SOFTWARE LIBRE PARA LA EVALUACION DE CALIDAD DE SERVICIO EN VOZ SOBRE IP

OSCAR JOVANNI MAESTRE SANMIGUEL
JAVIER EDUARDO GARCÍA PRADA

UNIVERSIDAD PONTIFICIA BOLIVARIANA SECCIONAL BUCARAMANGA
ESCUELA DE INGENIERÍA
FACULTAD INGENIERÍA DE SISTEMAS E INFORMÁTICA
BUCARAMANGA
2015

SOFTWARE LIBRE PARA LA EVALUACION DE CALIDAD DE SERVICIO EN VOZ SOBRE IP

OSCAR JOVANNI MAESTRE SANMIGUEL
JAVIER EDUARDO GARCIA PRADA

TRABAJO DE GRADO PARA OPTAR POR EL TÍTULO DE INGENIERO DE SISTEMAS E INFORMÁTICA

DIRECTORA
ANGÉLICA FLÓREZ ABRIL

UNIVERSIDAD PONTIFICIA BOLIVARIANA SECCIONAL BUCARAMANGA
ESCUELA DE INGENIERÍA
FACULTAD INGENIERÍA DE SISTEMAS E INFORMÁTICA
BUCARAMANGA
2015

AGRADECIMIENTOS

Agradecemos a nuestras familias por intentar creer en nosotros y nunca perder las esperanzas, sobre todo por el apoyo económico brindado en todos estos años de formación.

A la docente Diana Teresa Gómez por su incondicional apoyo y el tiempo dedicado para brindarnos sus valiosos aportes.

A nuestros profesores por toda esa paciencia y por compartirnos sus conocimientos ya sea por obligación o devoción pero de alguna manera lo hicieron.

A todos nuestros amigos, conocidos y desconocidos que nos motivaron a seguir adelante, mil gracias a todos ellos.

CONTENIDO

LISTA DE TABLAS

LISTA DE FIGURAS

LISTA DE ANEXOS

RESUMEN GENERAL DE TRABAJO DE GRADO

TITULO:	SOFTWARE LIBRE PARA LA EVALUACION DE CALIDAD DE SERVICIO EN VOZ SOBRE IP
AUTOR(ES):	Javier Eduardo García Prada Oscar Jovanni Maestre Sanmiguel
FACULTAD:	Facultad de Ingeniería de Sistemas e Informática
DIRECTOR(A):	Angélica Flórez Abril

RESUMEN

Este documento contiene el informe del proyecto de grado donde se plasma el desarrollo de iQos, un software libre para evaluación de métricas calidad de servicio en voz sobre IP. Durante el proceso de creación de este software, se hizo necesario la adopción de una metodología acorde a los requerimientos y también se escogieron un conjunto de herramientas y estándares abiertos, que se integraron para medir el desempeño de una red de datos en el momento que se realiza una o múltiples llamadas telefónicas de voz sobre IP. El proyecto fue llevado a cabo por etapas de análisis, diseño, selección, integración y pruebas, se involucran diversas tecnologías de comunicación y arquitecturas de software.

PALABRAS CLAVES:

Calidad de servicio, software libre, VoIP, redes de datos

V° B° DIRECTOR DE TRABAJO DE GRADO

GENERAL SUMMARY OF WORK OF GRADE

TITLE: A FREE SOFTWARE FOR EVALUATE VOICE OVER IP QUALITY OF SERVICE

AUTHOR(S): Javier Eduardo García Prada
Oscar Jovanni Maestre Sanmiguel

FACULTY: Facultad de Ingeniería de Sistemas e Informática

DIRECTOR: Angelica Flórez Abril

ABSTRACT

This document contains the report of the graduation project. It contains a description of the development of iQos, an open source software that evaluates quality of service metrics in voice over IP. During the software creation process, a methodology was adopted to comply with the requirements. Additionally, a set of tools and standards were chosen to measure the data network performance at the same moment that calls were made through voice over IP. The project was completed through states of analysis, design, selection, integration, and testing. Many communication technologies and software architectures were used.

KEYWORDS:

Quality of service, free software, VoIP, computer networks

V° B° DIRECTOR OF GRADUATE WORK

INTRODUCCIÓN

Internet ha logrado posicionarse como la red global de las comunicaciones, esto gracias a la masificación y acogida que ha tenido por parte de numerosos usuarios alrededor del mundo. Con el transcurrir del tiempo, el conjunto de tecnologías que se han derivado de Internet ofrecen una gama de servicios que se adaptan a las necesidades específicas de las personas, lo que ha conllevado al desplazamiento de formas tradicionales de comunicación como es el caso de la telefonía tradicional. Las comunicaciones por voz han escalado a un nivel complejo gracias a la tecnología denominada Voz sobre IP (VoIP). Las empresas se han interesado por el abanico de servicios y ventajas que éste tipo de tecnología ofrece en el ámbito de los negocios; hoy en día es común hablar de aplicativos de software como Skype, Hangouts, Viber, Tango y muchas otras donde millones de usuarios las utilizan y se conectan diariamente, logrando a través de las mismas comunicarse sin importar distancias ni ubicación geográfica.

La implementación de VoIP en las organizaciones requiere de una evaluación técnica y de un análisis detallado, donde se debe tener en cuenta la capacidad de la infraestructura tecnológica y el estado actual del canal de comunicación por donde van a viajar los paquetes para garantizar la calidad de dicho servicio. Para llevar a cabo esas tareas existen herramientas de software que pueden ayudar a diagnosticar la viabilidad en la implementación de servicios de VoIP en las organizaciones. En este documento se describe una propuesta para el desarrollo de un software libre denominado iQos que evalúa la calidad de servicio de una red de datos para el uso de tecnología VoIP.

1. ESPECIFICACIÓN DEL PROYECTO

1.1 RESEÑA HISTÓRICA DEL PROYECTO iQos

En el año 2013 se reunió un grupo de estudiantes pertenecientes al Semillero de Investigación en Redes y Seguridad Informática (SIRESI) de la Universidad Pontificia Bolivariana Seccional Bucaramanga con el fin de llevar a cabo un proyecto de investigación orientado a evaluar el comportamiento de los protocolos involucrados en las comunicaciones de Voz sobre IP. A raíz de esto se propuso el desarrollo de un software para evaluar métricas de calidad de servicio usando herramientas de código y estándares abiertos, cuyo nombre es iQos. Lo anterior permitiría recopilar información y obtener medidas que ayudan a analizar el comportamiento de los enlaces de comunicaciones bajo condiciones particulares.

El equipo de trabajo propuso entonces una metodología de desarrollo y un conjunto de herramientas de código abierto que se pudieran integrar con el software iQos junto con tecnologías modernas, siendo necesario involucrar las Redes de datos y la ingeniería del software.

1.2 ESPECIFICACIÓN DE LA SITUACIÓN PROBLEMÁTICA

El éxito que ha tenido Internet en estos últimos años ha traído consigo un constante descenso en el uso de la telefonía fija tradicional por parte de organizaciones y personas alrededor del mundo; para el año 2018, según estimaciones del servicio de investigación y consultoría de TeleGeography[1], el número de suscriptores en servicios de VoIP tendrán un incremento del 250% y la telefonía fija un descenso del 33% con respecto al 2008.

Las empresas son quizás quienes más se benefician de las ventajas inherentes que posee VoIP, atraídas por el bajo costo y la versatilidad que ofrece esta tecnología para mejorar la competitividad, sobre todo si se tiene en cuenta el auge de los mercados globales que requieren formas nuevas de comunicación para favorecer la comunicación empresarial. Muchas de estas empresas optan por implementar sus propias centrales telefónicas IP o contratar con un proveedor servicios donde el único requisito es contar con una red de datos, que permita el flujo de información entre los nodos que participan en la comunicación.

[1] TELEGEOGRAPHY, International. Fixed line telephony: a far cry from dead. En: Telegeography. [En línea]. (30, Abril, 2015). Disponible en: <https://www.telegeography.com/products/commsupdate/articles/2014/10/15/fixed-line-telephony-a-far-cry-from-dead/>

No obstante, existe una necesidad relevante que exige ciertos requisitos a la hora de implementar VoIP, debido a que en las redes de datos se utiliza la conmutación por paquetes (contrario a la conmutación de circuitos usada por la telefonía fija tradicional) lo que conlleva a compartir el tráfico de red con otros protocolos y servicios. Esto es importante porque la calidad del servicio depende de ciertos parámetros que deben ser medidos y evaluados en la infraestructura de red y en el canal de comunicación. VoIP es sensible a las distorsiones en la voz, retardos y fluctuaciones que deben ser identificadas y mitigadas aplicando pruebas rigurosas para garantizar la calidad del servicio.

Si bien, existe software especializado para realizar este tipo de mediciones, cabe resaltar que son costosos y en muchas ocasiones complicados de usar. En la actualidad existen pocos aplicativos de software libre que tengan una interfaz gráfica intuitiva y fácil de usar para un administrador de red o personal de un Departamento de Tecnologías de Información de una organización, que dicho software permita unificar un conjunto de pruebas como por ejemplo mediciones de tráfico personalizado entre dos extremos, sea unidireccional o bidireccional, que lleve a cabo la recreación de llamadas telefónicas con VoIP y que realice pruebas de Jitter, latencia y pérdida de paquetes.

En este proyecto se busca responder la pregunta: ¿Cómo evaluar la calidad de servicio en llamadas VoIP mediante una herramienta de software libre que recree llamadas usando el protocolo RTP?

1.3 OBJETIVOS

1.3.1 Objetivo General

Desarrollar un software libre que permita recrear llamadas de Voz sobre IP usando el protocolo de transmisión de información en tiempo real RTP para evaluar la calidad del servicio de VoIP en una red de datos.

1.3.1 Objetivos Específicos

- Analizar el funcionamiento y aplicabilidad de la calidad de servicio en el desarrollo tecnológico actual, mediante la recopilación de información.

- Formalizar los requerimientos apropiados del software a partir de las funcionalidades y el alcance del proyecto.

- Diseñar e implementar una interfaz web sencilla e intuitiva mediante el uso de estándares abiertos para gestionar las pruebas de calidad de servicio en VoIP.

- Desarrollar una herramienta de software para recrear llamadas VoIP a través de la manipulación de paquetes TCP/IP, de acuerdo a las especificaciones de del protocolo RTP.

- Establecer las métricas de calidad de servicio a evaluar en términos de retardo, perdida de paquetes y fluctuación en el tiempo (*Jitter*) para determinar la factibilidad de implantar VoIP en una red de datos.

- Ofrecer un servicio REST para la realización de las pruebas de QoS y recreación de llamadas VoIP a partir de la integración del *backend* y *frontend* en el sistema.

- Generar los resultados de las pruebas de evaluación de la calidad de servicio en VoIP mediante el uso de gráficos estadísticos para su posterior interpretación y análisis.

1.4 JUSTIFICACIÓN

Según datos de la ITU (*International Telecommunications Union*): en el año 2014 el número de personas con acceso internet a nivel mundial alcanzó la cifra de 3000 millones[2], las tecnologías de Voz sobre IP se fortalecerán con este aumento porque va ligado con la adaptación de internet en diferentes entornos, generando la apertura de nuevos mercados, incluyendo nuevos actores y modelos de negocios[3], lo cual conlleva a que se produzcan continuos cambios sociales y económicos[4].

En el contexto actual adaptarse a las tendencias tecnológicas hace que los procesos internos y la prestación de servicios se modifiquen, creando un nuevo paradigma organizacional que altera la competitividad de una organización[4]. Al final, son los usuarios quienes perciben los efectos de implantar tecnologías que van de la mano con el desarrollo local, nacional e internacional.

Evaluar el aspecto de calidad de servicio es una prioridad si se desea implantar la tecnología de voz sobre IP en una organización. Tener un software capaz de realizar pruebas rigurosas y confiables que evalúen métricas de calidad de servicio puede ayudar en la toma de decisiones para su implementación. También es relevante tener en cuenta que muchas empresas, especialmente las pequeñas y medianas, cuentan con un capital limitado por lo cual invertir en una herramienta de software costosa se sale de su

[2] ITU. The World in 2014: ICT Facts and Figures. En: Union Internacional de las Telecomunicaciones. [En línea]. (30, Abril, 2015). Disponible en: <http://www.itu.int/en/ITU-D/Statistics/Documents/facts/ICTFactsFigures2014-e.pdf>
[3] ITU. El protocolo de transmisión de voz por Internet (VoIP) llega a la mayoría de edad, En: Unión internacional de las telecomunicaciones, [En línea]. (30 Mayo 2015). Disponible en: <https://www.itu.int/net/itunews/issues/2009/07/21-es.aspx>
[4] OHMAE, Kenichi. “Adaptarse o morir: los tres niveles de cambio” HARVARD DEUSTO BUSINESS REVIEW. UCLASchool of Public and Social Research. Noviembre 2008.

capacidad económica. Los administradores de red y los departamentos de TI (Tecnologías de Información) de muchas organizaciones deben someterse a herramientas difíciles de usar y poco intuitivas, lo que hace que realizar pruebas de calidad de servicio sea una tarea desgastante y complicada.

Por lo anterior, se propone construir una herramienta de software libre con las nuevas tecnologías de la información y los estándares abiertos para reducir costos, que facilite las tareas de implementación de la tecnología VoIP. Este software tendrá la capacidad de realizar tareas unificadas para la evaluación de QoS, recreando sesiones y llamadas telefónicas mediante la creación de paquetes que tengan los valores propios de los protocolos RTP. Una ventaja adicional será la flexibilidad de un servicio web para que las pruebas puedan ser accedidas desde cualquier lugar usando un navegador, con soporte de tecnologías actuales como HTML5, convirtiéndose en una alternativa libre, abierta y multiplataforma.

1.5 IMPACTO ESPERADO

El presente proyecto busca dar solución a una necesidad de evaluación de calidad de servicio sobre VoIP utilizando software libre para beneficio de las organizaciones en general y, especialmente en pequeñas y medianas empresas. A continuación se listan los tipos de impacto del presente proyecto:

- **Administrativo**

Un impacto se percibirá en la productividad y la competitividad de las organizaciones, ya que las respuestas a exigencias comerciales u organizacionales serán ágiles y flexibles si se tiene un canal de comunicaciones debidamente analizado para comunicarse con tecnología VoIP.

- **Académico**

El desarrollo de un software libre donde se integran diversas tecnologías y herramientas permitirá dar inicio a investigaciones en el campo de los protocolos de comunicación usados por TCP/IP en tecnología VoIP. Los semilleros y grupos de investigación podrán intercambiar conocimiento con la oportunidad de profundizar en las tendencias tecnológicas que ayuden a propiciar un ambiente adecuado para la innovación, involucrando ramas de la Ingeniería de Sistemas e Informática, tales como las redes de datos y la ingeniería del software.

- **Tecnológico**

Un aspecto positivo destacable es la posibilidad de tener acceso a un conjunto de pruebas

unificadas en una sola herramienta, estableciendo una comunicación directa que permita evaluar el estado de la infraestructura tecnológica y la configuración de la red de datos. Importante también es poder ofrecer una herramienta que posibilite sacar mayor provecho a los recursos que brindan las tecnologías de la información, que vaya de la mano con las tendencias y traspase las barreras de la arquitectura computacional debido a que el software para evaluación de QoS es portable y soporta arquitecturas de hardware x86, x64 y ARM lo que representa una ventaja tecnológica.

- **Económico**

El uso de herramientas y estándares abiertos para el desarrollo de este proyecto de software libre, busca que las instituciones y organizaciones con bajos recursos puedan acceder a esta herramienta y realizar pruebas de evaluación de QoS en VoIP sobre su infraestructura, considerando su limitado presupuesto de operación comparado con el uso de herramientas de licencia comercial.

1.6 METODOLOGÍA

Este proyecto se sustenta en una metodología de desarrollo ágil (RAD) llamada TDD (*Test-Driven Development*), que efectúa pruebas unitarias o conjuntas en cada iteración y posteriormente posibilita refactorizar y ajustar el código, lo que permite mejorar la calidad del software y tener entregas en poco tiempo.

1.6.1 Metodología TDD

TDD (*Test-Driven Development*), es una metodología RAD que se orienta en pruebas continuas que depuran fallos o errores en cada una de las etapas y genera las condiciones para la creación de un código robusto en concordancia con nuevas tecnologías, lenguajes y *frameworks*[5].

En la Figura 1 se explica el ciclo de vida de TDD por medio del diagrama de flujo donde se generan más iteraciones a medida que el número de errores aumenta y viceversa. El primer paso consiste en agregar una prueba general y derivar si es necesario, un subconjunto de casos para validar todas las posibilidades. Si algún caso de prueba específico falla, entonces se ajusta o refactoriza el código para hacer la correcciones pertinentes.

