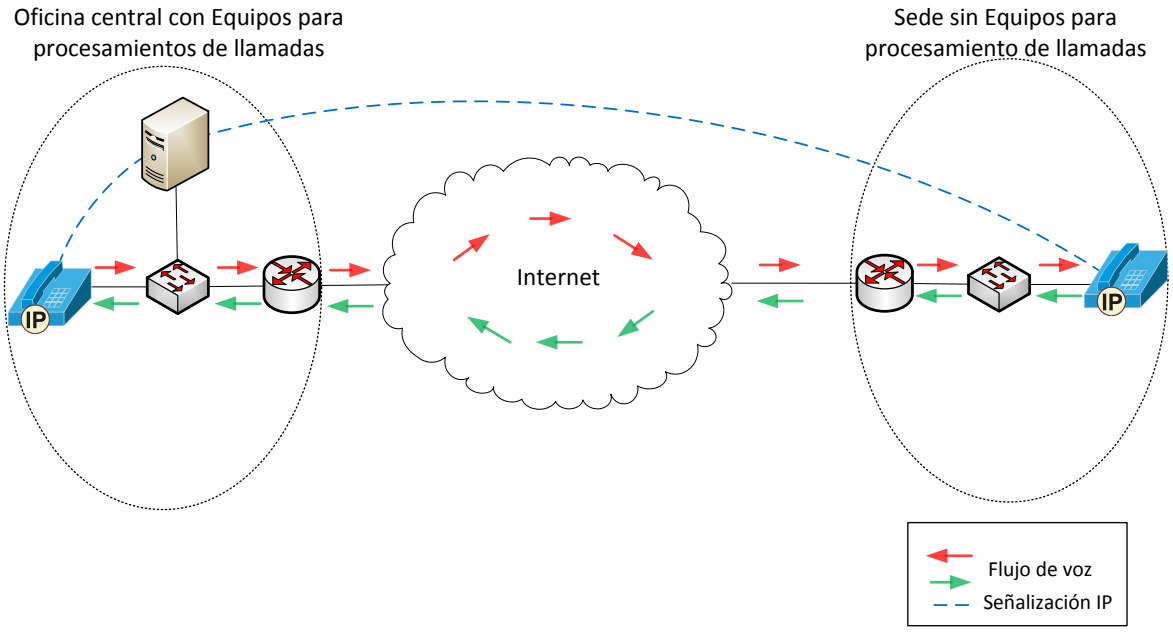


Figura 3.7. Modelo centralizado para procesamiento de llamadas

El modelo centralizado nos permite también flexibilidad para el procesamiento de llamadas, como lo realiza el modelo distribuido, sin importar su distancia geográfica, por medio de una conexión a Internet, se puedan realizar llamadas entre sedes, e incluso a través de la oficina central desde y hacia la *RTB* como se muestra en la Figura 3.8, pero se debe considerar que las sedes remotas pierden su total independencia en la gestión de llamadas, al depender de una oficina central para realizarlas [13].

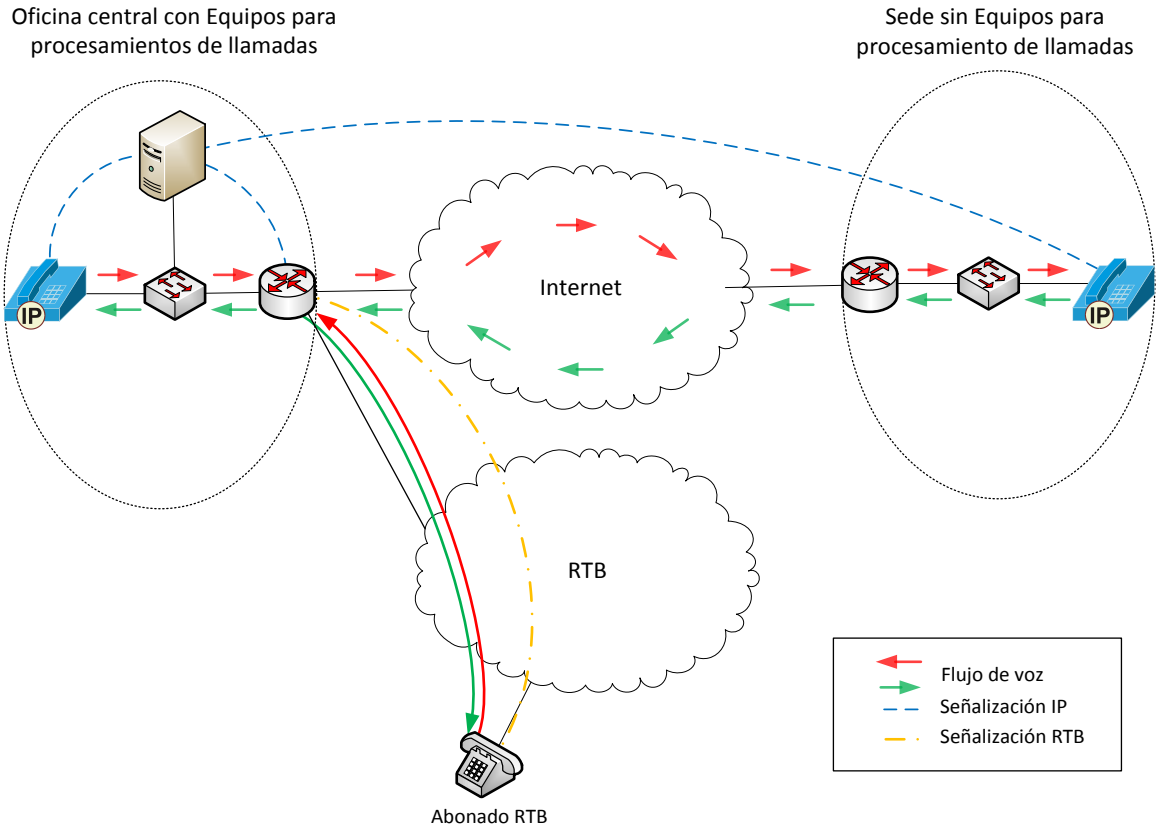


Figura 3.8. Capacidad de comunicaciones en cualquier dirección en modelo centralizado

Otro punto a considerar es que este tipo de modelo también estaría en capacidad de soportar una conexión directa con la *RTB* en las sedes remotas a pesar de no contar con equipos de procesamiento de llamadas propios, pero los teléfonos *IP* o cualquier otro dispositivo de comunicación necesita establecer una comunicación de señalización primero con la sede central para poder establecer comunicación con Internet o la *RTB* local, como es el caso de la Figura 3.9.

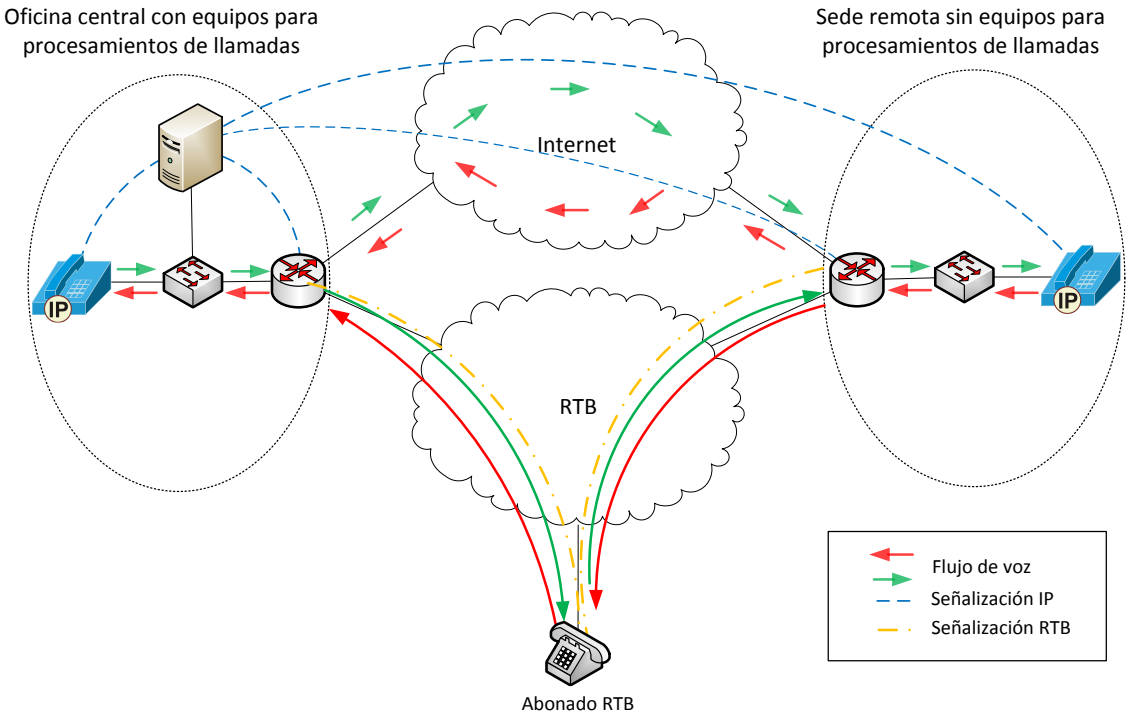


Figura 3.9. Capacidad de comunicaciones en cualquier dirección en un modelo centralizado

3.2.3 Modelo para procesamiento de llamadas en la Nube

El modelo centralizado para procesamiento de llamadas de la Figura 3.8, es el que da nacimiento a lo que conocemos como *VoIP* en la Nube, el mismo que es básicamente un escenario en el que los equipos conmutadores del sistema telefónico no se encuentran en el mismo lugar de los teléfonos u otros dispositivos de comunicación del usuario o suscriptor, sino que normalmente se encuentran fuera de las dependencias, es decir en un proveedor de servicios externo. Y los teléfonos u otros dispositivos del usuario, se comunican con los equipos conmutadores y de procesamiento de llamadas a través de Internet, como se aprecia en la Figura 3.10.

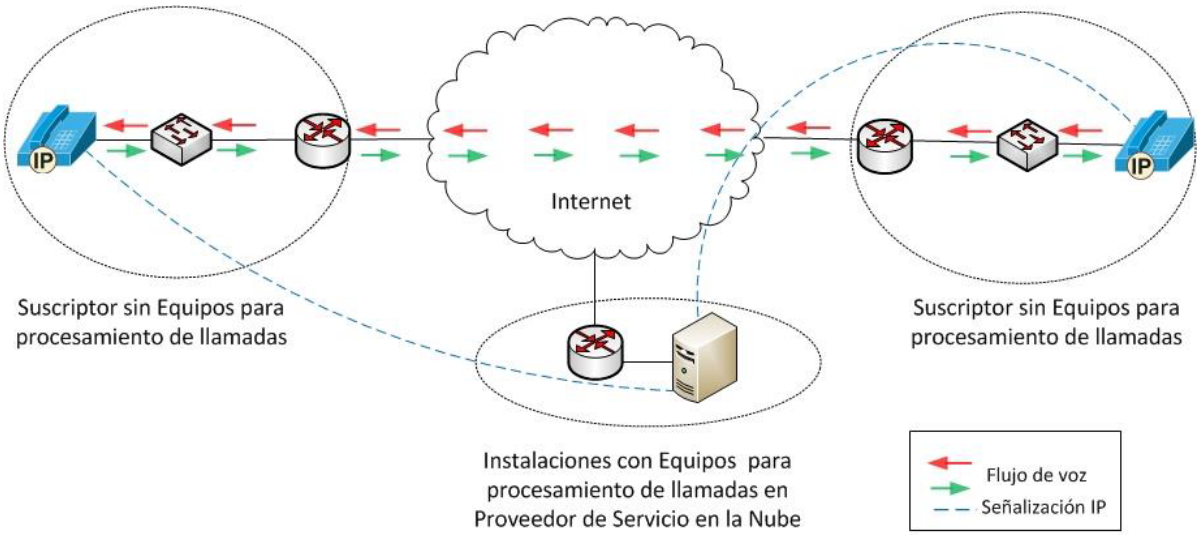


Figura 3.10. Modelo básico de *VoIP* en la Nube para llamadas en Internet

Una de las mayores ventajas de este tipo de modelo se da, en que los usuarios no necesitan una conexión directa a la *RTB* para acceder llamadas a teléfonos clásicos, sino que lo hacen a través de un proveedor externo por medio de la Nube de Internet, como se aprecia en la Figura 3.11. Por lo que ofrece ventajas de uso y configuración para los usuarios, escalabilidad bajo demanda, y minimización los costos iniciales de arranque (*CAPEX*), aunque se debe considerar que los *OPEX* dependen de un análisis comparativo de tarifas y de situación inicial, para establecer si son mayores o menores que un modelo *VoIP* tradicional. Nótese que este es el único modelo válido para usuarios en movilidad que no dispongan de infraestructura necesaria para instalar sus propios equipos de procesamiento de *VoIP*.

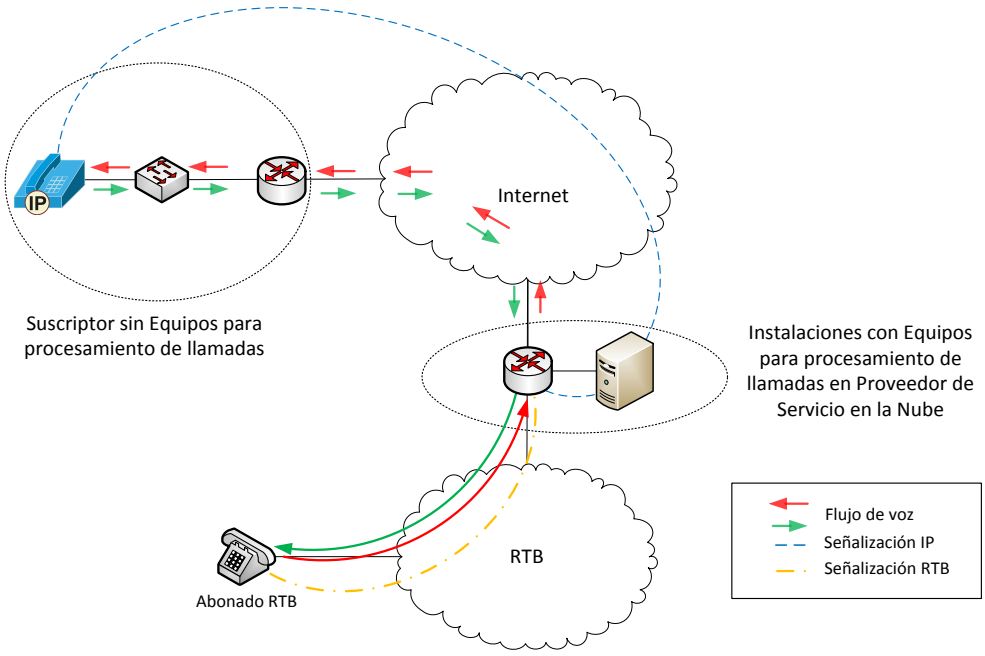


Figura 3.11. Modelo Básico de *VoIP* en la Nube para llamadas desde y hacia la *RTB*

Entre las desventajas de este tipo de tecnología se encuentran que posee muy pocas opciones de selección para los planes de llamadas y costos por minuto, así como existe poco control sobre el registro de llamadas, una dependencia total con el proveedor del servicio hasta la culminación del contrato de prestación, y no existen opciones de conmutación por fallo propias sino las exclusivas del proveedor.

Así mismo los proveedores de servicios cobran una tarifa mensual por su servicio de *VoIP* en la Nube, la cual depende del número líneas contratadas. La misma que puede variar grandemente sobre la base de las características y funcionalidades que se elijan, especialmente si se decide agregar tecnologías como mensajería unificadas y/o videoconferencia.

A lo expuesto se pueden comparar las características de los tres tipos de tecnología para telefonía, en la Tabla 1.

	Telefonía Tradicional	VoIP	VoIP en la Nube
Conmutación	Circuitos	Paquetes	Paquetes
Gestión de llamadas	Independiente	Independiente/Dependiente	Dependiente
Personalización Solución	Alta	Alta	Baja
Escalabilidad	Baja	Media	Alta
Calidad Voz	Alta	Media	Media
Tasa de bits	Fija 64 kbps	Variable 16-128 Kbps	Variable 16-128 Kbps
Estándares	Bien definidos	No existe un estándar definido, sino que compiten entre sí	No existe un estándar definido, sino que compiten entre sí
Servicios	Tradicionales (llamadas, buzones de voz, caller ID)	Tradicionales + Adicionales (Mensajería Instantánea, Video, etc)	Tradicionales + Adicionales (Mensajería Instantánea, Video, etc)
Infraestructura	Dedicada para Telefonía	Compartida con Datos	Compartida con Datos
Operadores	Usualmente por un solo operador	Múltiples (por ej acceso Internet y VoIP ofrecido por diferentes operadores)	Múltiples (por ej acceso Internet y VoIP ofrecido por diferentes operadores)
Numeracion	Dependiente ubicacion geográfica	Independiente ubicación geográfica	Independiente ubicación geográfica
Números Emergencia	El llamante puede ser ubicado rápidamente	El llamante no puede ser ubicado	El llamante no puede ser ubicado
Flexibilidad	Baja	Media	Alta
Gestión ENUM	-	Independiente	Dependiente
CAPEX	Alto	Alto	Bajo
OPEX	Alto	Medio	Bajo

Tabla 1. Comparación tecnologías de conmutación de llamadas de voz

Entonces desde el punto de vista del usuario, este modelo comparado con la telefonía tradicional y de VoIP clásico, podría suponer un retraso importante en el establecimiento de las llamadas si la sede de procesamiento de las llamadas de *VoIP* estuviera muy distante (en Internet). Si además toda la

señalización, control y gestión de las llamadas se tuviera que llevar a cabo hasta un único punto de Internet podría suponer un cuello de botella muy importante con un nivel de escalabilidad limitado. Por ello es necesario hacer tres últimos apuntes sobre este modelo:

1. *Disminución del retraso de establecimiento de llamadas*: la sede central con el equipamiento de procesamiento de las llamadas se puede distribuir a lo largo y ancho de Internet estableciendo varias sedes en distintos lugares de Internet. Si bien geográficamente estos equipos están distribuidos, lo que es más importante es que su gestión y control se hace centralizadamente como si fuese una única sede. De esta manera se puede acercar geográficamente a cada empresa los equipos de procesamiento de la *VoIP*.
2. *Escalabilidad*: una empresa proveedora de servicios de *VoIP* mediante un modelo en la Nube podría balancear sus recursos entre distintos equipos de procesado de la *VoIP* para ganar en escalabilidad. Esto se puede hacer mediante grandes servidores y virtualización en una única sede (contratando varias líneas de alta velocidad de Internet y líneas telefónicas) o bien alojando dichos equipos en distintos lugares geográficos.
3. *Externalización de la infraestructura*: las empresas que provean un servicio de *VoIP* mediante un modelo centralizado podrían seguir varias estrategias de comercialización para ofrecer servicios en la

Nube: a) *SaaS*: simplemente proveen el servicio alquilando los equipos de procesamiento de la *VoIP* a una empresa externa, b) *IaaS*: pueden proveer sus infraestructuras a une empresa externa para que ella provea el servicio de *VoIP* y c) *PaaS*: la empresa puede incluso proporcionar una plataforma completa de *VoIP* para que otras empresas desarrollen sus servicios personalizadas a otras empresas.

3.3 PROTOCOLOS DE VOIP EN LA NUBE

Para el proceso de establecimiento de una llamada utilizando *VoIP* en la Nube, se necesitan los mismos mecanismos que se utilizan en *VoIP*, es decir los protocolos necesarios para establecer la comunicaciones entre dos puntos serán de señalización, encaminamiento y transporte.

3.3.1 Protocolos de Señalización

La señalización en la telefonía es uno de los aspectos más importantes, ya que es la encargada del establecimiento, terminación, control de las llamadas y ofrecer servicios suplementarios [14], esto es realizado mediante códigos y órdenes propias de cada protocolo de señalización.

Se presentan a continuación los protocolos de señalización *IP* para voz más utilizados los cuales son: H.323, *SIP*, *IAX* y *MGCP*.

H.323

H.323 históricamente es el primer protocolo estandarizado en el que se especifican los componentes y procedimientos para comunicaciones multimedia en tiempo real sobre redes de conmutación de paquetes. Fue ratificado por la *UIT* y es un protocolo de señalización fuera de banda (del inglés, *out-of-band*), es decir los paquetes de voz y señalización se comunican por vías independientes; los paquetes de señalización se comunican con los participantes de la llamadas y los equipos de procesamiento de llamada, y los paquetes de voz se comunican directamente entre los llamantes y llamados, sin tener la necesidad de pasar los equipos de procesado de llamadas, es decir, ambos tipos paquetes pueden tener caminos distintos [15].

Una red H.323 se compone de varias zonas. Cada una se compone de los siguientes elementos [14]:

Terminal: es un dispositivo final del cliente, en el que los flujos de datos y la señalización H.323 se originan y terminan. Un terminal puede ser un teléfono *IP*, un computador o cualquier otro dispositivo con capacidad multimedia para comunicación bidireccional.

Gateway: es un elemento opcional que ofrece servicios de interconexión y traducción entre terminales pertenecientes a la red H.323, con redes que utilizan protocolos diferentes.

Gatekeeper: es un componente muy útil pero también opcional que provee varios servicios como plan de marcación, traducción de direcciones, control de acceso, gestión de ancho de banda, para los dispositivos de su zona de control.

Unidad de control Multipunto (*MCU*, del inglés, *Multipoint Control Unit*): es un dispositivo opcional, y su función básica es proveer servicios de multiconferencia entre terminales o *Gateways*.

Por otra parte H.323 es un conjunto de protocolos, cada uno tiene un rol específico en el proceso de comunicación [14], entre los cuales podemos mencionar:

H.225/RAS (*Registration, Admission and Status*): protocolo que se utiliza para el registro de los terminales en el Gatekeeper, admisión de llamadas, asignación de ancho de banda y el intercambio de mensajes de estado. También se utiliza para la comunicación entre Gatekeepers a través de varias zonas para fines de resolución de direcciones.

H.225: es el componente de señalización, por lo tanto lleva a cabo todas las operaciones para establecer, mantener y terminar una sesión de llamada.

H.245: es el protocolo de control. Los datos de control son necesarios para el intercambio de información sobre las capacidades de cada terminal y manejar la operación de canales lógicos.

En la Figura 3.12 se muestra un esquema de la señalización en H.323 en dos esquemas diferentes.

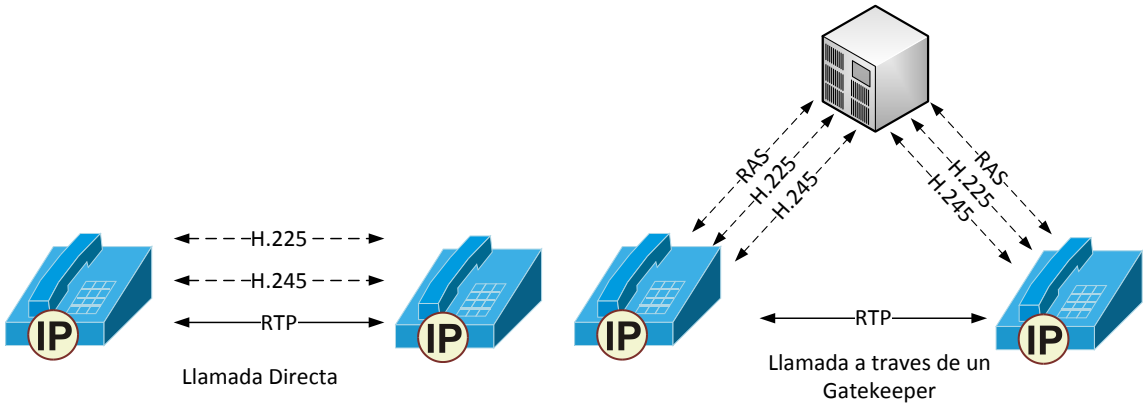


Figura 3.12. Señalización en el Protocolo H.323 en dos modelos de llamadas distintos. [14]

SIP

El Protocolo de Iniciación de Sesión o *SIP* (del inglés, *Session Initiation Protocol*), es un estándar de la IETF, mediante el RFC (Petición de Comentarios, del inglés, *Request for Comments*) 3261 [16]. *SIP* es un protocolo de señalización para iniciar, mantener y terminar las sesiones de llamadas, que deben ser establecidas entre los usuarios. Es un protocolo de

señalización fuera de banda y es una alternativa a otros protocolos como H.323, pero más ligero.

Una red *SIP* se compone básicamente de dos elementos [17]:

Agente Usuario: un dispositivo habilitado para *SIP* que recibe o realiza las sesiones de llamadas, puede ser este un dispositivo final como un teléfono o un *Gateway* conectado a otra red. Puede funcionar como Agente Usuario Cliente (del inglés, *UAC*: *User Agent Client*) que es quien realiza una solicitud de llamada, o un Agente Usuario Servidor (del inglés, *UAS*: *User Agent Server*) quien recibe y responde a esta solicitud. Típicamente un dispositivo final *SIP* puede actuar como *UAS* o *UAC*, pero no ambos a la vez.

Servidor: incluye al Servidor *Proxy*, Servidor de Registro y Servidor de Redireccionamiento y son completamente independientes de los *UAS*. El Servidor *Proxy* es un elemento de la red intermediario que recibe una solicitud de un Agente Usuario u otro Servidor *Proxy* y en su nombre reenvía o atiende a la solicitud. Se diferencia de un Agente Usuario en que no emite solicitud, sino que responde solamente a ellas. El Servidor de Registro es un servidor que procesa solicitudes de autenticación de los *UAC* en el sistema. Y el Servidor de Redireccionamiento que responde a solicitudes, pero no las reenvía a

otros, y utiliza un sistema de búsqueda de direcciones y responde al solicitante la nueva ubicación del llamado.

En la Figura 3.13 se muestran dos ejemplos de establecimiento de sesión para *VoIP* usando *SIP*.

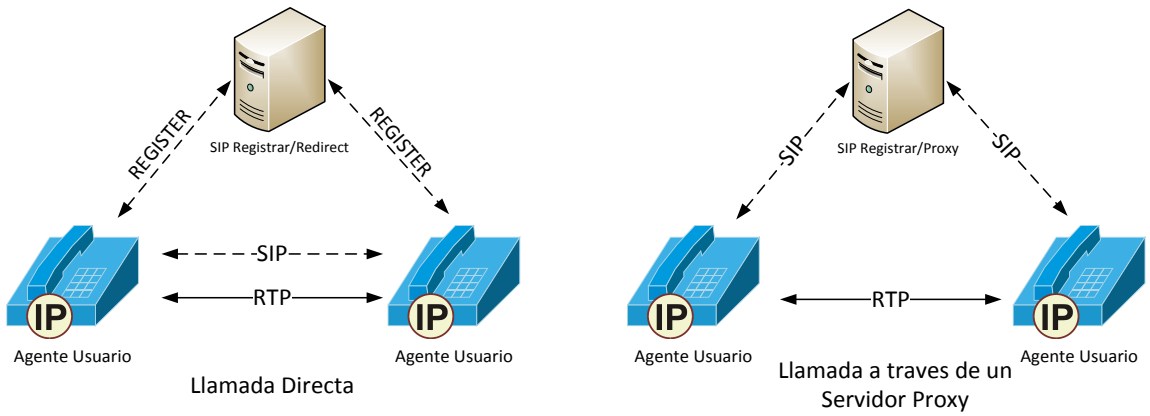


Figura 3.13. Señalización en el Protocolo *SIP* en dos modelos de llamadas distintos. [14]

IAX

El protocolo *IAX* (del inglés, *Inter-Asterisk eXchange*) se estandarizó en la RFC 5456 [18]. *IAX* es uno de los protocolos utilizados para manejar conexiones *VoIP* entre servidores denominados Asterisk, y entre servidores y clientes que también utilizan protocolo *IAX*. El protocolo *IAX* ahora se refiere generalmente al *IAX2* (segunda versión).

IAX2 es robusto, lleno de novedades y muy simple en comparación con otros protocolos. Utiliza un único puerto *UDP*, para comunicaciones entre dispositivos finales para señalización y datos [19]. Por lo que ambos tipos de tráfico no toman caminos independientes, por tanto se dice que la señalización se transmite en banda (del inglés, *in-band*), y necesariamente debe pasar por el servidor *IAX*, lo cual lo diferencia de otros protocolos como *SIP* y H.323 en los que el trafico *RTP* puede viajar de extremo a extremo sin tener que pasar por los equipos de señalización.

En la Figura 3.13 se muestra un diagrama de conexión de servidores y teléfonos *IP* usando *IAX*.

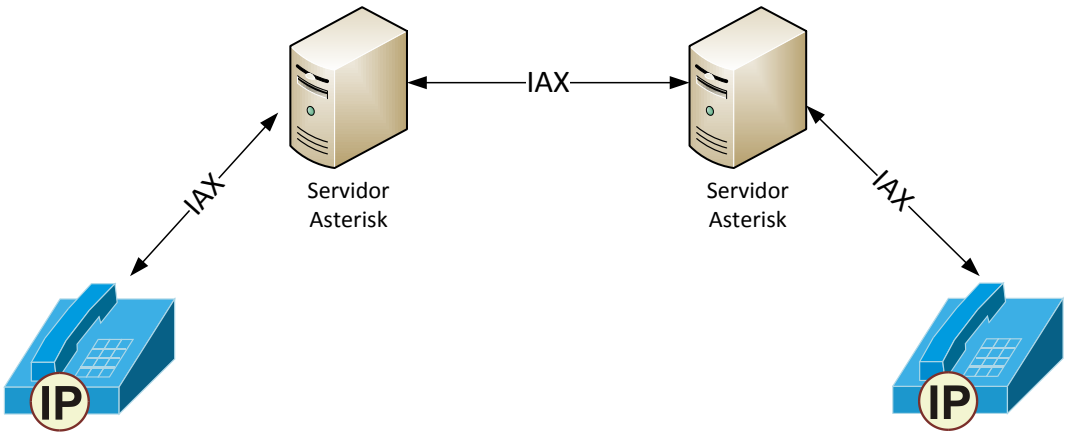


Figura 3.13. Señalización en el Protocolo *IAX*. [14]

MGCP

Es un protocolo de señalización estandarizado por medio del RFC 3435 [20]. *MGCP* (del inglés, *Media Gateway Control Protocol*) asume una arquitectura centralizada para el control de llamadas, y donde esta función es realizada por un Agente de Llamada (del inglés, *CA*: *Call Agent*). *MGCP* asume que estos elementos de control o *CA*, contienen la inteligencia para enviar órdenes a los *Gateways* o dispositivos finales bajo su control para establecer una llamada [21]. *MGCP* no define un mecanismo de sincronización para *CA*, y es en esencia un protocolo maestro-esclavo, por lo cual simplifica su configuración y administración. *MGCP* puede utilizar otros protocolos como *SIP* o H.323 para comunicarse con otras sedes fuera de su sistema centralizado de control.

En la Figura 3.14 se muestra un diagrama de conexión de servidores y teléfonos *IP* usando *MGCP* (entre servidores se podría usar *SIP* o H.323).

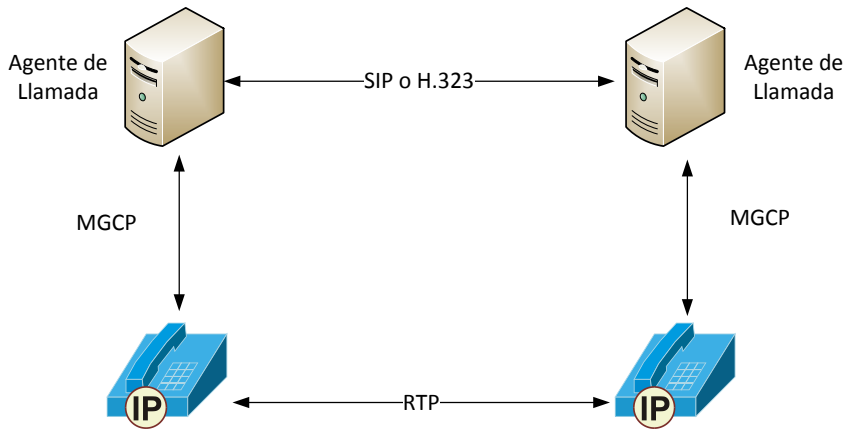


Figura 3.14. Señalización en el Protocolo MGCP. [21]

3.3.2 Protocolos de encaminamiento

Los paquetes de *VoIP* y de *VoIP* en la Nube, toman sus caminos en Internet a través de los encaminadores (del inglés, *routers*). Para lo cual en primer lugar, el encaminador crea una tabla de encaminamiento para recabar información de otros encaminadores de la ruta óptima para cada paquete. Esta tabla puede ser estática o dinámica, pero esta última es considerada superior, ya que se adapta a las condiciones cambiantes de la red [22].

Se puede clasificar a los protocolos de encaminamiento en Protocolos de *Gateway* Interior (del inglés, *IGP*: *Interior Gateway Protocols*) que se utilizan para el encaminamiento dentro de un sistema autónomo como por ejemplo *RIP* (Protocolo de Información de Encaminamiento, del inglés, *Routing Information Protocol*), *EIGRP* (Protocolo de Encaminamiento de Pasarela interior mejorado, del inglés, *Enhanced Interior Gateway Routing Protocol*), *IS-IS* (Sistema Intermedio a Sistema Intermedio, del inglés, *Intermediate System to Intermediate System*), *OSPF* (Primer trayecto más corto abierto, del inglés, *Open Shortest Path First*); y en Protocolos de *Gateway* Exterior (del inglés, EGP: *Exterior Gateway Protocol*) que se usan para el encaminamiento entre sistemas autónomos, como por ejemplo *BGP* (Protocolo de la Pasarela Externa, del inglés, *Border Gateway Protocol*) [22].

Para estos algoritmos existen métricas para poder evaluarlos como por ejemplo número de saltos, tasa de bits, retraso, confiabilidad y de ellos se

elige el protocolo a implementar y en base a este protocolo el algoritmo escoge el camino óptimo.