Teniendo en cuenta los bajos recursos de personal, el tiempo limitado y la necesidad de llevar al mínimo los errores, esta metodología resulta apropiada para cumplir con los

[5] AMBYSOFT, Introduction to Test Driven Development. En: Agile Data. [En línea]. (30, Abril, 2015). Disponible en: <http://agiledata.org/essays/tdd.html>

objetivos planteados en las condiciones anteriormente mencionadas. Para dar cumplimento a cada uno de los objetivos planteados en el proyecto, se hizo necesario también enlistar una serie de actividades ordenadas como se puede verificar en la Tabla 1, donde cada una de estas va asociada a un resultado (para consultar en detalle las actividades y el tiempo requerido, se recomienda consultar el Anexo A con el diagrama de Gantt).

Figura 1. Ciclo de vida de TDD[5].

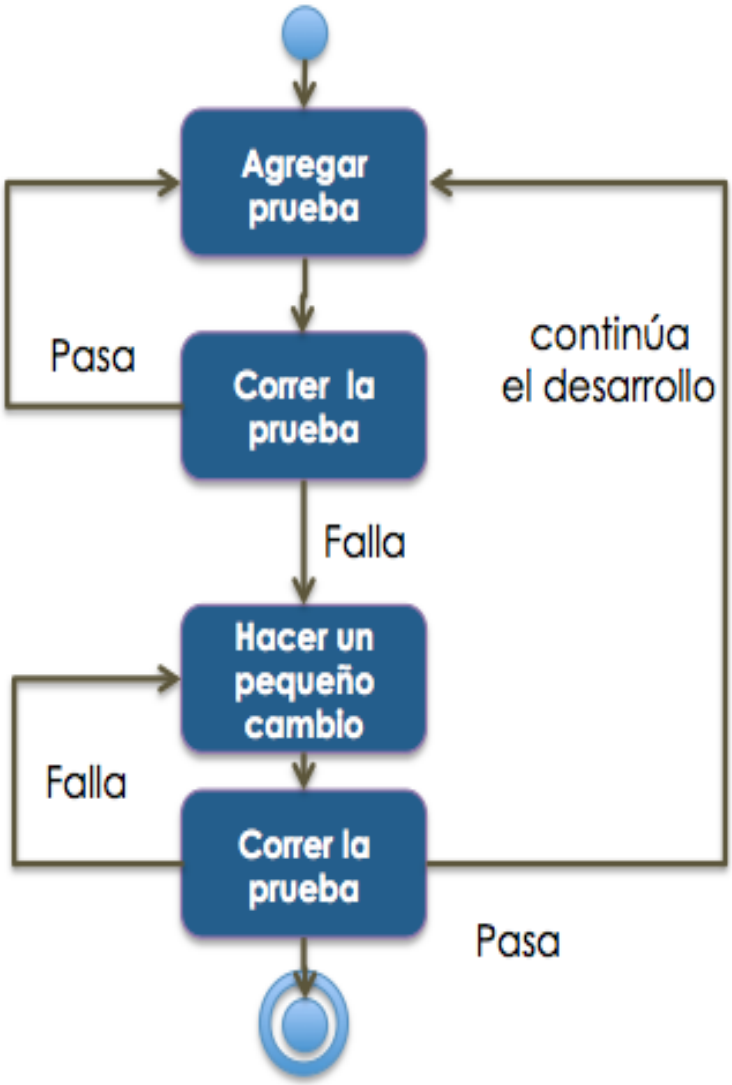


Tabla 1. Lista de actividades y entregables

N°	Actividad	Resultado
1	Definición de requerimientos y funcionalidades del software	Documento con diagrama de procesos y arquitectura cliente-servidor.
2	Diseñar un diagrama de arquitectura	
3	Diseño de la interfaz gráfica	*Frontend* implementado en HTML5 y CSS3.
4	Relacionar los elementos de la vista con las funcionalidades del software	
5	Codificar archivos de audio mp3 a G711	Archivos de audio con el códec G711a o G711u.
6	Manipular paquetes RTP con el módulo Gstreamer.	Archivos con paquetes de red construidos (extensión cap o pcap) para cada flujo de llamada.
7	Definir las métricas de QoS	Documento con definición de cada métrica aplicada en el QoS para VoIP.
8	Definir herramientas de software para aplicar las métricas	
9	Emular condiciones latencia, *jitter*, perdida de paquetes y ancho de banda	Script en Python que manipula el *kernel* de linux por medio de iproute2.
10	Integrar herramientas con *Backend*	Código fuente escrito en Python con las funciones asociadas a las herramientas de software libre.
11	Realizar las pruebas de funcionalidad del *Backend* con las herramientas	
12	Integración del *Backend* y *Frontend*	Implementación del Servicio Web con REST.
13	Desplegar resultados estadísticos de la evaluación de métricas QoS	Generación de gráficos estadísticos como tortas y polígonos de frecuencia en formato PNG.
14	Definir vista de los resultados	
15	Depuración de errores y validaciones	Software cliente-servidor con código fuente.
16	Prueba final de integración y conexión cliente-servidor	
17	Elaboración del manual de uso de la aplicación	Manual de uso de la aplicación.

2. MARCO REFERENCIAL

2.1 ANTECEDENTES

Las comunicaciones por voz han experimentado cambios importantes a través del tiempo, desde la telefonía análoga, la radio, telefonía digital, telefonía celular, telefonía satelital y en la época actual con la masificación del Internet, la transmisión de paquetes de voz sobre dicha red.

Usar una red de datos para realizar una llamada telefónica sobre IP, implica comprimir la voz y enviarla en paquetes por un canal donde hay diferente tipo de tráfico (texto, voz, video, audio, entre otros), donde se espera tener una calidad de servicio aceptable. Para ello se han desarrollado diversas soluciones de software que evalúan métricas de QoS, que analizan y detectan anomalías en el canal de comunicaciones. Entre las herramientas de software más destacadas para la evaluación de calidad de servicio sobre VoIP se pueden encontrar las siguientes:

2.1.1 Solarwinds: VoIP& Network Quality Manager

Es un software que monitoriza la calidad de servicio de llamadas VoIP con métricas como el *jitter*, latencia, pérdida de paquetes y MOS (*Mean Option Score*) la cual es una prueba de la percepción del servicio de VoIP desde la perspectiva del usuario. Tiene una herramienta que correlaciona los problemas de llamadas con el rendimiento de la red de datos[6]. Entre sus principales funciones están

- Monitoreo del desempeño de llamadas VoIP
- Detección y solución de problemas del desempeño de las llamadas VoIP
- Búsqueda y filtrado de los registros de detalles de las llamadas
- Paneles de control de monitoreo VoIP y WAN
- Alertas inteligentes de desempeño de VoIP y WAN
- Vista de cuadro de señal de llamada
- Configuración automática de IP SLA (*Internet Protocol Service Level Agreement*)
- Planificación de capacidad de VoIP y WAN

Posee también opciones de búsqueda y filtros con los registros detallados de llamadas. El precio de la versión básica es de 1.495 dólares, la versión completa 11.995 dólares (precios consultados en Junio 4 de 2014)[7] y la licencia es privativa para ambos casos.

---

[6] SOLARWINDS, "VoIP&NetwokQuality Manager (DataSheet)". Versión 4.1, 2014. [En línea]. (15 Marzo 2014). Disponible en: <http://www.solarwinds.com/documentation/IPSLA/docs/VNQMAdministratorGuide.pdf>

[7] SOLARWINDS. "VoIP Network Quality Manager | IP SLA | SolarWinds". [En línea]. (24 Marzo 2014) Disponible en: <http://www.solarwinds.com/voip-network-quality-manager.aspx>

2.1.2 Cisco: SAA and RTTMON

Son herramientas integradas en el IOS (*Internetwork Operating System*) de los dispositivos de Cisco, actualmente están disponibles en las versiones 12.0 y superiores. Entre sus características están: capacidad de hacer pruebas para detectar retardo, *jitter*, pérdida de paquetes y generar estadísticas sobre la red de datos[8].

SAA y RTTMON son utilidades que se pueden utilizar aplicando las métricas mediante la implementación de pequeños enrutadores Cisco IOS como agentes, para simular las estaciones en los extremos de la red donde se conectan los clientes. Los agentes de monitoreo despliegan alarmas y eventos que se desencadenan una vez que se han determinado los valores permitidos en un umbral y funcionan como sondas de fluctuación y retraso. El precio va ligado al costo de los equipos de red Cisco.

2.1.3 Paessler: PRTG Network Monitor

Es un software de monitoreo VoIP que supervisa del rendimiento de la red con un sensor PRTG (Paessler Router Traffic Grapher), analiza el tráfico de red y localiza fallos sensibles e incluso problemas leves en la conectividad. También brinda la posibilidad de actuar con rapidez y de forma proactiva. El sensor de calidad de servicio supervisa la conexión de red y detecta parámetros como el *jitter* de VoIP, pérdida de paquetes, la variación del retardo de paquetes, los paquetes duplicados y otras lecturas.

Este software de monitorización de VoIP funciona enviando paquetes UDP entre dos extremos, analizando y corrigiendo eventos que pueden afectar a la calidad del sonido de VoIP[9]. En la versión básica el costo de la licencia es de 440 dólares y aumenta hasta 47.250 dólares (precios consultados en Junio 4 de 2014) en la versión corporativa dependiendo de la cantidad de sensores y funciones que se quieran agregar.

2.1.4 Mausezahn

Es un generador de tráfico escrito en C el cual envía paquetes entre dos extremos de una red[10], a menudo se usa para pruebas de seguridad en IDS y *Firewalls* con el propósito de auditar y encontrar fallos en redes de datos.

En cuanto a pruebas de QoS permite el envío de tráfico RTP usando el códec G.711 para medir *jitter*, perdida de paquetes y retardos entre dos host. Su interfaz funciona bajo línea

---

[8] CISCO, Medición de retraso, fluctuación y pérdida de paquetes con SAA y RTTMON del IOS de Cisco. Mayo 2008. [En línea]. (9 Marzo 2014). Disponible en: <http://www.cisco.com/cisco/web/support/LA/7/77/77854_saa.pdf>

[9] LADNER, Ralf. "Funkschau Kommunikationstechnik für Profis", Revista de IT funkschau, Tomo 20. 2011. [En línea]. (4 Marzo 2014). Disponible en: <http://cdn.paessler.com/common/files/product-reviews/funkschau-2011-es.pdf>

[10] HERBERT, Haas. Mausezahn User's Guide. Versión 0.38.1. 2010 [En línea]. (15 Mayo 2014) Disponible en:<http://www.perihel.at/sec/mz/mzguide.html>

de comandos para plataformas Linux o Unix y tiene una licencia GPL (*General Pubic License*) v2 desde la versión 0.33.

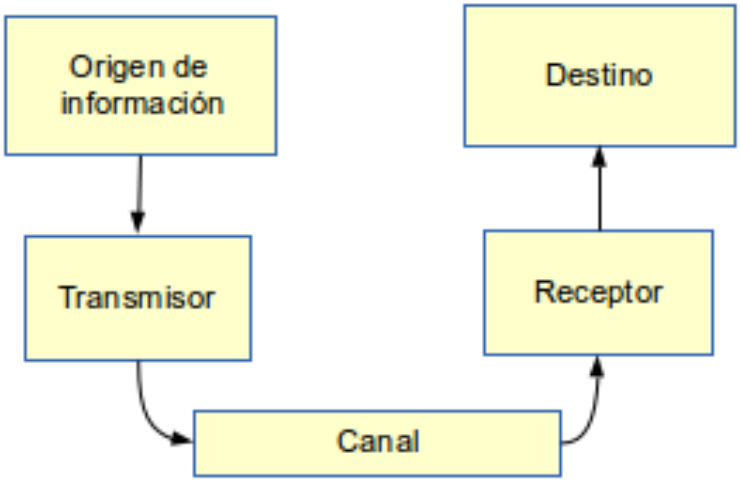
2.2 MARCO TEÓRICO

2.2.1 Telefonía fija convencional

Sus inicios se remontan desde la invención del teléfono en 1877 por el norteamericano Alexander Graham Bell[11], después de darse cuenta que para transmitir voz se podía usar la corriente continua, para ello fabricó el primer teléfono capaz de transmitir y recibir voz humana.

Los elementos de la comunicación usados por la telefonía son compatibles con los principios de la teoría de la información enunciada por Shannon[12], quien postuló que se requiere de cinco componentes para construir un sistema de comunicaciones (ver Figura 2).

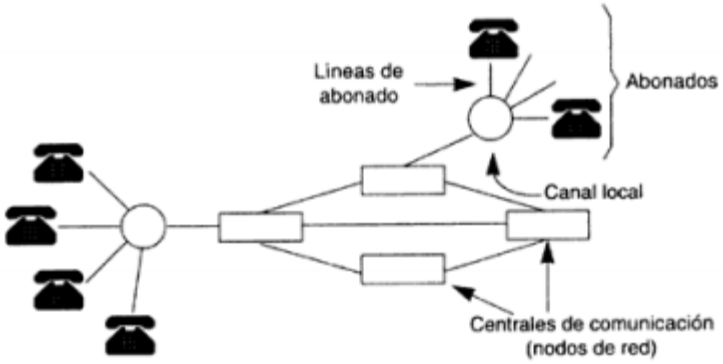
Figura 2. Elementos de un sistema de comunicación.



La telefonía fija se basa en la conmutación de circuitos[12], la cual crea un circuito físico entre los dos extremos para establecer la comunicación. La red de telefonía convencional está formada por los abonados, el cableado de los abonados que se comunican con las centrales locales, estas a su vez se interconectan con centrales de jerarquía superior por una red troncal de transporte como se muestra en la Figura 3.

[11] PEREA, Jose. "Teoría de la información". Red Tercer Milenio. 2012.
[12] ESCOBAR, Manuel. "Telefonía y conmutación". Primera Edición. Red Tercer Milenio. 2012.

Figura 3. Esquema de una red de telefonía fija convencional[12]



Cuando se establece una comunicación entre dos extremos, ocurre previamente una secuencia de eventos que se describen a continuación:

- La central telefónica de origen identifica la otra central a la cual está conectado el abonado destino con el que se requiere establecer la llamada.
- Se establece una ruta, encaminando la llamada entre la central del abonado origen hacia la central del abonado destino.
- Si la comprobación de la disponibilidad de línea está libre, la central indicará al abonado destino, por medio de una señal sonora o timbre que está recibiendo una llamada debido a que se emite una señal eléctrica por el cableado del abonado.
- Se realiza la comunicación por medio de un circuito entre los dos extremos y la línea queda ocupada.
- Se finaliza la comunicación y de nuevo las líneas quedan disponibles.

2.2.2 Voz sobre IP

VoIP (*Voice over Internet Protocol*), es una tecnología que permite la transmisión de voz en tiempo real utilizando una red de datos conmutada por paquetes. El servicio de VoIP requiere de un proceso ordenado de pasos, donde la voz se cuantifica, codifica y finalmente se comprime[12].

Una vez realizado el anterior proceso al que se llama digitalización, se procede a encaminar los paquetes de voz hacia su destino por medio de la red de datos, apoyándose en el protocolo IP[12]. VoIP está compuesto por un conjunto de protocolos que definen una serie de parámetros y reglas, donde cada uno tiene una función específica sea de control, señalización, calidad de servicio o transporte de datos[13]. A pesar de la diversidad de protocolos, estos se pueden clasificar en dos tipos:

[13] WALLINGFORD, Ted. "Switching to VoIP". O'Reilly & Associates. Enero 2005

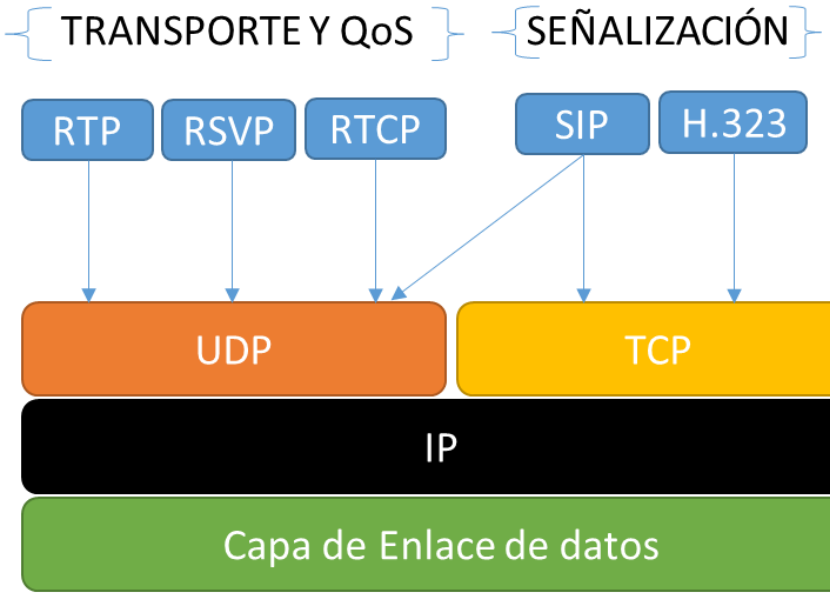
- **Señalización y control**

Los protocolos de señalización para VoIP más conocidos son H.323 definido por la ITU-T (Sector de Normalización de las Telecomunicaciones) y SIP *(Session Initiation Protocol)* definido por la IETF (*Internet Engineering Task Force*), los cuales permiten autenticar usuarios por medio de una identificación de sesión. Por otro lado permiten iniciar llamadas, transferirlas, mantenerlas y finalizarlas por medio de un servicio de control[13].

- **Transporte y calidad de servicio multimedia**

Existen una serie de protocolos diseñados para llevar datos multimedia como mensajes, voz y video en tiempo real. Debido a las características que varían según el tipo de tráfico, se hizo necesario garantizar la calidad de servicio por medio de protocolos que permitieran la clasificación y priorización de los paquetes[13], entre los más utilizados se encuentran RTP, RSVP, y RTCP.

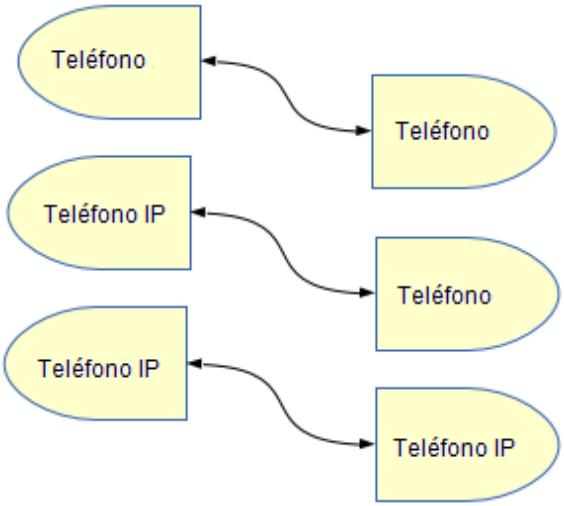
Figura 4. Diagrama de protocolos que conforman VoIP



En la Figura 4 se puede apreciar la pila de protocolos asociadas a las capas TCP/IP, y se evidencia como RTP (*Real Time Protocol*), RSVP (*Resource reSerVation Protocol*) y RTCP (*Real Time Control Protocol*) que manejan paquetes de multimedia se apoyan en protocolos no orientados a conexión como UDP. Por el contrario los protocolos de señalización como SIP y H.323 utilizan por defecto la capa de transporte TCP que es más confiable y orientada a conexión, aunque cabe anotar que SIP puede apoyarse también en UDP pero debe ser configurado especialmente para ello.

En VoIP, intervienen los mismos elementos de la comunicación representados en la Figura 1. VoIP utiliza redes conmutadas por paquetes como canal de comunicaciones y en cuanto a los transmisores y receptores, existen diversos elementos como los teléfonos fijos convencionales (análogos o digitales) y teléfonos IP (tanto por software y por hardware) que pueden intervenir en el proceso de una llamada telefónica[13] como se ilustra en la Figura 5:

Figura 5. Formas de comunicación con VoIP



a) ***Teléfono a Teléfono***: Ocurre cuando ambos dispositivos pueden comunicarse con centrales que poseen acceso a Internet o a una red de datos.

b) ***Teléfono IP a Teléfono***: La comunicación es posible desde un computador con acceso a Internet que se comunica con un teléfono fijo en una red de telefonía pública.

c) ***Teléfono IP a Teléfono IP****:* Esta comunicación es posible con el uso de *softphones* o teléfonos IP que se interconectan con la central telefónica que tiene acceso a Internet o a una red de datos.

2.2.3 Servicios Web

La W3C (*World Wide Web Consortium*) define los Servicios Web como sistemas de software diseñados para soportar una interacción máquina a máquina sobre una red de datos permitiendo a diferentes aplicativos con diferentes fuentes comunicarse unos con otros por medio de un estándar ajeno a la plataforma, sistema operativo o lenguaje de

programación[14]. En cuanto al modelo Cliente-Servidor, los servicios web comparten la lógica del negocio, los datos y los procesos, por medio de una interfaz que comunica estos programas a través de la red.

Internet ha crecido en popularidad y los servicios han comenzado a mejorar los procesos de negocios y la experiencia de los usuarios a pesar que estos procesos internos se realizan de forma transparente a ellos. La integración de diferentes tecnologías que trabajan conjuntamente con los estándares está emergiendo, para asegurar la seguridad y operatividad entre varios servicios independientemente de los fabricantes que los proveen. A continuación se listan una serie de tecnologías utilizadas en el despliegue de los servicios orientados a la web:

Llamadas a Procedimientos Remotos (RPC)*:* Este tipo de tecnología provee una interfaz de llamado a funciones distribuidas y fueron utilizadas en los inicios de los de los servicios web[14].

Arquitectura Orientada a Servicios (SOA): La unidad básica de comunicación es el mensaje, y no la función a diferencia de RPC. Esto es típicamente referenciado como servicios orientados a mensajes[15] .

Transferencia de Estado Representacional (REST): Su funcionamiento se basa sobre el protocolo HTTP usando las acciones GET, POST, PUT y DELETE los cuales actúan sobre los recursos con estados y no con métodos o mensajes como RPC y SOA respectivamente[15].

2.3 TECNOLOGÍAS UTILIZADAS

Para la realización de iQos se utilizaron aplicativos de software libre que permitieron ejecutar tareas específicas para evaluar el desempeño de una de red de datos, extraer información de las sesiones RTP, capturar y recibir paquetes. A continuación se describen las herramientas usadas en el desarrollo del proyecto:

2.3.1 GStreamer

Es un framework multimedia de software libre escrito en lenguaje C y con licencia GPL para sistemas Linux, Unix y Windows[16], permite crear aplicaciones de vídeo, sonido y codificación. GStreamer posee un núcleo que secciona el flujo de datos y

[14] BOOTH, David. Web Services Architecture. En: The World Wide Web Consortium. 2004. [En línea]. (30 Abril 2015). Disponible en: <http://www.w3.org/TR/ws-arch/>
[15] BERNERES, Lee. "Uniform Resource Identifiers (URI): Generic Syntax". En: Xerox Corporation. 1998. [En línea]. (30 Abril 2015). Disponible en: <http://www.ietf.org/rfc/rfc2396.txt>
[16] GSTREAMER. The the open source multimedia framework. [En línea] (19 Abril 2015). Disponible en: <http://gstreamer.freedesktop.org>

manejo/negociación de distintos tipos de medio, aplicar filtros, codificar y decodificar múltiples formatos[16]. Para efectos de este proyecto fue usado para transmitir audio en tiempo real creando paquetes RTP e insertando el payload con códec G.711.

2.3.2 Iperf

Es una herramienta de software libre con la cual se puede medir el desempeño de una red mediante la inyección de tráfico TCP y UDP de forma unidireccional o bidireccional[17]. Iperf puede ser programado por tiempo o por el tamaño de datos a transferir. Entre sus características principales se encuentran las siguientes:

- Medición de ancho de banda.
- Reporta el tamaño del MSS/MTU.
- Soporte para la ventana deslizante en TCP mediante *buffer* en sockets.
- Soporta hilos y conexiones simultaneas asíncronas.
- Medición de pérdida de paquetes, latencia y *jitter*.
- Soporte *multicast.*
- Utiliza una arquitectura cliente servidor que puede operar bajo plataformas Linux, Windows, OSX y Solaris.

2.3.3 Tcpdump

Es un analizador de paquetes multiplataforma que funciona por línea de comandos, basado en la librería libpcap, está escrito en C++ y liberado bajo licencia BSD (*Berkeley Software Distribution*)[18]. Tcpdump permite visualizar y capturar el tráfico emitido y recibido en el computador donde está instalado. iQos se apoya en tcpdump para registrar el tráfico bidireccional en el momento que se realizan las llamadas telefónicas por voz sobre IP.

2.3.4 Rtpbreak

Es un software libre liberado bajo licencia GPLv2 que puede detectar, reconstruir y analizar sesiones RTP[19]. Funciona mediante heurísticas aplicadas en el tráfico del protocolo UDP, permitiendo extraer información de las llamadas telefónicas de VoIP. iQos hace uso de rtpbreak para decodificar los *streams* de RTP y ordenar los paquetes para su posterior análisis con tshark.

2.3.5 Tshark

Es un analizador de paquetes, brinda una amplia información del flujo de datos en una

[17] IPERF. What is Iperf? . The TCP/UDP Bandwidth Measurement Tool. [En línea] (4 Marzo 2014). Disponible en: https://iperf.fr/. Julio 2010
[18] TCPDUMP. Tcpdump/Libpcap, public repository. [En línea]. (15 Enero 2015). Disponible en: <http://www.tcpdump.org/>
[19] RTPBREAK. Reconstruct and analyze any RTP session. [En línea]. (15 Enero 2015). Disponible en: <https://nucleo.fedorapeople.org/rpms/rtpbreak/rtpbreak.spec>

red, funciona mediante una interfaz de línea de comandos y permite la captura de paquetes mediante la activación del modo promiscuo en la interfaz de red, o leyendo éstos desde un archivo de capturas previamente almacenado[20]. Posee además filtros para protocolos y hosts específicos, es multiplataforma y esta liberado con licencia GPL v2.

2.3.6 Python

Es un lenguaje de programación de alto nivel, interpretado, multiparadigma, multiplataforma y multipropósito con licencia PSF (*Python Software Foundation*). Posee una semántica dinámica y es usado para en aplicaciones donde se requiere un desarrollo rápido[21]. Python hace énfasis en la legibilidad del código para reducir el costo del mantenimiento del software y además permite anexar módulos o paquetes para hacer reúso del código.

En el desarrollo del proyecto fue de vital importancia el uso de Python debido a que actuó como *backend* de la plataforma web, así mismo permitió la interconexión entre los nodos cliente y servidor, además facilitó la ejecución de comandos internos del sistema.

2.3.7 HTML5

Es la nueva versión del lenguaje de etiquetado HTML (*Hyper Text Markup Language*) para la *World Wide Web*, cuyo estado actual está bajo revisión del W3C. HTML5 tiene como objetivo permitir la creación, estructuración y presentación de contenidos como páginas web. Cabe destacar algunas mejoras que permiten brindar un mejor manejo de la información, como el soporte de contenidos multimedia, especialmente para audio, video e imágenes vectoriales[22]. Por otro lado, ésta versión de HTML posee compatibilidad con versiones anteriores y mejora el rendimiento en los computadores que actúan como clientes[21].

Existen unos principios desde el punto de vista del diseño que se mejoraron en esta versión que son: la compatibilidad, utilidad, interoperabilidad y acceso universal[23]. Otro paso importante de HTML5 es la separación de la presentación y el contenido para delegar detalles de la vista a CSS *(Cascading Style Sheets)* y así tener más accesibilidad, simplicidad y documentos de tamaño pequeño. Pero sin duda lo que hace poderoso a HTML5 es su rica colección de API *(Application Programming Interface),* entre las cuales están:

---

[20] TSHARK. tshark - The Wireshark Network Analyzer 1.12.2. [En línea]. (23 Junio 2015). Disponible en: < https://www.wireshark.org/docs/man-pages/tshark.html >

[21] PYTHON. The Python Tutorial. Python Software Foundation. [En línea]. (11 Enero). Disponible en: <https://docs.python.org/2/tutorial/index.html#tutorial-index>

[22] MATTHEW, David. "HTML5: Desiging Rich Internet Applications (Visualizingthe Web)". Focal Press; Edición: 2. Noviembre de 2012

[23] LUBBERS, Peter. "Pro HTML5 Programming: Powerful APIs for Richer Internet Application Development". Apress; Edición . 2011

- *Canvas (2D and 3D)*
- *Channel messaging*
- *Cross-document messaging*
- *Geolocation*
- *MathML*
- *Microdata*
- *Server-Sent events*
- *Scalable Vector Graphics (SVG)*
- *WebSocket API and protocol*
- *Web origin concept*
- *Web storage*
- *Web SQL database*
- *Web Workers*
- *XMLHttpRequestLevel 2*

Estas API hacen posible acceder a funcionalidades que con versiones anteriores de HTML no se podrían, como por ejemplo el acceso a cámara y micrófono, animaciones en lienzo, efectos 3D, geolocalización, almacenamiento local, acceso a base de datos, entre otros[22]. HTML5 hizo mejoras significativas en la estructuración de contenido por medio de etiquetas semánticas para describir el contenido de los elementos en secciones (*ver Tabla2*).

Tabla 2. Nuevos elementos de sección de HTML5

Sección	Descripción
header	Contenido de cabecera
footer	Contenido para pie de la página
section	Estructura en secciones
article	Contenido independiente de un artículo
aside	Es un contenido resumido
nav	Se usa como el menú de navegación

3. CALIDAD DE SERVICIO (QoS - *QUALITY OF SERVICE*)

3.1 ¿QUÉ ES QoS?

La calidad de servicio (QoS) se define como un conjunto de tecnologías y técnicas aplicadas a las redes de datos[24], cuyo objetivo es intentar garantizar el buen desempeño de la red para ofrecer servicios, asegurando que el tráfico y los requisitos de funcionamiento previamente definidos puedan ser satisfechos teniendo en cuenta ciertas características o métricas, tales como la disponibilidad del servicio, el ancho de banda, la latencia, el orden de llegada de los paquetes y la tasa de errores[23]. Para aplicar QoS se realizan tareas para priorizar o reservar los recursos que requiera el tráfico. Existen determinados servicios que son afectados por eventos adversos que se presentan en un canal de comunicaciones, la calidad de servicio ayuda mitigar estos eventos no deseados logrando mantener un desempeño adecuado de la red, porque somete al flujo de datos a una serie de reglas, asignando prioridades y recursos que ayudan a identificar los servicios diferenciándose unos de otros[25].

3.1.1 ¿Por qué es importante la calidad de servicio?

En las redes de datos se transportan diferentes tipos de tráfico (Ej. texto, voz, video, entre otros). Existe también un tipo de tráfico que requiere un especial cuidado entre los cuales se encuentran video y voz en tiempo real, debido a que son sensibles a ciertos fenómenos como los retardos, pérdidas de paquetes y fluctuaciones que se presentan en la red[24]. Para mitigar estos eventos, se hace necesario que en las redes de datos se pueda brindar calidad de servicio, para garantizar su óptimo desempeño mediante una combinación de buenas prácticas y técnicas, que posibiliten la administración adecuada de los recursos y que operen de forma transparente para los usuarios. Por lo tanto, si el desempeño de la red es el adecuado, se logra obtener una convergencia de servicios y se tendrá una percepción favorable de los mismos. Desplegar QoS en una red de datos ofrece ciertas ventajas[24] entre las cuales se encuentran:

Convergencia de servicios: Debido a que facilita la interoperabilidad entre diferentes protocolos, de esta manera se pueden ofrecer múltiples servicios con el rendimiento adecuado.

Control sobre los recursos*:* Se pueden administrar parámetros importantes como el ancho de banda y establecer prioridades para protocolos específicos.

---

[24] SZIGETI. Tim; HATTINGH. Christina; BARTON Robert; Briley Kenneth. End-to-End QoS Network Design: Quality of ServiceforRich-Media & Cloud Networks. 2nd Edición. 2013

[25] KUROSE. James; ROSS Keith. Redes de Computadores un enfoque descendente. Capítulo 7. Pearson Educación. 2010

3.1.2 Cambios hechos en TCP/IP para adoptar QoS

El diseño inicial con el que fue concebido Internet, tenía como propósito brindar un servicio del “Mejor Esfuerzo” (*Best Effort*)[23], es decir, tratar todos los paquetes de la misma forma sin importar el tipo de tráfico. No obstante, los servicios de video conferencias, llamadas telefónicas sobre IP, entre otros requieren de un funcionamiento en tiempo real, no van a tener un buen desempeño en redes congestionadas que funcionen en un tipo de arquitectura del “Mejor Esfuerzo” [23]. Con el tiempo fue necesario realizar modificaciones en el protocolo IP para que empezar identificar cada paquete y procesarlo de una manera especial de acuerdo al servicio, entre los cambios que se introdujeron en el protocolo TCP/IP se encuentran los siguientes:

Campo ToS (*Type of Service*)*:* Es un campo de 8 bits que se incluyó en la cabecera IPv4 (año 1981) cuya finalidad es identificar la prioridad de la llamada y la ruta que debía tomar el paquete.

Protocolo RSVP (*Resource Reservation Protocol*)*:* Se introdujo en el año 1994, su función principal es reservar recursos en la ruta que el paquete va recorrer para adaptarse al volumen del flujo de datos que demanda un servicio determinado.