En la Tabla 2 se muestra una comparación entre los protocolos de encaminamiento más utilizados.

	RIPv2	EIGRP	OSPF	IS-IS	BGP
Interior o Exterior	Interior	Interior	Interior	Interior	Exterior
Tipo de Protocolo	Vector Distancia	Vector Distancia	Estado de Enlace	Vector Distancia	Vector Camino
Métrica	Numero de Saltos	Numero de Saltos	Costo	Costo	Multiples Atributos
Limite Numero de Saltos	15	224	Ninguno	Ninguno	-
Tamaño de la red	Pequeño	Grande	Grande	Grande	Muy Grande
Tiempo de convergencia	Lento	Muy rápido	Rápido	Rápido	Muy lento
Algoritmo	Bellman-Ford	Dual	Dijkstra	Dijkstra	Algoritmo Mejor Ruta
Direccionamiento sin clase	Si	Si	Si	Si	Si
Requiere diseño jerárquico	No	No	Si	Si	No
Distancia administrativa	120	5/90/170	110	115	20/200
Actualizaciones	30 seg	Solo cuando ocurren cambios	Solo cuando ocurren cambios	Solo cuando ocurren cambios	Solo cuando ocurren cambios

Tabla 2. Características de los protocolos de encaminamiento más utilizados. [23]

Y a pesar que no es un propósito del presente trabajo realizar un estudio detallado de cada protocolo de encaminamiento, y su desenvolvimiento

frente a *VoIP*, y también que los usuarios finales no pueden tener un control sobre el mismo sobre todo cuando en la Nube se está conectado a un conjunto de redes como es Internet, donde esta administración corresponde a los proveedores del servicio, si se puede mencionar que la *VoIP* está directamente relacionada a los tiempos de convergencia de los protocolos de encaminamiento elegidos. Y al ser *BGP* el protocolo utilizado para interconexión entre los Proveedores de Servicio de Internet (del inglés, *ISP*: *Internet Service Provider*), así como el protocolo de encaminamiento dentro de los grandes *ISP* [24], por lo que sus ventajas y desventajas, determinan directamente el éxito o fracaso de una solución de *VoIP* en la Nube.

Entre las ventajas de *BGP* se encuentran por ejemplo la asignación de muchos atributos y sus reglas complejas para selección de una ruta, su escalabilidad [25], y entre las desventajas la lenta convergencia [26] [27]. Pero a pesar de ello *BGP* encuentra la ruta más corta hacia el destino, es decir la ruta que atraviesa la menor cantidad de sistemas autónomos (del inglés, *AS*: *autonomous system*) característicos de este protocolo [28].

Por ello es importante que una implementación de *VoIP* en la Nube cuente con proveedores de internet, que implementen *BGP* directamente, para así los tiempos de establecimiento de las llamadas sean los mínimos posibles. Por esta razón *VoIP* en la Nube, presenta ventajas en comparación a *VoIP* tradicional, porque puede realizar el encaminamiento directo en *BGP* en la Nube, sin la necesidad de realizarlo sobre protocolos de encaminamiento

internos como *OSPF* o *IS-IS*, en sedes que poseen estos protocolos para su acceso a Internet.

3.3.3 Protocolos de Transporte de la Voz

Las funciones de los protocolos de encaminamiento y de señalización son de disponer los caminos que van a tomar los flujos de voz y el establecimiento y desconexión de las llamadas, respectivamente. Por lo tanto, los protocolos de transporte son los encargados de llevar los paquetes con la voz digitalizada (y comprimida de ser el caso) de un lugar a otro, y que los mismos puedan ser reconstruidos en el orden correcto.

RTP

El Protocolo de Transporte en Tiempo Real (del inglés, *Real-time Transport Protocol*) se estandarizó en la RFC 3550 [29], es un protocolo diseñado para el transporte de datos con características de tiempo real, como lo es la voz. La capa de transporte puede utilizar ya sea *TCP* (Protocolo de Control de Transmisión, del inglés, *Transmission Control Protocol*) o *UDP* (Protocolo de datagrama de usuario, del inglés, *User Datagram Protocol*), sin embargo, la extra fiabilidad y la retransmisión de paquetes perdidos ofrecidos por *TCP*, no tiene sentido en la comunicaciones de voz. En consecuencia, para realizar llamadas de voz sobre paquetes *IP*, el *RTP* se encapsula en *UDP* para una entrega rápida. Por otra parte, *RTP* sobre *UDP* complementan la

falta de fiabilidad, proporcionando secuencia de la numeración y la reordenación de los paquetes. *RTP* sobre *UDP* se utiliza para los protocolos de señalización fuera de la banda, como H.323 y *SIP*, para la transmisión de voz. En la Figura 3.15 se muestra la encapsulación de la voz de *RTP* en *UDP*.

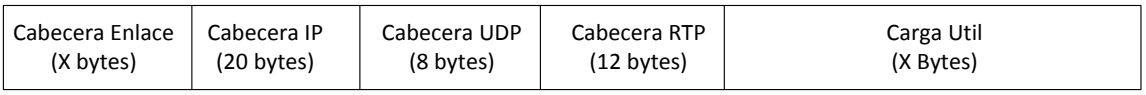


Figura 3.15. Encapsulación de la Voz de *RTP* en *UDP*. [13]

3.4 CONSIDERACIONES EN EL DISEÑO DE VOIP EN LA NUBE

Una de las características de los sistemas que operan servicios en la Nube, entre ellos los servicios de *VoIP*, es la confianza en las redes, y si a este se suma que cada día ingresan más usuarios y aplicaciones a la Nube, las redes de acceso y de “columna vertebral” (del inglés, *backbone*) deben estar en la capacidad de soportar más tráfico, con niveles de entrega satisfactorios. Y es así que los requerimientos del tiempo operativo (del inglés, *uptime*) son cada vez mayores. A continuación se muestra en la Tabla 3, porcentajes de *uptime* con sus correspondientes tiempos sin servicios en un año y un mes.

Disponibilidad del Servicio	Tiempo sin Servicio en 1 Año	Tiempo sin Servicio proporcional en 1 Mes
95%	> 18 dias	36 horas
99%	> 3.6 dias	7 horas
99.50%	> 1.8 dias	3.6 horas
99.90%	8.7 horas	43 minutos
99.99%	53 minutos	4.3 minutos
99.999%	> 5 minutos	26 segundos
99.9999%	31 segundos	< 3 segundos

Tabla 3. Disponibilidad del Servicio vs. Tiempo sin servicio

Como consecuencia para poder ofrecer una alta disponibilidad del servicio se deben tomar consideraciones para la planificación, diseño e implementaciones de las redes en sí misma, entre las principales características con las que se podemos categorizar una red para *VoIP* se encuentran [30]:

Retraso: es el tiempo que toma a un paquete en ir desde el dispositivo transmisor al dispositivo receptor. Según el estándar G.114 de la ITU-T recomienda que si el retraso en una vía es menor a 150 ms, entonces es aceptable para una buena comunicación de voz, pero si este es superior a 400 ms la comunicación es inaceptable, e intermedia entre estos dos valores.

Jitter: es la variación del retraso de los paquetes, que incurre a través de la red debido al procesamiento, encolamiento, almacenamiento en memoria temporal (del inglés, *buffering*), congestión, o la variación del tiempo de propagación debido a las diferentes rutas que puede tomar un paquete *IP*.

Paquetes perdidos y con errores: son los paquetes que no llegan a su destino o que al hacerlo llega con información de carga útil diferente al enviado. Por tanto la red debe ser planificada para proveer una suficiente tasa de bits tanto para la señalización, como para el tráfico de voz principalmente, por lo tanto se deben aplicar técnicas conocidas como de Calidad de Servicio para evitar la congestión o la perdida de paquetes, como por ejemplo debido a un control de flujo deficiente. Dependiendo del protocolo de señalización pueden estos paquetes utilizar el protocolo *TCP* para el establecimiento de la llamada como H.323, por lo tanto existirían retransmisiones si es necesario, lo cual podrían producir retrasos en el establecimiento, desconexión o solicitud de servicios en una llamada.

Calidad de Servicio (del inglés, *QoS: Quality of Service*): son un conjunto de técnicas que ofrecen capacidad de priorizar, es decir ofrecer mayor velocidad a ciertos paquetes sobre otros, por lo que entonces proveen niveles de tráfico consistentes y predecibles entre los partícipes de una aplicación.

Tasa de Bits: es el número de bits que se transmiten típicamente en un segundo, por lo que proveer de una cantidad correcta de tasa de bits o velocidad de transmisión entre los partícipes de las llamadas, tomando en consideración los tipos y números de dispositivos, Códec utilizado, tiempo de fragmentación y características adicionales como seguridad por ejemplo. Por lo tanto, la tasa de bits aprovisionada debe contar con políticas de *QoS* habilitadas, para poder proporcionar la priorización y la programación de las diferentes clases de tráfico.

En la Tabla 4 se muestran las tasas de bits para dos Códecs de voz muy utilizados. De lo cual se aprecia que las velocidades son mayores a las comúnmente conocidas para estos Códec G.711 y G.729, es decir 64 Kbps y 8 Kbps respectivamente [13], que serían las tasas de bits que corresponderían a la carga útil de la voz, a ellos se suman a la longitud de las cabeceras correspondientes de *RTP*, *UDP*, *IP* y de enlace tal como se aprecia en la Figura 3.15.

Codec	Intervalo de Fragmentacion	Tasa de bits por llamada
G.711	20 ms	80 kbps
G.711	30 ms	74 kbps
G.729	20 ms	24 kbps
G.729	30 ms	19 kbps

Tabla 4. Tasa de Bits promedio para G.711 y G.729. [13]

Un aspecto a considerar es que ni la *QoS*, ni la tasa de bits por sí solos son la solución, sino que ambos deben ser planificados en la infraestructura de red, por ello se recomienda seguir un Modelo Jerárquico [31]. Este modelo mediante la especialización de funciones, combinada con una organización jerárquica, simplifica las tareas requeridas para construir una red que cumpla con los requerimientos actuales y pueda crecer para satisfacer las necesidades futuras. Los modelos jerárquicos utilizan niveles para simplificar las tareas de interconexión. Cada nivel puede centrarse en funciones específicas, que le permite elegir los sistemas adecuados y las características de cada nivel. Los modelos jerárquicos se aplican tanto al diseño de *LAN* y *WAN*. En la Figura 3.16 se muestra un diseño general basado en el Modelo Jerárquico.

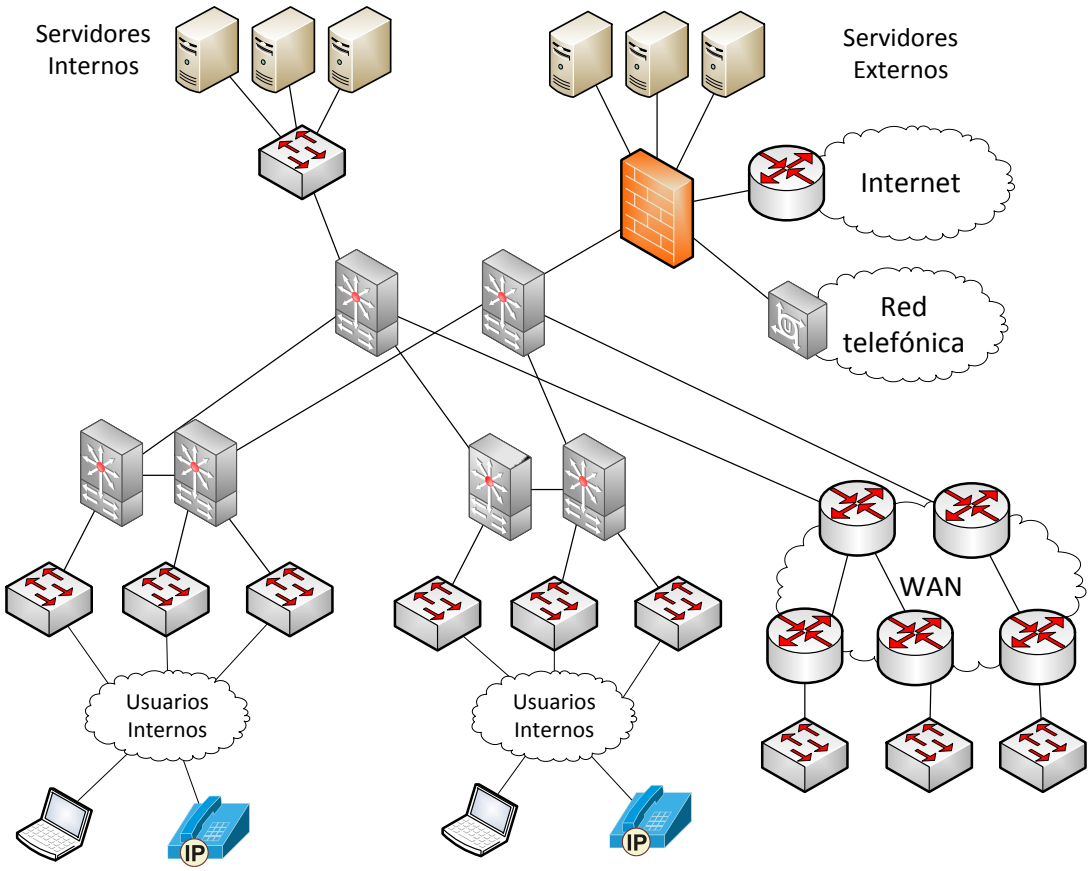


Figura 3.16. Modelo Jerárquico de diseño para redes. [31]

La naturaleza modular de este modelo permite el uso apropiado de la tasa de bits dentro de cada nivel de la jerarquía, reduciendo la provisión de la velocidad por adelantado a la necesidad actual. La meta por lo tanto es mantener cada elemento de diseño simple y funcional, centrado en su necesidad, facilitando la implementación y compresión de la red, lo que facilita su gestión y escalabilidad.

Por lo cual una buena forma de planificar la red, es empezar desde los aspectos más generales de la aplicación que necesitamos transportar, con lo cual se adapta la infraestructura física a la necesidad real, y no viceversa, que es un modelo de diseño arriba-abajo (del inglés, *Top-Down*) [31]. En la Figura 3.17 se muestra un esquema de diseño de redes, siguiendo el esquema jerárquico y modular.

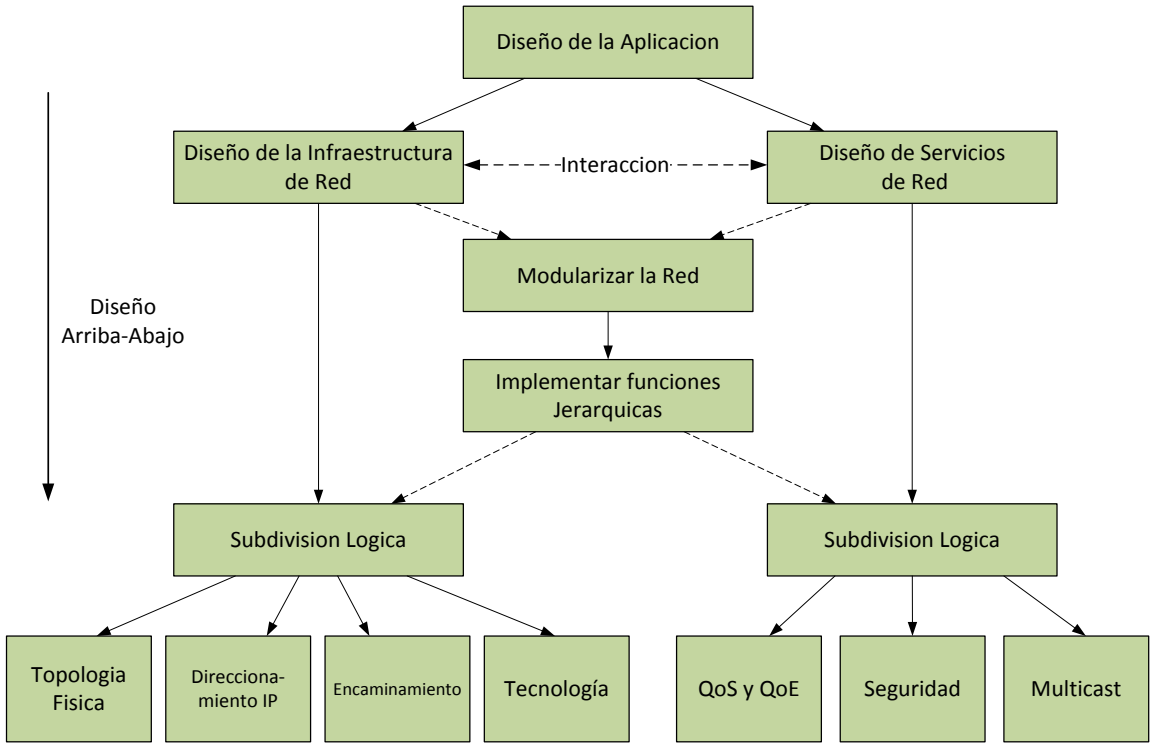


Figura 3.17. Principios de Diseño Jerárquico y Modular. [31]

3.4.1 Diseño de LAN

El diseño de *LAN* es tan importante como el diseño de *WAN*, dado que los usuarios están conectados a ellos directamente y es quien distribuye de manera ordenada, el acceso de los usuarios de una sede a la *WAN*.

Un diseño tradicional de *LAN* jerárquica está compuesto de tres niveles [31], tal como se aprecia en la Figura 3.18.

1. El Nivel de Acceso (del inglés, *Access Layer*) provee la conectividad de las estaciones de trabajo y servidores, así como una posible clasificación en grupos de trabajo, como son las V*LAN*s (*LAN* virtuales, del inglés, *Virtual LAN*); y herramientas de seguridad de primera mano.
2. El Nivel de Distribución (del inglés, *Distribution Layer*) provee conectividad basada en políticas de entrega de tráfico, filtrado de seguridad, balanceo de cargas, *QoS*, encaminamiento entre *VLANs* y demás políticas de una organización. También es utilizada para aislar problemas en la red.
3. El Nivel del Núcleo (del inglés, *Core Layer*) provee de un *backbone* de alta velocidad para los conmutadores de distribución (del inglés, *Distribution Switches*). Al ser este nivel crítico para las

comunicaciones debe proveer una alta confiabilidad de la red y adaptarse muy rápido a los cambios.

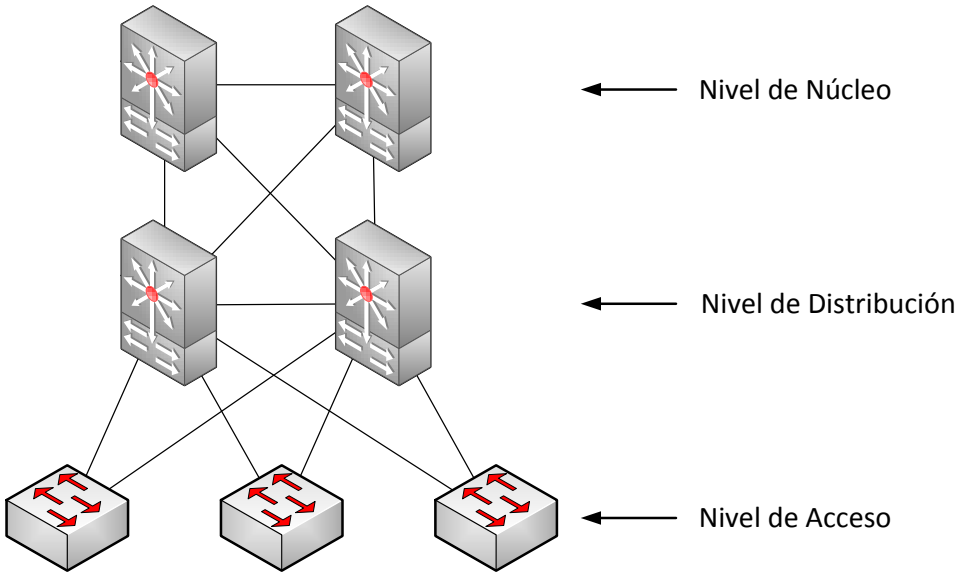


Figura 3.18. Diseño de interconexión de conmutadores para una *LAN* jerárquica en tres niveles. [31]

Cada nivel proporciona la funcionalidad necesaria para la red. Cabe mencionar que no es estrictamente necesario implementar cada nivel por separado en entidades físicas diferentes, pueden compartir recursos en un mismo equipo, pero cuando el número de usuarios y por lo tanto el tráfico

aumenta de manera considerable, se recomienda su separación, para una gestión más eficaz.

3.4.2 Diseño de WAN

Hay varios factores que deben considerarse al seleccionar una tecnología de transporte para *WAN*, como que por su naturaleza pueden ser públicas como el Internet, o privadas. La geografía juega también un papel clave en lo referente a que tecnologías *WAN* están disponibles en un área determinada. Las ciudades más importantes disponen de más opciones de transporte para *WAN*, y las zonas rurales son más limitadas en cuanto a la disponibilidad de opciones.

En la Tabla 5 se examinan algunas tecnologías *WAN*, y se realiza una comparación básica entre las mismas y que nos ayudan a seleccionar la mejor alternativa disponible.

Tipo	Tecnologia WAN	Tasa de Bits	Confiabilidad	Retardo
Acceso	DSL	Medio	Baja	Medio
	Cable Módem	Medio	Baja	Medio
	Wifi	Medio	Baja	Medio
	Celular	Bajo-Medio	Baja	Medio
Troncal	Metro Ethernet	Medio-Alto	Alta	Baja
	SONET/SDH	Alto	Alta	Baja
	MPLS	Alto	Alta	Baja
	Fibra Oscura (Dark Fiber)	Alto	Alta	Baja
	DWDM	Alto	Alta	Baja

Tabla 5. Comparación Tecnologías *WAN* de Acceso y Troncal. [31]

La alta disponibilidad del servicio es lo que la mayoría de los usuarios desean para sus aplicaciones de red, y es altamente crítica cuando empezamos a migrar aplicaciones a la Nube. Los componentes claves de la disponibilidad de las aplicaciones son el tiempo de respuesta, el rendimiento y la fiabilidad. Para las aplicaciones en tiempo real tales como la voz y el vídeo, estas no son muy tolerantes a la fluctuación y el retraso. La Tabla 6 identifica diversos requisitos de tráfico para aplicaciones de datos, voz y vídeo.

	Aplicaciones Comunes		Aplicaciones Tiempo-Real	
	Transferencia de Archivos	**Navegacion Web**	**Voz**	**Video**
Tasa de bits necesaria	Alta	Baja	Baja	Alta
Tolerancia del sistema a paquetes perdidos y con errores	Alta (TCP)	Alta (TCP)	Baja (UDP)	Media (UDP)
Tolerancia del sistema a *Downtine*	Media (TCP)	Media (TCP)	Baja (UDP)	Media (UDP)
Tolerancia a Retraso de paquetes	Medio	Medio	Bajo	Bajo
Tolerancia a *Jitter*	Medio	Medio	Bajo	Bajo

Tabla 6. Requerimientos de las Aplicaciones. [31]

En la Tabla 6 se aprecia que la voz es la más sensible a aspectos como su tolerancia a paquetes perdidos y con errores, porque *UDP* es utilizado como protocolo de transporte, así como una mínima tolerancia al *downtime*, que pueda hacer que termine la conexión anticipadamente. Entre otras desventajas también se encuentran mínimas tolerancias a los retrasos de paquetes y *jitter*, pero presenta ventajas significativas como son sus bajas tasas de bits necesarias para la voz. Por lo cual el diseño de una red para *VoIP* en la Nube, debe poner un énfasis muy especial sobre todos estos aspectos.

En la Tabla 7 se comparan una serie de medios diferentes de transporte para las redes *WAN*, junto con las velocidades y tipos de tecnologías asociadas a ellos.

Medio	< 2 Mbps	2 Mbps - 45 Mbps	45 Mbps - 100 Mbps	100 Mbps- 10 Gbps
Cobre	ISDN, Frame Relay, TDM, ADSL	Frame Relay, Ethernet, ADSL, Cable	Fast Ethernet	Gigabit Ethernet, 10 Gigabit Ethernet
Fibra Óptica	N/D	Ethernet	Fast Ethernet, ATM	Gigabit Ethernet, 10 Gigabit Ethernet, SONET/SDH, Fibra Oscura
Inalámbrico	802.11b, GPRS, Edge	802.11b, 3G, HSPA	802.11a/g	802.11n

Tabla 7. Comparación de Tasas de Bits en Medio Físicos. [31]

Las diferentes técnicas de *QoS* son cada vez más importantes debido al aumento de tráfico sensible al retraso, como lo es la *VoIP*, a esto sumando las limitadas tasas de bits disponibles en *WAN*, mientras que en la *LAN* generalmente son muy abundantes. Por lo que las diferentes técnicas de *QoS* están encaminadas mayormente al tratamiento de tráfico en *WAN*. En la Tabla 8 se muestran algunas técnicas comunes de *QoS*, para asegurar de que el tráfico más crítico reciba el mejor tratamiento posible en base a la velocidad disponible en los momentos de congestión.

Categoría QoS	Descripción
Clasificación	Identifica y marca flujos y provee prioridad a ciertos flujos
Gestión de la Congestión	Mecanismos para manejar el desbordamiento utilizando algoritmos de encolamiento
Mecanismos Eficiencia del Enlace	Reduce el retraso y *jitter* en enlaces de baja velocidad
Traffic Shaping y Policing	Evita la congestión mediantes políticas de ingresos y egresos de flujos

Tabla 8. Técnicas comunes de *QoS*. [31]

3.4.3 Diseño de Redes Privadas Virtuales

Una red privada virtual o *VPN* es una tecnología que utiliza el Internet u otra red intermedia para conectar equipos terminales a redes remotas que de otro modo serían inaccesibles. La *VPN* proporciona seguridad para el tráfico enviado, de forma que permanezca aislado de otros equipos de la red intermedia a través de la conexión [32]. Las *VPN* pueden conectar a usuarios individuales a una red remota o conectar varias redes juntas a través de una red pública. A través de la *VPN*, los usuarios pueden tener acceso a múltiples recursos en las redes remotas, tales como archivos, bases de datos, sitios Web internos, servicios de telefonía, etc., tal como si estuvieran en una conexión punto a punto, como se muestra en la Figura 3.19.

Existen básicamente dos arquitecturas de conexión *VPN*: de acceso remoto y punto a punto [22]. La *VPN* de acceso remoto es la más comúnmente utilizada para usuarios que se conectan a una red por medio de autenticación a través de Internet como medio de acceso. La *VPN* punto a punto, se utiliza para conectar oficinas remotas con la sede central.

El servidor o encaminador *VPN*, que posee un vínculo permanente a Internet, acepta las conexiones vía Internet provenientes de los sitios y establece un túnel *VPN*. Los servidores o encaminadores de las sucursales se conectan a Internet utilizando los servicios de su proveedor local de Internet, típicamente mediante conexiones de banda ancha. Esto permite eliminar los costosos vínculos punto a punto tradicionales, sobre todo en las comunicaciones internacionales.

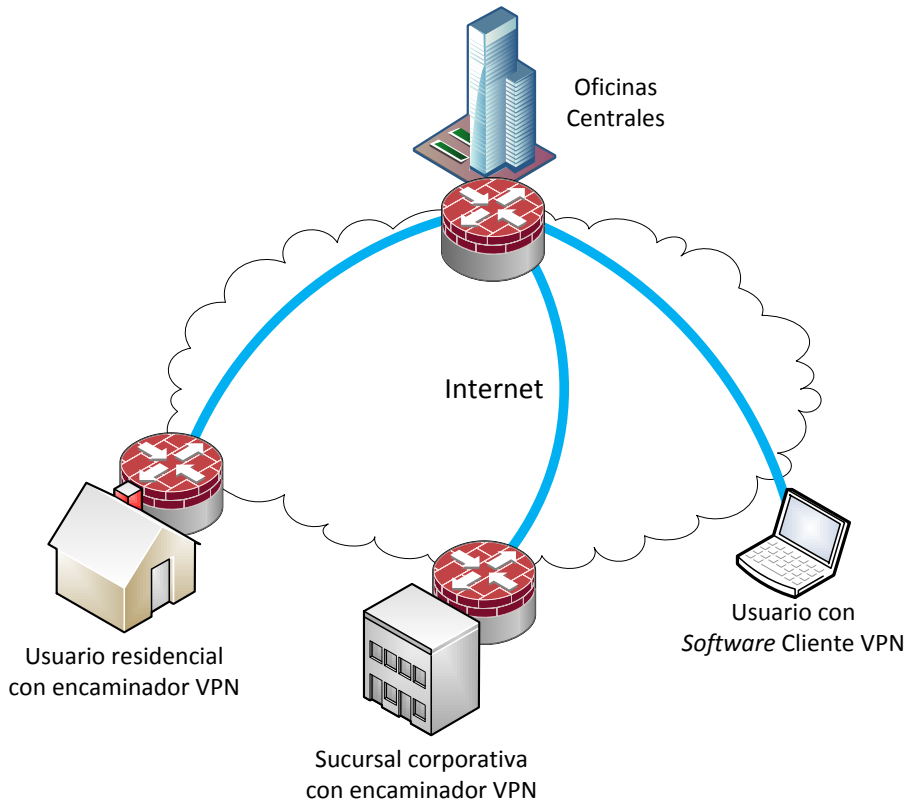


Figura 3.19. Creación de *VPN* a través de Internet. [31]

Los protocolos de *VPN* incluyen los siguientes [33]:

1. *Internet Protocol Security* (*IPSec*) fue desarrollado por la *IETF*. Su diseño se ajusta a la mayoría de los objetivos de seguridad: autenticación, integridad y confidencialidad. *IPSec* funciona a través de la codificación y la encapsulación de un paquete *IP* dentro de un paquete *IPSec*. La desencapsulación sucede al final del túnel, donde se decodifica el paquete *IP* y es reenviado a su destino previsto.

2. *Secure Sockets Layer* (*SSL*) y su sucesor *Transport Layer Security* (*TLS*) son protocolos criptográficos que proporcionan comunicaciones seguras por una red, comúnmente Internet. Una *VPN* SSL puede conectar varias sedes donde *IPSec* no funciona correctamente por políticas en los cortafuegos (del inglés, *firewall)* y reglas de *NAT* (Traducción de direcciones de Red, del inglés, *Network AddressTranslation*).

3.4.4 Diseño de Zona Desmilitarizada

La Zona Desmilitarizada o *DMZ* se la puede definir como una red separada, localizada en una zona neutral entre una red privada y una pública, por razones de seguridad y control [34]. La *DMZ* actúa básicamente como una etapa intermedia entre Internet y los recursos de la red privada, previniendo a usuarios externos tener acceso directo a servidores internos y a los datos, como se aprecia en la Figura 3.20. En la mayoría de redes actuales con servicios públicos, tales como correo electrónico, navegación Web, y ahora los servidores de *VoIP* en la Nube, se encuentran en la *DMZ* para ofrecer servicio a los usuarios autorizados para los servicios.

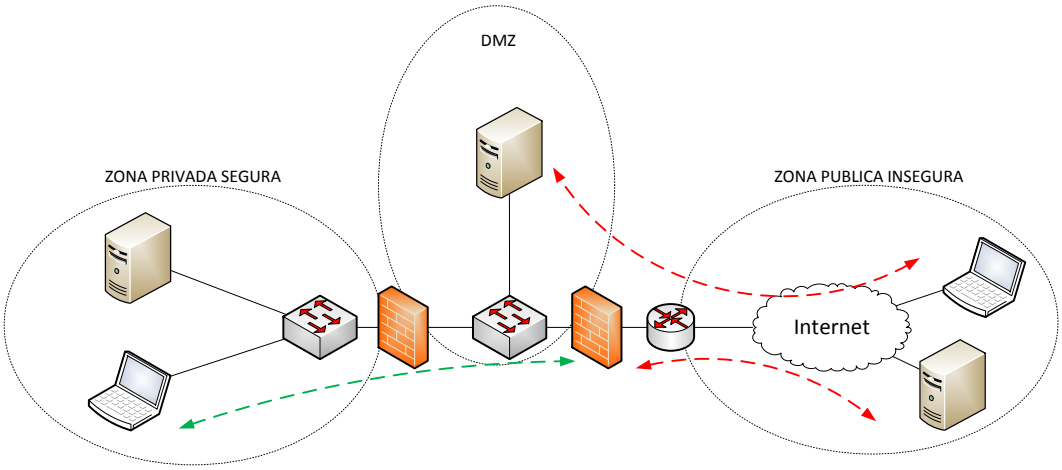


Figura 3.20. Esquema de zona neutral de la *DMZ*

De ahora en adelante se dibuja los dos *firewalls* de la *DMZ* mediante un solo equipo, como se muestra en la Figura 3.21.

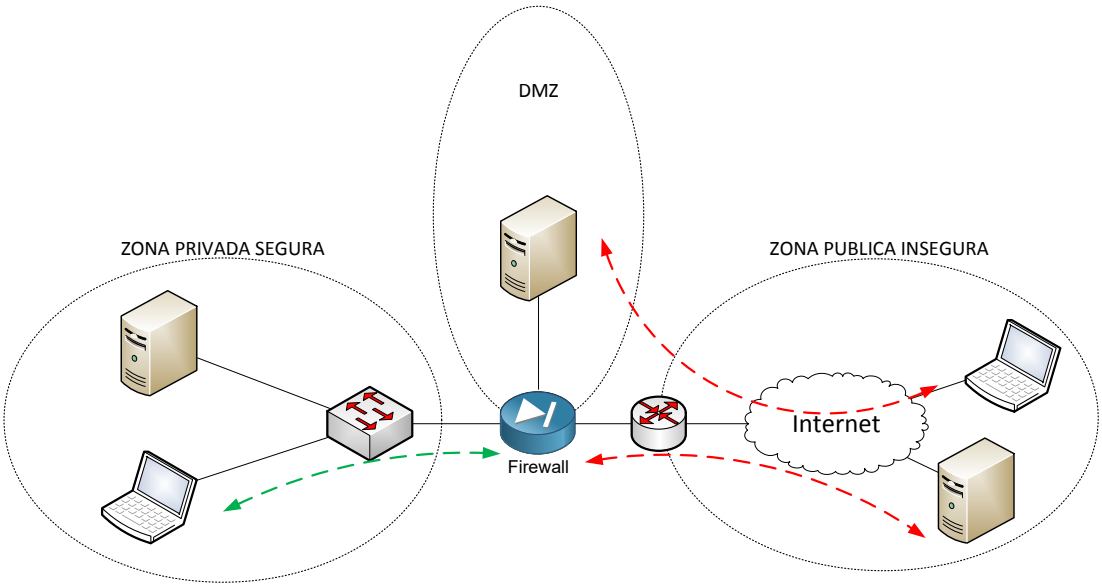


Figura 3.21. Esquema simplificado de la *DMZ*. [34]

El acceso remoto se autentica y asegura por medio de *SSL* o *IPSec*. Las políticas de control de acceso también se pueden hacer cumplir para limitar el acceso sólo a los recursos necesarios y de acuerdo a la función del usuario.