Prioridades y etiquetas en IPv6*:* Introducida en el año 1995, se agregaron 4 bits en la cabecera de IPv6 para definir una prioridad del tráfico con 16 valores posibles de mayor a menor. También se adicionó un campo de 24 bits para identificar un flujo de datos por medio de etiquetas.

Campo DS (*Differentiated Service*): Desarrollado en 1998, es un campo que consta de 8 bits, su objetivo es almacenar la información de QoS en las cabeceras de IPv4 e IPv6, los primeros seis bits identifican la acción que se debe ejecutar sobre el paquete y los dos restantes son usados para control de congestión[26].

3.2 ARQUITECTURAS QoS

Cuando se desea dar un tratamiento especial a algún tipo de tráfico los enrutadores deben configurarse para tratar los paquetes. Existen categorías o arquitecturas de QoS como la del Mejor Esfuerzo, Servicios Diferenciados y Servicios Integrados, cada uno proporciona diferentes niveles de complejidad y eficacia en el manejo de tráfico[27].

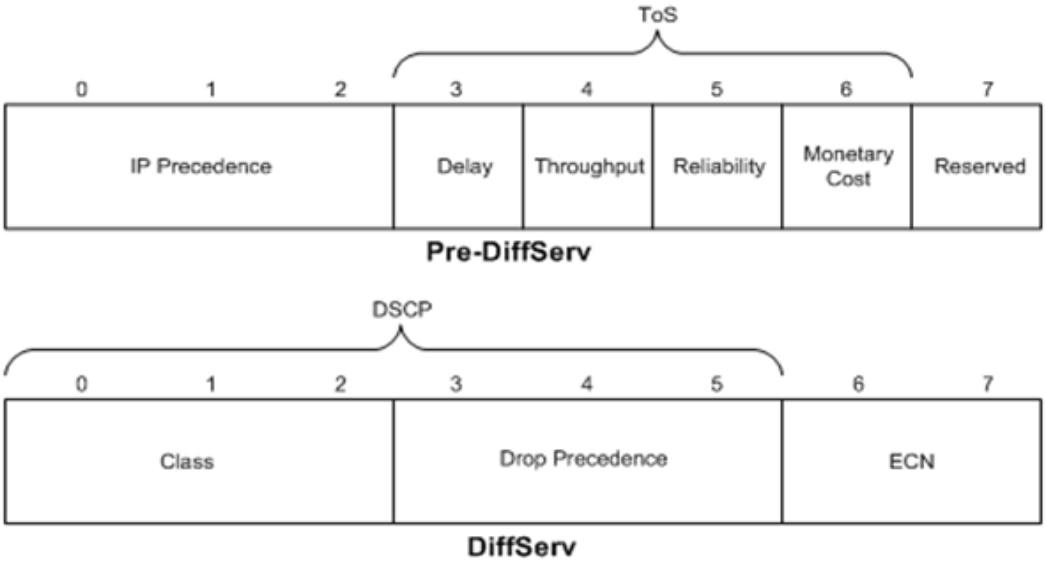
[26] CISCO SYSTEMS, EMC CORPORATION, "Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers", RFC 2474, Diciembre 1998.
[27] XEROX; BBN; IBM. "Specification of Guaranteed Quality of Service". RFC 2212. Septiembre 1997.

- **Mejor Esfuerzo o *Best Effort***: es el más común donde se intenta enviar el mayor número de paquetes en el menor tiempo, pero no se garantiza la entrega del paquete al destino final. En general no se controlan parámetros como el retraso y la variación. El comportamiento de este servicio tiene un valor por omisión de 0 en la cabecera DSCP (*Differentiated Services Code Point*)[26], lo cual quiere decir que el paquete no posee un tratamiento especial o de marcado.

- **Servicios Integrados o Intserv:** Es un servicio garantizado descrito en el RFC 2212 donde se negocian parámetros de red entre los extremos apoyándose en RSVP. Los parámetros que se manejan son: retardo, caudal de datos, confiabilidad, costo y reservación. Todos los dispositivos de red a lo largo del trayecto necesitan RSVP y se debe indicar la reservación QoS requerida. Las reservas son temporales y se hacen asignando un ancho de banda para ciertos servicios. Las reservas deben ser actualizadas en intervalos de tiempo periódicos por lo cual se agrega tráfico adicional a la red y se aplican sobre el campo TOS (*Type of Service*) de la cabecera IP [28].

- **Servicios Diferenciados o Diffserv:** Funciona mediante la clasificación de los paquetes, usa mecanismos de encolamiento y definición de clases de tráfico a nivel de la capa de red para dar prioridad a los paquetes. La clasificación y marcado ocurre en las fronteras de la red sobre el campo DS (*Differentiated Service*) de la cabecera IP. Para *Intserv* la negociación de los parámetros ocurre al marcar y leer el campo TOS de la cabecera IP de cada paquete. Por otro lado *Diffserv* usa el campo llamado "DS" para gestionar la calidad de servicio como se aprecia en la Figura 6.

Figura 6. Campos TOS y DS[29].



[28] EVANS, John; FILSFILS, Clarence. "Deploying IP and MPLS QoS for Multiservice Networks", ISBN 978-0-12-370549-5. 2007.
[29] CISCONINJA. Marking DSCP values with Policy-Based Routing. En: Cisconinja. [En línea]. (29, Abril, 2015). Disponible en: <https://cisconinja.wordpress.com/2008/11/26/marking-dscp-values-with-policy-based-routing>

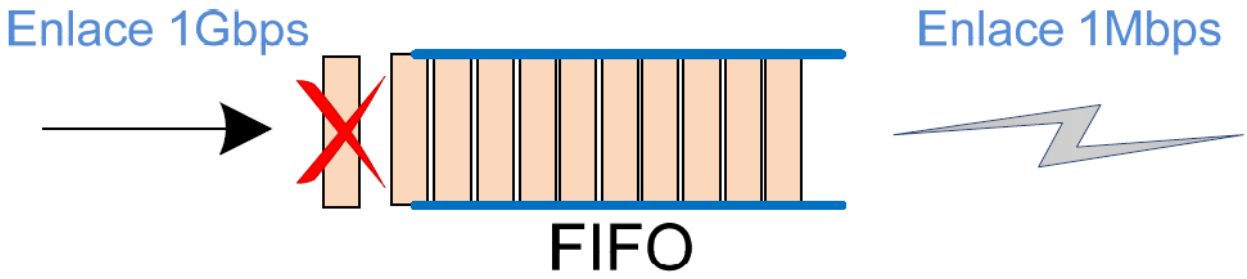
3.2.1 Clasificación de Paquetes y encolamiento

El objetivo de la clasificación consiste en identificar flujos de paquetes y asociarlos en clases para luego aplicarles técnicas de QoS. Los flujos se pueden identificar por los campos de la cabecera IP y también por los campos en los protocolos de transporte como UDP y TCP[30]. Antes de clasificar el tráfico se realiza un marcado de los paquetes alterando los campos TOS o DS según la arquitectura que se vaya a manejar, este marcado puede realizarse en el origen cuando los paquetes se envían, o en el destino a medida que los paquetes llegan.

El objetivo de marcar paquetes es que estos se puedan identificar, clasificar y procesar por los enrutadores. La forma como se seleccionan los paquetes va de acuerdo a la disciplina de planificación para la cual se manejan métodos de encolamiento para su posterior transmisión[29]; entre las disciplinas de encolamiento más importantes se encuentran las siguientes:

FIFO (*First Input First Output*): A medida que llegan los paquetes estos se van almacenando en un buffer para luego ser enviados, este tipo de encolamiento va transmitiendo cada paquete en el mismo orden que llegan[29], cabe resaltar que si el buffer se llena los paquetes entrantes son rechazados como se observa en la Figura 7:

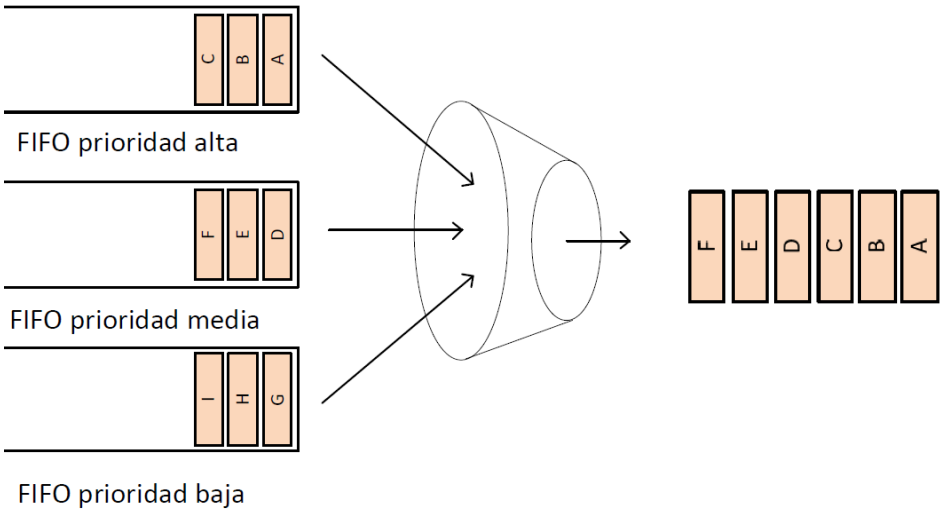
Figura 7. Esquema de funcionamiento de una cola FIFO[31].



Colas de Prioridad: Son un conjunto de clases ordenadas en alta y baja prioridad, donde las primeras son atendidas siempre que existan paquetes de esta clase en el buffer, de lo contrario se transmitirán las colas de baja de prioridad[29] como se ilustra en la *Figura 8.*

[30] HUBERT, Bert. Linux Advanced Routing & Traffic Control. [En línea]. (10, Julio, 2015). Disponible en: <http://www.lartc.org>
[31] DTE. Gestión del tráfico y calidad de servicio. En: Universidad de Sevilla. [En línea]. (29, Abril, 2015). Disponible en: <http://www.dte.us.es/docencia/etsii/gii-ti/tecnologias-avanzadas-de-la-informacion/Tema3-QoS.pdf>

Figura 8. Esquema de funcionamiento de las Colas de Prioridad[30].



Colas de turno rotatorio: Usa un método de selección equitativa basado en *Round Robin*, las colas se recorren según el método que puede ser CFQ *(Completely Fair Queuing)* la cual es una disciplina conservadora donde a cada cola se recorre circularmente enviando un solo paquete de forma completamente equitativa como se ilustra en la *Figura 9*. Por otro lado existe una disciplina WFQ (*Weighted Fair Queuing*) donde se asigna un peso *Wn* por cola y se recorre circularmente enviando *n* paquetes según el peso de la cola[29] *(ver Figura 10).*

Figura 9. Esquema de funcionamiento de CFQ[30].

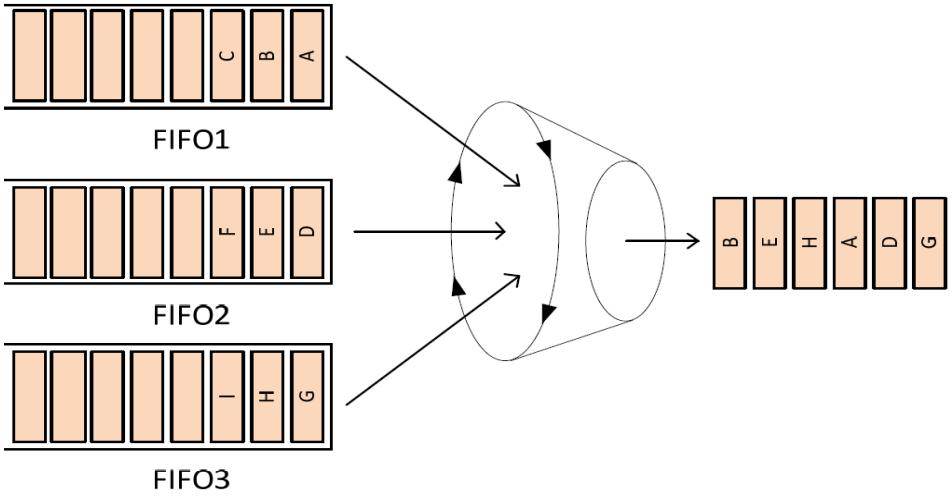
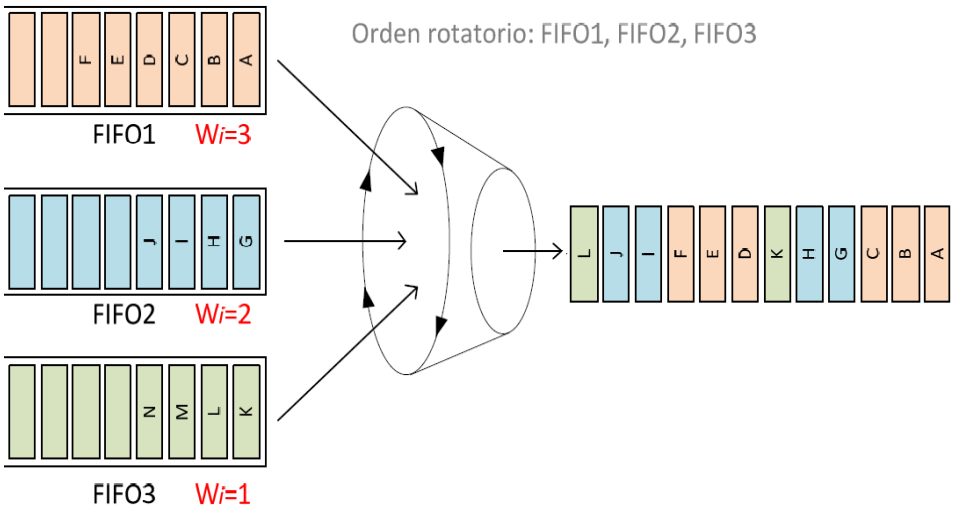


Figura 10. Esquema de funcionamiento de WFQ[30].



3.3 PROTOCOLOS USADOS EN VOIP

3.3.1 SIP (*Session Initiation Protocol)*

El protocolo de inicio de sesiones (SIP) es un protocolo de señalización a nivel de capa de aplicación desarrollado por la IETF (*Internet Engineering Task Force*) [28], emplea la arquitectura Cliente-Servidor y es utilizado para registrar, modificar y finalizar sesiones de llamadas telefónicas que utilizan tecnologías como VoIP o contenidos multimedia[32]. El protocolo SIP posee cinco propiedades para crear y terminar comunicaciones multimedia (voz, video y chat)[31]:

- Localización del usuario: Determina los puntos extremos que van a participar en la comunicación.

- Disponibilidad del usuario: Verifica la voluntad de participación de los actores en la comunicación.

- Capacidades del usuario: Se selecciona el medio de transmisión y los parámetros que se usarán en la comunicación.

- Configuración de sesión: Se establecen parámetros propios de la sesión tanto del usuario que llama como del que recibe la llamada.

- Administración de sesión: Involucra la creación y finalización de las sesiones, también algunos parámetros para invocar otros servicios.

[32] COLUMBIA U; ERICSSON; AT&T. SIP: Session Initiation Protocol. RFC 3261. Junio 2002.

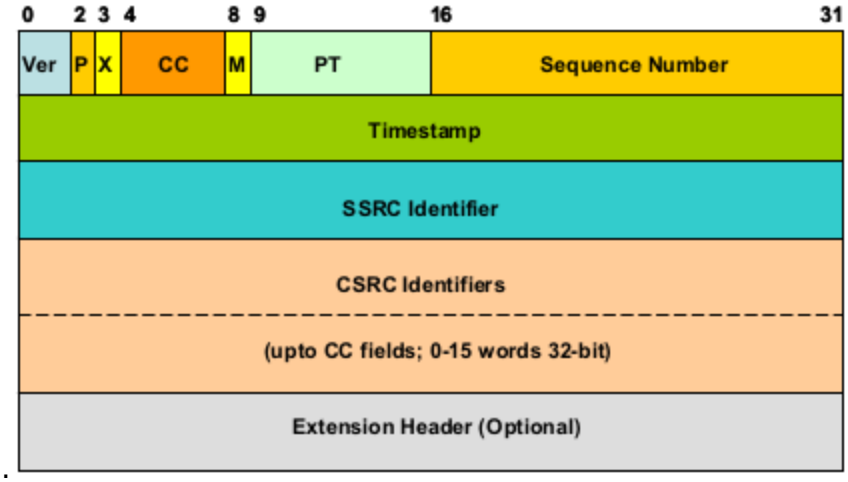
3.3.2 RTP (*Real-time Transport Protocol*)

Es un protocolo para la transmisión de datos en tiempo real y adoptado como estándar en internet[33], desarrollado por la IETF. La primera versión de RTP fue aprobada en 1995 y quedo plasmada en el RFC 1889. La versión 2 está definida en el RFC 3550, siendo ampliamente utilizada hoy en día por cientos de aplicaciones multimedia para transmisión y recepción de contenido de audio y video. RTP trabaja a nivel de capa de sesión junto con el protocolo UDP. Importante también es que el protocolo tiene soporte para difusión *unicast* y *multicast*, permite la identificación de los paquetes a través de sus *payloads*, marcas de tiempo y posteriormente los clasifica accediendo a las cabeceras[34].