Por lo tanto, el proveedor de servicios de *VoIP* en la Nube puede tener una infraestructura como se muestra en la Figura 3.22, la cual ha sido adaptada del Modelo Compuesto para Empresas de Cisco [35]. En la red interna se encuentra la granja de servidores, donde residen datos como facturación, bases de datos, y demás sistemas necesarios para la operación, y desde la *DMZ* ofrecer servicios de telefonía a sus usuarios. Y en una etapa intermedia del Borde de Distribución, donde residen conmutadores de distribución, porque como se mencionó en la sección 3.4.1, es aquí donde se aplican las políticas de red de la organización, y que en este caso sirven como elemento de frontera.

Pero en cambio si el servicio se ofrece, realizando una conexión *VPN* entre los dos puntos, a través del Internet, se necesita un equipo que autentique la conexión, tal como lo muestra la Figura 3.23. Cabe mencionar que tanto el *firewall* como el equipo que permite el acceso remoto vía *VPN*, pueden estar incluidos en el mismo *hardware*, o se puede duplicar el *hardware* para que cada una realice una función distinta. Igual estos equipos se conectan al borde de distribución.

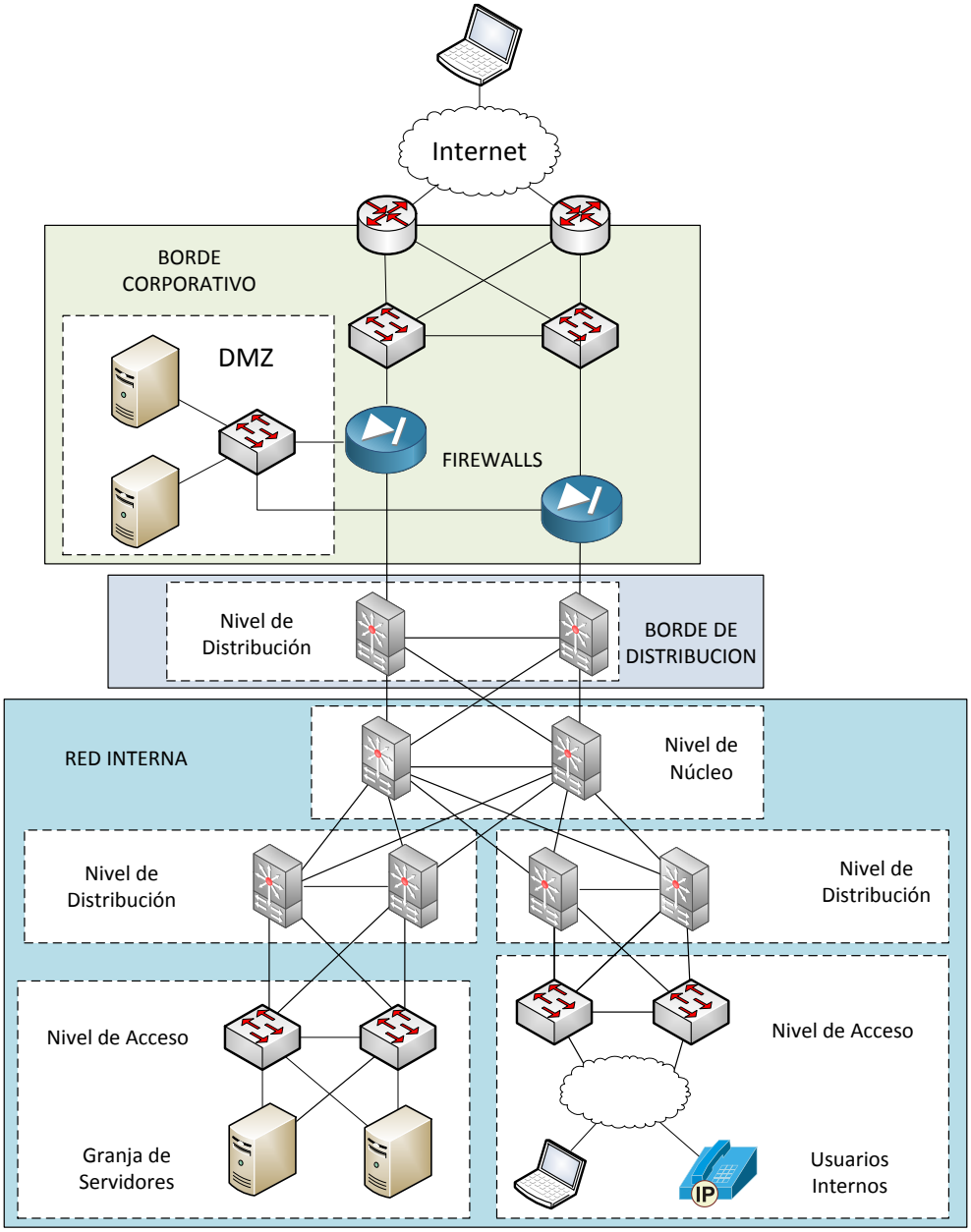


Figura 3.22. Modelo para ofrecer servicios a usuarios externos a través de *DMZ*.

[34]

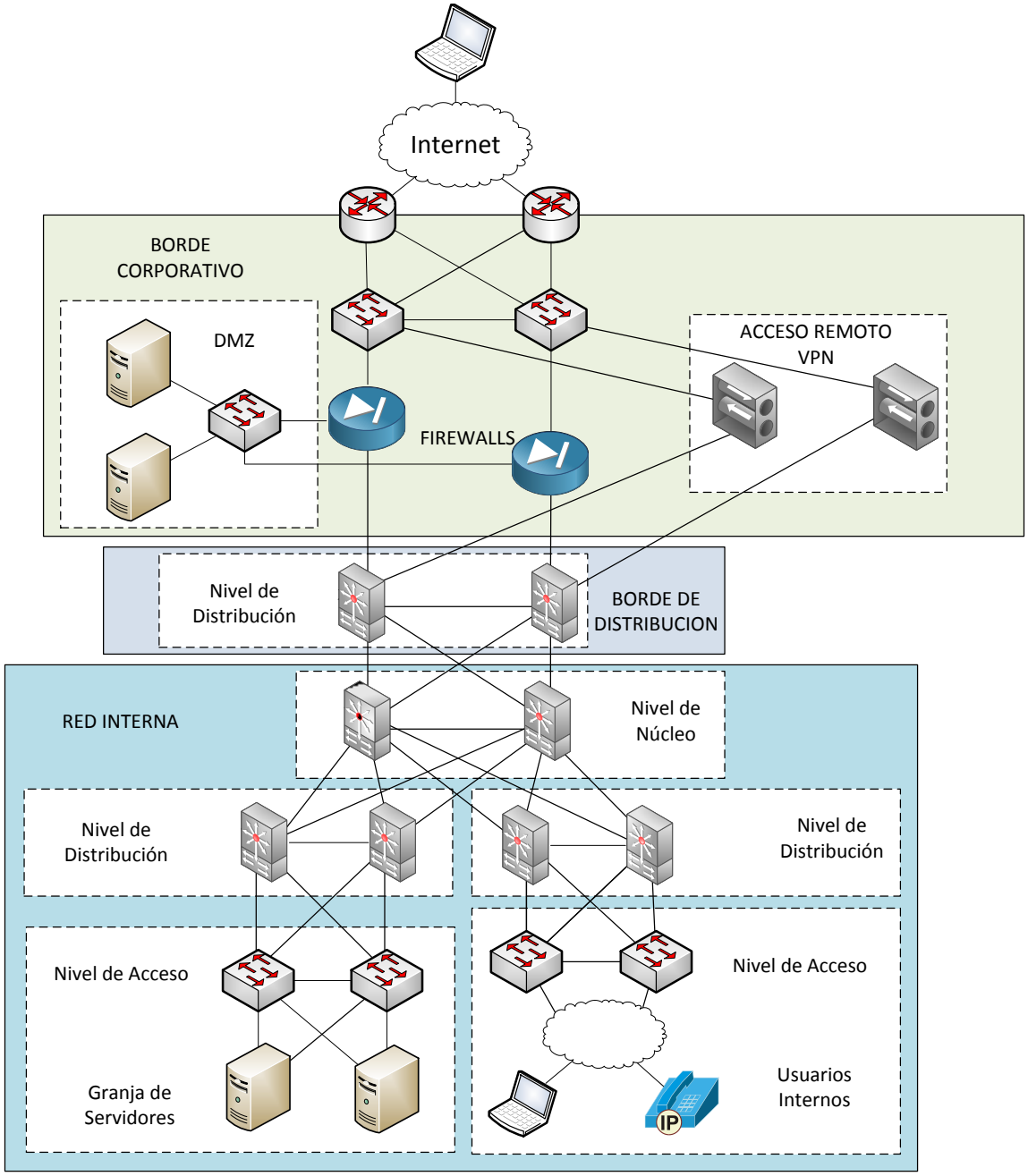


Figura 3.23. Modelo para ofrecer servicios a usuarios externos a través de *DMZ* y *VPN*. [34]

3.4.5 Dimensionamiento de Gateways de Voz

El dimensionamiento de los *Gateways* de voz, es muy importante también, dado que para poder ofrecer servicios de *VoIP* en la Nube, se necesita tener interconexión con la *RTB*. Por lo tanto se debe tener en cuenta las necesidades del tráfico de voz en el diseño de una red. Dado que un sistema sobredimensionado podría ser económicamente inviable, porque la probabilidad de acceso simultáneo de todos los usuarios es prácticamente nula. Dos medidas que nos ayudan a determinar la magnitud de este tráfico son el Grado de Servicio y los Erlangs:

Grado de Servicio: probabilidad de que una llamada sea bloqueada por un *Gateway* de voz cuando intenta ocupar un circuito durante la hora más ocupada. Matemáticamente se puede definir como:

$$Grado\ de\ Servicio = \frac{Llamadas\ pérdidas}{Intentos\ de\ llamada}\ x\ 100\% \qquad (3.1)\ [36]$$

Erlangs: uno de los modelos probabilísticos más utilizados para medir la intensidad de tráfico, que es una razón adimensional entre el volumen del tráfico y el intervalo de tiempo durante el cual es realizada la medición. Matemáticamente se puede definir como:

$$Erlangs = \frac{Número\ de\ llamadas\ x\ Tiempo\ medio\ de\ llamada\ (minutos)}{60} \quad (3.2)\ [36]$$

Dos de los modelos más comúnmente utilizados para medición de tráfico son Erlang B y Erlang B extendido.

Los Erlang B calculan la capacidad de canales requeridos dado una valor de Erlang (trafico hora pico) y el Grado de Servicio deseado. Se utiliza para medir la probabilidad de pérdida de llamadas en un grupo de circuitos. Estadísticamente sigue una distribución de *Poisson*.

$$Probabilidad_{Bloqueo} = \frac{\frac{Erlangs^{\#\ de\ Canales}}{\#\ de\ Canales!}}{\sum_{x=0}^{\#\ de\ Canales}\frac{Erlangs^{x}}{x!}}\ x\ 100\% \quad (3.3)\ [36]$$

Los Erlang B extendidos se utiliza cuando algunas llamadas bloqueadas o fallidas vuelven a ser reintentadas, por lo cual se añade un porcentaje de reintento a los Erlang B.

En la Tabla 9 se muestra Erlangs B para 1 a 30 canales, después 60, 90, 120 y 150 canales, que representaría 2, 3, 4 y 5 E1 de voz respectivamente.

# Canales	Grado de Servicio							
(N)	0.10%	0.50%	1%	2%	3%	5%	7%	10%
1	0.001	0.005	0.01	0.02	0.031	0.053	0.075	0.111
2	0.046	0.105	0.153	0.223	0.282	0.381	0.47	0.595
3	0.194	0.349	0.455	0.602	0.715	0.899	1.06	1.27
4	0.439	0.701	0.869	1.09	1.26	1.52	1.75	2.05
5	0.762	1.13	1.36	1.66	1.88	2.22	2.5	2.88
6	1.15	1.62	1.91	2.28	2.54	2.96	3.3	3.76
7	1.58	2.16	2.5	2.94	3.25	3.74	4.14	4.67
8	2.05	2.73	3.13	3.63	3.99	4.54	5	5.6
9	2.56	3.33	3.78	4.34	4.75	5.37	5.88	6.55
10	3.09	3.96	4.46	5.08	5.53	6.22	6.78	7.51
11	3.65	4.61	5.16	5.84	6.33	7.08	7.69	8.49
12	4.23	5.28	5.88	6.61	7.14	7.95	8.61	9.47
13	4.83	5.96	6.61	7.4	7.97	8.83	9.54	10.5
14	5.45	6.66	7.35	8.2	8.8	9.73	10.5	11.5
15	6.08	7.38	8.11	9.01	9.65	10.6	11.4	12.5
16	6.72	8.1	8.88	9.83	10.5	11.5	12.4	13.5
17	7.38	8.83	9.65	10.7	11.4	12.5	13.4	14.5
18	8.05	9.58	10.4	11.5	12.2	13.4	14.3	15.5
19	8.72	10.3	11.2	12.3	13.1	14.3	15.3	16.6
20	9.41	11.1	12	13.2	14	15.2	16.3	17.6
21	10.1	11.9	12.8	14	14.9	16.2	17.3	18.7
22	10.8	12.6	13.7	14.9	15.8	17.1	18.2	19.7
23	11.5	13.4	14.5	15.8	16.7	18.1	19.2	20.7
24	12.2	14.2	15.3	16.6	17.6	19	20.2	21.8
25	13	15	16.1	17.5	18.5	20	21.2	22.8
26	13.7	15.8	17	18.4	19.4	20.9	22.2	23.9
27	14.4	16.6	17.8	19.3	20.3	21.9	23.2	24.9
28	15.2	17.4	18.6	20.2	21.2	22.9	24.2	26
29	15.9	18.2	19.5	21	22.1	23.8	25.2	27.1
30	16.7	19	20.3	21.9	23.1	24.8	26.2	28.1
60	40.8	44.8	46.9	49.6	51.6	54.6	57.1	60.4
90	66.5	71.8	74.7	78.3	80.9	85	88.5	93.1
120	93	99.4	103	107.4	110.7	115.8	120.1	126.1
150	119.9	127.4	131.6	136.8	140.7	146.7	151.9	159.1

Tabla 9. Erlangs B. [36]

Y dependiendo del número de canales necesitados al dimensionar el *Gateway* de Voz, se puede utilizar interfaces de conexión analógicas, si existen pocos canales necesitados, y utilizar interfaces digitales cuando el

número de canales lo justifique. En la Tabla 10 se muestran las interfaces de conexión más utilizadas con la *RTB*.

Tipo	Interface	Número de Canales	Uso Típico
Analógico	FXO	1	Conexión Residencial o Pequeña Empresa
Digital	T1 CAS	24	Usado principalmente en Norte América
Digital	T1 PRI	23	Usado principalmente en Norte América
Digital	E1 PRI	30	Usado principalmente en Europa y Sudamérica
Digital	E1 R2	30	Usado principalmente en Asia y Sudamérica

Tabla 10. Interfaces de conexión a la *RTB* más utilizadas. [13]

3.4.6 Esquema de la solución para VoIP en la Nube

Siguiendo el esquema de modularidad y jerarquía, que es muy útil cuando la red crece en tamaño, adicionalmente considerando los diseños y dimensionamientos de *LAN*, *WAN*, *VPN*, *DMZ* y *Gateways* de voz descritos en las secciones 3.4.1 a 3.4.5, y tomando como referencia el Modelo Compuesto para Empresas de Cisco [35], y el Modelo de Arquitectura para Empresas de Cisco [31], se realizaron algunas adaptaciones a estos modelos, y en consecuencia se puede dividir el diseño de la red para servicios de *VoIP* en la Nube, en cinco grandes áreas que son: Módulo de Red Interna, Módulo de Borde de Distribución, Módulo de Borde Corporativo, Módulo de Red de Acceso y Módulo Equipos Terminales.

Este modelo de arquitectura que se aprecia en la Figura 3.24, mantiene el concepto de Nivel de Núcleo, el cual provee un *backbone* de alta velocidad a la sede, a la cual se conectan las equipos del Nivel de Distribución tanto para Acceso a usuarios internos, granja de servidores y el Borde de Distribución, al cual se conecta el Borde Corporativo donde residen la *DMZ*, *Firewalls*, Acceso *VPN*, *WAN* Corporativa y *Gateways* de Voz. El Borde Corporativo presenta conexión directa con la Red de Acceso, donde se encuentra el Internet, *WAN*, y la *RTB*. Y por último los equipos terminales que tienen conexión a los servicios necesitados, estos equipos pueden ser computadores, teléfonos inteligentes, tabletas, centros de datos, e incluso sucursales corporativas que accedan mediante enlaces *WAN* o *VPN* a servicios de *VoIP* en la Nube.

Cabe mencionar también que en redes más pequeñas, los niveles pueden traslaparse en uno solo, incluso un solo dispositivo, pero las funciones permanecen.

En la Figura 3.25, se muestra el mismo esquema modular pero más detallado a nivel de conexiones redundantes de equipos para aumentar la disponibilidad del servicio, aspecto clave para las comunicaciones en la Nube. En este esquema de servicio de *VoIP* sobre una Nube Pública, los usuarios pueden conectarse directamente a la *DMZ* o por medio de *VPN*, dependiendo del sistema del servicio contratado. Así mismo bases de datos

como de facturación, o de clientes, pueden reposar con seguridad en la *LAN* detrás del *firewall*, en la granja de servidores.

Y en la Figura 3.26, en cambio se aprecia un esquema de servicio de *VoIP* sobre una Nube Privada, que generalmente la tendría una empresa o institución para proveer de servicios a sus usuarios. Por facilidad de lectura se simplifica el esquema, pero las funciones de la *DMZ*, *VPN*, *LAN* y *WAN* serían las mismas que para las redes públicas, solo que aquí las redes de acceso de los usuarios corporativos serán el Internet o enlaces privados dedicados, donde las medidas de seguridad podrían ser un poco menores.

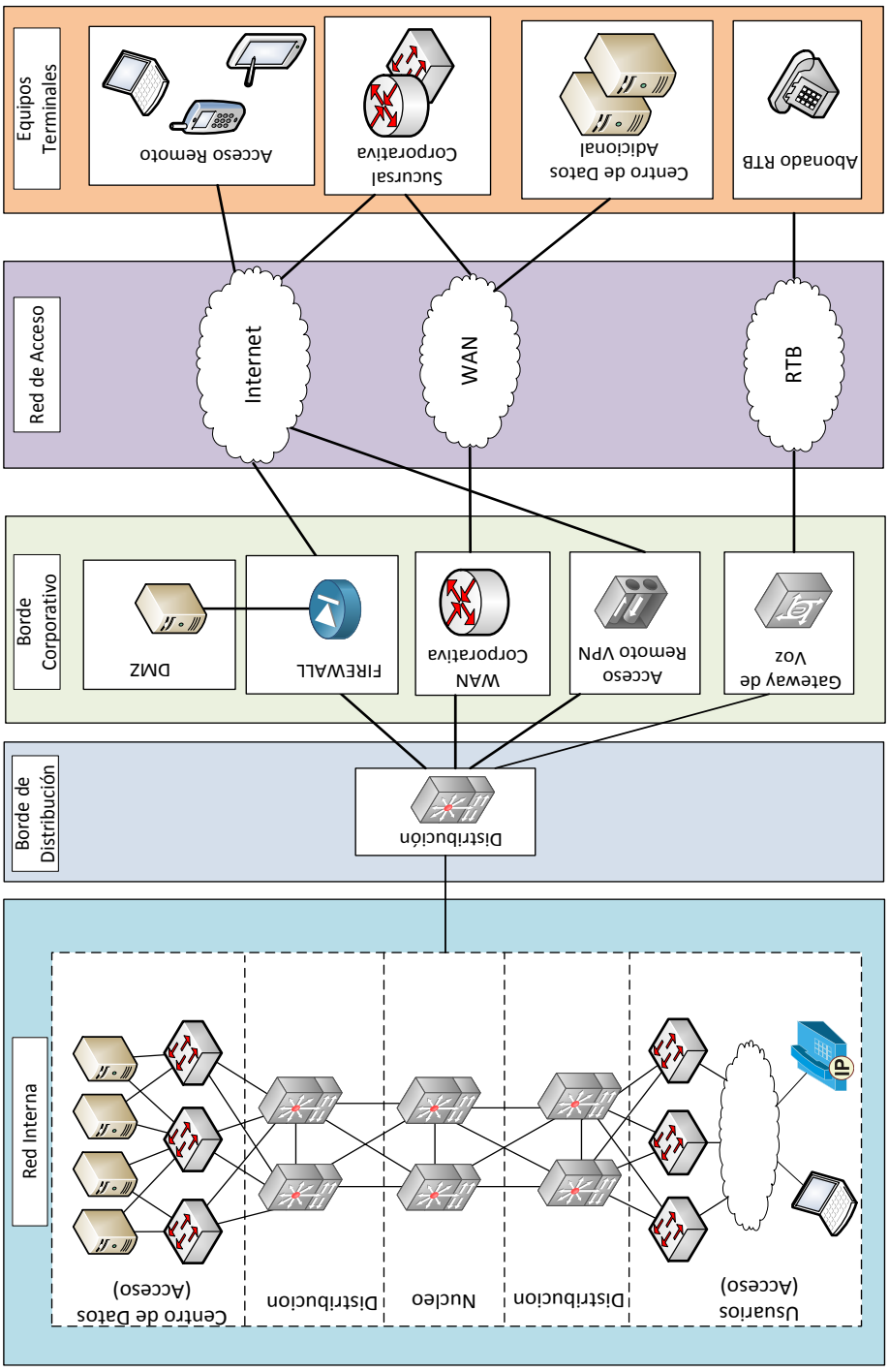


Figura 3.24. Diseño modular para *VoIP* en la Nube

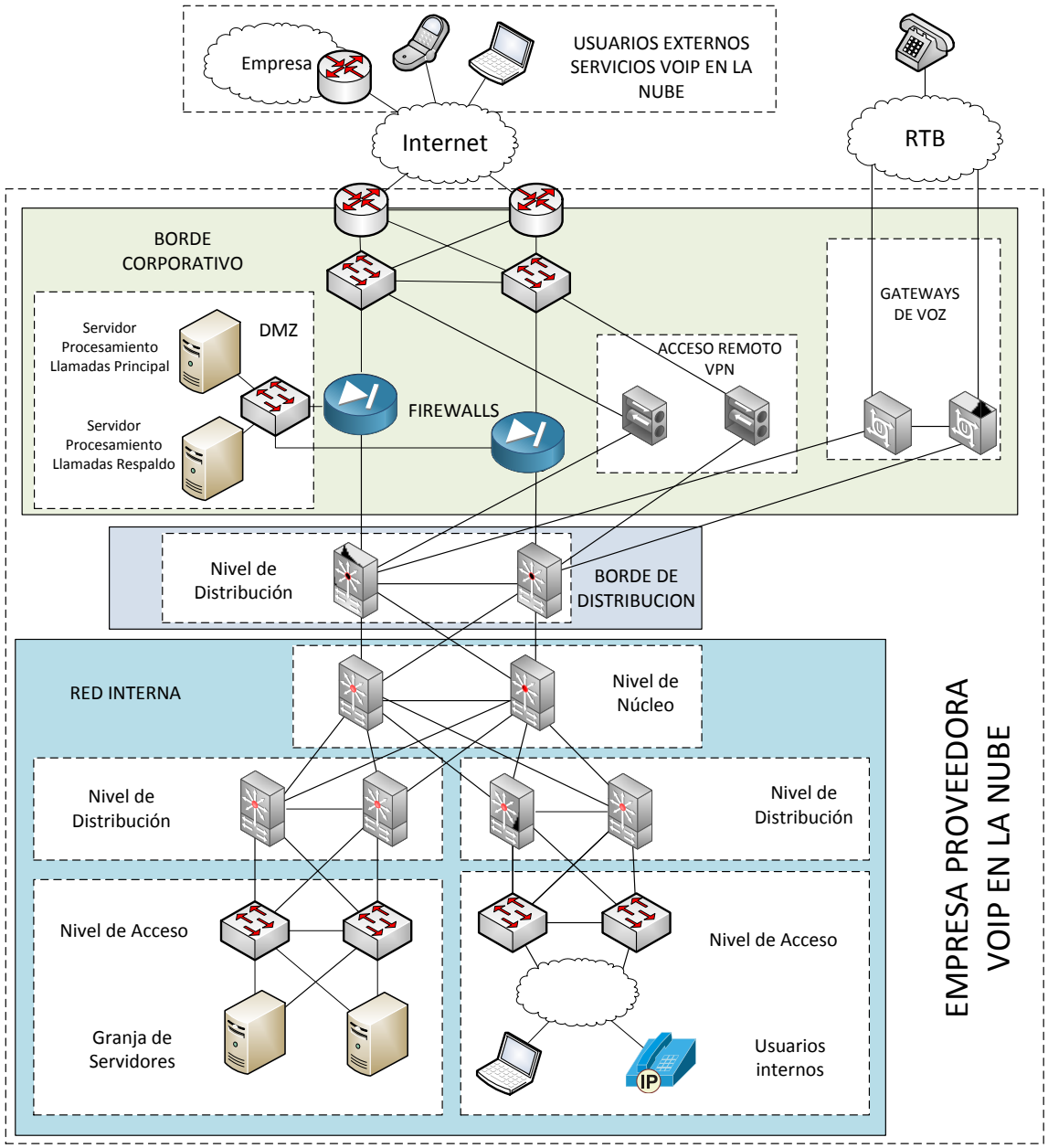


Figura 3.25. Esquema de Servicio *VoIP* para Nube Pública

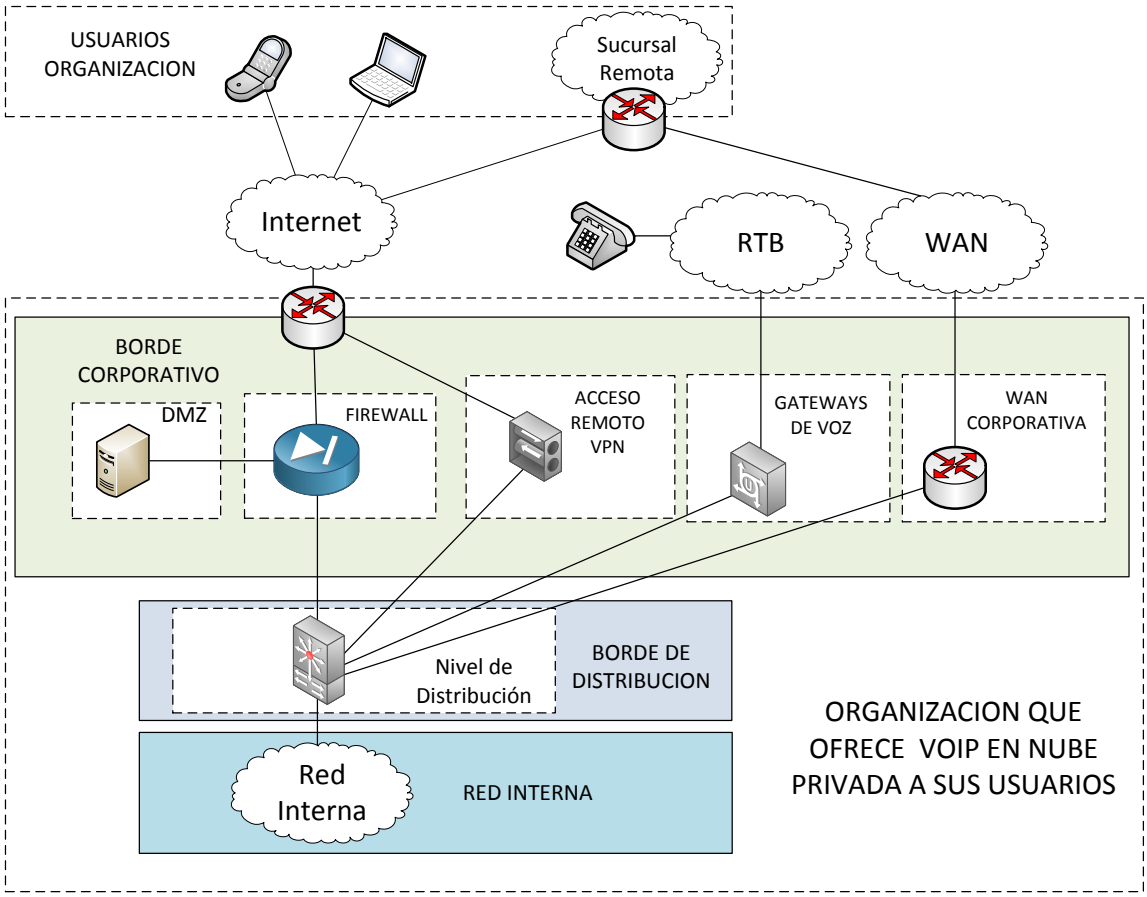


Figura 3.26. Esquema de Servicio *VoIP* para Nube Privada

Por su parte en la Figura 3.27 se muestra un esquema de servicio de *VoIP* sobre una Nube Híbrida, que sería una combinación de las Nubes Privadas y Públicas, y por lo tanto podrán existir varias combinaciones, y en este ejemplo se aprecia un *WAN* privada de una empresa que en una de sus sedes, podría tener la conectividad con el proveedor de servicios en la Nube.

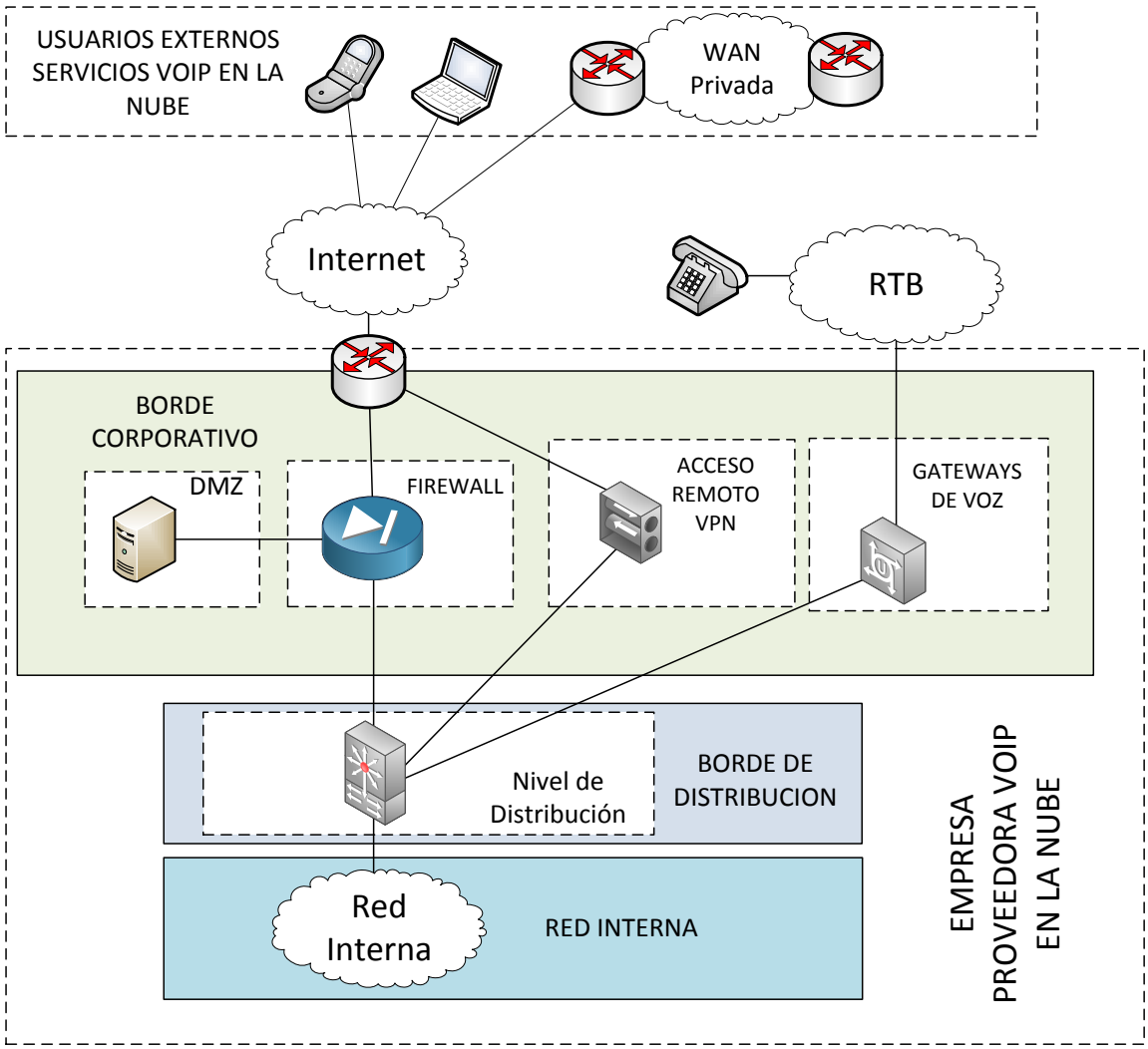


Figura 3.27. Esquema de Servicio *VoIP* para Nube Híbrida

Otro aspecto a considerar es que los usuarios de *VoIP* en la Nube, tienen opciones como del tipo *SaaS* donde los proveedores del servicio, instalan y operan los equipos de procesamiento de llamadas, y por ende los clientes no tiene gestión sobre los mismos. Pero también existen soluciones del tipo

IaaS, donde se pueden tener gestión sobre estos equipos, y tener la opción de tener soluciones más personalizadas, aunque en la mayoría de casos a un costo mayor.