Una característica de RTP es la creación de perfiles y la extensión de las cabeceras en los paquetes, esto ayuda a las aplicaciones a dar un tratamiento personalizado de las funciones que se desean implementar. Al mismo tiempo ofrece flexibilidad al protocolo para que siga evolucionando y evite que éste se quede obsoleto.

En la *Figura 11* se observa la estructura de la cabecera del paquete RTP, los campos son de gran importancia porque con ellos se pueden detectar anomalías como falsificaciones, spam y con ello obtener otros datos estadísticos[33], además facilitan las mediciones de métricas de QoS.

Figura 11. Cabecera de un paquete RTP[35].



A continuación se describen los campos de la cabecera RTP:

[33] FRANCE TELECOM R&D; ANTIPOLIS, Sophia. "Detection and Comparison of RTP and Skype Traffic and Performance". Proceedings of the IEEE Global Telecommunications Conference (GLOBECOM '06). 2006.
[34] NEC UNIFIED SOLUCTIONS, INC. "What VoIP Requires From a Data Network". 2006
[35] PANAYOTIS, Fouliras. On RTP filtering for network traffic reduction. MoMM '08 Proceedings of the 6th International Conference on Advances in Mobile Computing and Multimedia. 2008

- Ver: Se utiliza para identificar la versión de RTP (puede ser 1 ó 2).

- P: Se refiere al *padding*, cuando se activa indica que existen uno o varios octetos de relleno después del *payload*.

- X: Permite la expansión de la cabecera RTP cuando se activa, es usada a menudo para insertar perfiles.

- CC: Contador de CSRC *(Content Source)*

- M: Es el marcador para identificar eventos especiales en la cabecera.

- PT: Es el tipo de *Payload*, indica alguna codificación especial, por ejemplo si es audio o video.

- *SequenceNumber* (Numero de Secuencia)*:* Se incrementa a medida que se envía un paquete RTP, lo que permite identificar perdida de paquetes.

- *Timestamp:* Permite la sincronización las sesiones RTP agregando marcas de tiempo.

- *SSRC Identifier*: Es un identificador del origen del flujo RTP por medio un campo de 32 bits.

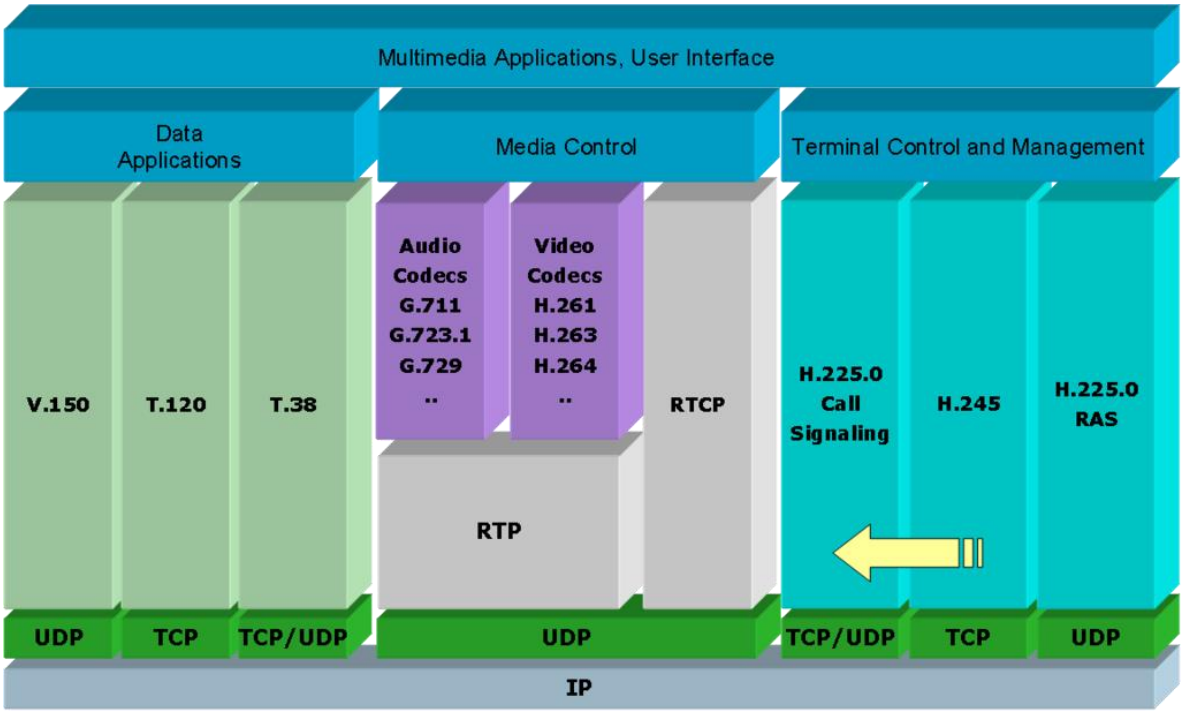
- *CSRC Identifiers*: Es una lista formada con los identificadores SSRC para determinar el mezclador, con ello se puede conocer el número de participantes en un flujo de datos, como por ejemplo cuando se realiza una conferencia.

3.3.5 Protocolo H.323

Es un estándar definido por la ITU-T donde se describe un conjunto de especificaciones para el transporte de servicios multimedia basados en redes de datos conmutadas por paquetes[36]. H.323 está compuesto de una pila de protocolos (ver Figura 12) interconectados entre sí donde se especifican los modelos de las llamadas y procedimientos en la señalización. Entre los protocolos internos de H.323 se encuentran los siguientes:

[36] ITU-T. H.323 : Sistemas de comunicación multimedia basados en paquetes. [En línea].(30, Agosto, 2014). Disponible en: <https://www.itu.int/rec/T-REC-H.323/recommendation.asp?lang= es&parent=T-REC-H.323-200912-I>

Figura 12. Protocolos que conforman H.323[37]



- **H.225.0***:* Opera con un subconjunto de protocolos de señalización que son: RAS (Registro, Admisión, Estado) y Q.931 para señalizar las llamadas cuando se inician, se mantienen y finalizan.

- **H.245***:* Tiene la función de control multimedia en las sesiones de comunicación de H.323. Entre sus características principales esta la definición de la codificación, control de flujo y del *jitter*.

- **RTP***:* Es usado para transmitir los flujos de audio (usando códec G.7xx) y vídeo (usando códec H.26x) en tiempo real.

- **RTCP** *(Real-Time Control Protocol):* Es usado para control del flujo de multimedia, informa la calidad de servicio accediendo a las cabeceras de RTP.

- **T.120**: Protocolo que permite conexiones multipunto usado para conferencias.

- **V.150**: Usado en comunicaciones por medio de *módems.*

- **T.38:** Usado en comunicaciones por fax.

[37] WIKIPEDIA. A complete, sophisticated protocol stack En: Wikipedia. [En línea]. (03, Mayo, 2015). Disponible en: <http://en.wikipedia.org/wiki/H.323>

3.4 MÉTRICAS QoS USADAS EN VoIP

VoIP requiere que la red tenga la capacidad de entregar la mayor cantidad de paquetes en el menor tiempo posible y en el orden correcto. El protocolo IP en el cual se apoya VoIP, no garantiza que todos los paquetes lleguen a su destino[38], y adicional a esto, la configuración de los enrutadores que se encuentran a lo largo de la red puede manejar diferentes tipos tratamientos para los paquetes, lo que puede causar alteraciones cuando se realizan llamadas telefónicas en redes conmutadas por paquetes[37].

Aplicar QoS tiene efectos sobre el rendimiento de la red y requiere de implementaciones desde el punto de vista técnico para solventar aquellas problemáticas a las que está expuesta VoIP, ya que éstas se manifiestan como degradaciones en las señales de audio influyendo sobre la percepción de los usuarios. Para medir parámetros que afectan la calidad de servicio en VoIP se usan una serie de métricas (sugeridas en la ITU Y.1540 y Y.1541) entre las se tienen principalmente las siguientes:

- ***Jitter***:

Se define como las variaciones en el tiempo (demoras) cuando se transmiten los paquetes. Generalmente esas demoras son causadas por un encolamiento al existir altos niveles de tráfico en la red[39]. El *Jitter* se puede medir de muchas formas pero existen unas reglas recomendadas en los RFC3550, RFC3611.

- **Perdida de paquetes***:*

Se presenta cuando los niveles de tráfico son muy altos y los paquetes tienden a perderse al ser rechazados por los dispositivos de red, especialmente cuando los buffers se llenan. En VoIP la perdida de paquetes es un problema que afecta la calidad del servicio si este es mayor al 1%.

- **Latencia***:*

Es la demora en la entrega de los paquetes. En el documento G.114 la UIT recomienda que para la transmisión de voz por internet se debe tener un umbral menor o igual al 150 milisegundos[38].

[38] MARSIC, Ivan. Computer Networks: Performance and Quality of Service.Department of Electrical and Computer Engineering, Rutgers University. Diciembre 2010.
[39] COLUMBIA U.;PACKET DESIGN; BLUE COAST SYSTEMS INC. RTP: A Transport Protocol for Real-Time Applications. Julio 2003

3.5 CODECS

Teniendo en cuenta que la voz humana está compuesta de señales de audio analógicas (continuas en el tiempo)[40], se hace necesario convertirlas en información digital, ya que en VoIP es utilizada la conmutación de paquetes para transmitir la voz. Los códecs (Codificador - Decodificador) son algoritmos que convierten señales de audio analógicas a formato de audio digital y viceversa, funcionan a nivel de capa de presentación en dónde se definen las reglas de cómo se deben procesar e interpretar los datos transmitidos. Existen diferentes tipos de *codec* para VoIP, entre los más conocidos se encuentran los definidos por la ITU-T como el G.711, G729, G.722, G726 y GSM los cuales varían según en nivel de compresión y frecuencia de muestreo. La utilización de un *codec* determinado va a tener repercusiones que afectan directamente el ancho de banda (bps) y la calidad de la voz[39]. En general se puede asumir que a mayor compresión se va a obtener mayor distorsión y por consiguiente un deterioro en la calidad del audio, debido a que se pierden datos con frecuencias de muestreo bajas. Para comparar un *codec* con respecto a otro se debe considerar la capacidad de ofrecer mejor calidad de voz usando la misma cantidad de ancho de banda.

Tabla 3. Lista de códec usados en VoIP

Codec	Flujo de datos (Kbits/s)	Muestreo (KHz)	Latencia (ms)
G.711	64	8	15 - 20
G.722	64	16	40
G.723.1	5.6/6.3	8	37,5
G.726	16/24/32/40	8	20
G.729	8	8	15
GSM 06.10	13	8	20

Para calcular el ancho de banda que va consumir un *codec* se debe conocer información como el tiempo periódico con el cual se envía cada paquete y el flujo de datos; como se ilustra en la *Tabla 3*, se pueden apreciar los datos relevantes asociados a cada *codec*, por ejemplo para el *codec* G.711 cada paquete se envía en un lapso de 20 ms uno tras otro, por lo tanto en cada segundo de audio se deben enviar 50 paquetes (debido a que un segundo equivale a 1000 ms, y este valor dividido en 20 ms es igual a 50), cada uno con una velocidad de transmisión de 64 Kbps[41]; adicionalmente se debe considerar sumar la longitud de las cabeceras RTP, UDP, IP

[40] ITU-T. G Series: Transmission systems and media, digital systems and networks. [En línea]. (10, Mayo, 2015). Disponible en: <http://www.itu.int/net/itu-t/sigdb/speaudio/Gseries.htm>
[41] ITU-T. G.711: “Pulse Code Modulation (PCM) of voice frequencies”. CCITT. 1988.

y la capa física, por lo cual se debe considerar el siguiente cálculo para una red Ethernet:

```
Codec G.711 = 64 Kbits/s * 20 ms  = 160 Bytes
Cabecera RTP                      = 12 bytes
Cabecera UDP                      = 8 bytes
Cabecera IP                       = 20 bytes
Cabecera Ethernet                 = 28 bytes
Total                             = 238 bytes
```

Los 50 paquetes mencionados con anterioridad poseen un tamaño total de 238 bytes (1904 bits) que se envían cada segundo, lo que equivale a 95.2 Kbps en una comunicación unidireccional y se puede calcular de la siguiente forma:

$$50 \times 1904 \text{ bits/seg} = 95200 \text{ bps} = 95.2 \text{ Kbps}$$

3.6 REQUERIMIENTOS DE QoS PARA VoIP

Implementar Voz sobre IP requiere de una prioridad explícita para este servicio, donde se garantice el ancho de banda suficiente tanto para los protocolos de señalización y de transporte de datos. Por ejemplo, si se usa una arquitectura DiffServ, todo tráfico de voz debe ser marcado con un DSCP "EF" (*Expedited Forwarding*) y la pérdida de paquetes no debe superar un umbral del 1%[1]. En cuanto a la latencia unidireccional el valor mínimo deberá ser de 150 ms (300 ms bidireccional) y el Jitter unidireccional no deberá superar los 30 ms según la ITU G.114. Del mismo modo para la velocidad de transmisión requerida por llamada, se deben considerar el códec utilizado, tasa de muestreo y por supuesto el tamaño de las cabeceras de los protocolos a nivel de capa 2 (ver Tabla 4).

Tabla 4. Tamaño de las cabeceras para protocolos de capa 2.

Protocolo	Tamaño de la cabecera (Bytes)
802.1Q Ethernet	32
PPP	12
MLP	13
Frame Relay	4
ATM	53

Según la base de datos de recomendación de la ITU Y.1540 y Y.1541 los tres factores que afectan directamente la calidad de voz son la pérdida de paquetes, latencia y el Jitter.

Aunque la latencia no hace que sea ininteligible la conversación, si afecta el carácter de la misma, porque altera la sincronización entre el emisor y receptor al tener una

percepción diferente de los tiempos de respuesta entre los dos puntos extremos. La pérdida de paquetes causa vacíos y omisiones que dificultan el entendimiento por parte del receptor, entre más perdidas consecutivas de paquetes, hay una mayor degradación de la voz. El Jitter causa que los paquetes lleguen en desorden y por consiguiente la voz se distorsiona, a menudo se usan buffers para reordenar los paquetes.

4. DEFINICIÓN E IMPLEMENTACIÓN DE LAS MÉTRICAS UTILIZADAS EN iQos

4.1. JUSTIFICACIÓN

Las comunicaciones en sistemas digitales se ven afectadas por degradaciones al momento de tratar las señales vocales, para el caso de voz sobre IP no es la excepción, ya que es susceptible a sufrir distorsiones en las señales de audio debido a fenómenos que afectan a las redes basadas en conmutación de paquetes, y que producen las degradaciones del canal de comunicación.

VoIP está implementado a nivel de capa de transporte con el protocolo UDP, no orientado a conexión, con lo cual, si un paquete se pierde o la entrega del mismo tiene una demora considerable, no hay ningún mecanismo para reenviar o regular la velocidad de transmisión de los datos. A medida que hay más llamadas simultáneas se va reduciendo la calidad de la voz, y se ve reflejado en el rendimiento de red, esto afecta directamente la experiencia del usuario final. Debido a que el software genera paquetes RTP encapsulados dentro del protocolo UDP, se optó por tomar métricas como la demora (*delay*), variación de demora (*jitter*) y la pérdida de paquetes definidas en el RFC 2475 y en las recomendaciones ITU Y.1540 y Y.1541, entidad normalizadora que estudia los aspectos técnicos y publica recomendaciones sobre los mismos, con miras a la normalización de las telecomunicaciones en el plano mundial.

Las métricas mencionadas anteriormente, son indicadores de desempeño que pueden ser representados como valores estadísticos durante intervalos de tiempo. En éste proyecto se propone medir estos indicadores en rangos de tiempo que varían según la duración de las llamadas, los resultados serán entregados al usuario para su posterior interpretación; es importante señalar que el software no interpreta, ni decide, ni califica emitiendo juicios cualitativos al definir como buena o mala la calidad del servicio.

4.1 ¿QUÉ ES UNA MÉTRICA?

El documento RFC 2230 tiene un conjunto de conceptos y definiciones de lo que es una métrica, la cual es descrita como una unidad de medida de carácter cuantitativo, deben ser concretas y bien delimitadas y se obtienen a través de un método acordado o específico. Cada métrica de QoS escogida para este proyecto es medida por diferentes métodos.