3.5 FLUJOS DE SEÑALIZACIÓN EN LA NUBE

En la Figura 3.28 se muestra un ejemplo de un esquema en el que los clientes *SIP* pueden ser teléfonos *IP* o equipos terminales con un *software* preexistente que haga las mismas funciones de un cliente *SIP*.

SIP al utilizar nombres con esquema tipo *URI* (Identificador Uniforme de Recursos, del inglés, *Uniform Resource Identifier*) para el direccionamiento de llamadas. Los *SIP URI* pueden manejar números de teléfono, parámetros de transporte y un número de otros artículos. Lo más importante de los mismos, aunque no es parte del presente trabajo es que un *SIP URI* es un nombre que se resuelve en una dirección *IP* mediante un servidor *SIP Proxy* y la búsqueda de DNS en el momento de la llamada [17].

En la Figura 3.28 se muestra un ejemplo donde el cliente A llama al cliente B a través de dos servidores *SIP Proxy*, los mismos que se encuentran en la Nube de Internet, para lo cual se realiza principalmente el registro de los equipos terminales a los mismos, por medio de mensajes REGISTER. En la misma figura el Cliente *SIP* A realiza una llamada al Cliente *SIP* B, a través

de mensajes INVITE, los cuales son redireccionados por los respectivos servidores *SIP Proxy* a cual pertenecen ambos clientes. Una vez que recibe la invitación el cliente *SIP* B envía una confirmación aceptando la llamada a través del mensaje 200 OK, y el cliente *SIP* A envía su confirmación por medio del mensaje ACK.

En la mayoría de casos los clientes *SIP* no cuentan con un dirección *IP* pública, sino más bien de tipo privada, por lo tanto es necesario la utilización del protocolo *ICE* para que pueda existir comunicación entre ambos clientes, lo cual se detalla más detalladamente en el Apéndice C.

Una vez que existe la comunicación directa entre ambos clientes, los paquetes de voz son transportados por medio de *RTP*.

El paso a continuación una vez se desea concluir la llamada, es enviar un mensaje de tipo BYE, el cual es confirmado por el otro cliente por medio de un mensaje 200 OK.

Cabe considerar que al ser modelos de procesamiento de llamadas en la Nube podrían ser de tipo *SaaS* donde las llamadas gestionadas por un proveedor externo, que normalmente sería en la mayoría de casos, pero también podría existir la opción tipo *IaaS* donde se puede tener gestión directa de los equipos tanto de procesamiento como de acceso a otras redes telefónicas como puede ser la *RTB*.

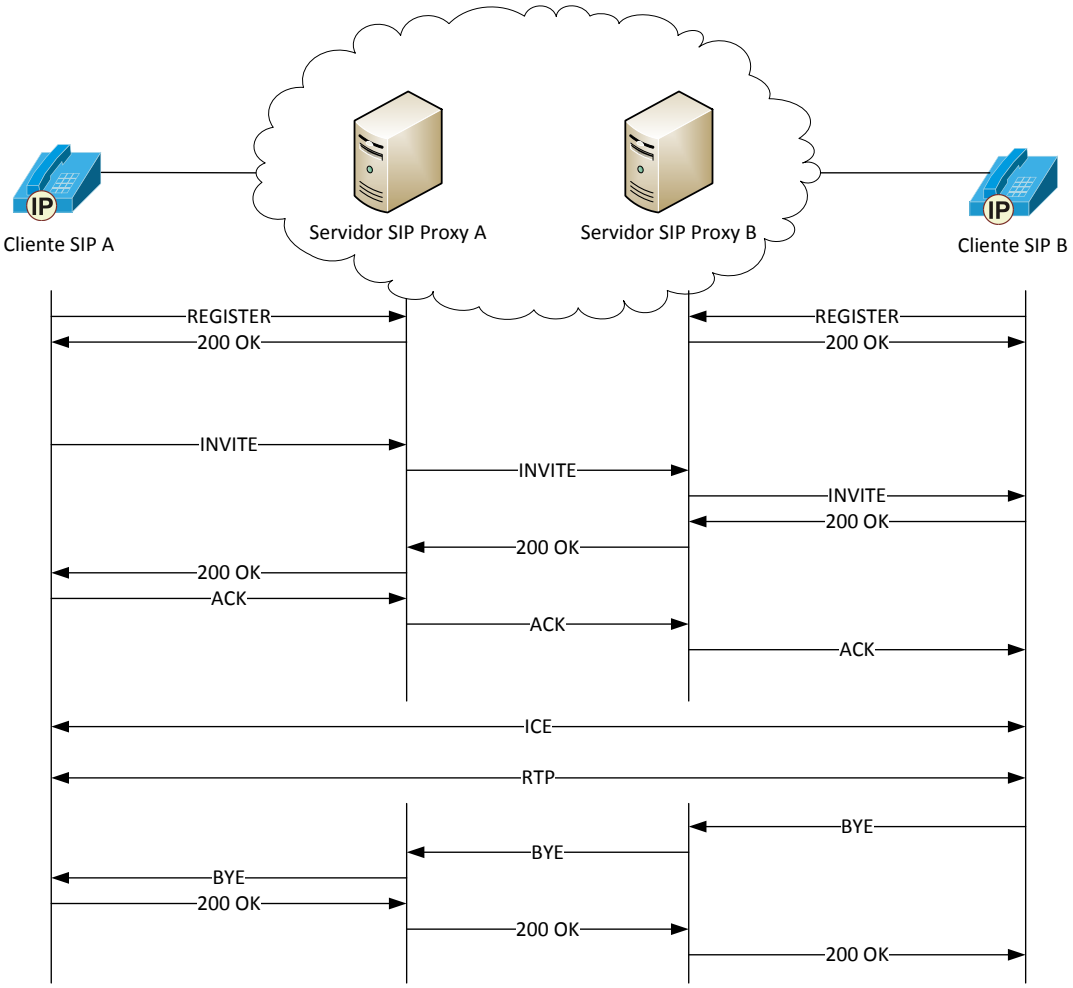


Figura 3.28. Esquema de señalización de *VoIP* en la Nube para un esquema de clientes en modalidad *SaaS*. [17]

CAPÍTULO 4

HTML5

4.1 INTRODUCCIÓN

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 computador, teléfono inteligente, tableta, o algún otro capaz de entender este tipo de contenido [37], 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 "*Markup*

Language” 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*. W3C ha estado activo desde el comienzo mismo de la Web. Este organismo de normalización ha desarrollado varias versiones de *HTML* con el paso de los años. La última versión en alcanzar la etapa final de la recomendación fue *HTML* 4.01 en 1999, conocida básicamente como *HTML*. De allí 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 [38]. Aunque *HTML5* no ha sido finalizado, casi todas sus etiquetas pueden ser utilizadas con seguridad en las páginas Web de hoy.

4.2 DIFERENCIACIÓN ENTRE HTML4 Y HTML5

HTML5 se distingue principalmente en dos categorías de *HTML*: Estructura y *Media*.

Estructura: la mayoría de las páginas Web de hoy en día están típicamente estructuradas por divisiones genéricas a través de la etiqueta <*div*>. Por tanto, un diseño que requiera cabecera, contenido principal, y área de pie de página tendría un mínimo de tres etiquetas <*div*>. *HTML5*, por el contrario, ofrece etiquetas específicas como por ejemplo <*header*> y <*footer*>, así como las de contenido, tales como <*article*> y <*summary*> [39]. En la Figuras 4.1 y 4.2 se muestran las estructuras semánticas tanto de *HTML*4 y *HTML5*.

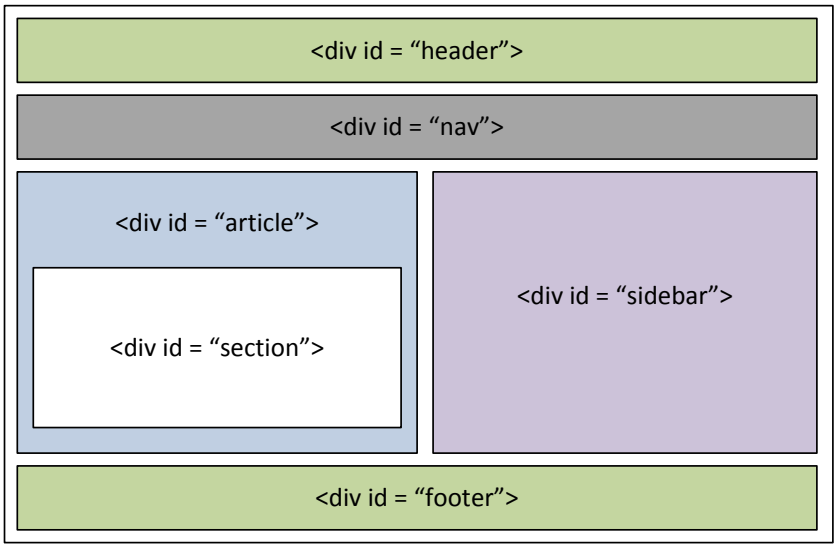


Figura 4.1. Estructura semántica de *HTML*4. [39]

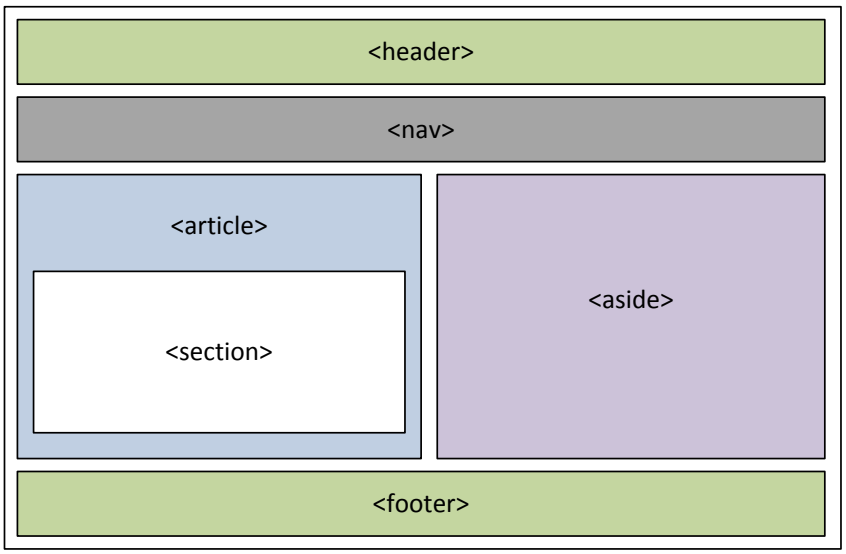


Figura 4.2. Estructura semántica de *HTML5*. [39]

HTML5 contiene numerosos otros elementos estructurales para la manipulación de figuras, formas y navegación también. La mayoría de estos aún no han sido aplicados por los navegadores actuales, pero siguen en fase de desarrollo. En la Tabla 11 se aprecia los nuevos elementos que aporta *HTML5*.

Categoría	Elementos
Elementos semánticos para estructurar una pagina	<article>, <aside>, <figcaption>, <figure>, <footer>, <header>, <hgroup>, <nav>, <section>, <details>, <summary>
Elementos semánticos para el texto	<mark>, <time>
	<wbr> (previamente soportado pero no oficialmente parte del lenguaje)
Formas Web e interactividad	<input> (no nuevo, pero tiene muchas subclasificaciones)
	<datalist>, <keygen>, <meter>, <progress>, <command>, <menu>, <output>
Audio, video y plug-ins	<audio>, <video>, <source>
	<embed> (previamente soportado pero no oficialmente parte del lenguaje)
Canvas	<canvas>

Tabla 11. Nuevos Elementos *HTML5.* [37]

Media: en versiones como *HTML*4 y anteriores, si se deseaba mostrar una animación o reproducir un vídeo, era necesario utilizar en un navegador un complemento (del inglés, *plugin*), como por ejemplo el reproductor de *Adobe Flash Player*. En cambio *HTML5* incluye soporte nativo para reproducir vídeo y audio a través de las etiquetas <*video*> y <*audio*>, respectivamente. Por esta razón en este proyecto entenderemos por *Media* la comunicación de audio y/o imágenes en movimiento, a través de flujos, que son un conjunto de paquetes que viajan por la red de manera secuenciada y dependiente.

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 [37].

4.3 CSS3

CSS u Hojas de estilo en cascada son un mecanismo utilizado para agregar estilo que incluye tipo de fuente, color y diseño de 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 X*HTML* (*Lenguaje de marcado de hipertexto extensible*, del inglés, *eXtensible HyperText Markup Language*) [38]. *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.

Debemos considerar que *HTML* contiene las estructuras, mientas *CSS* da el formato al contenido estructurado [38]. La lista de códigos muestra cómo

CSS está incrustado en un documento *HTML* usando la etiqueta <*style*> como se muestra en la Figura 4.3.

```
<! DOCTYPE HTML>
<HTMLLANg="en-us" >
<head>
<title>Font Test</title>

<style>
@font-face {
font-family: EraserDust;
}
body {
background: #333333;
font-family: EraserDust;
font-size: 24pt;
}
</style>

</head>
<body>
<p>Esto es una prueba</p>
</body></HTML>
```

Figura 4.3. Código *HTML* con *CSS*

Las diferentes versiones de *CSS* también se denominan niveles de *CSS*. Cada nivel de *CSS* se basa en el nivel anterior y agrega nuevas propiedades y características [39].

Las tres versiones más significativas son *CSS1*, CSS 2.1 y *CSS3*. Las especificaciones que forman *CSS3* son llamados módulos *CSS*. La especificación *CSS3* está todavía en desarrollo por el W3C.

4.4 DEFINICION DE API

API son 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 [40]. Estas *API* permiten mayor versatilidad y facilidad al desarrollar programas, porque los programadores no tiene la necesidad de programar desde un principio cada detalle.

Cuando se utilizan en el contexto del desarrollo de la Web, existen básicamente dos maneras de realizarlos, utilizando *REST* (Transferencia de Estado Representacional, del inglés, *Representational State Transfer*) directamente o desde *JavaScript*. *REST* es un estilo de arquitectura de *software* que proporciona un enfoque práctico y consistente a solicitudes y modificación de datos, y son realizados utilizando un conjunto de mensajes de petición *HTTP* como por ejemplo *POST, GET, PUT* o *DELETE* [41]; o utilizando llamadas en *JavaScript* utilizando mensajes *JSON* (Notación de Objetos *JavaScript*, del inglés, *JavaScript Object Notation*).

4.5 DEFINICIÓN DE SCRIPT

Un *script* es un código de programa que no necesita pre-procesamiento antes de que se ejecute [39]. En el contexto de un navegador Web, los *script* generalmente se refieren a códigos de programa escritos en *JavaScript* que son ejecutados por el navegador, cuando una página se descarga o en

respuesta a un evento activado por el usuario. Por lo cual los *scripts* pueden hacer a las páginas Web más dinámicas. Por ejemplo, sin volver a actualizar una página Web, permite modificaciones en el contenido de la página, o permite que el contenido sea enviado desde la página. Esta interactividad adicional hace que las páginas Web se comporten como una aplicación de *software* tradicional.

4.6 JAVASCRIPT

JavaScript ha existido mucho antes de *HTML5* y es uno de los más populares lenguajes de programación de la Web hoy en día, y es una plataforma orientada a objetos desarrollado por Netscape. Es un lenguaje ligero que no puede funcionar independiente (del inglés, *standalone*), sino que debe ser utilizado dentro de otros medios como documentos Web y navegadores [37].

A nivel general existe la idea errónea, que *JavaScript* tiene relación con *Java*. *JavaScript* es un lenguaje de *script* que reside en documentos *HTML*, por lo que sólo se ejecuta en un navegador, mientras *Java* es un lenguaje de programación que puede funcionar en *standalone*, y fue desarrollado por *Sun Microsystems*. En consecuencia, es importante considerar que un navegador habilitado para *Java* no es automáticamente un navegador habilitado para *JavaScript*.

En los documentos Web, *JavaScript* funciona mediante la identificación de elementos de la página y su posterior modificación o inserción de nuevos elementos. Por ejemplo se puede utilizar *JavaScript*, para cambiar dinámicamente el texto en una parte de la página Web cuando el usuario pasa sobre otra parte de la página con el ratón.

El código de la Figura 4.4 muestra un ejemplo de cómo *JavaScript* se inserta en un documento *HTML* usando la etiqueta <*script*>. En el ejemplo aparece un cuadro de alerta, el mismo que es realizado por *JavaScript*.

```
<!DOCTYPE HTML>
<HTMLLANg="en">
<head>
<meta charset="utf-8">
<title>A Simple JavaScript Example</title>
</head>
<body>
<p>En algun punto del procesamiento de la pagina, un script
aparecera.</p>
<script>
alert("Interrumpimos la pagina con un anuncio en JavaScript");
</script>
<p>Si esta aqui, es porque se vio el script.</p>
</body>
</HTML>
```

Figura 4.4. Código *HTML* con *JavaScript*

Para identificar los diversos elementos de la página, *JavaScript* utiliza el Modelo de Objetos para Documentos (*DOM* del inglés, *Document Object Model*). Y es esencialmente, una hoja de ruta a cualquier página Web

determinada. *JavaScript* utiliza el *DOM* para identificar un elemento preciso en una página Web, y luego analizar, modificar o eliminar su contenido [37].

JavaScript es un lenguaje de programación muy potente. También se ha convertido en el único lenguaje que los navegadores más populares, comparten soporte. Es el lenguaje de los navegadores Web, y ahora con *HTML5* es el único lenguaje para *scripts*, aprobado en su *draft* más reciente [42].

Con todo ello *HTML5* pretende incluir las tres siguientes especificaciones:

1. *HTML* 4.01 (*HTML*4)
2. eXtensible *HTML* 1.1 (X*HTML*1)
3. *DOM* Level 2 *HTML* (*DOM2 HTML*)

X*HTML* es una variante de *HTML* que utiliza la sintaxis de *XML* (Lenguaje de marcas extensible, del inglés, eX*tensible Markup Language*). X*HTML* tiene todos los mismos elementos que *HTML*, pero la sintaxis es ligeramente diferente.

HTML5 también estandariza muchas de las características que han sido utilizados por desarrolladores Web durante años, pero no han sido formalmente documentadas como normas.

A diferencia de anteriores especificaciones, las *API* y *DOM* son partes fundamentales de la especificación *HTML5*. Además, para muchas nuevas características sintácticas tales como un número de nuevas etiquetas, *HTML5* también introduce varias *API* nuevas para el desarrollo de aplicaciones Web complejas.

4.7 COMPATIBILIDAD DE NAVEGADORES

Todos los principales navegadores como Google Chrome, Opera, Safari, Internet Explorer y Mozilla Firefox soportan *HTML5* en mayor o menor grado a pesar de que las especificaciones de *HTML5* aún están en desarrollo.

La Tabla 12, provista por *html5test.com*, muestra los resultados de sus pruebas de la forma que los navegadores dan soporte a *HTML5*, la cual está basada en un puntaje máximo de 500 puntos, y 15 puntos adicionales asignados por los elementos de *audio* y *video*, y como las especificaciones aún no definen un Códec final, por esta razón por cada Códec que soporta se entrega este puntaje adicional. Como el estándar sigue en desarrollo, muy probablemente los esquemas de puntuación varíen en un futuro cercano.

Navegador	Versión	Puntaje /500	Extra /15
Chrome	23	448	13
Opera	12.10	419	9
Safari	6.0	378	8
Firefox	16	372	10
Internet Explorer	10	320	6

Tabla 12. Comparación de navegadores frente a *HTML5*

En la Figura 4.5, se muestra la evolución del soporte de los distintos navegadores a *HTML5* desde el 2009, según la Web *HTML5test.com*.

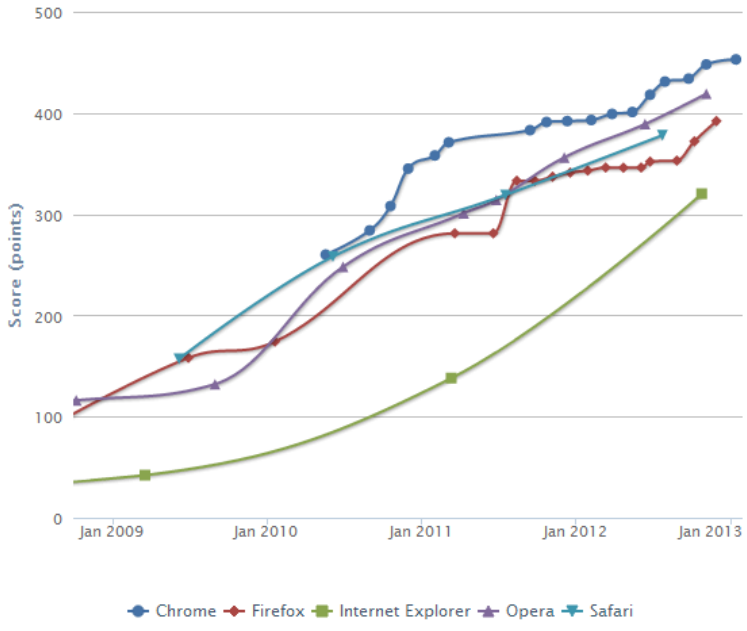


Figura 4.5. Soporte acumulativo a *HTML5* desde el año 2009

CAPÍTULO 5

UTILIZACIÓN DE HTML5 PARA LA SEÑALIZACIÓN EN LAS TECNOLOGÍAS DE VOIP EN LA NUBE

5.1 INTRODUCCIÓN

Para el transporte de señalización a través de *HTML5* es necesario introducir el concepto de *WebSocket*, sin el cual no es posible la integración de señalización en plataformas Web.

5.1.1 WebSocket

WebSocket define un protocolo que habilita la comunicación *full-dúplex*, sobre la cual se pueden enviar mensajes entre un cliente (navegador) y un servidor. Esto se realiza mediante una conexión simple en *TCP*, en lugar de

la creación de una nueva conexión *TCP* cada vez que el servidor o el cliente desean comunicarse. *WebSocket* no podría ser definido como solo una mejora a las comunicaciones *HTTP*, sino con un avance considerable para las comunicaciones en tiempo real en aplicaciones Web, simplificando la complejidad de una comunicación bidireccional, la gestión de sus conexiones y que permite un aumento la velocidad de las mismas.

El protocolo definido para *WebSocket*, fue normalizado a través de la RFC 6455 [43] y fue desarrollado como parte de la iniciativa *HTML5*. El protocolo definido para *WebSocket* necesita ser combinado con el *API* para *WebSocket* para proveer comunicación bidireccional, este *API* está siendo normalizado por el W3C [44]. El *API* para *WebSocket* habilita a las páginas Web para utilizar *WebSocket* para la comunicación de dos vías con un *host* remoto, a través de una *socket* único en la Web.

Un *socket* puede ser definido como la combinación de una dirección *IP* y un puerto, y a través del cual dispositivos finales pueden intercambiar información en la red [45]. Los *sockets* son comúnmente utilizados para la interacción entre clientes y servidores.

En una configuración típica de un sistema, los clientes se conectan al servidor, intercambian información, y luego se desconectan [46]. Por lo cual los servidores se encuentran siempre en modo de escucha (del inglés,

listening). En la Figura 5.1 se muestra un organigrama de funcionamiento de la comunicación Cliente-Servidor mediante *sockets*.

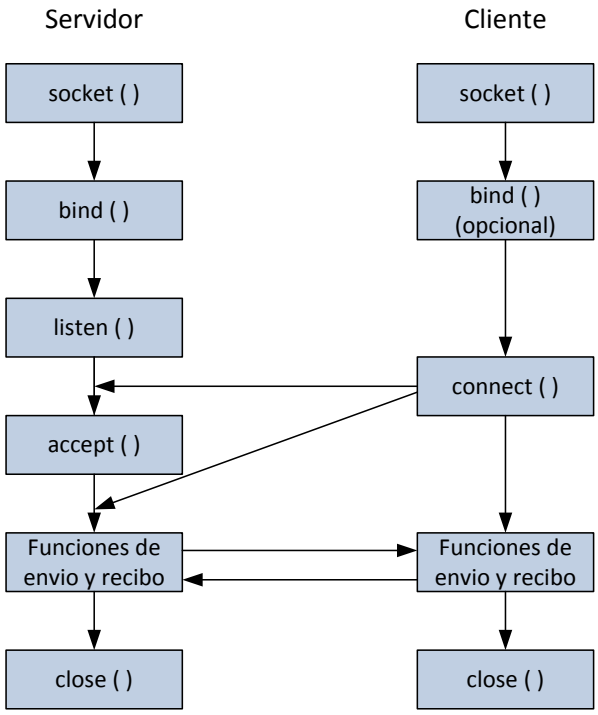


Figura 5.1.Organigrama de comunicación típico Cliente-Servidor. [46]

Por esta razón es que los navegadores sólo admiten mecanismos de extracción del cliente (del inglés, *client pull mechanisms*), y cuando se desarrolló *HTTP* no se proporcionó directamente a los desarrolladores Web, la capacidad de inserción por el servidor (del inglés, *server push*) [46]. Por tanto los servidores no eran capaces de informar a los clientes cuando existía alguna actualización, como por ejemplo cuando un navegador visita una página Web, una solicitud *HTTP* se envía al servidor Web que aloja la

página, el servidor Web acepta la solicitud y devuelve una respuesta al cliente. Por ello, si deseaba tener comunicación en tiempo real, era necesario refrescar manualmente la página Web de manera constante, lo cual no era eficiente.

Consecuentemente los navegadores empezaron a utilizar sondeo (del inglés, *Polling*) [47], en el cual el navegador envía peticiones *HTTP* a intervalos regulares e inmediatamente recibe una respuesta. Esta técnica fue el primer intento de los navegadores para ofrecer información en tiempo real, y es una buena solución si es conocido exactamente el intervalo de entrega de los mensajes, sin embargo, los datos en tiempo real a menudo no son previsibles, haciendo inevitable peticiones innecesarias y como resultado, muchas conexiones se abren y cierran innecesariamente. Como se aprecia en la Figura 5.2, el modelo de conexión es *half-duplex*, lo cual es uno de las principales inconvenientes de este tipo de tecnología.

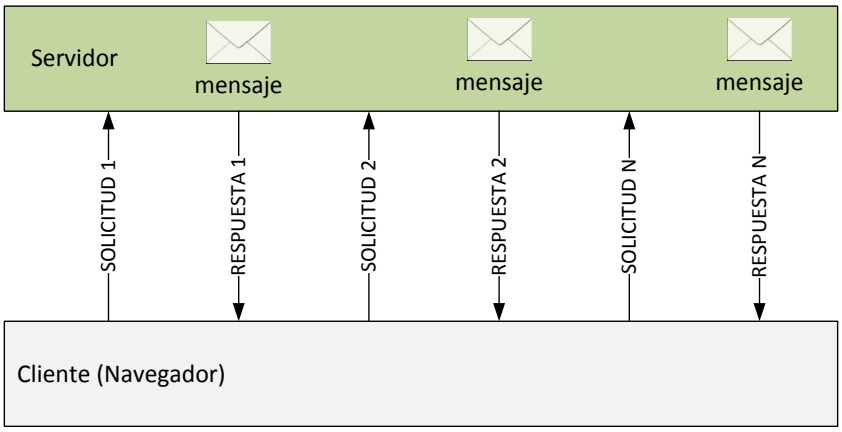


Figura 5.2. Arquitectura *Polling*. [47]

Este método y subsiguientes para proveer comunicación en tiempo real como sondeo largo (del inglés, *long-polling*) y *streaming*, implicaban peticiones y respuestas *HTTP*, que contenían gran cantidad de datos y cabeceras, que introducían latencia y tráfico en la red innecesarios.

Para una comunicación *full-duplex* se requieren dos conexiones, una de subida (del inglés, *upstream*) y otra de bajada (del inglés, *downstream*), lo cual aumenta la complejidad del sistema para mantener y coordinar esta conexión. Por ello *HTTP* no fue diseñado para comunicación en tiempo real. Mientras que *WebSockets* proporciona una enorme reducción en el tráfico innecesario en la red y la latencia en comparación con *Polling* y *Streaming,* que se utilizan para emular una conexión *full-duplex* [48].

El protocolo definido para *WebSocket*, fue diseñado para trabajar con la infraestructura Web existente. Como parte de este principio de diseño, la especificación del protocolo define que la conexión *WebSocket* comienza su vida como una conexión *HTTP*, lo que garantiza la plena compatibilidad con el mundo pre *WebSocket*. Luego el protocolo solicita conmutar de *HTTP* a *WebSocket*, a esto se lo conoce como un apretón de manos *WebSocket (del inglés, WebSocket handshake*), por lo que el navegador envía una petición al servidor, que indica que quiere cambiar de *HTTP* a *WebSocket*. Seguido el cliente expresa su deseo a través de la actualización de la cabecera (del inglés, *upgrade header*). En este punto la conexión *HTTP* se rompe y se sustituye por la conexión *WebSocket* sobre la misma conexión *TCP*. La

conexión *WebSocket* utiliza los mismos puertos de *HTTP* (80) y HTTPS (443), de manera predeterminada [43].

Una vez establecida, las tramas de datos *WebSocket* se pueden enviar de ida y vuelta entre el cliente y el servidor en modo *full-duplex*. Ambas tramas de texto o binario puede ser enviadas en ambas direcciones al mismo tiempo, tal como se muestra en la Figura 5.3.

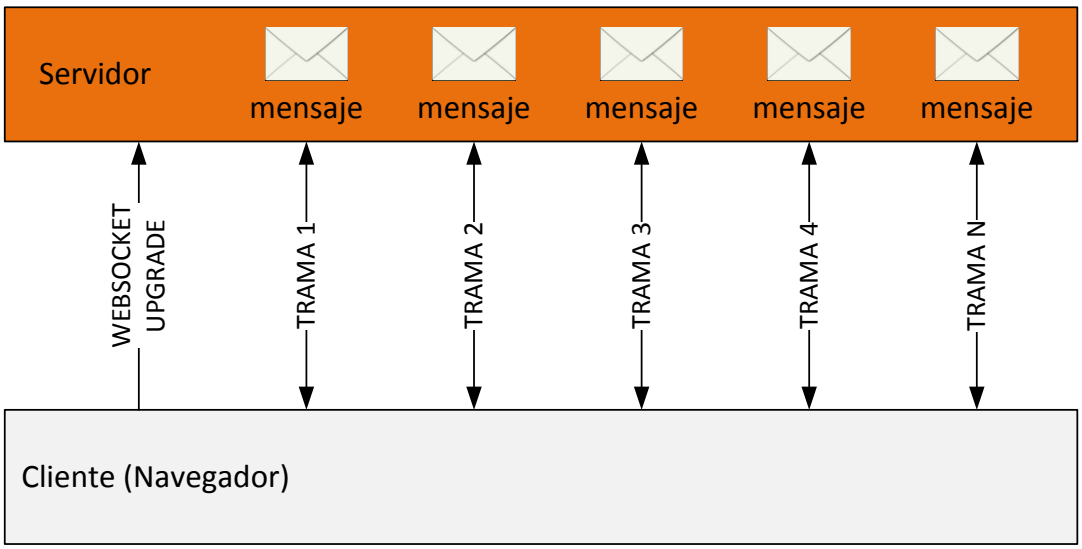


Figura 5.3. Arquitectura *WebSocket*. [47]

El *API* para *WebSocket* que funciona sobre *HTML5* es la primera estandarización para ayudar a los navegadores a implementar un mecanismo de *server push*. El *API* para *WebSocket* requiere de soporte de

componentes del servidor. Por lo tanto, no es suficiente si solo se implementa en el navegador el *API* para *WebSockets*, el servidor también tendría que implementar el protocolo para soportar esta comunicación.

Más detalles sobre la conexión *WebSockets* y el *API* para *WebSocket* se presentan en el Apéndice A.

5.2 HTML5 PARA EL TRANSPORTE DE ARCHIVOS EN TIEMPO REAL

Para el transporte de archivos, y en el caso del presente estudio de la voz, a través de *HTML5*, debemos considerar dividirlo en dos partes, el uno en lo concerniente a la señalización, y el otro con respecto al transporte de la voz. Para el primero utilizaremos *WebSockets* y para el segundo *WebRTC* (Comunicaciones Web en Tiempo Real, del inglés, *Web Real Time Communications*), que se detalla en la sección 5.4; donde ambos utilizan como soporte *HTML5*.

5.2.1 WebSockets para el transporte del protocolo de señalización

Como los *WebSockets* habilitan el intercambio de mensajes entre clientes y servidores a través de una infraestructura *HTTP* existente, esta herramienta puede ser utilizada para transporte de mensajes de señalización, y en el presente estudio de *SIP*. Esta misma especificación está siendo

desarrollada actualmente en la cual se define la utilización de un nuevo protocolo basado en *WebSocket* para el transporte de *SIP* [49]. En la misma tenemos dos agentes: el Cliente *SIP WebSocket*, el cual es una entidad *SIP* capaz de abrir conexiones salientes a servidores *WebSockets*, y el Servidor *SIP WebSocket*, que por lo contrario es capaz de escuchar conexiones entrantes. Ambos se comunican utilizando el protocolo basado en WebSocket para transportar mensajes de *SIP*, el cual se puede definir como un protocolo a nivel de aplicación sobre una conexión *WebSockets* para llevar solicitudes y respuestas *SIP*.

Como se describió en la sección 5.1.1, cuando el protocolo conmuta de *HTTP* a *WebSocket*, se lo conoce como *WebSocket handshake*. Ahora en este *handshake* inicial, el cliente debe incluir el valor de *SIP* en su cabecera, específicamente en *Sec-WebSocket-Protocol*, como se aprecia en la Figura 5.4.

```
GET / HTTP/1.1
Host: sip-ws.example.com
Upgrade: Websocket
Connection: Upgrade
Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
Origin: HTTP://www.example.com
Sec-WebSocket-Protocol: sip
Sec-WebSocket-Version: 13
```

Figura 5.4. Handshake inicial del cliente para el subprotocolo *WebSocket SIP*. [49]

El paso siguiente consiste en que el servidor debe responder con el mismo tipo de mensaje, como se observa en la Figura 5.5.

```
HTTP/1.1 101 Switching Protocols
Upgrade: Websocket
Connection: Upgrade
Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=
Sec-WebSocket-Protocol: sip
```

Figura 5.5. Respuesta del servidor para el protocolo basado en WebSocket para transporte de mensajes de *SIP*. [49]

Con ello se establece la conexión inicial para el transporte de mensajes *SIP* siguientes, los mismos que se envían de manera individual en un solo mensaje *Websocket* cada uno.

Un ejemplo de la conexión utilizando el protocolo basado en *WebSocket* para transportar mensajes de *SIP* se presenta en el Apéndice B.

5.3 LECTURA DE MUESTRAS DE AUDIO

HTML5 ofrece la posibilidad de incorporar fácilmente elementos de *Media* en documentos a través de las etiquetas <*audio*> y <*video*> sin la necesidad de instalar un complemento. Pero los mismos son solo para reproducción. En el caso del presente estudio se necesita tener la posibilidad de comunicar voz en tiempo real, por lo cual se necesitan Códec de voz para la codificación y

decodificación de la voz, y una *API* que nos permita el acceso a dispositivos de equipo como el micrófono.

5.3.1 Codificación de Audio

Los Códec mínimos a implementar para asegurar la comunicación por *WebRTC* (que se detalla en la sección 5.4.1), son los siguientes [50]:

1. Opus
2. G.711
3. *Telephone event*

Las recomendaciones indican que tanto Opus como G.711 se deben implementar en *WebRTC*. La razón de esta decisión es que Opus es capaz de soportar audio de alta calidad a través de tasas de bits variables, mientras que G.711 se incluye como una opción de respaldo cuando existan situaciones complejas de interoperabilidad e integración sobre todo con equipos pertenecientes a la *RTB* [51]. Esto radica en que G.711 es un Códec PCM de banda estrecha, el cual fue diseñado para redes de conmutación por circuitos con tasas de bits fijas de 64 Kbps, en sus dos versiones *u-law* (utilizada en Norteamérica) y *a-law* (utilizada en Latinoamérica, Europa y resto del mundo) y que no implementa compresión, por lo cual no experimenta retraso por compresión.

Mientras que Opus es un Códec abierto y estandarizado mediante RFC 6716 [52], y fue diseñado específicamente para redes de conmutación por

paquetes. Se puede ajustar perfectamente entre tasas altas y bajas de bits, las cuales van desde 6 kbps en canal de banda estrecha mono hasta 510 Kbps para música estéreo.

Pero al no ser un estándar *WebRTC* sino en desarrollo bajo código abierto se pueden agregar nuevos Códecs de acuerdo a las preferencias de los desarrolladores.

Mientras que *Telephone Event*, es considerado para la transmisión de tonos DTMF para señalización por medio de paquetes *RTP*.

El *API* que realiza la función de acceso a elementos del dispositivo como micrófono es *MediaStream API*, que tiene embebido el Códec a utilizar (se explica en la sección 5.4.1).

5.3.2 Decodificación de Audio

Como se muestra en la Figura 5.6, el proceso de decodificación de audio, es similar a la codificación realizada por el Códec seleccionado por el desarrollador, solo que a éste se adiciona un *buffer* de reproducción, para contrarrestar el *jitter*. Otro aspecto a considerar es que en el proceso existen algunos retrasos, que se producen en el transporte de la voz, como son los retrasos producidos por el procesamiento del codificador, paquetizador, *buffer* de reproducción, despaquetizador y codificador, y a este sumado el

retraso por el transporte de los paquetes por Internet (recomendable que no sea mayor a 150 ms), como se detalló en la sección 3.4.

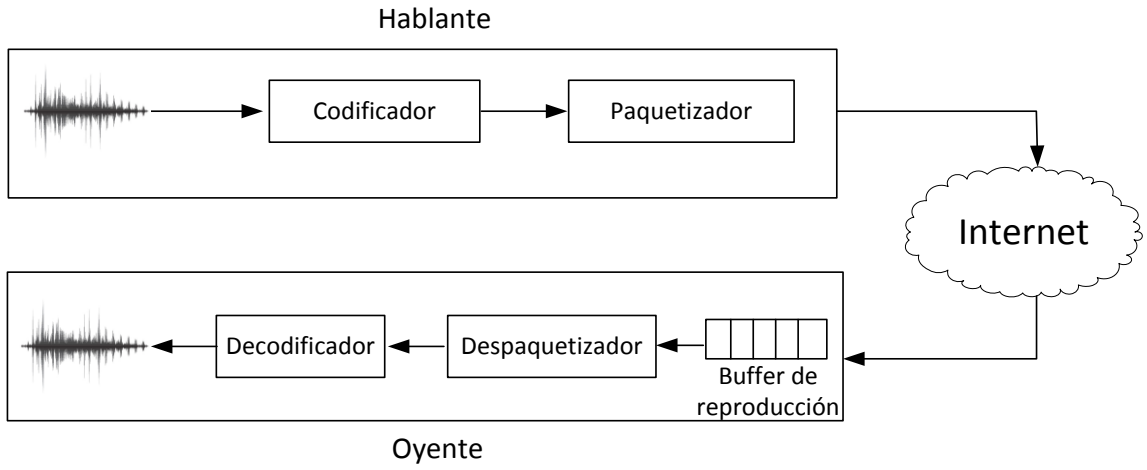


Figura 5.6. Proceso de codificación y decodificación del audio en la Nube. [30]

5.4 TRANSPORTE DE AUDIO

Para el transporte de audio y específicamente de la voz en los navegadores Web, materia del presente estudio, proponemos *WebRTC*, que habilita comunicación *UDP*, en vez de *TCP* tradicional en este tipo de clientes. En los siguientes apartados presentamos el proyecto *WebRTC* y su funcionamiento específico.

5.4.1 WebRTC

WebRTC es un proyecto de código abierto y normalizado por dos entidades, los protocolos e interactividad lo está realizando el *IETF*, por medio del *IETF*

RTCWeb, mientras que el *API* para el desarrollo Web lo está realizando el W3C, por medio del *WebRTC Working Group* [56]. Todo esto en conjunto para habilitar la transferencia de audio, video y datos entre navegadores Web o equipos terminales, sin la necesidad de instalar *plugins* o *native apps*, sino mediante la utilización de *API*s en *JavaScript* y *HTML5*, para el transporte de *RTP* [53].

Si se analiza retrospectivamente, en el inicio se tenía comunicación *half-duplex* entre un navegador y un servidor Web por medio de *HTTP*, después por medio de *WebSockets* se pudo entre los dos tener una comunicación *full-duplex*, y ahora con *WebRTC* se puede tener la posibilidad de *comunicación full-duplex* entre navegadores, como se aprecia en la Figura 5.7 [54].

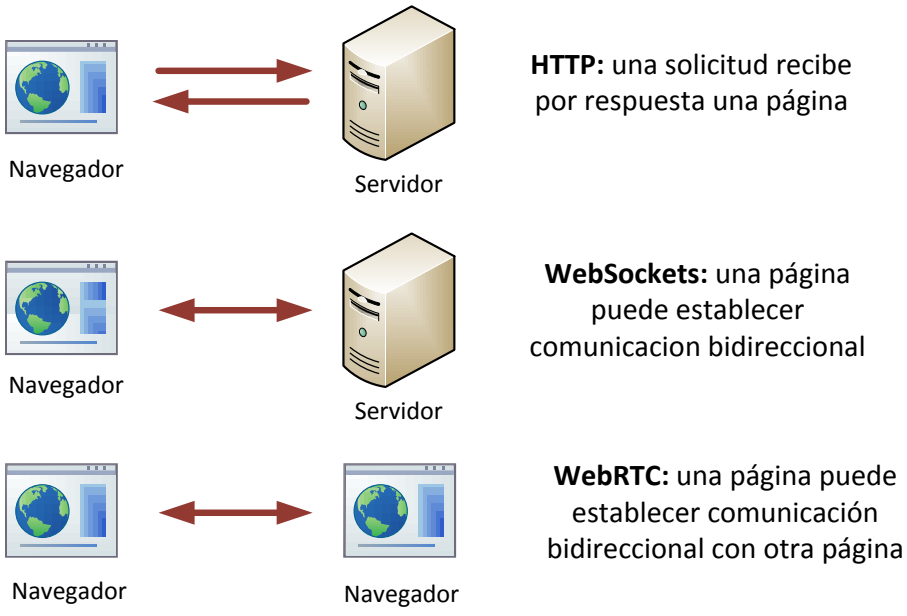


Figura 5.7. Evolución de comunicación vía Web. [54]

En la Figura 5.8 se muestra el modelo de un navegador y el rol de las funciones de comunicación en tiempo real. *WebRTC* interactúa con las aplicaciones en servidores Web utilizando *API* estándares, y se comunica con el sistema operativo utilizando el navegador [56].

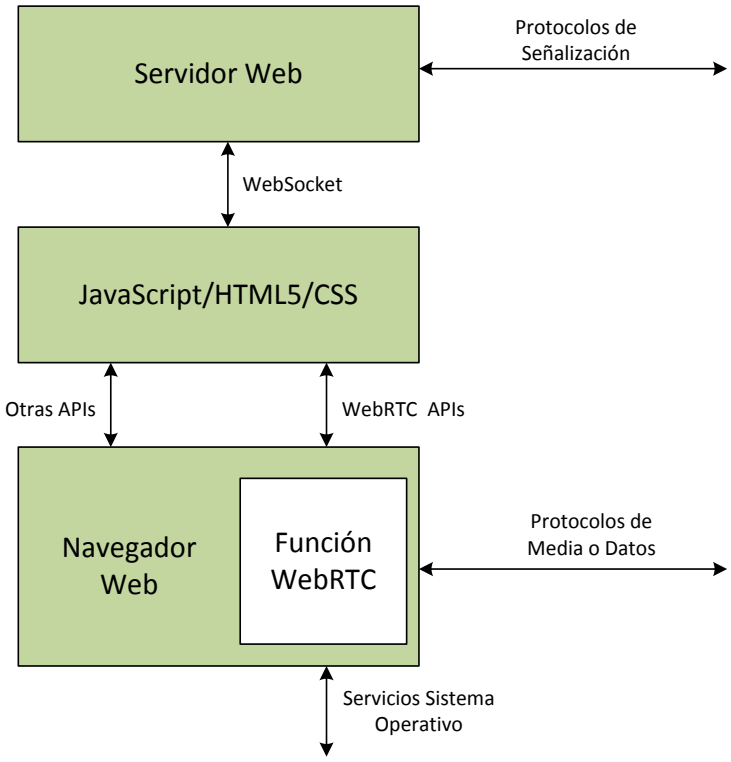


Figura 5.8. Modelo del Navegador. [56]

Un nuevo aspecto de *WebRTC* es la interacción navegador a navegador, a través de una conexión entre pares (del inglés, *peer connection*), y esta conexión puede utilizar protocolos de transporte como *UDP*, y no

necesariamente *TCP* como lo hace el tráfico Web convencional. El escenario básico de conexión se muestra en la Figura 5.9, donde ambos navegadores por medio de *HTML5* acceden a la misma aplicación *WebRTC* desde un mismo servidor. En donde la señalización pasa por el servidor Web y el tráfico de audio, video o datos fluye directamente entre los navegadores.

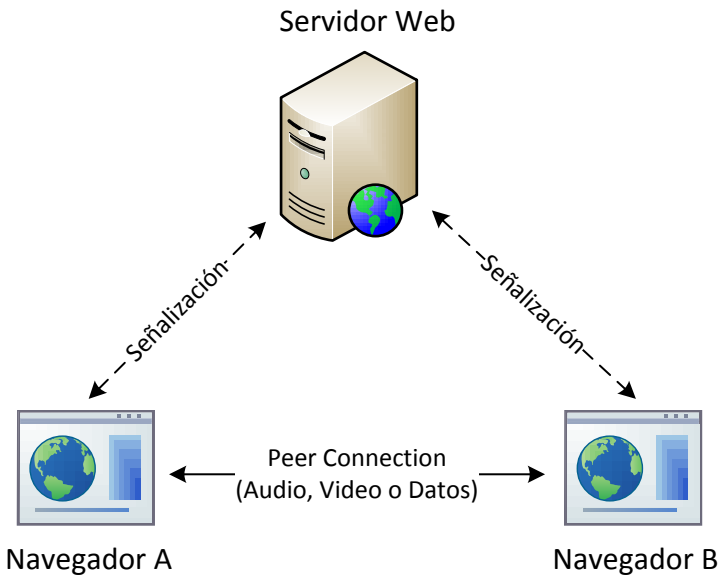


Figura 5.9. Escenario básico de conexión mediante *WebRTC*. [56]

Un aspecto muy importante a considerar es que la señalización, no es la misma utilizada como en los sistemas telefónicos convencionales, sino más bien que como la señalización no está normada para *WebRTC*, sino que se debe ejecutar sobre *WebSockets* al servidor Web que brinda el acceso a las

páginas o puede estar en un servidor aparte que solo maneje esta señalización. O puede darse el caso es que lo navegadores se conecten a su correspondiente servidor *WebRTC*, utilizando *WebSockets SIP* [49], y la señalización entre estos dos servidores pueda darse directamente en *SIP*, como se muestra en la Figura 5.10.

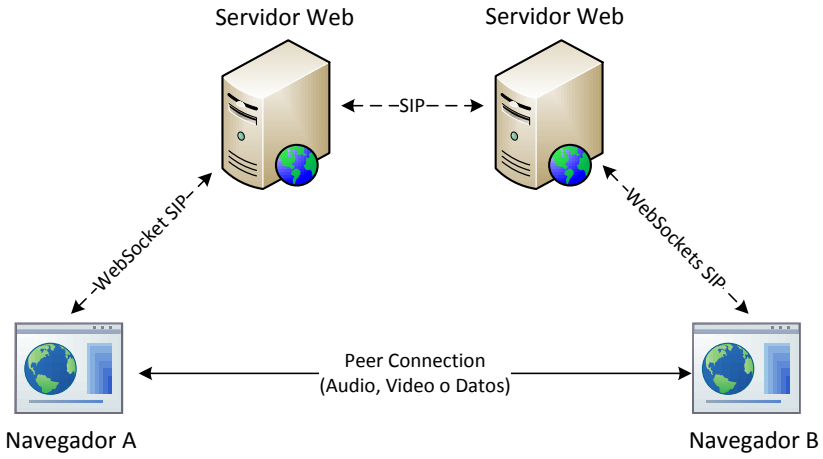


Figura 5.10. Conexión de navegadores por *WebRTC* y servidores mediante *SIP*. [56]

A su vez, también puede existir el caso en el que en una parte de la comunicación se tenga navegadores Web y en el otro extremo dispositivos convencionales de comunicación como teléfonos *IP*, que en el caso del *SIP* se conocen como clientes *SIP*, como se muestra en la Figura 5.11.

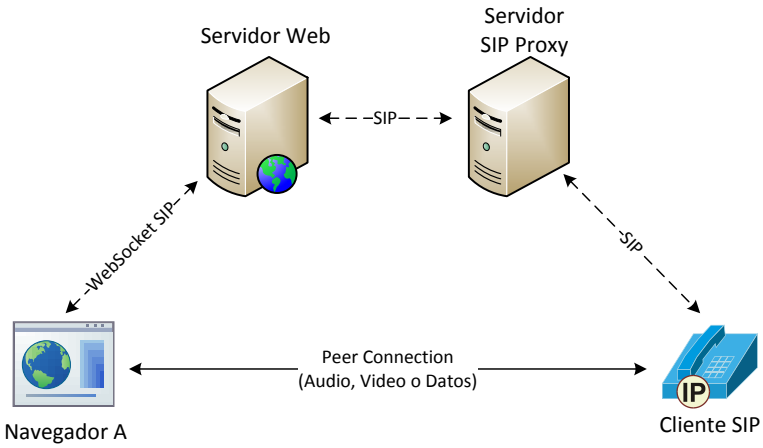


Figura 5.11. Interoperación de *WebRTC* y *SIP.* [56]

Incluso *WebRTC* podría tener una conexión con la *RTB*, a través de *Gateways*, como se muestra en la Figura 5.12.

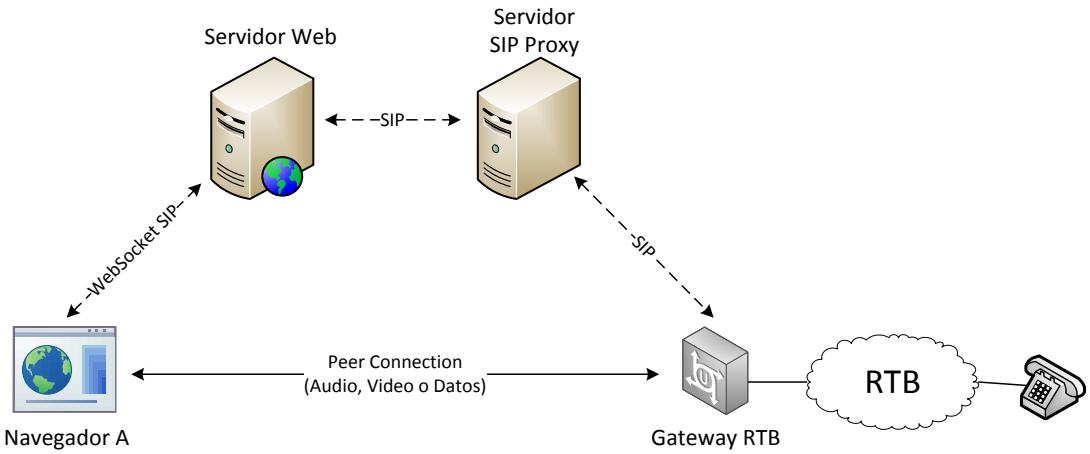


Figura 5.12. Interoperación de *WebRTC*, *SIP* y *RTB.* [56]

Establecimiento básico de sesiones mediante WebRTC

Para, el establecimiento de una sesión mediante *WebRTC*, existen cuatro pasos principales, los cuales son los siguientes [56]:

1. Acceso al audio y video local.

2. Establecimiento de la conexión entre el navegador y el par que puede ser otro navegador o un dispositivo final.

3. Intercambio de flujos de *Media* en la conexión.

4. Cerrar la conexión.

Para el acceso al audio y video local del equipo, *WebRTC* utiliza el *MediaStream API*, el cual se utiliza para obtener acceso a flujos de *Media* de video y audio (voz), tales como la cámara del usuario y el micrófono u otro dispositivo de captura. Para obtener acceso a estos flujos de información, las aplicaciones Web solicitan acceso a través de la función *getUserMedia*.

Mientras que para el establecimiento de la conexión entre pares (del inglés, *peer*), se utiliza el *RTCPeerConnection API*, y una vez se establezca conexión, los objetos *MediaStream* se pueden enviar directamente entre los pares. El mecanismo de conexión utiliza *Interactive Connectivity Establishment* (*ICE*), *Traversal Using Relays around NAT* (*TURN*) y *Session Traversal Utilities for NAT* (*STUN*) para atravesar *firewalls* y *NAT*, los cuales se describen en el apéndice D.

Para el intercambio de voz, una vez se establece la conexión entre los pares, es necesaria una negociación entre los navegadores de representación de los datos en el canal. Cuando se realiza una solicitud local o remotamente, para agregar o remover flujos de voz, el navegador puede generar un objeto *SessionDescription*, los cuales son utilizados para describir las características de las sesiones *WebRTC*, como por ejemplo el Códec por medio de *SDP* (Protocolo de Descripción de Sesión, del inglés, *Session Description Protocol*). Cuando se realiza algún cambio a la conexión, se produce en un cambio en la descripción del objeto *SDP*. Así mismo *SDP* está estandarizado y se utiliza mucho en *VoIP* por lo cual permite la comunicación entre navegadores y dispositivos convencionales como teléfonos *IP*, *Gateways.* *SDP* fue estandarizado por medio del RFC 4566 [57]. Entonces una vez que los navegadores hayan intercambiado sus objetos *SDP* la sesión se puede establecer. Y por último para cerrar la conexión, se invoca al método *close( )* del objeto *RTCPeerConnection*.

Una sesión básica de establecimiento con *WebRTC* se muestra en la Figura 5.13.

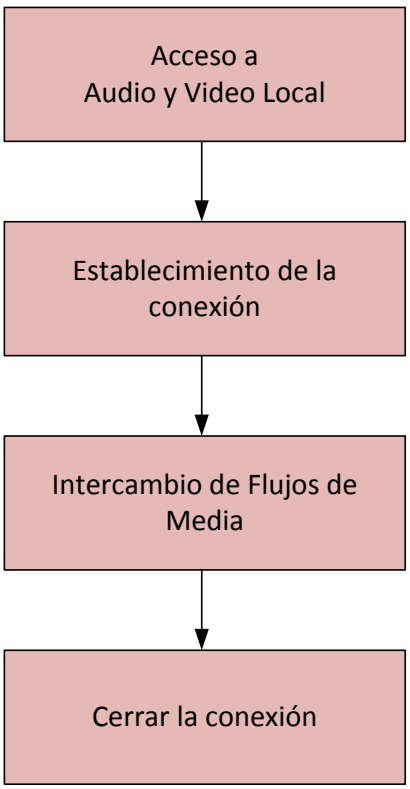


Figura 5.13. Establecimiento de una sesión *WebRTC*. [56]

Arquitectura de WebRTC

En la Tabla 13 se muestran algunos protocolos que necesita *WebRTC* para poder funcionar correctamente.

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

Tabla 13. Protocolos que utiliza *WebRTC* para sesiones. [56]

De estos protocolos podemos destacar, que las propuestas de conexión utilizan ICE para establecer la conexión y *SRTP* (Protocolo de Transporte Seguro en Tiempo Real, del inglés, *Secure Real-time Transport Protocol*) sobre *UDP* para transferir flujos de *Media* a un interlocutor remoto para comunicación en tiempo real [58], este comportamiento se muestra en la Figura 5.14.

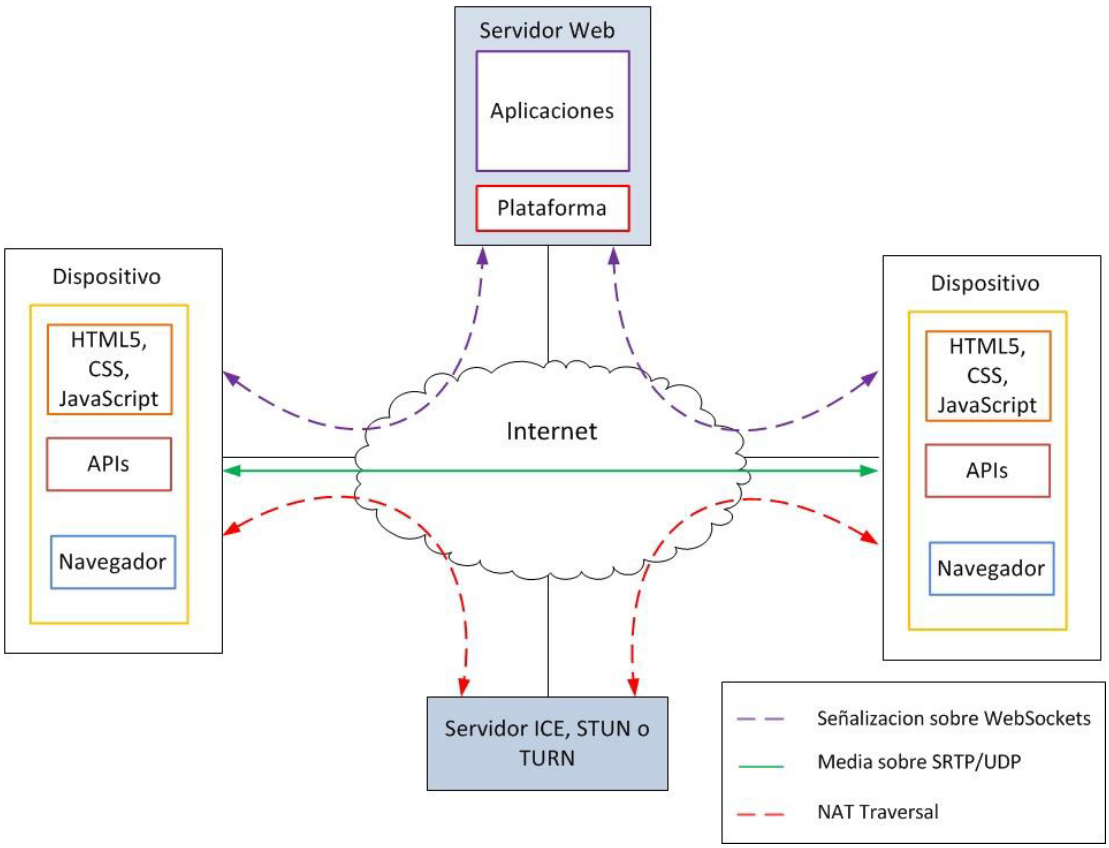


Figura 5.14. Señalización y Media en una sesión *WebRTC*. [58]

Como se mencionó en la sección 5.4.1, *WebRTC* ofrece comunicación y aplicaciones multimedia en tiempo real sin la necesidad de instalar *plugins*, y al ser de código abierto permite a desarrolladores la capacidad de implementar nuevas aplicaciones fácilmente. La arquitectura de *WebRTC* es mostrada en la Figura 5.15 [59].

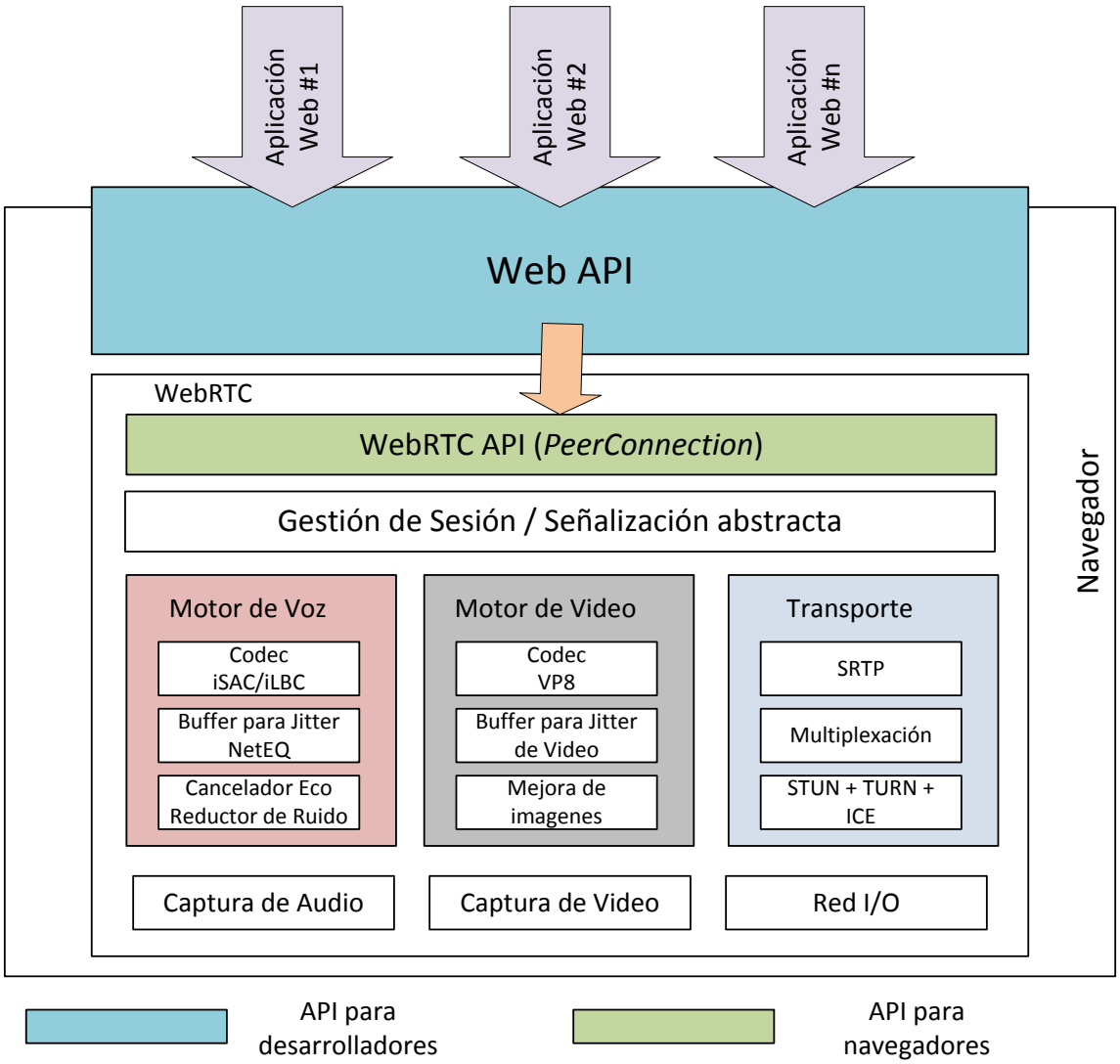


Figura 5.15. Arquitectura *WebRTC*. [59]

Las aplicaciones Web son realizadas 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 [60]. Y *WebRTC API*

permite a los desarrolladores de navegadores, implementar más fácilmente las propuestas de Web *API*.

Por su parte la Gestión de Sesión es un nivel abstracto, que permite la gestión de llamadas y su gestión por medio de señalización, que 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* [56].

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

5.5 DISEÑO DE UNA RED PARA VOIP EN LA NUBE CON HTML5

Tomando como referencia el diseño para redes con servicios *VoIP* en la Nube descrito en la sección 3.4.6 del presente estudio, y las nuevas formas de acceso a servicios de telefonía por medio de *WebSockets* para señalización y *WebRTC* para transporte de *SRTP*, en la Figura 5.16 se muestra una solución que integra ambos escenarios.

En este modelo usuarios con teléfonos celulares inteligentes, tabletas, o computadores, pueden acceder al servicio por medio de la Web a través de *HTML5*, los usuarios se registran en el servidor *WebRTC*, y a continuación

puede realizar una solicitud de llamada sea a otro usuario Web, a un teléfono *IP* físico *SIP* en Internet, o a un número convencional de la *RTB*. El usuario *WebRTC* se puede registrar directamente a través del servidor *WebRTC* que deberán estar en una *DMZ* por seguridad, o este mismo usuario puede solicitar registro a través de acceso remoto por *VPN* si la necesidad de seguridad es mayor para la empresa que provee el servicio. Igual se recomienda que el acceso a la *RTB* y las bases de datos de los usuarios estén detrás de un *firewall*, para maximizar en lo posible las medidas de seguridad. Igual este esquema puede ser utilizado para nubes privadas e hibridas.

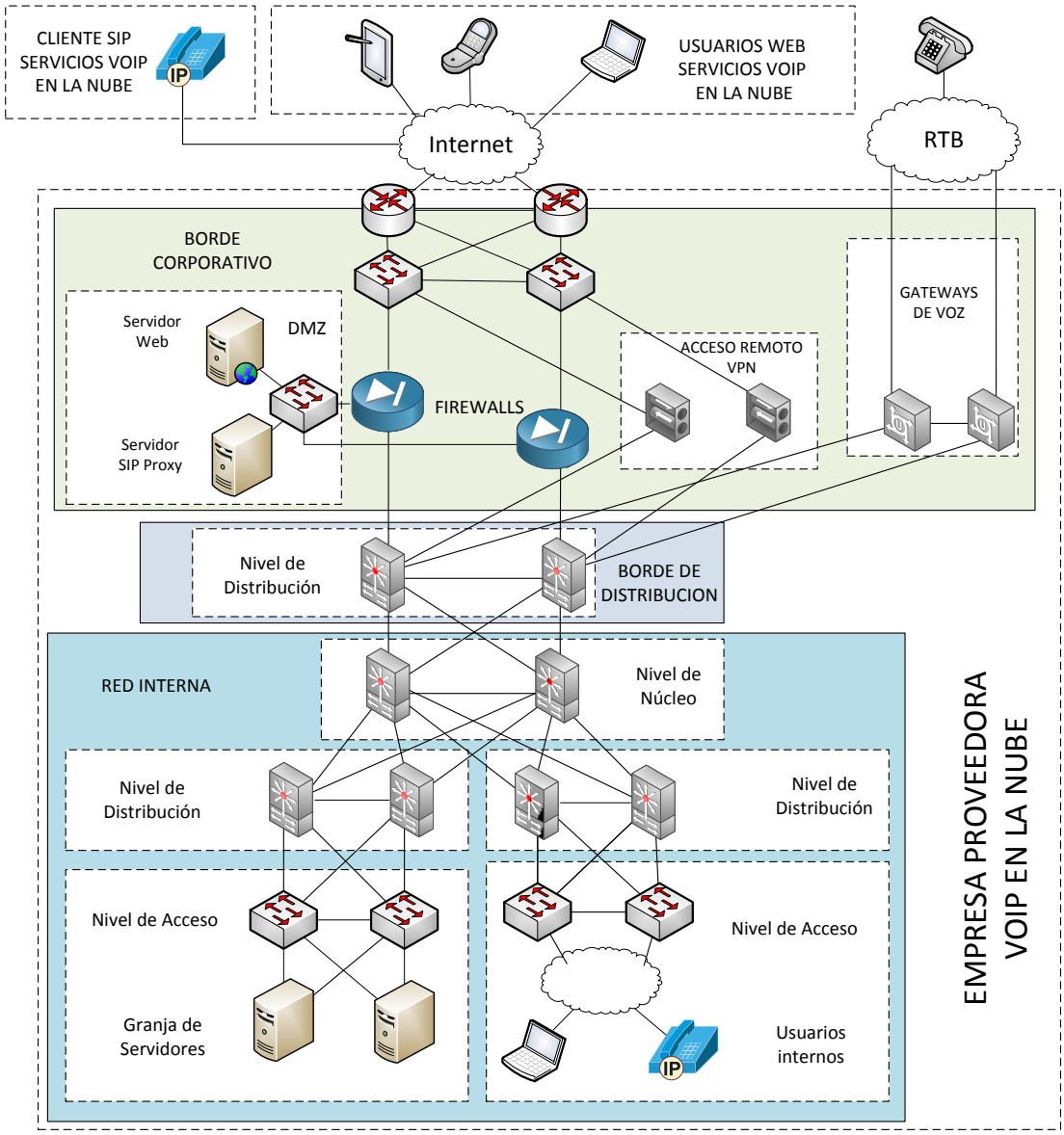


Figura 5.16. Diseño de una red para Servicios *VoIP* en la Nube para acceso Web y convencional

5.6 FLUJOS DE SEÑALIZACIÓN DE VOIP EN LA NUBE CON EL USO DE HTML5

En la Figura 5.17 se muestra los pasos en el establecimiento de una sesión *WebRTC*.