4.2 LATENCIA, JITTER Y PÉRDIDA DE PAQUETES

iQos se apoya en Tshark para tomar las métricas de las llamadas VoIP, en el caso de la latencia, se calcula un valor “delta” restando la marca de tiempo (*timestamp*) del paquete RTP que llega al destino, con la marca de tiempo del paquete anterior en el mismo destino

como se muestra en la Tabla 5, donde se aprecia la llegada de cinco paquetes, donde zi es la marca de tiempo actual, z0 la marca de tiempo anterior y, la variable yi, el valor de la resta definido en segundos.

Tabla 5. Ejemplo de cómo se registran las marcas de tiempo para obtener la latencia

i	Zi	z0	yi
1	1423459861.286	-	-
2	1423459862.047	1423459861.286	0.7609 seg
3	1423459862.806	1423459862.047	0.7590 seg
4	1423459863.662	1423459862.806	0.8559 seg
5	1423459864.462	1423459864.462	0.7999 seg

El cálculo se puede expresar matemáticamente de la siguiente forma:

$$y\,i = z\,i - \mathrm{z0}$$

yi: Es el valor delta de la latencia entre 2 paquetes
zi: Valor tiempo del paquete actual
z0: Valor del tiempo del paquete anterior

El *jitter* unidireccional es variación de la latencia en un sentido, Tshark hace los cálculos del *jitter* basado en la especificación del RFC 3550 *A Transport Protocol for Real-Time Applications*, en dicho documento se compara el campo *timestamp* de los paquetes RTP. El *jitter* se puede medir con la siguiente formula:

$$X\,i = \mathrm{X0} + \frac{1}{16}\,(|y\,i| - \mathrm{X0})$$

Xi: Jitter actual
X0: Jitter anterior
yi: El valor delta de la latencia actual

El porcentaje de pérdida de paquetes se calcula con respecto al campo de número de secuencia del paquete RTP, Tshark entonces cuenta el total de paquetes enviados, ordena los paquetes recibidos y si un valor en el número de secuencia falta se cuenta como un paquete perdido.

$$P = \left(1 - \frac{T - L}{T}\right) \times 100$$

En la formula se aprecia cómo se calcula el porcentaje total de paquetes perdidos:

P: Porcentaje paquetes perdidos

T: Total de paquetes enviados
L: Paquetes perdidos

4.3 EMULACIÓN DE MÉTRICAS

La emulación de métricas sirve por ejemplo, en situaciones donde se desea evaluar el comportamiento de diversos protocolos de la pila TCP/IP en redes de datos, verificar el tratamiento que se da a los paquetes con condiciones particulares, y con ello poder predecir posibles fallos o anomalías que puedan ocurrir. iQos usa un *framework* de Python llamado Shapy para emular parámetros en las interfaces de red, este se apoya en el módulo NetEM del *kernel* de Linux. NetEm proporciona control de tráfico por disciplina de colas, posee características de emulación de redes. Fue concebido originalmente con el propósito de probar la variación del retardo (*Jitter*), pérdida, duplicación y reordenación de paquetes. Aunque NetEM no refleja cómo las redes reales se comportan si permite tener una simplicidad en las mediciones repetibles que pueden ser estimadas. Posee también técnicas avanzadas para la emulación de la varianza, correlación y distribución gaussiana. Una característica distintiva de NetEm es su perfecta integración en el sistema operativo, permitiendo la emulación de redes, incluso en el ámbito de sistema local entre procesos de usuario reduciendo así la necesidad de una virtualización.

Shapy ofrece una instalación sencilla para administrar la configuración de la prueba de emulación sobre las interfaces de red seleccionadas, permite que el usuario cree un archivo que es accesible como si se tratara un módulo de Python. En la figura 13 se describen los elementos relevantes del archivo de configuración para Shapy:

Figura 13. Configuración Shapy

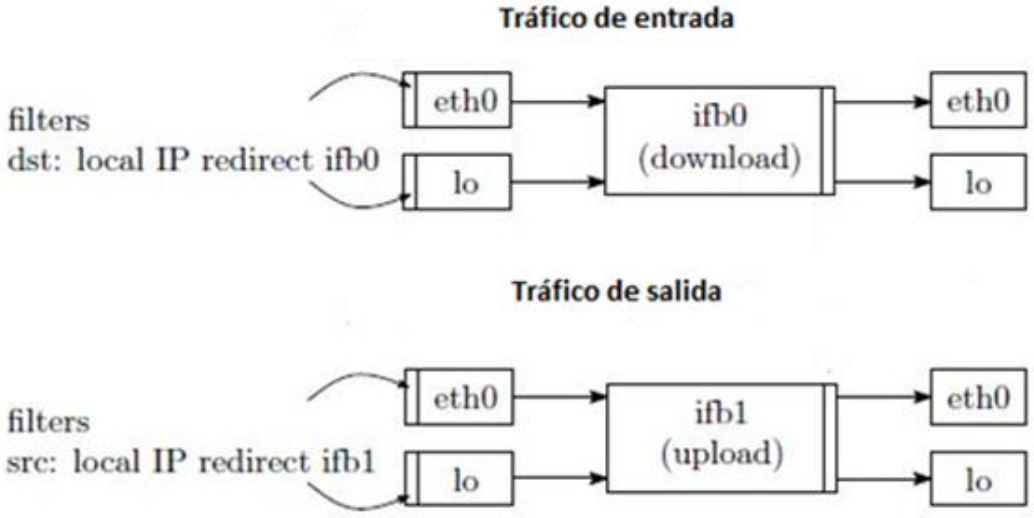
Donde UNITS son las unidades para limitar ancho de banda puede ser Kbits o Mbits, ENV es la variable de entorno con la ruta donde están los comandos de Linux, finalmente EMU_INTERFACES hace referencia las interfaces de red que se pretenden emular. Lo anterior debe ser guardado en un archivo con extensión ".py". Para aplicar los parámetros a emular se crea un archivo de Python de con el siguiente contenido (Figura 14):

Figura 14. Ejemplo de archivo de configuración de Shapy desde Python

```
#!/usr/bin/python
#IMPORTAR LIBRERIAS
from shapy.emulation.shaper import Shaper
from shapy import register_settings
#LLAMAR ARCHIVO DE CONFIGURACION
register_settings('archivo_configuracion.py')
#APLICAR REGLAS
ip        ="192.168.1.1"
subida    = 1000
bajada    = 500
latencia  = 250
jitter    = 25
regla  = {(ip,) : {'upload': subida, 'download': bajada, 'delay': latencia, 'jitter': jitter},}
sh  =  Shaper()
sh.set_shaping(regla)
```

En el ejemplo anterior se definen las reglas que se aplican en la interfaz correspondiente a la dirección IP 192.168.1.1, velocidad de subida de 1000 Kbps, de bajada 500 Kbps, latencia de 250 milisegundos y jitter de 25 milisegundos. Posteriormente en el momento de ejecución se crearán de forma automática dos interfaces de red virtuales IFB0 e IFB1 (ver Figura 15), en éstas se agrupa el tráfico entrante y saliente respectivamente, permitiendo enmascarar el comportamiento de las demás interfaces que se agregaron a la emulación (en el ejemplo: `eth0, lo`).

Figura 15. Esquema de control de tráfico sobre interfaces virtuales[42].



Para ejecutar Shapy se requiere tener permisos de administrador, tener instalado Python 2.6 ó 2.7, y con respecto a la arquitectura de hardware cabe destacar que se llevaron a cabo pruebas sistemas x86, x86-64 y ARM. Con respecto a la pérdida de paquetes, se

[42] CVUT. Framework for network management to support simulation of varying network conditions. En:Czech Technical University in Prague. [En línea]. (02, Mayo, 2015). Disponible en: <https://dip.felk.cvut.cz/browse/pdfcache/prauspet_2011bach.pdf>

puede emular con el comando iptables. La Figura 16 es un ejemplo donde se rechazarán el 15% de los paquetes entrantes en todas las interfaces:

Figura 16. Comando iptables para perdida de paquetes

```
iptables -A INPUT -m statistic --mode random --probability 0.15 -j DROP
```

4.4 MEDICIÓN DEL ANCHO DE BANDA

Para la medir el ancho de banda se utilizó la herramienta Iperf, usada a menudo para diagnosticar redes, funciona por una interfaz de línea de comandos en una arquitectura cliente servidor. Del lado del servidor basta con ejecutar "`iperf -s –u`", con ello se abre un *socket* UDP en el puerto 5001 como se aprecia en la Figura 17:

Figura 17.Ejemplo de ejecución de Iperf del lado del servidor.

```
ubuntu@ip-172-31-28-20:~$ iperf -s -u
------------------------------------------------------------
Server listening on UDP port 5001
Receiving 1470 byte datagrams
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
```

Del lado del cliente se debe especificar la dirección IP (en el ejemplo 1.1.1.1) del servidor para establecer la comunicación de forma bidireccional. Finalmente, se obtienen los datos de la velocidad en Mbits/s como se muestra a continuación (Ver Figura 18):

Figura 18. Ejemplo de ejecución de Iperf del lado del cliente.

```
[root@localhost ~]# iperf -u -c 1.1.1.1
------------------------------------------------------------
Client connecting to 1.1.1.1, UDP port 5001
Sending 1470 byte datagrams
UDP buffer size:  224 KByte (default)
------------------------------------------------------------
[  3] local 1.1.1.3 port 43002 connected with 1.1.1.1 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-10.0 sec  1.25 MBytes  1.05 Mbits/sec
[  3] Sent 893 datagrams
[  3] Server Report:
[  3]  0.0-10.0 sec  1.25 MBytes  1.05 Mbits/sec  18.923 ms    0/  893 (0%)
```

Los datos de salida arrojados por iperf fueron capturados usando Python, filtrados desde el *backend*, convertidos a formato JSON donde posteriormente se envían remotamente al navegador Web y por medio de Javascript se procesan dichos datos para desplegar la información en tiempo real, de esta forma se hace posible generar los gráficos estadísticos concernientes a la prueba sin emulación.

5. DESARROLLO DEL SOFTWARE iQos

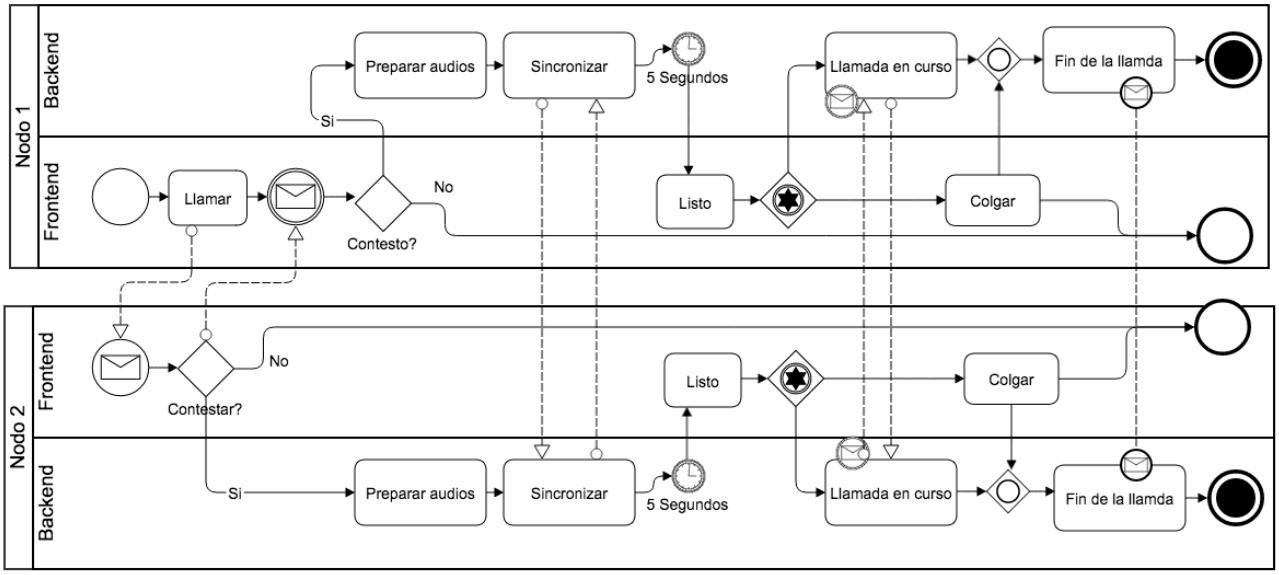
En este capítulo se presenta el proceso de desarrollo del software iQos para la evaluación de calidad de servicio en voz sobre IP, siguiendo el plan de trabajo establecido a través de la metodología de desarrollo ágil TDD.

5.1 DISEÑO

5.1.1 Procesos de negocio

Para definir y entender el comportamiento de las funcionalidades se realizó el modelado de procesos de negocio BPMN en las principales funciones del software. La figura 19 muestra un ejemplo el comportamiento del proceso para la recreación de una llamada de VoIP:

Figura 19. Proceso de recreación de llamadas de VoIP para iQos.



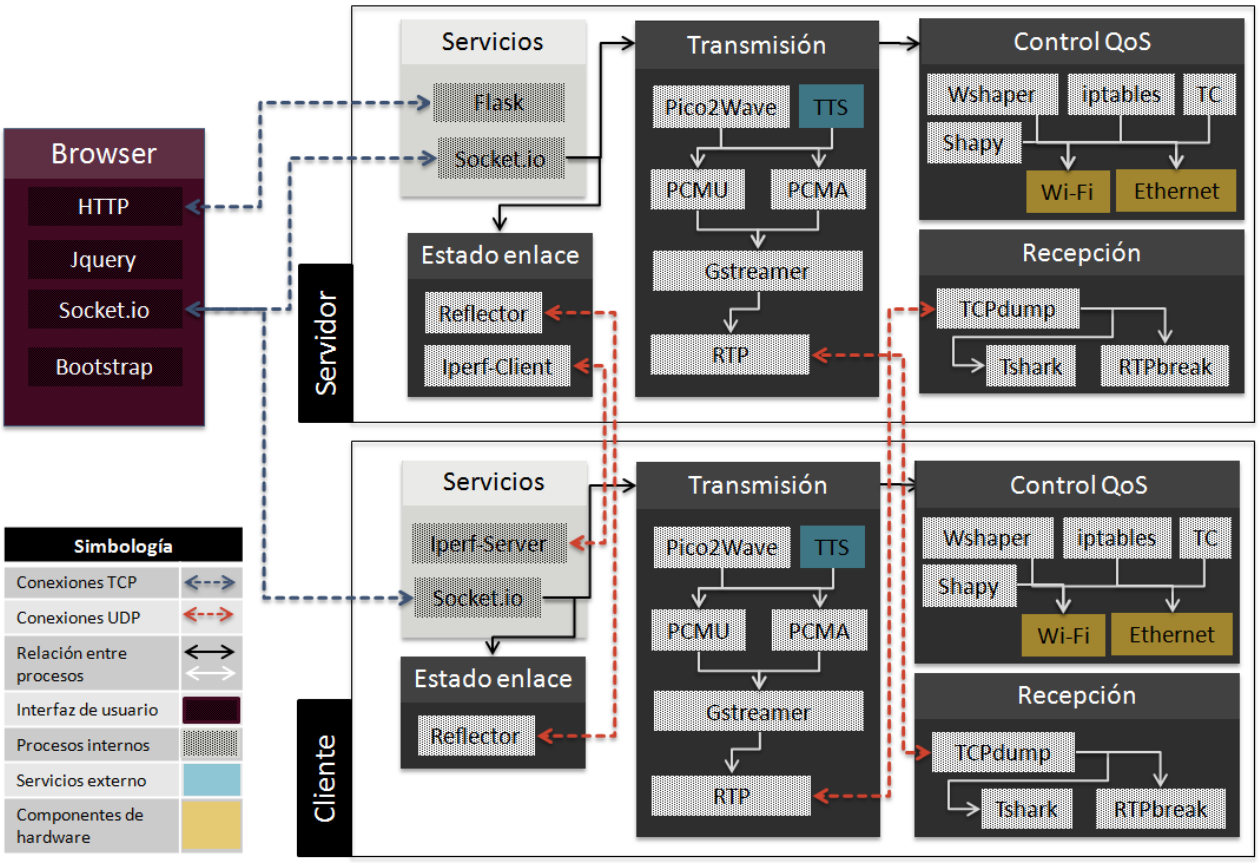
En el anterior diagrama, se observa la serie de pasos que se deben realizar durante el proceso de recreación de una llamada telefónica de VoIP entre dos nodos, en donde los primeros procesos requieren la interacción del usuario a través de una interfaz gráfica para ejecutar acciones como llamar, contestar y visualizar estados de la llamada, estos procesos son realizados en el lado del cliente y se define como *frontend*. Después se procede en el servidor (*backend*) a preparar los archivos de audio y establecer una sincronización entre los nodos para dar inicio a la llamada y mantenerla en curso hasta que el usuario indique la señal colgar desde el *frontend* para finalizarla.