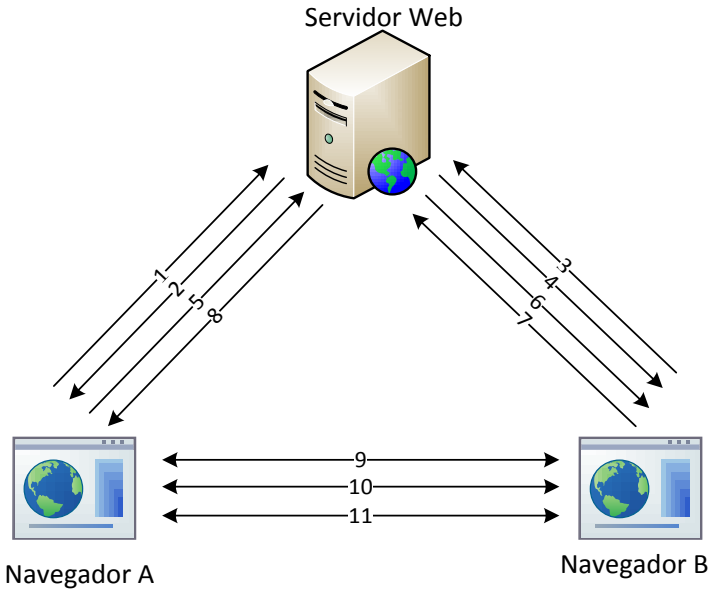


Figura 5.17. Vista de protocolos para el establecimiento de una sesión *WebRTC*. [56]

Lo que debe considerar es que ambos navegadores A y B están ejecutando la misma aplicación *WebRTC* en *JavaScript* descargado desde el mismo servidor Web. Y cuando un usuario desea comunicarse con otro, comienza una negociación entre los navegadores, esta negociación es la forma en que los navegadores llegan a un acuerdo a una sesión aceptable para

intercambio de *Media*, y la misma es realizada por medio de un intercambio de solicitudes y respuestas [56], las mismas que son:

1. El navegador A solicita una página Web al servidor Web.
2. El servidor Web proporciona la página Web con *WebRTC JavaScript* al navegador A.
3. El navegador B solicita una página Web al servidor Web.
4. El servidor Web proporciona la página Web con *WebRTC JavaScript* al navegador B.
5. El navegador A decide comunicarse con B, entonces el navegador A envía una solicitud por medio de un objeto *SDP* de A al servidor Web.
6. El servidor Web envía el objeto *SDP* de A en *JavaScript* a B.
7. *JavaScript* en el navegador B envía una respuesta por medio de un objeto *SDP* al servidor Web.
8. El servidor Web envía el objeto *SDP* de B en *JavaScript* a A.
9. A y B comienzan un proceso para determinar la mejor ruta para alcanzarse directamente.
10. Después del proceso de mejor ruta, A y B empiezan un intercambio de llaves para seguridad del tráfico de *Media*.
11. A y B comienzan en intercambio de voz, datos y video.

La misma conexión pero vista como diagrama se muestra en la figura 5.18.

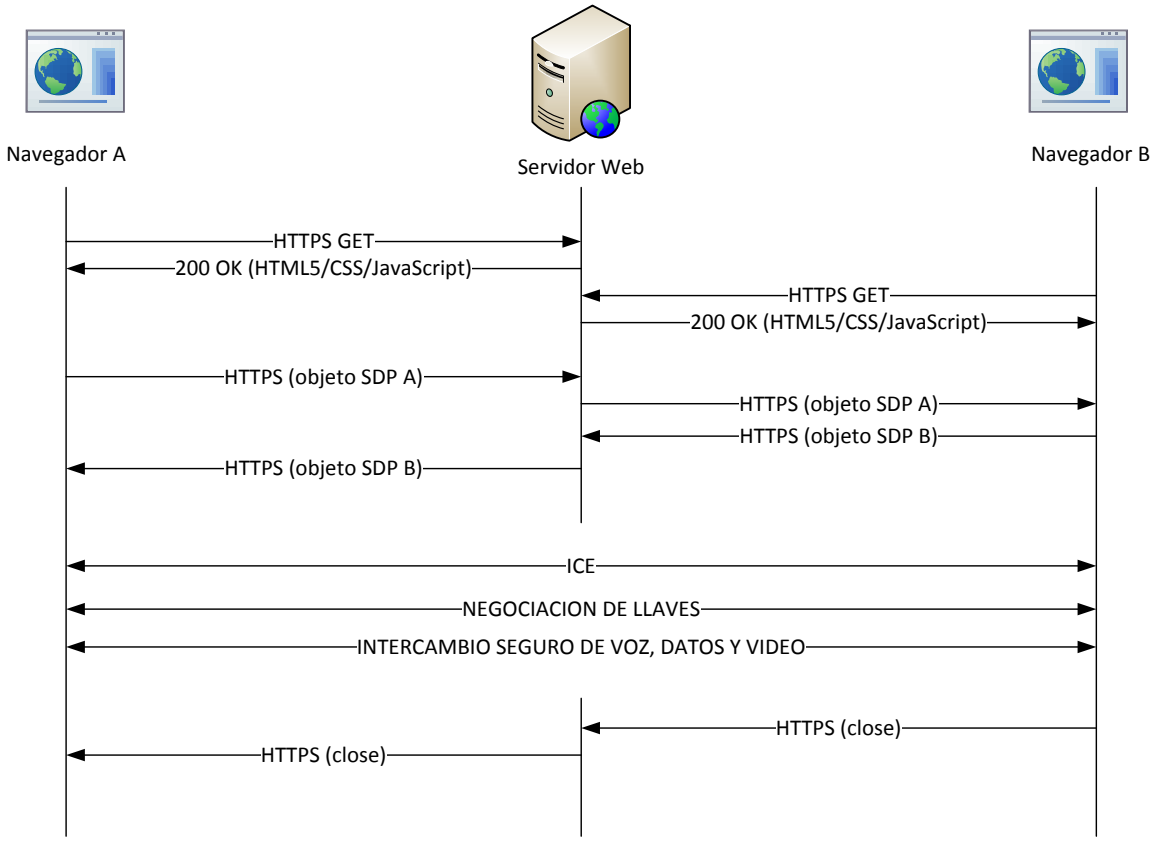


Figura 5.18. Vista de protocolos para el establecimiento de una sesión *WebRTC*.

[56]

Ahora en el caso que se aprecia en la Figura 5.19, de tener un cliente que sea un navegador denominado A, que no tenga instalada ninguna aplicación preexistente para realizar llamadas, sino directamente accediendo a una *URL*.

Para lo cual cuando el usuario desea comunicarse con otro, entre el navegador y el servidor Web comienza la negociación a través de intercambio de tipo solicitud y respuesta [56]. Esta negociación es la manera en la que las dos partes en una sesión de comunicación llegan a acuerdo sobre una sesión que sea aceptable a nivel de códec, conexión, etc.

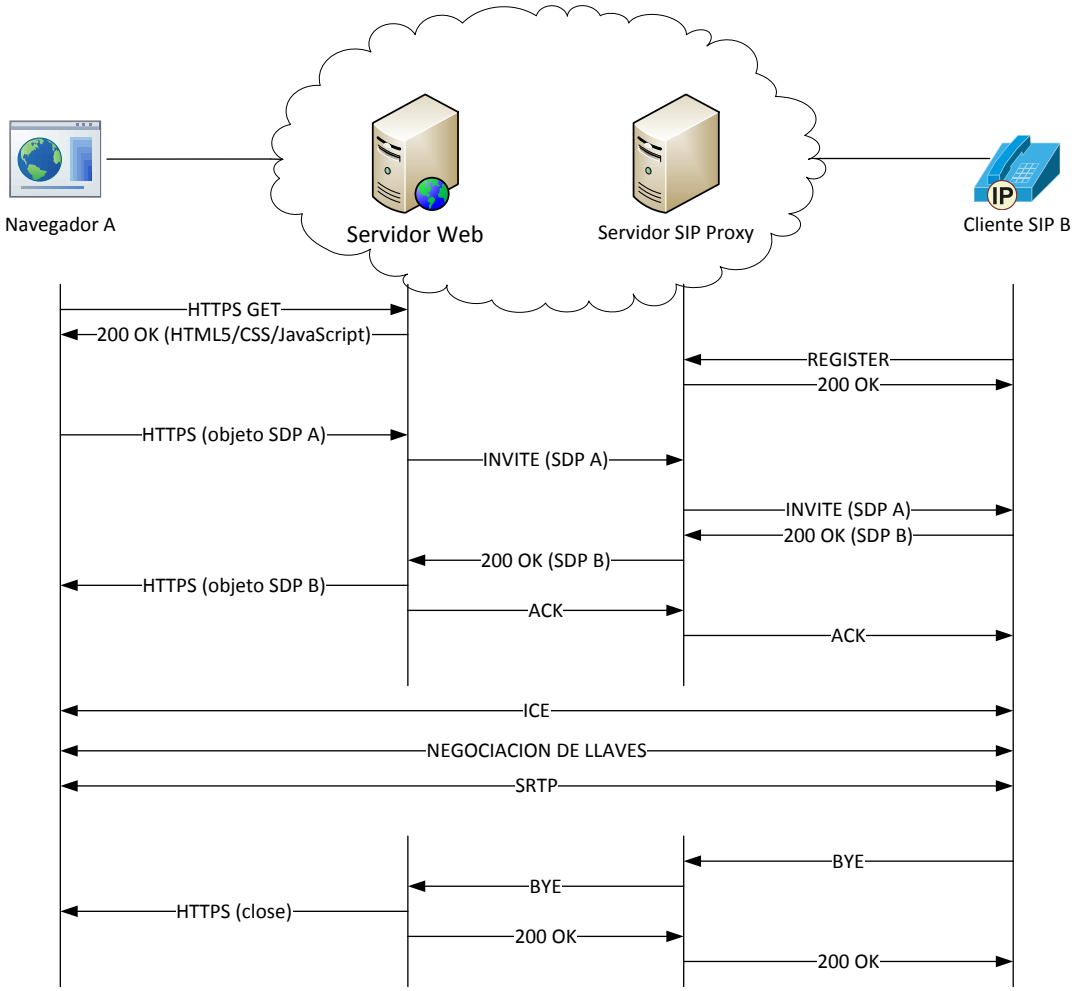


Figura 5.19. Flujo de Señalización de *VoIP* en la Nube mediante *HTML5*. [56]

Esta negociación comienza cuando el servidor Web proporciona la página Web con *WebRTC JavaScript* al navegador A, una vez que este lo ha solicitado a través de un mensaje HTTPS GET.

Un aspecto a considerar es que cuando el usuario del Navegador A decide realizar la comunicación, el código *JavaScript* en el navegador proporciona una restricción basada en la descripción de *Media*, solicita el medio de comunicación, y consigue permiso del usuario. Es importante que la concesión de permisos está ligada al dominio de la página Web, y que este permiso no se extiende a elementos emergentes. La información deseada de *Media* es capturada en un objeto *SDP*. Esta es la oferta, que es enviada al cliente *SIP* B por medio del servidor Web y reenviada al servidor *SIP Proxy* donde se encuentra registrado el cliente *SIP* B a través del mensaje INVITE.

Es importante notar que *WebRTC* no estandariza como los navegadores envían estas ofertas, por lo que aún está en desarrollo, pero puede ser utilizado por el momento solicitudes *XML HTTP* (del inglés, XML *HTTP Request*).

Entonces el Cliente *SIP* B recibe la invitación, y responderá por medio de un mensaje 200 OK, el cual sería enviado en formato *SIP* convencional hasta el servidor Web, donde será traducido por medio de *WebSockets SIP*, como se detalla en el Apéndice B.

Otro detalle a mencionar es que el Navegador A no envía al cliente *SIP* B un mensaje de confirmación ACK, sino que el servidor Web lo realiza por él.

Una vez realizado este proceso viene la etapa de comunicación directa entre ambos clientes, a través del protocolo *ICE* explicado en el Apéndice C. Paso seguido empieza un proceso de intercambio de llaves para seguridad del tráfico de media. Cuando finaliza el intercambio el tráfico de *Media*, será transportado por medio del *SRTP*, entre los clientes directamente.

Cuando la llamada finaliza el cliente *SIP* B envía un mensaje de tipo BYE a su servidor *SIP Proxy* quien lo reenvía al servidor B y este lo traduce a un formato HTTPS para cierre de la conexión. E igualmente el servidor Web confirma por el Navegador A al cliente *SIP* B.

CONCLUSIONES

Las principales conclusiones alcanzadas son las siguientes:

1. En esta tesis hemos realizado un estudio detallado de diversas tecnologías de *VoIP*. Partiendo de un esquema tradicional de la *VoIP* que abunda hoy en día en muchas empresas (modelo de *VoIP* distribuido), hemos planteado primero el modelo de *VoIP* para la empresa centralizado a partir del cual hemos derivado fácilmente un modelo de *VoIP* en la Nube.
2. Una vez concretada la arquitectura de *VoIP* en la Nube hemos ido un paso más allá para presentar un modelo de red apropiado para empresas que quisieran proporcionar el servicio de *VoIP* en la Nube,

para ello hemos partido del Modelo Compuesto para Empresas de Cisco y del Modelo de Arquitectura para Empresas de Cisco.

3. Teniendo claro que el beneficio implícito de servicios en la Nube, es el de ahorro de energía en le Empresa, hecho éste muy importante hoy en día, podemos considerar que nuestra propuesta es una propuesta de *VoIP* en la Nube verde (*Green*). Esto es, la empresa que opte por contratar este tipo de servicio ganaría en consumo de energía en sus instalaciones y además tendría todos los beneficios derivados de este tipo de soluciones: *CAPEX* y *OPEX*.
4. Destacamos que hemos diseñado una arquitectura completa de provisión de *VoIP* en la Nube para equipos de telefonía con interfaz Web (aunque reamente es universal y puede funcionar para cualquier tipo de terminal telefónico), teniendo en cuenta lo siguiente: La red del proveedor de *VoIP* está en la Nube, los tipos de servicio (*IaaS* y *SaaS*) y los compromisos de diseño teniendo en cuenta: encaminamiento, plan de numeración, G*ateways, VPN*, *DMZ*, *LAN*, *WAN*, protocolos de señalización y protocolos de transporte de voz
5. Dado que el estado de la tecnología en *WebRTC* y *HTML5* para equipos de telefonía no es definitivo, nosotros sólo pudimos hacer pruebas de funcionamiento preliminares.

RECOMENDACIONES

En esta tesis se ha abierto varias posibles líneas de ampliación que consideramos son muy interesantes:

1. Profundizar en el estudio del *software* Web que permite implantar eficazmente la *VoIP*.
2. Realizar pruebas de funcionamiento con terminales móviles.
3. Demostrar que la señalización sobre *WebSockets* es eficiente, que los Códec utilizados son adecuados y si el modelo de negocio es rentable.
4. Demostrar mediante técnicas de diseño de sistemas complejos que la viabilidad de nuestro diseño está asegurado para los próximos años.

5. Realizar estudios de comparación entre *JsSIP* y *sipML5* en su rendimiento para *WebSockets*.
6. Realizar estudios de comparación entre *OverSIP* y *Asterisk* como servidores *SIP Proxy* para *WebSockets*.
7. Realizar estudios de *PSSIP* como una nueva pila de código abierto para el protocolo *SIP.*

APÉNDICE A

WEBSOCKET

Antes de que el cliente y el servidor inicien el envío y recepción de mensajes, necesitan establecer una conexión primero. Esto se realiza mediante el *handshake*, donde el cliente envía una solicitud de conexión, y si el servidor desea, envía una respuesta aceptando la conexión. La especificación del protocolo deja claro que una de las decisiones es asegurar que tanto los clientes basados en *HTTP* y *WebSocket* pueden operar en el mismo puerto [61]. Esta es la razón por la que el cliente y el servidor realizan un *upgrade* desde un protocolo basado en *HTTP* a un protocolo basado en *WebSocket*.

En la Figura A.1 se muestra un ejemplo del *handshake* inicial del cliente, y en la Figura A.2 se enseña el *handshake* de respuesta del servidor [43].

```
GET /chat HTTP/1.1
   Host: server.example.com
   Upgrade: WebSocket
   Connection: Upgrade
   Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
   Origin: HTTP://example.com
   Sec-WebSocket-Protocol: chat, superchat
   Sec-WebSocket-Version: 13
```

Figura A.1. *Handshake* inicial del cliente. [43]

```
HTTP/1.1 101 Switching Protocols
   Upgrade: WebSocket
   Connection: Upgrade
   Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=
   Sec-WebSocket-Protocol: chat
```

Figura A.2. *Handshake* de respuesta del servidor. [43]

El cliente envía una clave en la cabecera *Sec-WebSocket-Key*, codificada en base64. El servidor envía una respuesta, toma esa cabecera y anexa a continuación la cadena *Globally Unique Identifier*(*GUID*):*258EAFA5-E914-47DA-95CA-C5AB0DC85B11* a ella, queda entonces de la forma *dGhlIHNhbXBsZSBub25jZQ==258EAFA5-E914-47DA-95CA-C5AB0DC85B11,* y luego calcula el hash SHA-1 de esta cadena (*b37a4f2cc0624f1690f64606cf385945b2bec4ea).* Luego se codifica ese valor *hash* en base64, y su valor *s3pPLMBiTxaQ9kYGzzhZRbK+xOo=* sería la cabecera de *Sec-WebSocket-Accept* en la respuesta del servidor.

En la Figura A.3 se muestra un resumen del establecimiento de una sesión *WebSocket*.

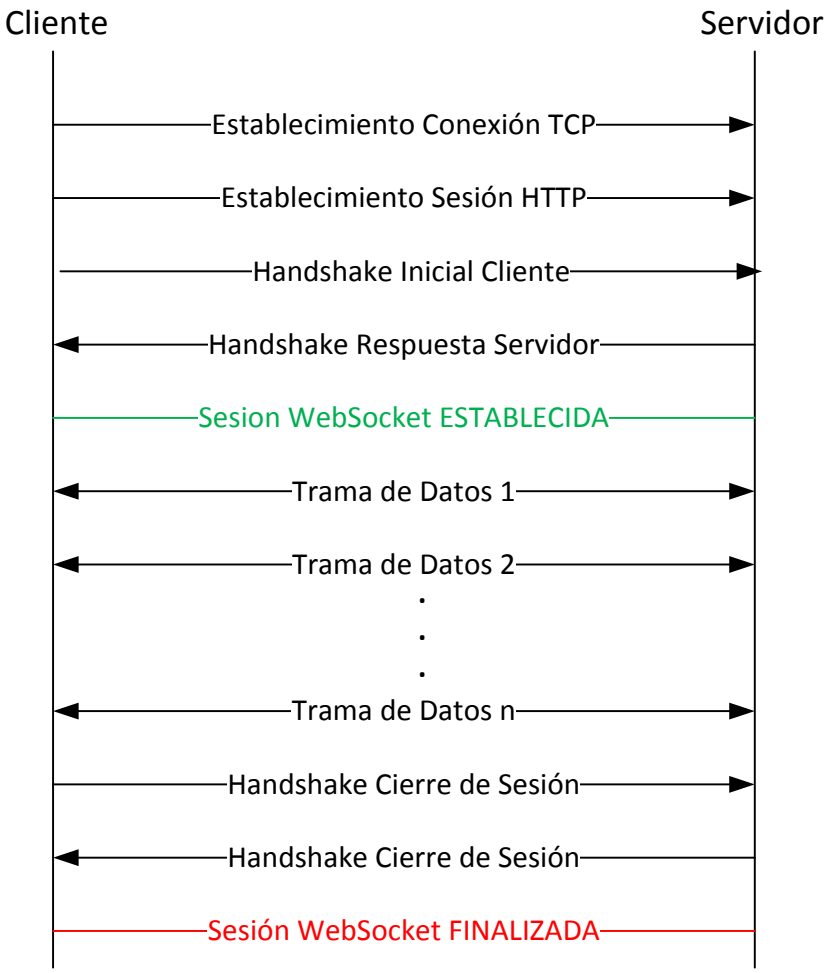


Figura A.3. Establecimiento Sesión *WebSocket*

API para Web Sockets

Existe un *API* en *JavaScript* para utilizar *WebSockets* en las aplicaciones Web, que está siendo normalizado por la W3C, la cual se muestra en la Figura A.4.

```
[Constructor(in DOMStringurl, in optional DOMString protocols)]
[Constructor(in DOMStringurl, in optional DOMString[]
protocols)]
Interface WebSocket {
readonly attribute DOMStringurl;

  // ready state
const unsigned short CONNECTING = 0;
const unsigned short OPEN = 1;
const unsigned short CLOSING = 2;
const unsigned short CLOSED = 3;
readonly attribute unsigned short readyState;
readonly attribute unsigned long bufferedAmount;

  // networking
attribute Function onopen;
attribute Function onmessage;
attribute Function onerror;
attribute Function onclose;
readonly attribute DOMString protocol;
voidsend(in DOMString data);
voidclose();
};
WebSocket implements EventTarget;
```

Figura A.4. *API* en *JavaScript* para *WebSockets*. [44]

El constructor *WebSocket* (*url,protocols*) toma uno o dos argumentos. El primer argumento, *url*, específica la URL a cual conectarse. Mientras que el segundo argumento opcional *protocols*, puede ser una cadena o un vector de cadenas. Cada cadena del vector es el nombre de un protocolo. La conexión sólo se establece si el servidor informa de que se ha seleccionado uno de estos protocolos.

El atributo *ready state* indica el estado de la conexión, y puede tener los siguientes valores: *CONNECTING* (la conexión aún no ha sido establecida), *OPEN* (la comunicación es establecida y la comunicación es posible),

CLOSING (la conexión está realizando el *handshake* para cerrar la conexión) y *CLOSE* (la conexión se ha cerrado o no se pudo abrir).

Los manejadores de eventos (*event handlers*) los cuales invocan un código *JavaScript* y son ejecutados cuando ocurre un evento en un elemento *HTML*. Los cuales también se incluyen en la *API* para *WebSockets* y se muestran en la Tabla 14.

Event handler	Tipo de evento	Causa del evento
onopen	open	Cuando un socket se ha abierto, después de un handshake TCP de tres vías TCP y upgrade
onmessage	message	Cuando un mensaje ha sido recibido desde un servidor WebSocket.
onerror	error	Cuando ocurrió el error.
onclose	close	Cuando un socket ha sido cerrado.

Tabla 14. Manejadores de evento *JavaScript* en *WebSockets*. [44]

El atributo *bufferedAmount* debe devolver el número de bytes de texto UTF-8 que ha sido encolado utilizando *send ( )*, pero que a partir de la última vez que el bucle de eventos comenzó a ejecutar una tarea, y que aún no se había transmitido a la Red. Esto no incluye cabecera adicional propia del protocolo.

En el ejemplo de la Figura A.5, se utiliza el atributo *bufferedAmount* para asegurar que las actualizaciones se envían cada 50 ms. Este atributo puede incluso saturar la red, por lo que debe ser monitoreado para ser utilizado con cuidado.

```
var socket = new WebSocket('ws://game.example.com:12010/updates');
socket.onopen = function () {
    setInterval(function() {
            if (socket.bufferedAmount == 0)
                    socket.send(getUpdateData());
    }, 50);
};
```

Figura A.5. Atributo *bufferedAmount*. [44]

En la Figura A.5 también se aprecia que para invocar a *WebSocket* en vez de http:// se utiliza ws://, en la Figura A.6 se indica otro ejemplo de llamado del *API* WebScoket desde una página *HTML*, con la correspondiente URL de conexión.

```
var socket=new WebSocket("ws://127.0.0.1:8080/");
```

Figura A.6. Llamado de *API WebSocket.* [43]

En lo referente al soporte que brindan los navegadores, en la Figura A.7 se muestra los niveles de soporte para los navegadores más utilizados extraídos de: *webcaniuse.com/WebSockets*.

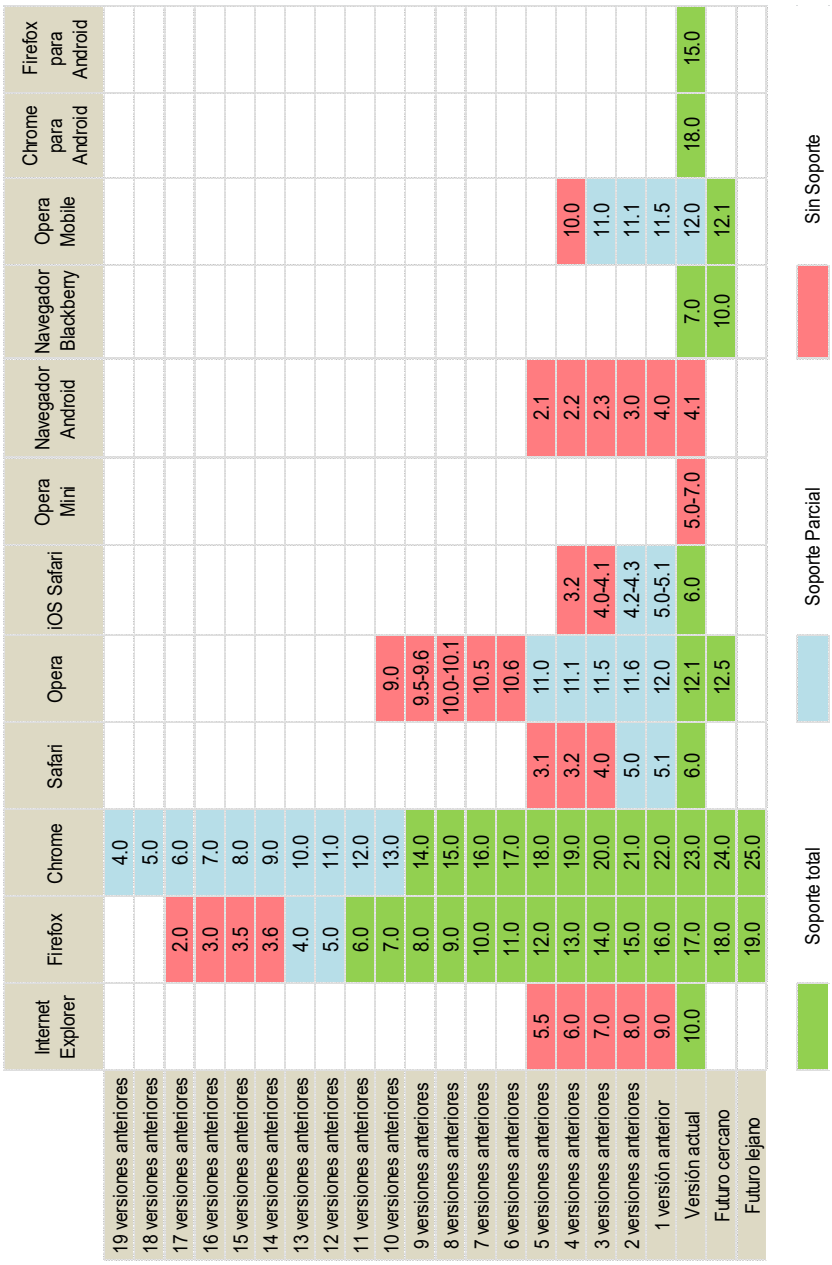


Figura A.7. Soporte de navegadores a *WebSocket*

APÉNDICE B

EJEMPLO DE REGISTRO SIP A TRAVÉS DE WEBSOCKET SIP

Handshake inicial y registro entre un cliente y servidor proxy

Alice carga una página Web utilizando su navegador y recupera código *JavaScript* que implementa el protocolo *WebSocket SIP* a través de una conexión *WebSocket* segura en el *handshake* inicial, entre el cliente (cliente *SIP WebSocket*) y un *SIP Proxy* (servidor *SIP WebSocket*). El cliente debe incluir el valor de *SIP* en su cabecera, específicamente en Sec-*WebSocket*-Protocol, como se detalló en la sección 5.2.1. Y una vez establecida la conexión *WebSocket*, Alice construye y envía una solicitud *SIP REGISTER* [49], como se muestra en la Figura B.1.

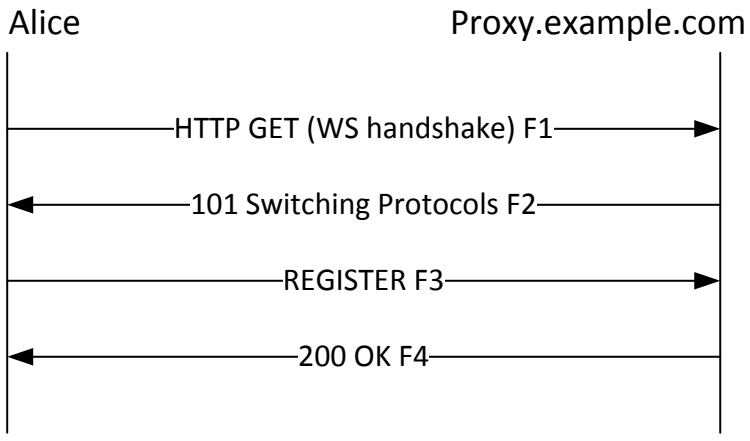


Figura B.1. *Handshake* inicial y registro *SIP*. [49]

Dado que la pila *JavaScript* en un navegador no tiene forma de determinar la dirección local de la que la conexión *WebSocket* realizó, esta aplicación utiliza un nombre aleatorio de dominio "*.invalid*" para en la cabecera de contacto. En las Figuras B.2 y B.3, se realiza una descripción de los mensajes de conexión del protocolo *WebSocket SIP* de la Figura B.1.

```
F1 HTTP GET (WS handshake) Alice -> proxy.example.com (TLS)
GET / HTTP/1.1
Host: proxy.example.com
Upgrade: Websocket
Connection: Upgrade
Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
Origin: HTTPS://www.example.com
Sec-WebSocket-Protocol: sip
Sec-WebSocket-Version: 13
```

Figura B.2. Mensaje F1: *HTTP* GET (WS handshake). [49]

```
F2 101 Switching Protocols proxy.example.com -> Alice (TLS)
HTTP/1.1 101 Switching Protocols
Upgrade: Websocket
Connection: Upgrade
Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=
Sec-WebSocket-Protocol: sip
```

Figura B.3. Mensaje F2: 101 Switching Protocols. [49]

En las Figuras B.4 y B.5 se puede apreciar que los mensajes *SIP* son sobre *WebSockets*.

```
F3 REGISTER Alice -> proxy.example.com (transport WSS)
REGISTER sip:proxy.example.com SIP/2.0
Via: SIP/2.0/WSS df7jal23ls0d.invalid;branch=z9hG4bKasudf
From: sip:alice@example.com;tag=65bnmj.34asd
To: sip:alice@example.com
Call-ID: aiuy7k9njasd
CSeq: 1 REGISTER
Max-Forwards: 70
Supported: path, outbound, gruu
Contact: <sip:alice@df7jal23ls0d.invalid;transport=ws>
;reg-id=1
;+sip.instance="<urn:uuid:f81-7dec-14a06cf1>"
```

Figura B.4. Mensaje F3: REGISTER. [49]

```
F4 200 OK proxy.example.com -> Alice (transport WSS)
SIP/2.0 200 OK
Via: SIP/2.0/WSS df7jal23ls0d.invalid;branch=z9hG4bKasudf
From: sip:alice@example.com;tag=65bnmj.34asd
To: sip:alice@example.com;tag=12isjljn8
Call-ID: aiuy7k9njasd
CSeq: 1 REGISTER
Supported: outbound, gruu
Contact: <sip:alice@df7jal23ls0d.invalid;transport=ws>
;reg-id=1
;+sip.instance="<urn:uuid:f81-7dec-14a06cf1>"
;pub-gruu="sip:alice@example.com;gr=urn:uuid:f81-7dec-14a06cf1"
;temp-gruu="sip:87ash54=3dd.98a@example.com;gr"
;expires=3600
```

Figura B.5. Mensaje F4: 200 OK. [49]

Dialogo INVITE a través de un servidor *Proxy*

En el mismo ejemplo, Alice realiza una conexión a Bob a través de un servidor *Proxy*, en el que la conexión de Alice con el servidor *Proxy* se hace a través del protocolo *WebSockets SIP* seguro, y la conexión entre el

servidor *Proxy* y Bob, se hace por medio de *SIP* convencional sobre *UDP*, como se muestra en la Figura B.6.

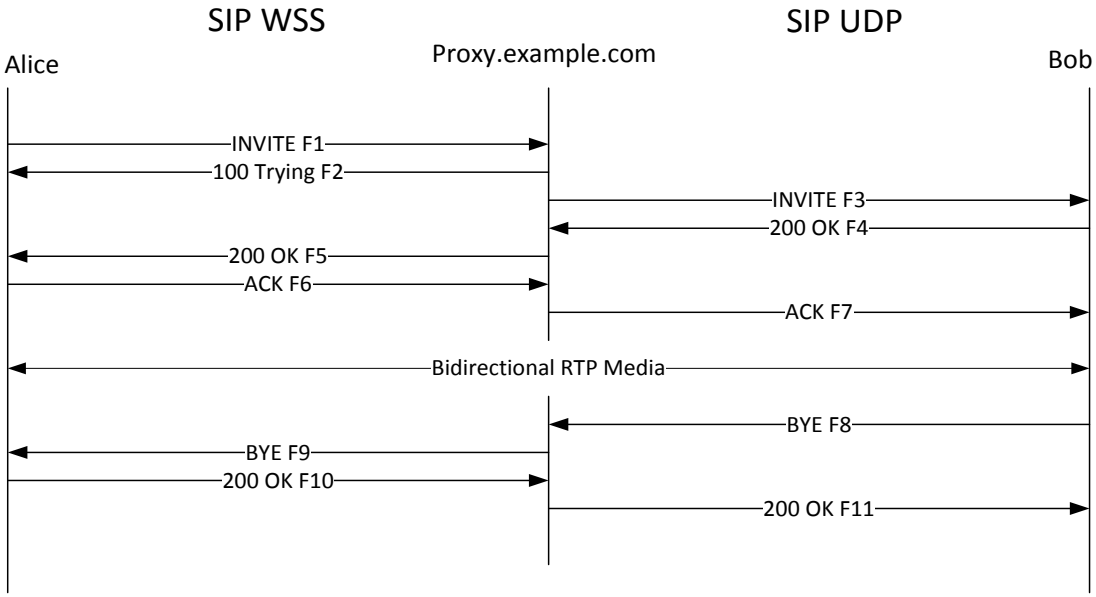


Figura B.6. Conexión por medio de un servidor proxy. [49]

En las Figuras B.7 a B.18 se realiza una descripción de los mensajes de conexión del protocolo *WebSocket SIP* y *SIP* sobre UPD de la Figura B.6.

```
F1 INVITE Alice -> proxy.example.com (transport WSS)
INVITE sip:bob@example.com SIP/2.0
Via: SIP/2.0/WSS df7jal23ls0d.invalid;branch=z9hG4bK56sdasks
From: sip:alice@example.com;tag=asdyka899
To: sip:bob@example.com
Call-ID: asidkj3ss
CSeq: 1 INVITE
Max-Forwards: 70
Supported: path, outbound, gruu
Route: <sip:proxy.example.com:443;transport=ws;lr>
Contact: <sip:alice@example.com
;gr=urn:uuid:f81-7dec-14a06cf1;ob>
Content-Type: application/SDP
```

Figura B.7. Mensaje F1: Invite en WSS. [49]

```
F2 100 Trying proxy.example.com -> Alice (transport WSS)
SIP/2.0 100 Trying-
Via: SIP/2.0/WSS df7jal23ls0d.invalid;branch=z9hG4bK56sdasks
From: sip:alice@example.com;tag=asdyka899
To: sip:bob@example.com
Call-ID: asidkj3ss
CSeq: 1 INVITE
```

Figura B.8. Mensaje F2: 100 Trying en WSS. [49]

```
F3 INVITE proxy.example.com -> Bob (transport UDP)
INVITE sip:bob@203.0.113.22:5060 SIP/2.0
Via: SIP/2.0/UDPproxy.example.com;branch=z9hG4bKhjhjqw32c
Via: SIP/2.0/WSS df7jal23ls0d.invalid;branch=z9hG4bK56sdasks
Record-Route: <sip:proxy.example.com;transport=udp;lr>,
<sip:h7kjh12s@proxy.example.com:443;transport=ws;lr>
From: sip:alice@example.com;tag=asdyka899
To: sip:bob@example.com
Call-ID: asidkj3ss
CSeq: 1 INVITE
Max-Forwards: 69
Supported: path, outbound, gruu
Contact: <sip:alice@example.com
;gr=urn:uuid:f81-7dec-14a06cf1;ob>
Content-Type: application/SDP
```

Figura B.9. Mensaje F3: Invite en *UDP*. [49]

```
F4 200 OK Bob -> proxy.example.com (transport UDP)
SIP/2.0 200 OK
Via: SIP/2.0/UDPproxy.example.com;branch=z9hG4bKhjhjqw32c
;received=192.0.2.10
Via: SIP/2.0/WSS df7jal23ls0d.invalid;branch=z9hG4bK56sdasks
Record-Route: <sip:proxy.example.com;transport=udp;lr>,
<sip:h7kjh12s@proxy.example.com:443;transport=ws;lr>
From: sip:alice@example.com;tag=asdyka899
To: sip:bob@example.com;tag=bmqkjhsd
Call-ID: asidkj3ss
CSeq: 1 INVITE
Contact: <sip:bob@203.0.113.22:5060;transport=udp>
Content-Type: application/SDP
```

Figura B.10. Mensaje F4: 200 OK en *UDP*. [49]

```
F5 200 OK proxy.example.com -> Alice (transport WSS)
SIP/2.0 200 OK
Via: SIP/2.0/WSS df7jal23ls0d.invalid;branch=z9hG4bK56sdasks
Record-Route: <sip:proxy.example.com;transport=udp;lr>,
<sip:h7kjh12s@proxy.example.com:443;transport=ws;lr>
From: sip:alice@example.com;tag=asdyka899
To: sip:bob@example.com;tag=bmqkjhsd
Call-ID: asidkj3ss
CSeq: 1 INVITE
Contact: <sip:bob@203.0.113.22:5060;transport=udp>
Content-Type: application/SDP
```

Figura B.11. Mensaje F5: 200 OK en WSS. [49]

```
F6 ACK Alice -> proxy.example.com (transport WSS)
ACK sip:bob@203.0.113.22:5060;transport=udp SIP/2.0
Via: SIP/2.0/WSS df7jal23ls0d.invalid;branch=z9hG4bKhgqqp090
Route: <sip:h7kjh12s@proxy.example.com:443;transport=ws;lr>,
<sip:proxy.example.com;transport=udp;lr>,
From: sip:alice@example.com;tag=asdyka899
To: sip:bob@example.com;tag=bmqkjhsd
Call-ID: asidkj3ss
CSeq: 1 ACK
Max-Forwards: 70
```

Figura B.12. Mensaje F6: ACK en WSS. [49]

```
F7 ACK proxy.example.com -> Bob (transport UDP)
ACK sip:bob@203.0.113.22:5060;transport=udp SIP/2.0
Via: SIP/2.0/UDPproxy.example.com;branch=z9hG4bKhwpoc80zzx
Via: SIP/2.0/WSS df7jal23ls0d.invalid;branch=z9hG4bKhgqqp090
From: sip:alice@example.com;tag=asdyka899
To: sip:bob@example.com;tag=bmqkjhsd
Call-ID: asidkj3ss
CSeq: 1 ACK
Max-Forwards: 69
```

Figura B.13. Mensaje F7: ACK en *UDP.* [49]

```
F8 BYE Bob -> proxy.example.com (transport UDP)
BYE sip:alice@example.com;gr=urn:uuid:f81-7dec-14a06cf1;ob
SIP/2.0
Via: SIP/2.0/UDP 203.0.113.22;branch=z9hG4bKbiuiansd001
Route: <sip:proxy.example.com;transport=udp;lr>,
<sip:h7kjh12s@proxy.example.com:443;transport=ws;lr>
From: sip:bob@example.com;tag=bmqkjhsd
To: sip:alice@example.com;tag=asdyka899
Call-ID: asidkj3ss
CSeq: 1201 BYE
Max-Forwards: 70
```

Figura B.14. Mensaje F8: BYE en *UDP*. [49]

```
F9 BYE proxy.example.com -> Alice (transport WSS)
BYE sip:alice@example.com;gr=urn:uuid:f81-7dec-14a06cf1;ob
SIP/2.0
Via: SIP/2.0/WSS proxy.example.com:443;branch=z9hG4bKmma01m3r5
Via: SIP/2.0/UDP 203.0.113.22;branch=z9hG4bKbiuiansd001
From: sip:bob@example.com;tag=bmqkjhsd
To: sip:alice@example.com;tag=asdyka899
Call-ID: asidkj3ss
CSeq: 1201 BYE
Max-Forwards: 69
```

Figura B.15. Mensaje F9: BYE en WSS. [49]

```
F10 200 OK Alice -> proxy.example.com (transport WSS)
SIP/2.0 200 OK
Via: SIP/2.0/WSS proxy.example.com:443;branch=z9hG4bKmma01m3r5
Via: SIP/2.0/UDP 203.0.113.22;branch=z9hG4bKbiuiansd001
From: sip:bob@example.com;tag=bmqkjhsd
To: sip:alice@example.com;tag=asdyka899
Call-ID: asidkj3ss
CSeq: 1201 BYE
```

Figura B.16. Mensaje F10: 200 OK en WSS. [49]

```
F11 200 OK proxy.example.com -> Bob (transport UDP)
SIP/2.0 200 OK
Via: SIP/2.0/UDP 203.0.113.22;branch=z9hG4bKbiuiansd001
From: sip:bob@example.com;tag=bmqkjhsd
To: sip:alice@example.com;tag=asdyka899
Call-ID: asidkj3ss
CSeq: 1201 BYE
```

Figura B.17. Mensaje F11: 200 OK en *UDP*. [49]

APÉNDICE C

NAT EN WEBRTC

Es posible establecer el flujo de *Media* sin el uso de *WebRTC* entre dos navegadores, sin embargo este flujo debe pasar por el servidor Web, y a continuación conectarse con el otro navegador como se muestra en la Figura C.1.

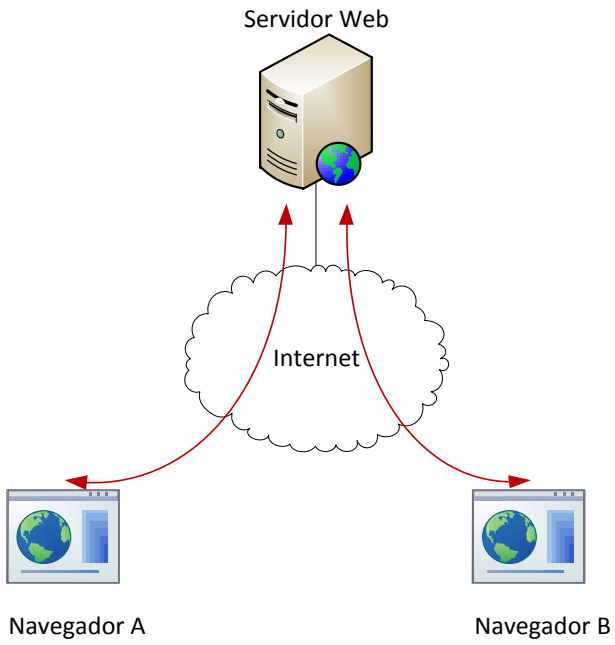


Figura C.1. Flujo de *Media* sin *WebRTC*. [56]

En cambio cuando se utiliza *WebRTC* por medio del *API RTCPeerConnection*, el flujo de *Media* es directo entre los navegadores,

como se muestra en la Figura C.2. Este flujo directo tiene la ventaja de tener menos saltos, por lo tanto menor latencia o retraso, lo cual es fundamental en la comunicación en tiempo real, además de reducir el consumo de tasa de bits al servidor Web.

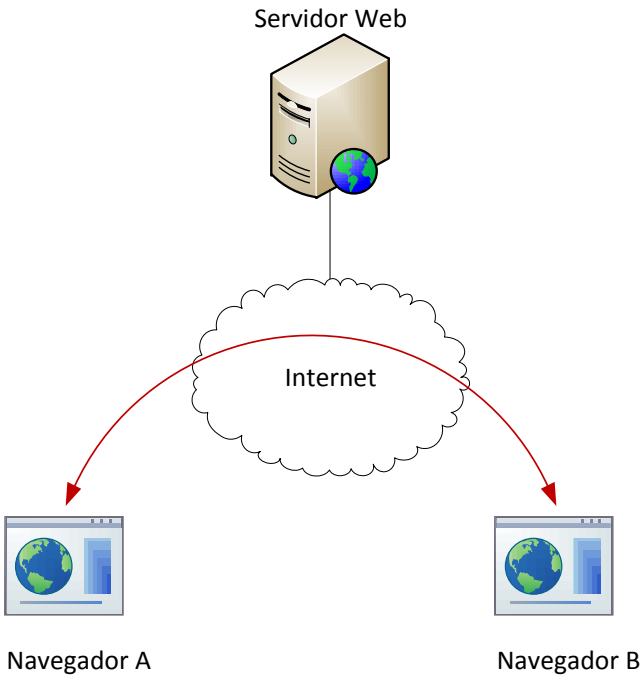


Figura C.2. Flujo de *Media* con *WebRTC*. [56]

Este tipo de comunicación directa entre navegadores sería la ideal, pero debido a que las dirección IPv4 públicas son finitas y existen muy pocas disponibles, se utiliza *NAT* para poder aumentar el número de equipos con conexión a internet. *NAT* es una manera de ocultar un rango de direcciones *IP* privadas detrás de una o varias direcciones *IP* públicas, generalmente.

Muchos protocolos especialmente aquellos que utilizan arquitecturas cliente-servidor y *TCP* como transporte, como se muestra en la Figura C.1 no tienen inconveniente con la utilización de *NAT*, pero para las conexiones entre pares como *P2P* (del inglés, *peer-to-peer*), la utilización de *NAT* y *UDP* como trasporte tiene mayores inconvenientes, como sería el caso la de la Figura C.2. Y como se describió en la sección 5.4.1, *WebRTC* utiliza ambos. Por lo tanto es necesario utilizar mecanismos para contrarrestar los inconvenientes de *NAT*, estos mecanismos son *STUN* y *TURN*.

STUN responde información sobre la dirección *IP* y los puertos de los paquetes en la que se recibieron, y que fue estandarizada por medio del RFC 5389 [62]. Y cuyo flujo se muestra en la Figura C.3.

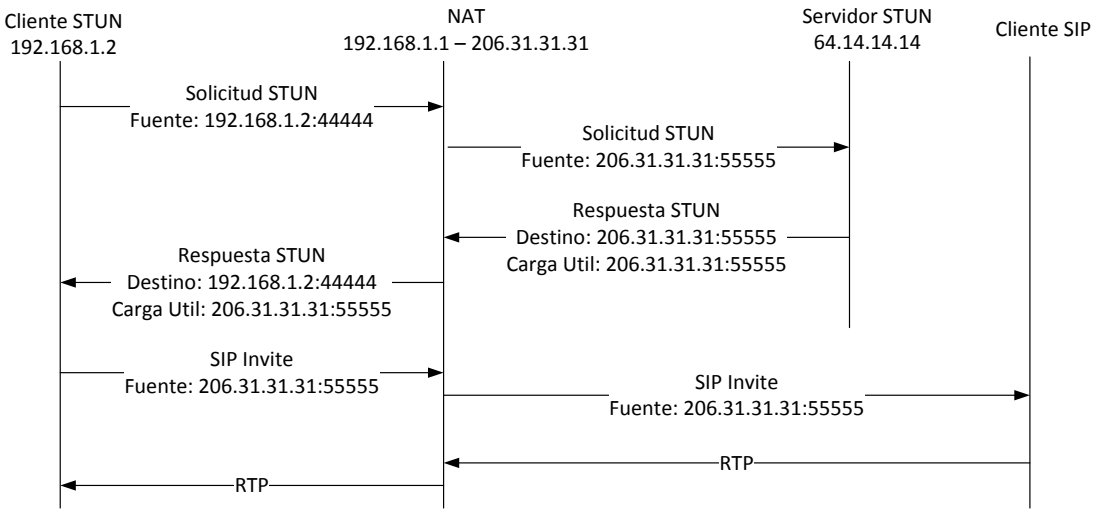


Figura C.3. Flujo de una llamada *SIP* con *STUN*. [65]

Los servidores *TURN* actúan como un relé, cualquier dato recibido se reenvía al cliente, y fue estandarizado por medio del RFC 5766 [63].

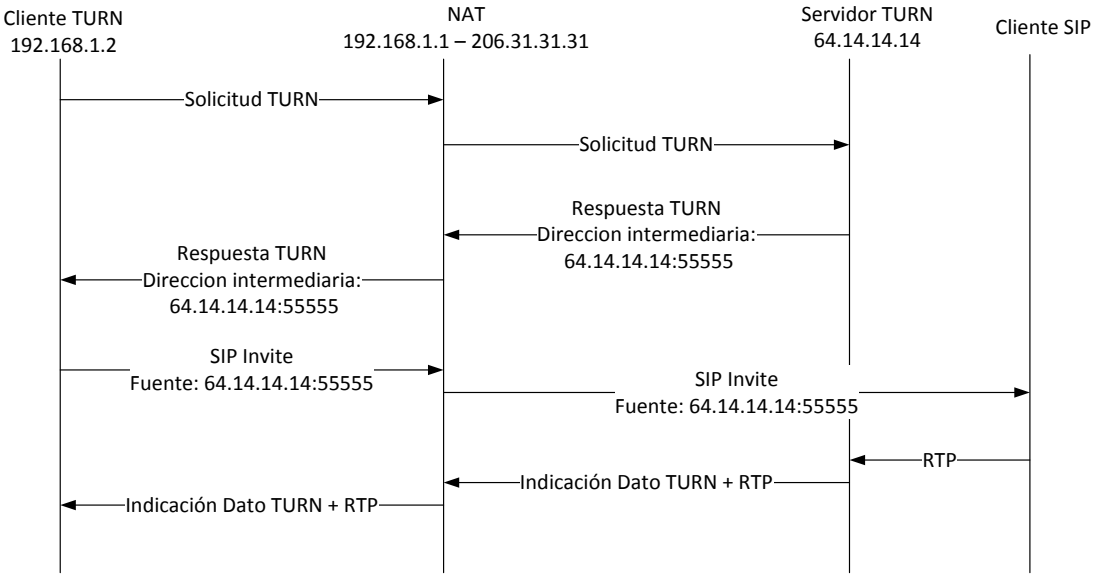


Figura C.4. Flujo de una llamada *SIP* con *TURN*. [65]

Sin embargo cuando *STUN* y *TURN* no funcionan correctamente sobre todo cuando existen *NAT* simétrico, se utiliza en estos casos *ICE* el cual hace una combinación de *TURN* y *STUN* para encontrar la forma más apropiada de comunicar los dispositivos finales. *ICE* fue estandarizado por medio del RFC 5245 [64]. *ICE* puede ser utilizado por cualquier protocolo que utilice el modelo solicitud-respuesta.

APÉNDICE D

PRUEBAS PRELIMINARES

Al ser *WebRTC* un esfuerzo para traer una *API* definida a los desarrolladores de *JavaScript*, que les permita acceder al mundo de las comunicaciones en tiempo real. Por lo cual para las implementaciones de *Asterisk* ha tenido el apoyo a *WebRTC* desde la versión 11, para ello un módulo *res_http_websocket* se ha creado para que permita a los desarrolladores de *JavaScript* interactuar y comunicarse con *Asterisk*. Así como el soporte para *WebSocket* como transporte ha sido añadido al módulo *chan_sip*, para permitir que *SIP* sea utilizado como protocolo de señalización. Como se muestra en la Figura D.1, toda la configuración se realiza en *sip.conf*, y se debe permitir utilizar *Websocket* como transporte, lo cual se logra con el contexto *ws*, y si se desea que solo los clientes Web utilicen *HTTPS* se debe configurar solo *wss*, o ambos si se permite el acceso de ambos. Ahora *WebRTC* utiliza *AVPF* como el perfil para los flujos de *Media*, y este no es agregado por defecto en *chan_sip*. Así como el cifrado de *Media* y que es necesario que sea agregado por ser un requisito.

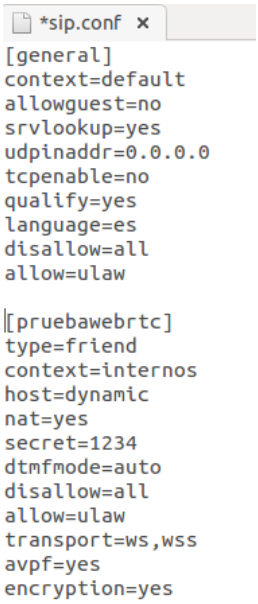


Figura D.1. Configuración de *sip.conf* para utilización de *WebSockets*

Otro requisito adicional es instalar librerías para la utilización de *SIP* con *WebSockets*, las cuales pueden ser *JsSIP* o *sipML5*. En las pruebas se instaló un demo de *JsSIP*, pero por falta de conocimientos en *Linux*, no se pudo hacer que funcione en su integración con *Asterisk* versión 11.1.0, pero muestra en la figura D.2 la versión instalada.

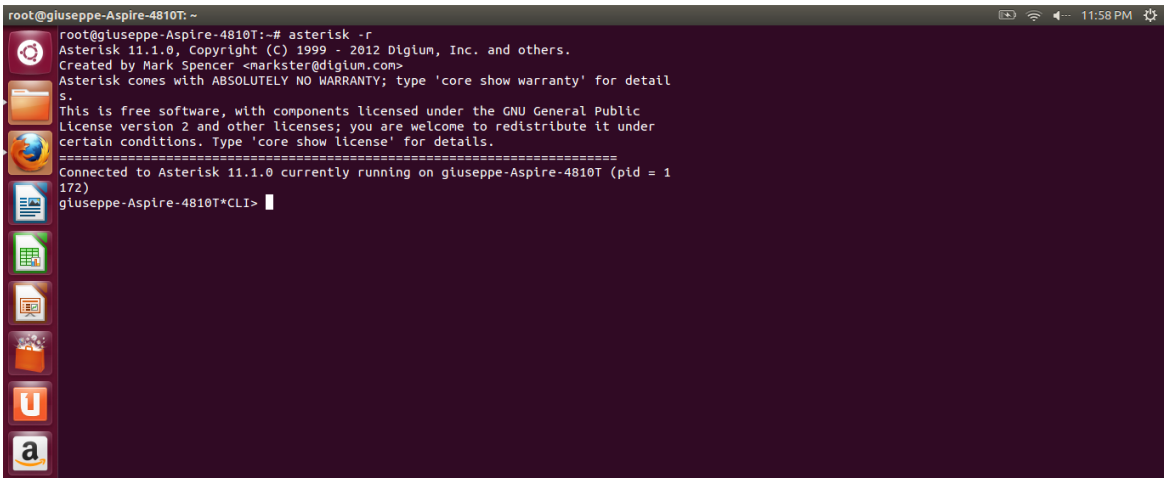


Figura D.2. *Asterisk* 11.1.0 instalado para pruebas con *WebSockets*

También existe otra versión de un servidor *SIP Proxy* que funciona directamente como un "SIP WebSocket Server", llamado *OverSIP* (Figura D.3) pero que no se lo pudo implementar por las mismas razones expuestas.

Figura D.3. *OverSIP*: SIP WebSocket Server

Otras pruebas que se realizaron fue con un demo de *WebRTC* el cual se lo encuentra en la página http://www.webrtc.org/demo, donde existe la aplicación AppRTC, https://apprtc.appspot.com/, al acceder a esta dirección se crea una sesión temporal a la cual puede acceder el otro equipo terminal (en la presente prueba fue https://apprtc.appspot.com/?r=26250700) y poder realizar una comunicación directa sin necesidad de ningún software preexistente, para lo cual el único requisito es que *WebRTC* solicita acceso a al audio y video local, tal como de describe en la sección 5.4.1 y como se muestra en la Figura D.4.

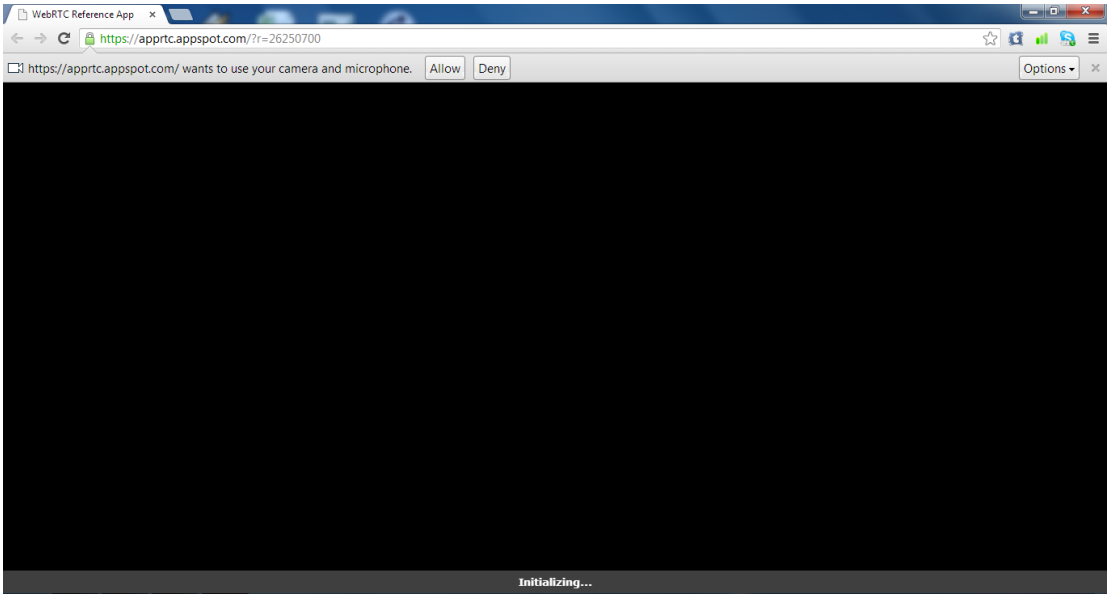


Figura D.4. *WebRTC* solicita acceso al audio y video local de equipo

Una vez se ha permitido el acceso al micrófono y cámara del equipo la comunicación se establece directamente entre ambos equipos, como se aprecia en la Figura D.5.

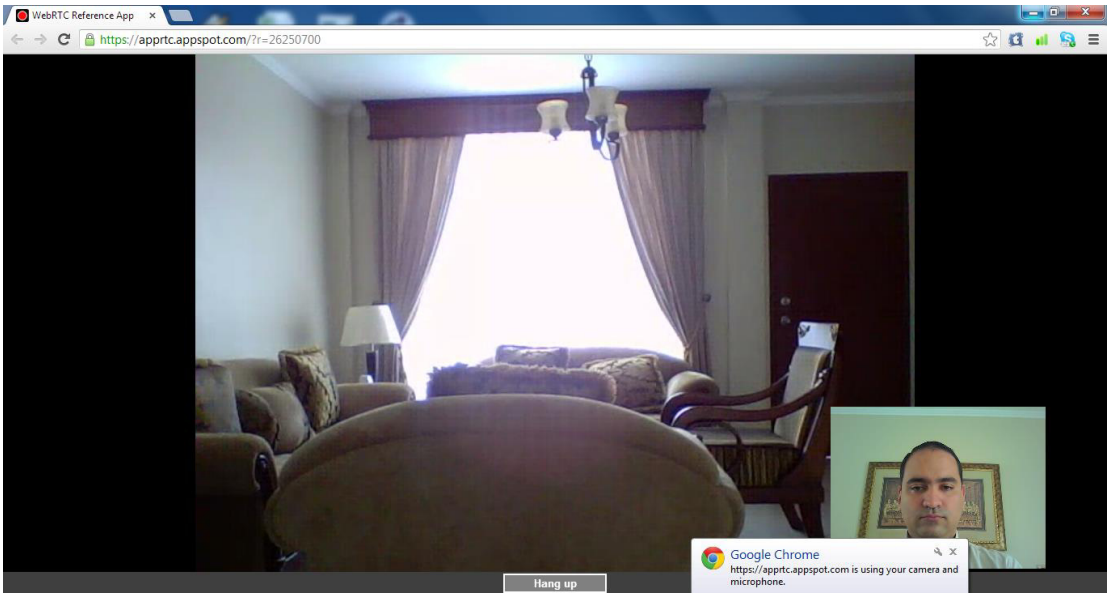


Figura D.5. Comunicación bidireccional a través de *WebRTC*

El navegador utilizado (Google Chrome) nos permite tener acceso al código fuente de la aplicación como se muestra en la Figura D.6, en la cual está escrito en *HTML5*, pero es mostrado en una vista resumen.