Para consultar el documento completo de los modelo de procesos de negocios y sus respectivos casos de pruebas se puede consultar el anexo D.

5.1.2 Diagrama de arquitectura

Con el fin de entender el comportamiento y la relación entre los componentes de iQos, se define una arquitectura tipo cliente-servidor propia de los sistemas distribuidos que da soporte para mantener un servicio web. La figura 20 muestra el diagrama de la arquitectura del software con su debida interacción entre servicios y componentes.

Figura 20. Diagrama de arquitectura iQos.



Browser: Es la capa ejecutada desde un navegador web, se encarga de gestionar las peticiones y respuestas del cliente y el servidor mediante enlaces sincrónicos, usa el protocolo HTTP (*Hypertext Transfer Protocol*) y las peticiones REST (*Representational State Transfer)* o asincrónicos mediante un canal de comunicación bidireccional conocido como *websockets* cuya implementación se hace con *Socket.io*. Esta capa también cuenta con *Boostrap* y *Jquery*, dos *frameworks* para diseño de sitios y aplicaciones web que facilitan el control de la interfaz de usuario a través de la manipulación de las hojas de estilo CSS (*Cascading Style Sheets*) y el DOM (*Document Object Model*) de la página HTML.

Servicios: En este bloque se cuenta con un servicio HTTP escrito en el lenguaje de programación Python con el *framework* para aplicaciones web Flask, este servicio será el

encargado de procesar las solicitudes que vienen del navegador ya sea por REST o *websocket*. El otro servicio que se encuentra en esta capa es *Iperf (Server/Client)*, una aplicación de software, utilizada para crear flujos de datos UDP para medir ancho de banda.

Estado de enlace: Capa que evalúa el estado del enlace en términos de latencia, pérdida de paquetes y ancho de banda entre los dos nodos, esto datos se toman a partir de los resultados generados por el Iperf-Client y el script reflector de Python.

Transmisión: En esta capa interactúa el programa TTS (*Text To Speech*) llamado Pico2Wave, un conversor de texto a voz artificial que emula la voz humana en lengua castellana, y genera un archivo de audio que se codifica en G.711 PCMA o PCMU, se envía como la carga útil al Gstreamer para crear un paquete RTP, éste último se transmite por la interfaz de red al nodo destino de la llamada. Este proceso se repite tantas veces como llamadas simultáneas se establezcan con anticipación.

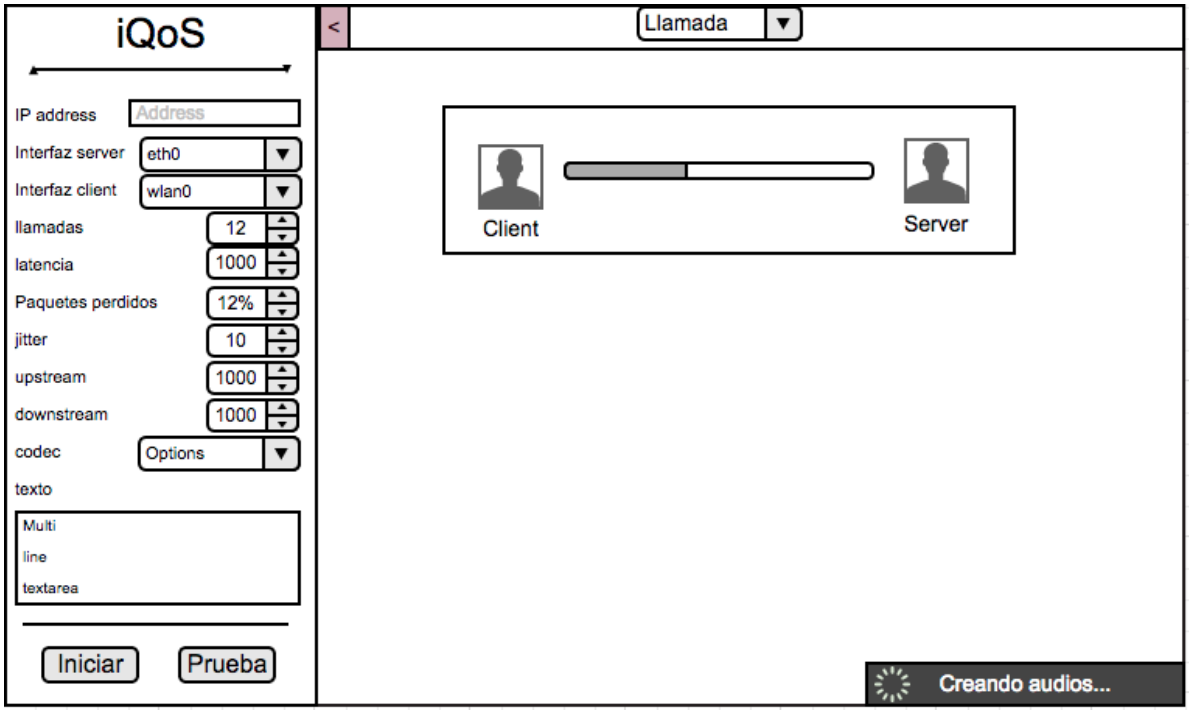
Recepción: Es el nivel que captura los paquetes UDP, filtrando únicamente los que contienen la cabecera RTP por medio de la herramienta de línea de comandos conocida como TCPdump, los paquetes son almacenados en un archivo con extensión .pcap para posteriormente ser procesados por el analizador de tráfico Tshark, que evalúa las métricas calculando un valor numérico. Rtpbreak es otra herramienta de software libre que analiza las sesiones RTP y facilita la extracción del *payload* en el receptor con el *codec* elegido previamente.

Control QoS: En la prueba de emulación se recrean parámetros de redes como latencia, pérdida de paquetes, *jitter* y ancho de banda, esto se logra combinando las herramientas de software libre iptables, el *framework* de python Shapy y wondershaper, un script que usa el *Traffic Control* (TC) de los sistemas UNIX. La alteración de estos parámetros de red tiene efecto sobre interfaces de red que soportan las tecnologías de capa 2 como Ethernet o IEEE 802.11a/b/g/n (Wi-Fi).

5.1.3 Interfaz gráfica

En esta etapa se planteó un diseño para solo una vista principal lugar donde contendrá todas las funcionalidades del software, esta interfaz se adapta a cada situación ocultando y mostrando contenedores cuando sea requerido por el usuario o el sistema, para así poder aprovechar el tamaño de la pantalla sin exceder sus límites teniendo en cuenta las diferentes resoluciones de pantalla.

Figura 21. Bosquejo de la vista durante la llamada



La Figura 21 es un ejemplo del bosquejo durante la realización de una prueba de calidad de servicio en VoIP, en el panel izquierdo se encuentra el formulario donde se indica los parámetros de configuración necesarios para la realización de la prueba; a continuación se describe el uso de cada campo.

- **IP address**: Corresponde a la dirección IP remota del servidor para iniciar la sincronización.

- **Interfaz de red servidor**: Hace referencia a la interfaz de red que va a usar el servidor en el momento de enviar y recibir los paquetes.

- **Interfaz de red cliente**: Hace referencia a la interfaz de red que va a usar el cliente en el momento de enviar y recibir los paquetes.

- **Número de llamadas**: Define la cantidad de llamadas VoIP simultáneas que se van realizar, el rango de valores va de 1 hasta 50.

- **Latencia**: Establece la latencia unidireccional en milisegundos. Valor máximo 1000 ms.

- **Paquetes perdidos**: Establece el porcentaje de pérdida de paquetes durante la

prueba.

- **Jitter**: Define la variación de la latencia unidireccional en milisegundos. Valor máximo 150 ms.

- **Upstream**: Establece la velocidad de transmisión de subida en Kbps. Valor mínimo 50 Kbps.

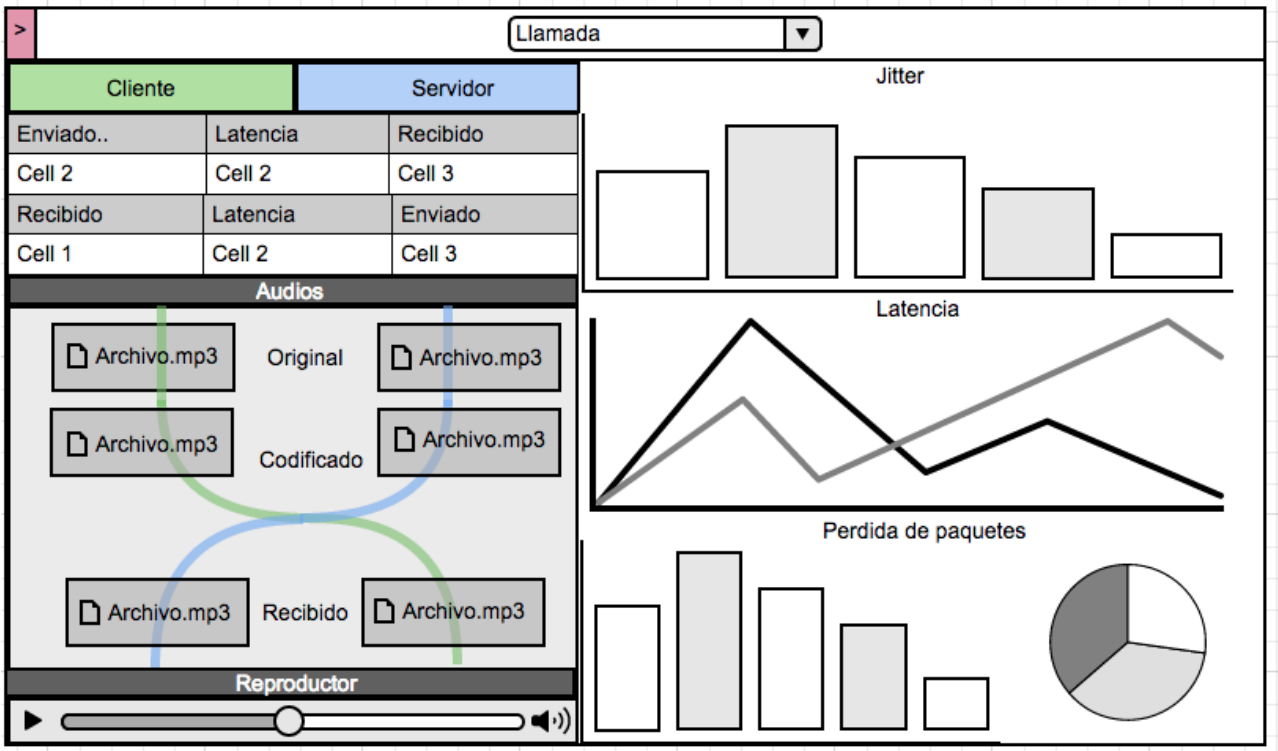
- **Dowstream**: Establece la velocidad de transmisión de bajada en Kbps. Valor mínimo 50 Kbps.

- **Codec**: Define el formato de codificación del audio G.711a (PCMA) o G.711u (PCMU).

- **Texto de prueba**: Campo opcional donde se crea un audio a partir del texto introducido, soporta un máximo de 1000 caracteres. Si el campo se deja en blanco, el software utilizará un conjunto de audios predeterminados.

El panel derecho muestra el progreso en tiempo real de la ejecución de la prueba y el combo box “llamada” permite seleccionar el número correspondiente a la llamada de la cual se desee visualizar los resultados.

Figura 22. Bosquejo vista resultados



La figura 22 representa el bosquejo de la interfaz de los resultados finales de la prueba de calidad de servicio, en esta vista se busca diferenciar y organizar los datos individuales de cada llamada, ubicando los gráficos estadísticos tales como los valores del *jitter* máximo y promedio de entrada y salida en cada uno de los nodos extremos, representados por medio de barras verticales; luego los valores de la latencia en los dos sentidos de la llamada donde se grafican usando un polígono de frecuencia; y al final de este panel lateral derecho se muestra el porcentaje de paquetes perdidos en la salida y entrada de la interfaz de red de cada nodo y así el total en toda la llamada usando una gráfica tipo torta. En el panel izquierdo están ubicados los datos numéricos obtenidos de la prueba, organizados en una tabla donde se relacionan los valores de las métricas y el sentido del flujo de datos. Finalmente en el lado inferior izquierdo se despliegan los archivos de audio generados (original), codificados (Enviado) y percibidos en cada nodo (Recibido), listos para ser reproducidos en la barra de control inferior.

5.2. IMPLEMENTACIÓN

La construcción de iQos fue realizada siguiendo los lineamientos de una metodología de desarrollo ágil TDD, partiendo de pruebas unitarias definidas en el Anexo C y utilizando un conjunto de herramientas de uso libre explicadas en la sección 2.3 del documento.

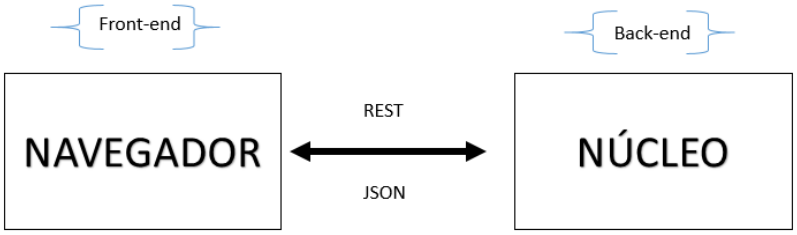
5.2.1 Etapas de la prueba con emulación

La prueba con emulación permite recrear una cantidad predefinida de llamadas simultáneas de VoIP e incluir parámetros en la red como latencia, jitter, ancho de banda, pérdida de paquetes y la generación de audios a partir de cadenas de texto (*Text to speech*). Cada vez que se inicia una nueva prueba esta tiene que atravesar un total de ocho etapas ordenadas, que están descritas en el esquema de la Figura 23. La etapa de desarrollo inicia con la construcción del núcleo del software en el lenguaje de programación Python, en el núcleo se encuentra el controlador de las funciones con mayor importancia para el proceso de control de calidad de servicio y emulación de llamadas de VoIP, a continuación se nombran los componentes incluidos en el núcleo.

Figura 23. Etapas de la prueba con emulación.

A. Definir prueba: Es la etapa inicial donde se establecen todas las características que se van a configurar en la prueba desde el navegador Web, básicamente consiste en un formulario en HTML5 compuesto de campos que deben ser diligenciados en el *Frontend*. Los datos del formulario se normalizan al formato JSON y son enviados mediante peticiones REST al *Backend* (ver Figura 24) para que dichos datos sean interpretados.

Figura 24. Comunicación entre *Frontend y Backend.*



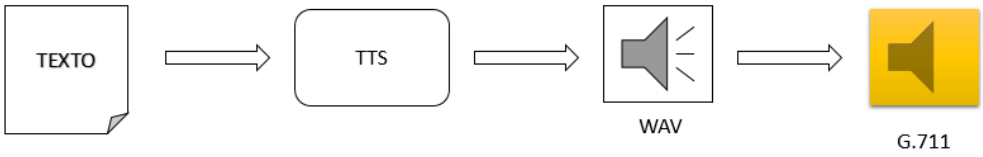
En la Figura 25 se muestra el ejemplo del código Javascript donde se almacenan los datos de los campos contenidos en los objetos DOM del formulario, se crea un selector que apunta al elemento con el identificador sendJson asociándolo con el evento click(), posteriormente se crea una estructura en formato JSON y se envía al nodo servidor por medio de websockets.

Figura 25. Código Javascript donde se normalizan y se envían los datos JSON

```
$('#sendJson').click(function(){
    datos = {};
    flag1= true;
    error=false;
    datos.ipRemota          = $("#ipRemota").val();
    datos.ipLocal           = ipLocal;
    datos.interfazRemota    = $("#interfazRemota").val();
    datos.interfazLocal     = $("#interfazLocal").val();
    datos.numeroLlamadas    = validadParametro($("#numeroLlamadas"),1);
    datos.latencia          = validadParametro($("#latencia"),0);
    datos.paquetesPerdidos  = validadParametro($("#paquetesPerdidos"),0);
    datos.jitter            = validadParametro($("#jitter"),0);
    datos.upstream          = validadParametro($("#upstream"),1);
    datos.downstream        = validadParametro($("#downstream"),1);
    datos.text              = $("#texto_google").val();
    datos.codec             = $('#codec').val();
    #envio de los datos de configuración a los servidores
    socket.emit('sendparametros', datos);
    socket2.emit('sendparametros', datos);
    }
});
```

B. Creación de audio con códec G.711: Una vez diligenciado el formulario, se selecciona el texto que se va a convertir en audio usando un TTS llamado Picowave, posteriormente se genera un archivo con extensión .wav que finalmente se codifica al estándar G.711 (ver Figura 26). Estos audios codificados son transportados en paquetes para llevar a cabo las llamadas telefónicas tanto en cliente como el servidor.

Figura 26. Proceso de generación de audio.