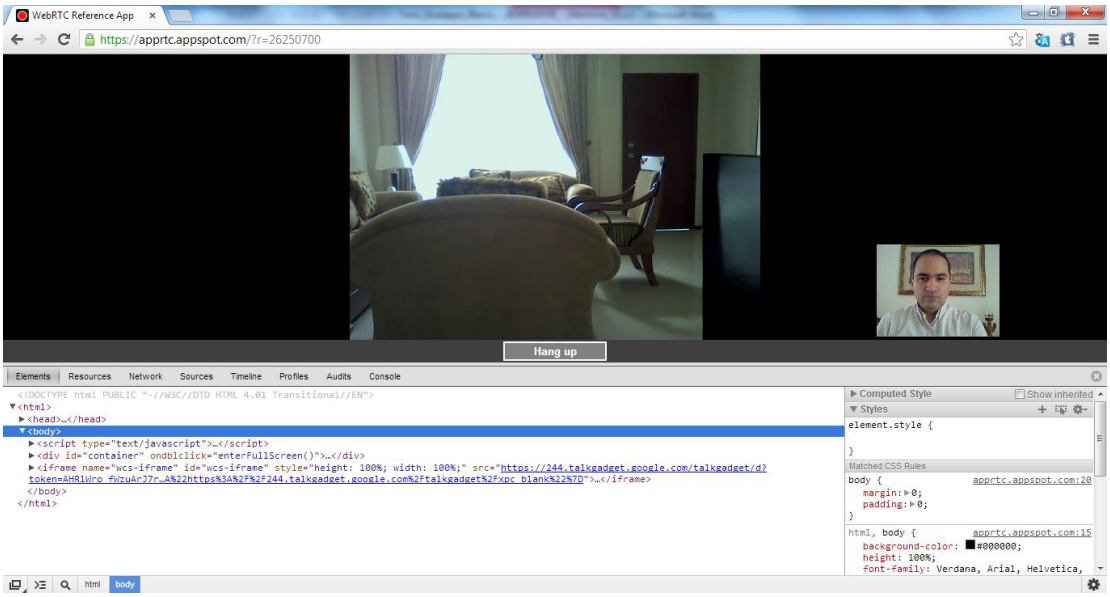


Figura D.6. Vista resumen de código fuente desde el navegador

Igualmente se puede acceder a todo el código de la aplicación como se muestra en la Figura D.7.

```
<html><head>
<title>WebRTC Reference App</title>
<link rel="canonical"
href="https://apprtc.appspot.com//?r=26250700">
<meta http-equiv="X-UA-Compatible" content="chrome=1">
<script src="/_ah/channel/jsapi"></script>

<!-- Load the polyfill to switch-hit between Chrome and Firefox
-->
<script src="/js/adapter.js"></script>

<style type="text/css">
  a:link { color: #ffffff; }
  a:visited {color: #ffffff; }
  html, body {
    background-color: #000000;
    height: 100%;
    font-family:Verdana, Arial, Helvetica, sans-serif;
  }
  body {
```

```
    margin: 0;
    padding: 0;
  }
  #container {
    background-color: #000000;
    position: relative;
    min-height: 100%;
    width: 100%;
    margin: 0px auto;
    -webkit-perspective: 1000;
  }
  #card {
    -webkit-transition-property: rotation;
    -webkit-transition-duration: 2s;
    -webkit-transform-style: preserve-3d;
  }
  #local {
    position: absolute;
    width: 100%;
    -webkit-transform: scale(-1, 1);
    -webkit-backface-visibility: hidden;
  }
  #remote {
    position: absolute;
    width: 100%;
    -webkit-transform: rotateY(180deg);
    -webkit-backface-visibility: hidden;
  }
  #mini {
    position: absolute;
    height: 30%;
    width: 30%;
    bottom: 32px;
    right: 4px;
    -webkit-transform: scale(-1, 1);
    opacity: 1.0;
  }
  #localVideo {
    opacity: 0;
    -webkit-transition-property: opacity;
    -webkit-transition-duration: 2s;
  }
  #remoteVideo {
    opacity: 0;
    -webkit-transition-property: opacity;
    -webkit-transition-duration: 2s;
  }
  #miniVideo {
    opacity: 0;
    -webkit-transition-property: opacity;
    -webkit-transition-duration: 2s;
  }
  #footer {
```

```
    spacing: 4px;
    position: absolute;
    bottom: 0;
    width: 100%;
    height: 28px;
    background-color: #3F3F3F;
    color: rgb(255, 255, 255);
    font-size:13px; font-weight: bold;
    line-height: 28px;
    text-align: center;
  }
  #hangup {
   font-size:13px; font-weight:bold;
   color:#FFFFFF;
   width:128px;
   height:24px;
   background-color:#808080;
   border-style:solid;
   border-color:#FFFFFF;
   margin:2px;
  }
  #logo {
    display: block;
    top:4;
    right:4;
    position:absolute;
    float:right;
    opacity: 0.5;
  }

</style>
<style>#wcs-iframe { display: none;
}</style><script>window["_GOOG_TRANS_EXT_VER"] =
"1";</script></head>
<body>
<script type="text/javascript">
  var localVideo;
  var miniVideo;
  var remoteVideo;
  var localStream;
  var remoteStream;
  var channel;
  var channelReady = false;
  var pc;
  var socket;
  var initiator = 0;
  var started = false;
  // Set up audio and video regardless of what devices are
present.
  var sdpConstraints = {'mandatory': {
                        'OfferToReceiveAudio':true,
                        'OfferToReceiveVideo':true }};
  var isVideoMuted = false;
```

```
  var isAudioMuted = false;

  function initialize() {
    console.log("Initializing; room=26250700.");
    card = document.getElementById("card");
    localVideo = document.getElementById("localVideo");
    miniVideo = document.getElementById("miniVideo");
    remoteVideo = document.getElementById("remoteVideo");
    resetStatus();
    // NOTE: AppRTCClient.java searches & parses this line;
update there when
    // changing here.

openChannel('AHRlWro_fWzuArJ7rScLQ2uKbvvyudTOJC6AWmTZJUm3Ce4ptLJ
ETYLMpInUNcGGeg4D3OrO0wqJQTNQJJz5uQSDUJRmz3pAoiG4CkYx7cUyw1p4jaE
eyhE');
    doGetUserMedia();
  }

  function openChannel(channelToken) {
    console.log("Opening channel.");
    var channel = new goog.appengine.Channel(channelToken);
    var handler = {
      'onopen': onChannelOpened,
      'onmessage': onChannelMessage,
      'onerror': onChannelError,
      'onclose': onChannelClosed
    };
    socket = channel.open(handler);
  }

  function resetStatus() {
    if (!initiator) {
      setStatus("Waiting for someone to join: <a
href=\"https://apprtc.appspot.com//?r=26250700\">https://apprtc.
appspot.com//?r=26250700</a>");
    } else {
      setStatus("Initializing...");
    }
  }

  function doGetUserMedia() {
    // Call into getUserMedia via the polyfill (adapter.js).
    var constraints = {"mandatory": {}, "optional": []};
    try {
      getUserMedia({'audio':true, 'video':constraints},
onUserMediaSuccess,
                   onUserMediaError);
      console.log("Requested access to local media with
mediaConstraints:\n" +
                  "  \"" + JSON.stringify(constraints) + "\"");
    } catch (e) {
```

```
      alert("getUserMedia() failed. Is this a WebRTC capable
browser?");
      console.log("getUserMedia failed with exception: " +
e.message);
    }
  }

  function createPeerConnection() {
    var pc_config = {"iceServers": [{"url":
"stun:stun.l.google.com:19302"}]};
    var pc_constraints = {"optional": [{"DtlsSrtpKeyAgreement":
true}]};
    // Force the use of a number IP STUN server for Firefox.
    if (webrtcDetectedBrowser == "firefox") {
      pc_config = {"iceServers":[{"url":"stun:23.21.150.121"}]};
    }
    try {
      // Create an RTCPeerConnection via the polyfill
(adapter.js).
      pc = new RTCPeerConnection(pc_config, pc_constraints);
      pc.onicecandidate = onIceCandidate;
      console.log("Created RTCPeerConnnection with:\n" +
                  "  config: \"" + JSON.stringify(pc_config) +
"\";\n" +
                  "  constraints: \"" +
JSON.stringify(pc_constraints) + "\".");
    } catch (e) {
      console.log("Failed to create PeerConnection, exception: "
+ e.message);
      alert("Cannot create RTCPeerConnection object; WebRTC is
not supported by this browser.");
        return;
    }

    pc.onaddstream = onRemoteStreamAdded;
    pc.onremovestream = onRemoteStreamRemoved;
  }

  function maybeStart() {
    if (!started && localStream && channelReady) {
      setStatus("Connecting...");
      console.log("Creating PeerConnection.");
      createPeerConnection();
      console.log("Adding local stream.");
      pc.addStream(localStream);
      started = true;
      // Caller initiates offer to peer.
      if (initiator)
        doCall();
    }
  }

  function setStatus(state) {
```

```
    footer.innerHTML = state;
  }

  function doCall() {
    var constraints = {"optional": [], "mandatory":
{"MozDontOfferDataChannel": true}};
    // temporary measure to remove Moz* constraints in Chrome
    if (webrtcDetectedBrowser === "chrome") {
      for (prop in constraints.mandatory) {
        if (prop.indexOf("Moz") != -1) {
          delete constraints.mandatory[prop];
        }
       }
     }
    constraints = mergeConstraints(constraints, sdpConstraints);
    console.log("Sending offer to peer, with constraints: \n" +
                "  \"" + JSON.stringify(constraints) + "\".")
    pc.createOffer(setLocalAndSendMessage, null, constraints);
  }

  function doAnswer() {
    console.log("Sending answer to peer.");
    pc.createAnswer(setLocalAndSendMessage, null,
sdpConstraints);
  }

  function mergeConstraints(cons1, cons2) {
    var merged = cons1;
    for (var name in cons2.mandatory) {
      merged.mandatory[name] = cons2.mandatory[name];
    }
    merged.optional.concat(cons2.optional);
    return merged;
  }

  function setLocalAndSendMessage(sessionDescription) {
    // Set Opus as the preferred codec in SDP if Opus is
present.
    sessionDescription.sdp = preferOpus(sessionDescription.sdp);
    pc.setLocalDescription(sessionDescription);
    sendMessage(sessionDescription);
  }

  function sendMessage(message) {
    var msgString = JSON.stringify(message);
    console.log('C->S: ' + msgString);
    // NOTE: AppRTCClient.java searches & parses this line;
update there when
    // changing here.
    path = '/message?r=26250700' + '&u=54132354';
    var xhr = new XMLHttpRequest();
    xhr.open('POST', path, true);
    xhr.send(msgString);
```

```
  }

  function processSignalingMessage(message) {
    var msg = JSON.parse(message);

    if (msg.type === 'offer') {
      // Callee creates PeerConnection
      if (!initiator && !started)
        maybeStart();

      pc.setRemoteDescription(new RTCSessionDescription(msg));
      doAnswer();
    } else if (msg.type === 'answer' && started) {
      pc.setRemoteDescription(new RTCSessionDescription(msg));
    } else if (msg.type === 'candidate' && started) {
      var candidate = new
RTCIceCandidate({sdpMLineIndex:msg.label,

candidate:msg.candidate});
      pc.addIceCandidate(candidate);
    } else if (msg.type === 'bye' && started) {
      onRemoteHangup();
    }
  }

  function onChannelOpened() {
    console.log('Channel opened.');
    channelReady = true;
    if (initiator) maybeStart();
  }
  function onChannelMessage(message) {
    console.log('S->C: ' + message.data);
    processSignalingMessage(message.data);
  }
  function onChannelError() {
    console.log('Channel error.');
  }
  function onChannelClosed() {
    console.log('Channel closed.');
  }

  function onUserMediaSuccess(stream) {
    console.log("User has granted access to local media.");
    // Call the polyfill wrapper to attach the media stream to
this element.
    attachMediaStream(localVideo, stream);
    localVideo.style.opacity = 1;
    localStream = stream;
    // Caller creates PeerConnection.
    if (initiator) maybeStart();
  }

  function onUserMediaError(error) {
```

```
    console.log("Failed to get access to local media. Error code
was " + error.code);
    alert("Failed to get access to local media. Error code was "
+ error.code + ".");
  }

  function onIceCandidate(event) {
    if (event.candidate) {
      sendMessage({type: 'candidate',
                   label: event.candidate.sdpMLineIndex,
                   id: event.candidate.sdpMid,
                   candidate: event.candidate.candidate});
    } else {
      console.log("End of candidates.");
    }
  }

  function onRemoteStreamAdded(event) {
    console.log("Remote stream added.");
    reattachMediaStream(miniVideo, localVideo);
    attachMediaStream(remoteVideo, event.stream);
    remoteStream = event.stream;
    waitForRemoteVideo();
  }
  function onRemoteStreamRemoved(event) {
    console.log("Remote stream removed.");
  }

  function onHangup() {
    console.log("Hanging up.");
    transitionToDone();
    stop();
    // will trigger BYE from server
    socket.close();
  }

  function onRemoteHangup() {
    console.log('Session terminated.');
    transitionToWaiting();
    stop();
    initiator = 0;
  }

  function stop() {
    started = false;
    isAudioMuted = false;
    isVideoMuted = false;
    pc.close();
    pc = null;
  }

  function waitForRemoteVideo() {
    // Call the getVideoTracks method via adapter.js.
```

```
    videoTracks = remoteStream.getVideoTracks();
    if (videoTracks.length === 0 || remoteVideo.currentTime > 0)
{
      transitionToActive();
    } else {
      setTimeout(waitForRemoteVideo, 100);
    }
  }
  function transitionToActive() {
    remoteVideo.style.opacity = 1;
    card.style.webkitTransform = "rotateY(180deg)";
    setTimeout(function() { localVideo.src = ""; }, 500);
    setTimeout(function() { miniVideo.style.opacity = 1; },
1000);
    setStatus("<input type=\"button\" id=\"hangup\" value=\"Hang
up\" onclick=\"onHangup()\" />");
  }
  function transitionToWaiting() {
    card.style.webkitTransform = "rotateY(0deg)";
    setTimeout(function() {
                localVideo.src = miniVideo.src;
                miniVideo.src = "";
                remoteVideo.src = "" }, 500);
    miniVideo.style.opacity = 0;
    remoteVideo.style.opacity = 0;
    resetStatus();
  }
  function transitionToDone() {
    localVideo.style.opacity = 0;
    remoteVideo.style.opacity = 0;
    miniVideo.style.opacity = 0;
    setStatus("You have left the call. <a
href=\"https://apprtc.appspot.com//?r=26250700\">Click here</a>
to rejoin.");
  }
  function enterFullScreen() {
    container.webkitRequestFullScreen();
  }

  function toggleVideoMute() {
    // Call the getVideoTracks method via adapter.js.
    videoTracks = localStream.getVideoTracks();

    if (videoTracks.length === 0) {
      console.log("No local video available.");
      return;
    }

    if (isVideoMuted) {
      for (i = 0; i < videoTracks.length; i++) {
        videoTracks[i].enabled = true;
      }
      console.log("Video unmuted.");
```

```
    } else {
      for (i = 0; i < videoTracks.length; i++) {
        videoTracks[i].enabled = false;
      }
      console.log("Video muted.");
    }

    isVideoMuted = !isVideoMuted;
  }

  function toggleAudioMute() {
    // Call the getAudioTracks method via adapter.js.
    audioTracks = localStream.getAudioTracks();

    if (audioTracks.length === 0) {
      console.log("No local audio available.");
      return;
    }

    if (isAudioMuted) {
      for (i = 0; i < audioTracks.length; i++) {
        audioTracks[i].enabled = true;
      }
      console.log("Audio unmuted.");
    } else {
      for (i = 0; i < audioTracks.length; i++){
        audioTracks[i].enabled = false;
      }
      console.log("Audio muted.");
    }

    isAudioMuted = !isAudioMuted;
  }

  setTimeout(initialize, 1);

  // Send BYE on refreshing(or leaving) a demo page
  // to ensure the room is cleaned for next session.
  window.onbeforeunload = function() {
    sendMessage({type: 'bye'});
  }

  // Ctrl-D: toggle audio mute; Ctrl-E: toggle video mute.
  // On Mac, Command key is instead of Ctrl.
  // Return false to screen out original Chrome shortcuts.
  document.onkeydown = function() {
    if (navigator.appVersion.indexOf("Mac") != -1) {
      if (event.metaKey && event.keyCode === 68) {
        toggleAudioMute();
        return false;
      }
      if (event.metaKey && event.keyCode === 69) {
        toggleVideoMute();
```

```
        return false;
      }
    } else {
      if (event.ctrlKey && event.keyCode === 68) {
        toggleAudioMute();
        return false;
      }
      if (event.ctrlKey && event.keyCode === 69) {
        toggleVideoMute();
        return false;
      }
    }
  }

  // Set Opus as the default audio codec if it's present.
  function preferOpus(sdp) {
    var sdpLines = sdp.split('\r\n');

    // Search for m line.
    for (var i = 0; i < sdpLines.length; i++) {
        if (sdpLines[i].search('m=audio') !== -1) {
          var mLineIndex = i;
          break;
        }
    }
    if (mLineIndex === null)
      return sdp;

    // If Opus is available, set it as the default in m line.
    for (var i = 0; i < sdpLines.length; i++) {
      if (sdpLines[i].search('opus/48000') !== -1) {
        var opusPayload = extractSdp(sdpLines[i], /:(\d+)
opus\/48000/i);
        if (opusPayload)
          sdpLines[mLineIndex] =
setDefaultCodec(sdpLines[mLineIndex], opusPayload);
        break;
      }
    }

    // Remove CN in m line and sdp.
    sdpLines = removeCN(sdpLines, mLineIndex);

    sdp = sdpLines.join('\r\n');
    return sdp;
  }

  function extractSdp(sdpLine, pattern) {
    var result = sdpLine.match(pattern);
    return (result && result.length == 2)? result[1]: null;
  }

  // Set the selected codec to the first in m line.
```

```
  function setDefaultCodec(mLine, payload) {
    var elements = mLine.split(' ');
    var newLine = new Array();
    var index = 0;
    for (var i = 0; i < elements.length; i++) {
      if (index === 3) // Format of media starts from the
fourth.
        newLine[index++] = payload; // Put target payload to the
first.
      if (elements[i] !== payload)
        newLine[index++] = elements[i];
    }
    return newLine.join(' ');
  }

  // Strip CN from sdp before CN constraints is ready.
  function removeCN(sdpLines, mLineIndex) {
    var mLineElements = sdpLines[mLineIndex].split(' ');
    // Scan from end for the convenience of removing an item.
    for (var i = sdpLines.length-1; i >= 0; i--) {
      var payload = extractSdp(sdpLines[i], /a=rtpmap:(\d+)
CN\/\d+/i);
      if (payload) {
        var cnPos = mLineElements.indexOf(payload);
        if (cnPos !== -1) {
          // Remove CN payload from m line.
          mLineElements.splice(cnPos, 1);
        }
        // Remove CN line in sdp
        sdpLines.splice(i, 1);
      }
    }

    sdpLines[mLineIndex] = mLineElements.join(' ');
    return sdpLines;
  }
</script>
<div id="container" ondblclick="enterFullScreen()">
  <div id="card" style="-webkit-transform: rotateY(180deg);">
    <div id="local">
      <video width="100%" height="100%" id="localVideo"
autoplay="autoplay" muted="true" src="" style="opacity: 1;">
    </video></div>
    <div id="remote">
      <video width="100%" height="100%" id="remoteVideo"
autoplay="autoplay"
src="blob:https%3A//apprtc.appspot.com/ea12c26d-8793-4ccb-82d9-
c47541ad6447" style="opacity: 1;">
      </video>
      <div id="mini">
        <video width="100%" height="100%" id="miniVideo"
autoplay="autoplay" muted="true"
```

```
src="blob:https%3A//apprtc.appspot.com/20df0914-ff63-4999-b65a-
af73698dfdc9" style="opacity: 1;">
      </video></div>
    </div>
  </div>
  <div id="footer"><input type="button" id="hangup" value="Hang
up" onclick="onHangup()"></div>
</div>

<iframe name="wcs-iframe" id="wcs-iframe" style="height: 100%;
width: 100%;"
src="https://244.talkgadget.google.com/talkgadget/d?token=AHRlWr
o_fWzuArJ7rScLQ2uKbvvyudTOJC6AWmTZJUm3Ce4ptLJETYLMpInUNcGGeg4D3O
rO0wqJQTNQJJz5uQSDUJRmz3pAoiG4CkYx7cUyw1p4jaEeyhE&amp;xpc=%7B%22
cn%22%3A%226mVrDjSKXU%22%2C%22tp%22%3Anull%2C%22osh%22%3Anull%2C
%22ppu%22%3A%22https%3A%2F%2Fapprtc.appspot.com%2F_ah%2Fchannel%
2Fxpc_blank%22%2C%22lpu%22%3A%22https%3A%2F%2F244.talkgadget.goo
gle.com%2Ftalkgadget%2Fxpc_blank%22%7D"></iframe></body></html>
```

Figura D.7. Código fuente demo *WebRTC*

A esta prueba se realizó una captura con el *software* Wireshark, como se aprecia en la Figura D.8, donde se aprecia la negociación de *STUN* para los terminales utilizados. Mientas en la Figura D.9 se muestra la captura de *Media* en *WebRTC*. Un aspecto a considerar que a pesar que *WebRTC* trabaja necesariamente con *SRTP* como protocolo de transporte, este es mostrado solo como *UDP*, lo cual es una limitante del *software* de captura.

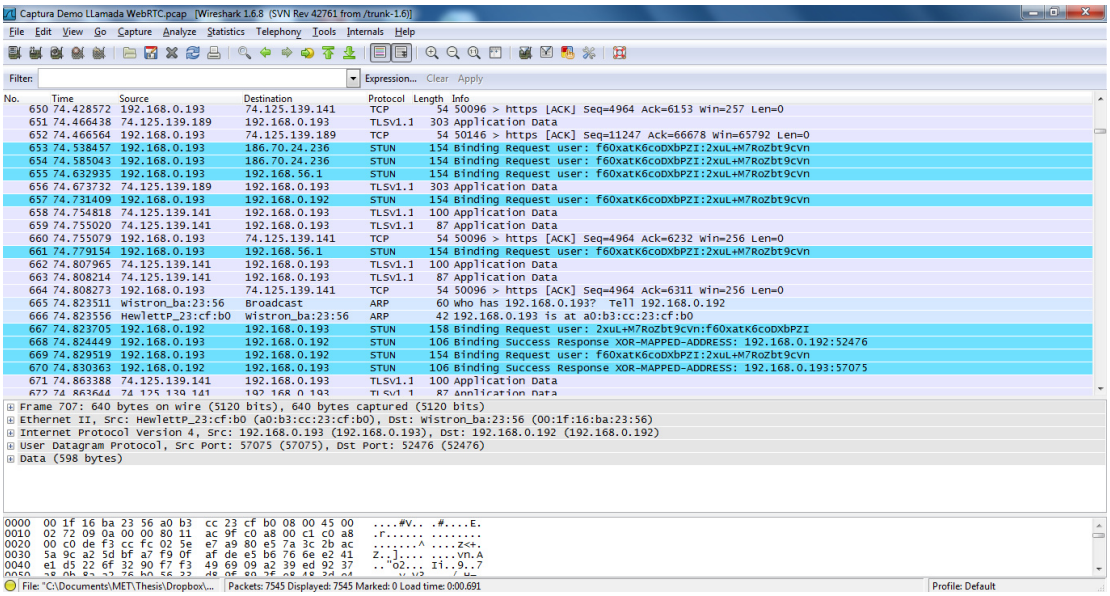


Figura D.8. Captura con Wireshark de negociación *STUN* para terminales

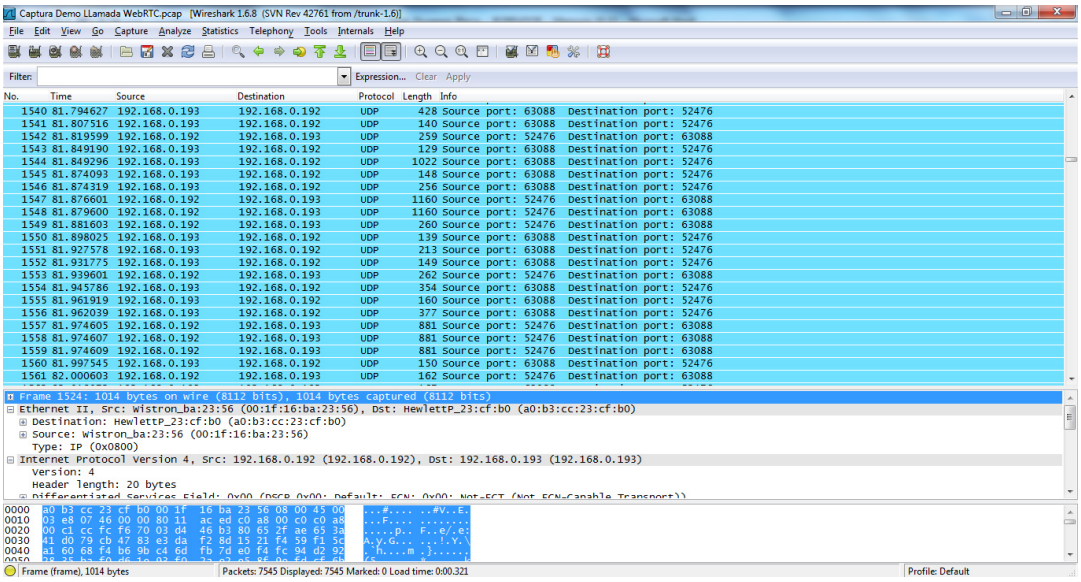


Figura D.9. Captura con Wireshark de media en *WebRTC*

En la Figura D.10 se muestra la información detallada de una trama en *WebRTC*, mientras que en la Figura D.11 se aprecia la jerarquía de protocolos de la captura donde el mayor porcentaje corresponde a *Media*.

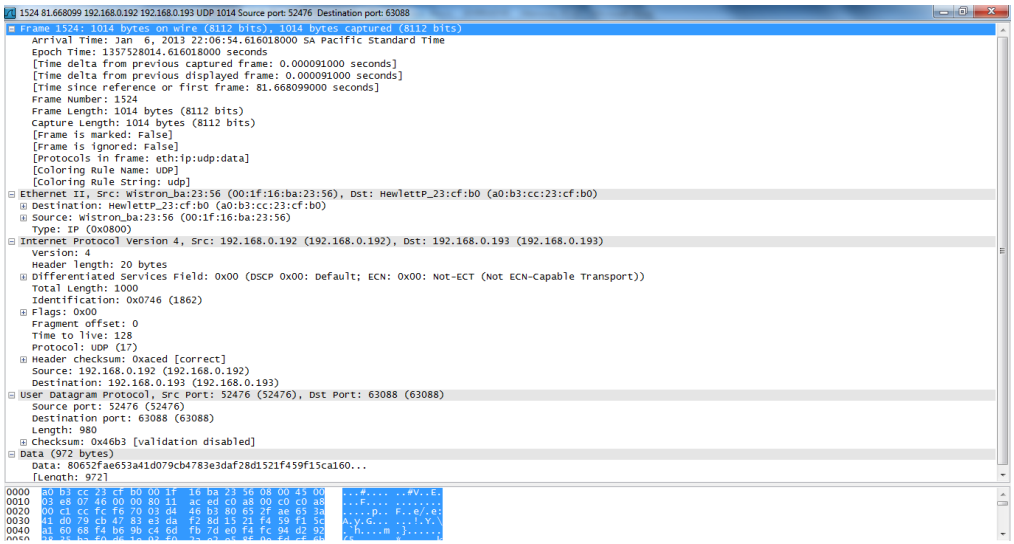


Figura D.10. Información detallada de una trama en *WebRTC*

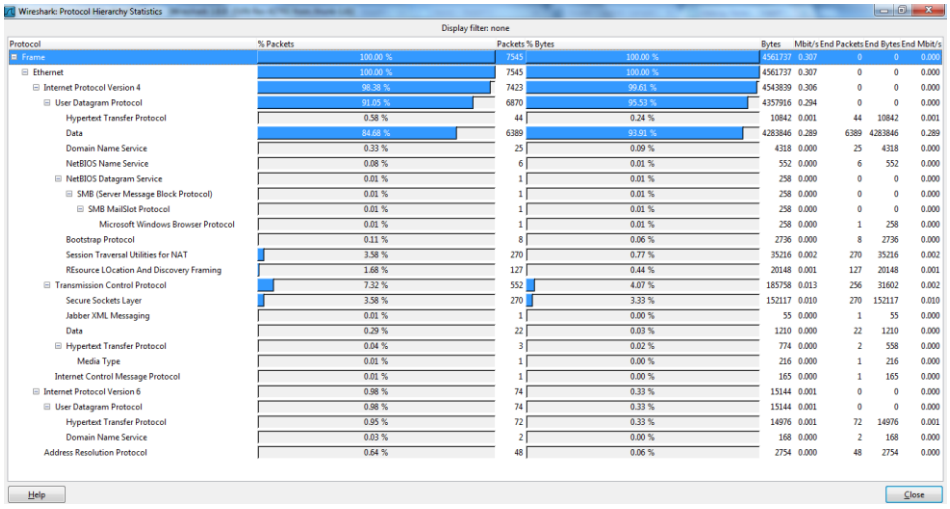


Figura D.11. Jerarquía de Protocolos en captura *WebRTC*

BIBLIOGRAFÍA

[1] Yankee Group, 2012 Mobility Predictions: A Year of Living Dangerously, 2011

[2] Vaughan-Nichols, S.J., Will HTML5 Standardize the Web?, IEEE Computer, Vol. 43 (4), 13-15, 2010

[3] Mateos, A. y Rosenberg, J., The Cloud at Your Service, 2011

[4] Sosinsky , B., Cloud Computing Bible, 2011

[5] Elsenpeter, R., Velte, A. y Velte, T., Cloud Computing: A Practical Approach, 2010

[6] Krutz, R. y Vines, R.V., Cloud Security: A Comprehensive Guide to Secure Cloud Computing, 2010

[7] National Institute of Standards and Technology, Cloud computing: A review of features, benefits, and risks, and recommendations for secure, efficient implementations. Information Technology Laboratory, Radack, S. extraído desde http://csrc.nist.gov/publications/nistbul/june-2012_itl-bulletin.pdf, 2012

[8] Miller, M., Cloud Computing: Web-Based Applications That Change the Way You Work and Collaborate Online, 2009

[9] Sun Microsystems, Introduction to Cloud Computing Architecture, extraído desde http://eresearch.wiki.otago.ac.nz/images/7/75/Cloudcomputing.pdf, 2009

[10] Dillon, T., Chang, E. y Wu, C., Cloud Computing: Issues and Challenges. 24th IEEE International Conference on Advanced Information Networking and Applications, extraído desde http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=5474674&url=http%3A%2F%2Fieeexplore.ieee.org%2Fxpls%2Fabs_all.jsp%3Farnumber%3D5474674, 2010

[11] Dalgic, I., y Fang H., Comparison of H.323 and SIP for IP Telephony Signaling, extraído desde http://www.cs.columbia.edu/~hgs/papers/others/1999/Dalg9909_Comparison.pdf, 1999

[12] Nagireddi, S., VoIP Voice and Fax Signal Processing, 2008

[13] Wallace, K., Cisco Voice over IP (CVOICE), Cisco Systems, Inc., 2009

[14] Pallis, G., A comparative study of in-band and out-of-band VoIP Protocols in layer 3 and layer 2.5 Environments, School of Computing, Edinburgh Napier University, extraído desde http://researchrepository.nAPIer.ac.uk/4005/1/george.pdf, 2010

[15] ITU-T, H.323, extraído desde http://www.itu.int/rec/T-REC-H.323-199611-S/en, 1996

[16] Handley, M., Schulzrinne, H., Scholler, E., Rosenberg, J., SIP: Session Initiation Protocol. IETF. RFC 2543, extraído desde http://www.ietf.org/rfc/rfc2543.txt, 1999

[17] Johnston, A., SIP: Understanding the Session Initiation Protocol, 2004

[18] Spencer, M., Capouch, B., Guy, E., Miller, F. y Shumard, K., IAX: Inter-Asterisk exchange Version 2. IETF RFC 5456, extraído desde https://tools.ietf.org/HTML/rfc5456, 2010

[19] Van Meggelen, J., Madsen, L. y Smith, J., Asterisk: The Future of Telephony, extraído desde http://cdn.oreillystatic.com/books/9780596510480.pdf, 2007

[20] Bergkvist, A., Burnett, D. , Jennings, C. y Narayanan, A, WebRTC 1.0: Real-time Communication Between Browsers extraído desde http://www.w3.org/TR/WebRTC/, 2012

[21] Ohrtman, F., Softswitch Architecture for VoIP, 2004

[22] Johnson, A., 31 Days before your CCNA Exam. Cisco Systems, Inc., 2009

[23] Balchunas, A, Routing Protocol Comparison v1.01, extraído desde http://www.routeralley.com/ra/docs/routing_protocol_comparison.pdf, 2007

[24] Cisco Systems, Inc., Cisco Active Network Abstraction 3.7 Reference Guide, 2010

[25] Pepelnjak, I., Advanced BGP network design for stability and security, extraído desde http://searchtelecom.techtarget.com/feature/Advanced-BGP-network-design-for-stability-and-security#reasons, 2008

[26] Pepelnjak, I., BGP essentials: The protocol that makes the Internet work, extraído desde http://searchtelecom.techtarget.com/feature/BGP-essentials-The-protocol-that-makes-the-Internet-work, 2007

[27] Kushman, N, Kandula, S. y Katabi, D, Can You Hear Me Now?! It Must Be BGP, 2007

[28] Davis, D., What you need to know about BGP routing protocol, extraído desde http://www.petri.co.il/csc_what_is_bgp.htm, 2009

[29] Schulzrinne, H., Casner, S., Frederick, R. y Jacobson, V., RTP: A Transport Protocol for Real-Time Applications. IETF. RFC 3550, extraído desde http://tools.ietf.org/HTML/rfc3550#page-8, 2003

[30] He, Q., Analyzing the Characteristics of VoIP Traffic, Department of Computer Science University of Saskatchewan, extraído desde

http://www.collectionscanada.gc.ca/obj/s4/f2/dsk3/SSU/TC-SSU-07132007120004.pdf, 2007

[31] Bruno, A. y Jordan, S., CCDA 640-864 Official Cert Guide. Cisco Systems, Inc. , 2011

[32] Scott, C., Wolfe, P. y Erwin, M., Virtual Private Networks, 1999

[33] Root, D. y Rissler, R., IPSec and SSL VPN Decision Criteria.Juniper Networks, extraído desde http://www.cadincweb.com/wordpress/wp-content/uploads/2010/11/Juniper-IPSec-vs-SSL-VPN.pdf, 2006

[34] Frahim, J. y Santos, O. Cisco ASA: All-in-One Firewall, IPS, and VPN Adaptive Security Appliance. Cisco Systems, Inc., 2005

[35] Teare, D., Implementing Cisco IP Routing (ROUTE) Foundation Learning Guide , 2010

[36] Pérez, N., Planificación y Dimensionamiento de Sistemas Celulares y de Radio Acceso, 2012.

[37] Lowery, J. y Fletcher, M., HTML5 24-Hour Trainer, 2011

[38] Hogan, B., HTML5 and CSS3: Develop with Tomorrow’s Standards Today, 2010

[39] Sikos, L., Web Standards, Mastering HTML5, CSS3 and XML, 2010

[40] Reddy, M., API Design for C++, 2011

[41] Google Developers, Google Books API Family, extraído desde https://developers.google.com/books/docs/v1/getting_started#background-concepts, 2012

[42] W3C, HTML 5.1. A vocabulary and associated APIs for HTML and XHTML, extraído desde http://www.w3.org/TR/HTML51/, 2012

[43] Fette, I. y Melnikov, A., The WebSocket Protocol, RFC 6455, extraído desde http://tools.ietf.org/HTML/rfc6455, 2011

[44] Hickson, I., The WebSocket API, W3C Candidate Recommendation, September 2012 extraído desde http://www.w3.org/TR/WebSockets/, 2012

[45] Stevens, W., Fenner, B. y Rudoff, A., UNIX Network Programming: Vol. 1: The Sockets Networking API, 2004

[46] IBM, IBM i 7.1 Information Center. How sockets work, extraído desde, http://publib.boulder.ibm.com/infocenter/iseries/v5r4/index.jsp?topic=%2Frzab6%2Fhowdosockets.htm, 2012

[47] Tonon, L., Marakana HTML5 Tutorial, extraído desde http://marakana.com/bookshelf/html5_tutorial/web_sockets.html, 2012

[48] Lubbers P. y Greco, F., Kaazing Corporation. HTML5 Web Sockets: A Quantum Leap in Scalability for the Web, extraído desde http://www.websocket.org/quantum.HTML, 2012

[49] Baz, I. y Millan J., The WebSocket Protocol as a Transport for the Session Initiation Protocol (SIP).draft-ietf-sipcore-sip-websocket-06, extraído desde https://datatracker.ietf.org/doc/draft-ietf-sipcore-sip-websocket/?include_text=1, 2012

[50] Valin, JM. y Bran, C., WebRTC Audio Codec and Processing Requirements, draft-ietf-rtcweb-audio-01, extraído desde http://tools.ietf.org/pdf/draft-ietf-rtcweb-audio-01.pdf, 2012

[51] Screene, R., WebRTC Audio Codecs: Opus and G.711 extraído desde http://thisisdrum.com/blog/tag/opus/, 2012

[52] Valin, JM ,Vos, K. y Terriberry, T., Definition of the Opus Audio Codec. (RFC 6716) extraído desde http://tools.ietf.org/pdf/rfc6716.pdf, 2012

[53] PKE Consulting, Introduction to WebRTC, extraído desde http://www.pkeconsulting.com/pkeWebRTC.pdf, 2012

[54] Dutton, S., WebRTC Plugin-free real-time communication, 2012

[55] Alvestrand, H., Overview: Real Time Protocols for Brower-based Applications, draft-ietf-rtcweb-overview-05, extraído desde http://tools.ietf.org/pdf/draft-ietf-rtcweb-overview-05.pdf, 2012

[56] Johnston, A. y Burnett, D., WebRTC: APIs and RTCWEB Protocols of the HTML5 Real-Time Web, 2012

[57] Handley, M., Jacobson, V. y Perkins, C., SDP: Session Description Protocol, IETF. RFC 4566, extraído desde http://tools.ietf.org/html/rfc4566, 2006

[58] Eriksson, G. y Håkansson, S., WebRTC: enhancing the web with real-time communication capabilities, extraído desde http://www.ericsson.com/res/thecompany/docs/publications/ericsson_review/2012/er-WebRTC-HTML5.pdf, 2012

[59] Google, WebRTC Code and API > Architecture, extraído desde http://www.webrtc.org/reference/architecture#TOC-Session-Management, 2011

[60] W3C, WebRTC 1.0: Real-time Communication Between Browsers, extraído desde http://www.w3.org/TR/webrtc/, 2013

[61] Dixit, S., An Introduction To WebSockets, extraído desde http://www.developerfusion.com/article/143158/an-introduction-to-WebSockets/, 2012

[62] Rosenberg, J., Mahy, R., Matthews, P. y Wing, D., Session Traversal Utilities for NAT (STUN).IETF. RFC 5389, extraído desde https://tools.ietf.org/HTML/rfc5389, 2008

[63] Mahy. R., Matthews, P. y Rosenberg, J., Traversal Using Relays around NAT (TURN). IETF. RFC 5766, extraído desde https://tools.ietf.org/html/rfc5766, 2010

[64] Rosenberg, J., Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols. IETF. RFC 5245, extraído desde https://tools.ietf.org/html/rfc5245, 2010

[65] Perreault, S. Viagénie. NAT and Firewall Traversal with STUN / TURN / ICE, extraido desde http://www.viagenie.ca/publications/2008-09-24-astricon-stun-turn-ice.pdf