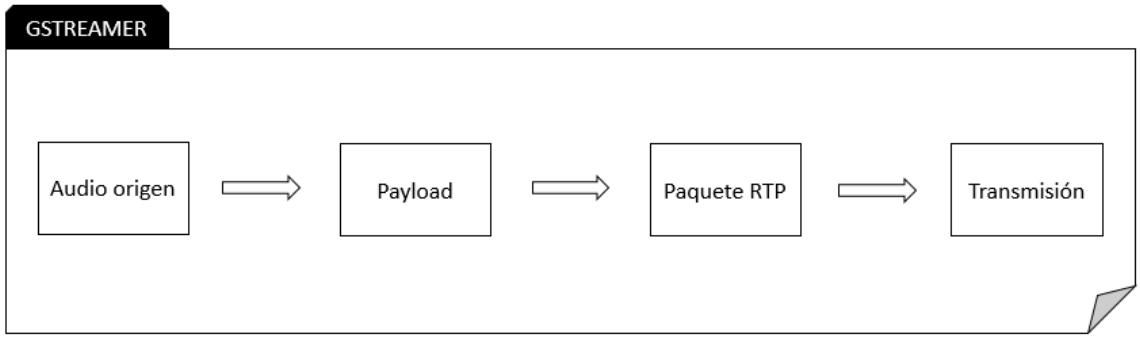
En la figura 27 se muestra la secuencia de comandos ejecutados desde Python para el proceso de creación de los audios, uno de ellos es el uso herramienta pico2wave donde se le especifica el texto, el lenguaje y la ruta del archivo de salida, luego con el comando sox se agregan audios al inicio y final del archivo generado para diferenciar la llamada con las otras que también se están ejecutando.

Figura 27. Método de Python para la generación de audio.

```
if not local:
      #Audio a partir de texto - API TTS GOOGLE
      googletts   =os.system("cp static/pruebas/google.mp3 " + ruta)
      googletts   =os.system("cp static/pruebas/google.mp3 " + ruta )
      pico2wave1  =os.system("pico2wave -l es-ES -w " + ruta + "inicio.wav" + " '" + texto1     + "'")
      pico2wave2  =os.system("pico2wave -l es-ES -w " + ruta + "final.wav" + " '" + texto2      + "'")
      audio       ="google.mp3"
      mp3_wav     =os.system("sox " + ruta + audio + " -r 16k " + ruta + audio.replace('.mp3','.wav'))
      combinar    =os.system("sox " + ruta + "inicio.wav " + ruta + audio.replace('.mp3','.wav') + " " + ruta + "final.wav " + ruta+archivo.replace('.mp3
      wav_mp3     =os.system("lame "+ ruta + archivo.replace('.mp3','.wav') +  " " + ruta+archivo)
      return True
else:
      #Audio predeterminado
      pico2wave1  =os.system("pico2wave -l " + cod_lenguaje + " -w " + ruta + "inicio.wav" + " '" + texto1  + "'")
      pico2wave2  =os.system("pico2wave -l " + cod_lenguaje + " -w " + ruta + "final.wav" + " '" + texto2   + "'")
      copia       =os.system("cp static/pruebas/default/audios/" + audio + " " + ruta)
      mp3_wav     =os.system("sox " + ruta + audio + " -r 16k " + ruta+audio.replace('.mp3','.wav'))
      combinar    =os.system("sox " + ruta + "inicio.wav " + ruta+audio.replace('.mp3','.wav') + " " + ruta + "final.wav " + ruta+archivo.replace('.mp3',
      wav_mp3     =os.system("lame "+ ruta + archivo.replace('.mp3','.wav') +  " " + ruta+archivo)
      return True
```

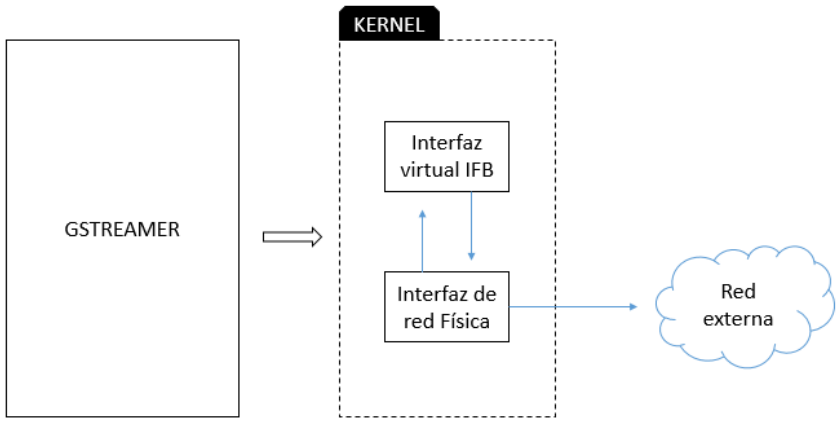
C. Creación de los paquetes RTP: Por medio del software Gstreamer se construyen los paquetes RTP usando un mecanismo de tuberías, en el cual se encapsula el archivo de audio de origen ya codificado siguiendo el estándar G.711, este archivo se segmenta para ser introducido como *payload* dentro de los paquetes que finalmente se van a transmitir (ver Figura 28).

Figura 28. Creación de paquetes RTP con Gstreamer.



D. Transmisión y control: Gstreamer a medida que va generando los paquetes RTP se comunica internamente con el *kernel* del sistema operativo para transmitir cada uno de ellos por la interfaz de red hacia la dirección IP destino con un orden de secuencia y una marca de tiempo (ver Figura 29).

Figura 29. Esquema de transmisión de paquetes donde interviene Gstreamer y el *Kernel* del sistema Operativo.



Gstreamer trabaja por medio de argumentos donde se le indica el tipo de códec a usar G.711a (PCMA) o G.711u (PCMU) dependiendo de la selección del usuario, la ubicación del archivo de audio y la dirección IP del destino, como se muestra en código de la figura 30.

Figura 30. Envío de audios

```
def txAudio(self, audio_generado, codec, encolar = False):
    try:
        self.auout  =   audio_generado
        comando     =   "gst-launch-0.10 -v "
        archivo     =   "filesrc location=" + self.auout + " "
        marca       =   "do-timestamp=true ! "
        pcm         =   "mad ! audioconvert ! audioresample ! alawenc ! "
        rtp         =   "rtppcmapay !"
        destino     =   "udpsink host=" + self.ipd + " port=" + str(self.pd)

        if codec == "pcmu":
            pcm     =   "mad ! audioconvert ! audioresample ! mulawenc ! "
            rtp     =   "rtppcmupay ! "

        if  encolar:
            rtp     =   "queue ! " + rtp

        salida, error = Popen(comando+archivo+marca+pcm+rtp+destino, stdin=PIPE, stdout=PIPE, stderr=PIPE, close_fds=True, shell=True)

        return salida

        except:
            print "*** ERROR: En transmision.txAudio(), gstreamer debe estar instalado"
            return False
```

La función de transmisión (txAudio) se ejecuta en paralelo para que todas las llamadas inicien al mismo tiempo y así evitar encolamientos o que las llamadas se lancen una después de otra (secuencial).

Tal como se describió en el apartado 4.3 del capítulo anterior, la emulación de métricas de calidad de servicio se realizó usando el *framework* Shapy de python, que de una manera simple facilitó el control de la pérdida de paquetes, latencia, jitter y ancho de banda, por lo tanto se creó un método para el control del tráfico a partir de una configuración especificada por el usuario

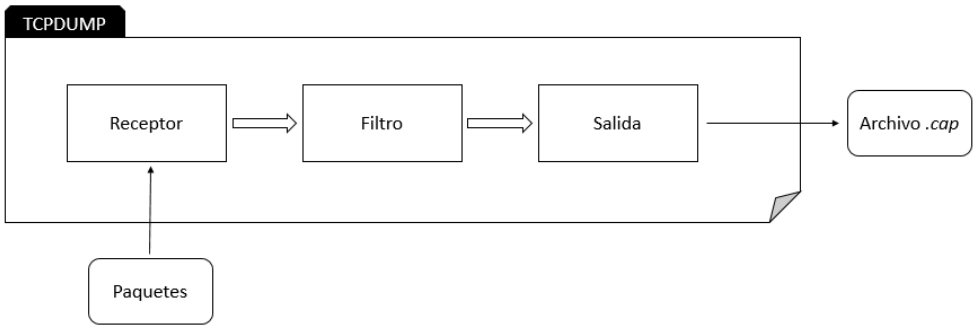
Figura 31. Método controlar QoS

```
#APLICA EL TRAFIC SHAPING PARA EMULAR LATECIA, PERDIDAS, JITTER
def controlar(self):
      try:
            "latencia ms, jitter ms, perdida % decimal, subida y descarga en kbits  "
            from shapy.emulation.shaper import Shaper
            from shapy import register_settings
            register_settings('configuracion')
            regla       = {(self.ipo,) : {'upload': self.bw_up, 'download': self.bw_down, 'delay': self.delay, 'jitter': self.jitter},}
            sh          =     Shaper()
            sh.set_shaping(regla)
            if self.drop:
                  os.system("tc qdisc add dev " + self.iface + " parent 1:1 handle 512: netem loss " + str(int(self.drop*100))+"%")
                  os.system("tc qdisc add dev " + self.iface + " parent 1:2 handle 513: netem loss " + str(int(self.drop*100))+"%")
                  os.system("tc qdisc add dev " + self.iface + " parent 1:3 handle 514: netem loss " + str(int(self.drop*100))+"%")
            return True
            except:
                  print "*** ERROR: En transmision.controlar(), verifique que se haya limpiado shaping en las interfaces"
            return False
```

En el segmento de código de la figura 31, se define una regla en formato JSON indicándole al Shapy los valores a emular, en el caso de la pérdida de paquetes se ejecuta el comando *traffic control* del sistema especificando el porcentaje de paquetes a perder.

E. Recepción y almacenamiento: Los paquetes entrantes se reciben por la interfaz de red y el programa Tcpdump se encarga de filtrar únicamente aquellos que pertenezcan al protocolo UDP y RTP, que contengan las direcciones origen y destino asociadas a los host cliente y servidor, donde finalmente son almacenados en archivos con extensión ".cap" en el orden de llegada (ver Figura 32).

Figura 32. Esquema de recepción y almacenamiento de paquetes con Tcpdump



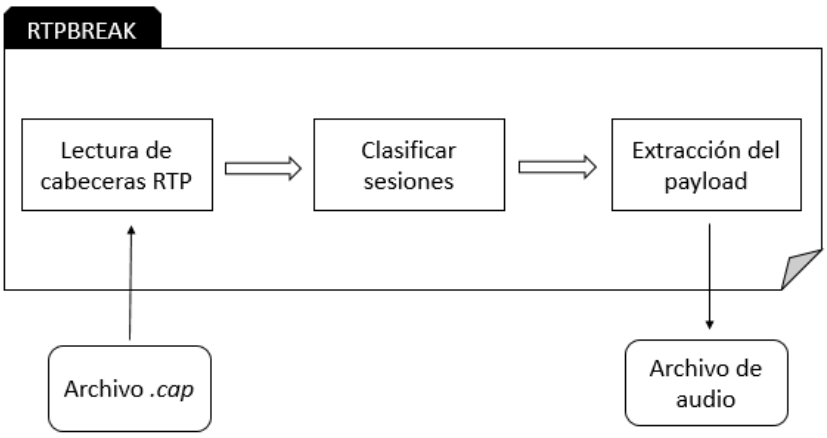
En la función rxAudio() del código de Python de la Figura 33, al igual que la función txAudio() de la etapa anterior, se ejecuta en un hilo separado para recibir todos los paquetes en la red interfaz seleccionada, en este caso se invoca el comando Tcpdump y por medio de parámetros se crea un filtro y un archivo destino como salida.

Figura 33. Método de recepción de paquetes.

```
#CAPTURA PAQUETES EN LA INTERFAZ SELECCIONADA
def rxAudio(self, nombre_prueba,codec):
    try:
        comando        =    "sudo tcpdump -i "
        interfaz     =    self.iface + " "
        puerto         =    self.pd
        filtro         =    "'(src " + self.ipd + " and dst " + self.ipo + ") or (src " + self.ipo + " and dst " + self.ipd +") and udp' -w "
        archivo_cap    =    nombre_prueba
        self.pcap_recv  =    archivo_cap
        self.codec        =codec
        verbose        =    " -vv -ttttt"
        #POPEN PERMITE CREAR HILOS USADO EN COMANDOS EN ESCUCHA QUE NO FINALIZAN
        salida1, error = Popen(comando + interfaz + filtro + archivo_cap + verbose, stdin=PIPE, stdout=PIPE, stderr=PIPE, close_fds=True,
l=True).communicate(None)
        print salida1
        print "TCPDUMP FINALIZADO"
        return True
    except:
        print "*** ERROR: En transmision.rxAudio(), tcpdump debe estar instalado"
        return False
```

F. Extraer audio: Una vez almacenados los paquetes, rtpbreak accede al archivo de la captura, identifica los paquetes RTP y hace una clasificación de las sesiones de VoIP, finalmente se extrae el contenido del *payload* para generar un archivo de audio de salida (Figura 34).

Figura 34. Proceso de extracción del *payload* con RTPbreak.



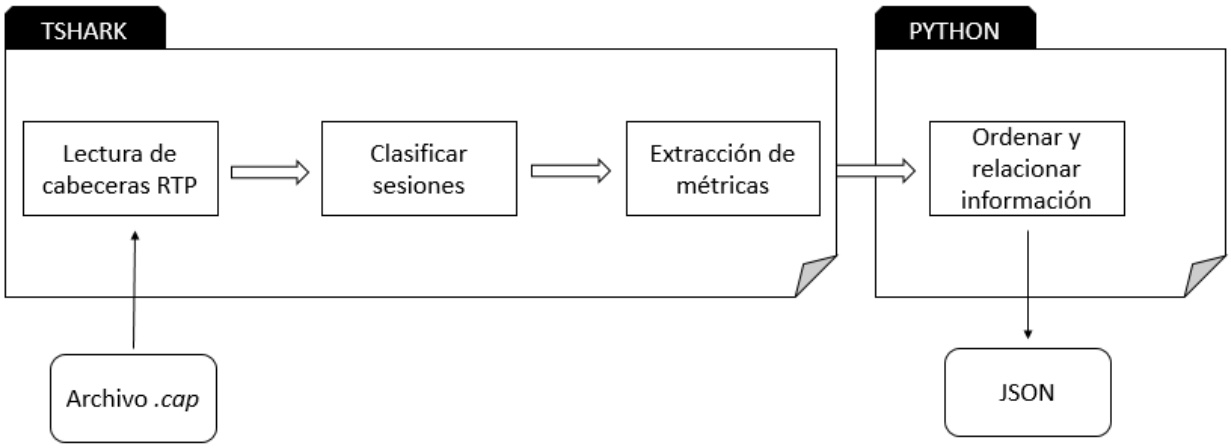
El código de python de la Figura 35 muestra la función extraerAudio() que recorre y procesa cada paquete almacenado en la ruta del archivo .cap indicada, renombra los archivos de audio y los convierte en formato mp3 para que puedan ser reproducidos en el navegador web.

Figura 35. Método para extraer y convertir audios

```
#EXTRAE EL AUDIO APARTIR DE UN ARCHIVO PCAP, RENOMBRA RAW A UL O AL Y CODIFICA DE NUEVO A MP3
def extraerAudio(self):
    try:
        archivo_cap    =    self.pcap_recv
        ruta_extraido    =    "static/pruebas/extraido/"
        salida2        =    commands.getoutput("mkdir -p " + ruta_extraido)
        #OPCIONES PARA PREVENIR PERDIDA DE PAQUETES
        salida3        =    commands.getoutput("rtpbreak -d " + ruta_extraido + " -r " + archivo_cap + " -g -n -P2 -t100 -T100")
        print salida3
        #CAMBIAR EXTENSION DE RAW A UL Y LUEGO CONVERTIR A MP3
        print "RENOMBRANDO"
        os.chdir("static/pruebas/extraido/")
        if self.codec == "pcma":
            os.system("rename 's/raw$/al/' *")
            os.chdir("../../../")
            archivos    =    commands.getoutput("ls  " + ruta_extraido+ "*.al").split("\n")
        else:
            os.system("rename 's/raw$/ul/' *")
            os.chdir("../../../")
            archivos    =    commands.getoutput("ls  " + ruta_extraido+ "*.ul").split("\n")
        for archivo_audio_pcm in archivos:
            if (archivo_audio_pcm.find("ul") >=0 ) or (archivo_audio_pcm.find("al") >=0):
                pcm2mp3(archivo_audio_pcm, archivo_audio_pcm[0:-2]+ ".mp3", self.codec)
    except:
        print "*** ERROR: En transmision.extraerAudio(), verifique que existan los archivos .raw"
        return False
```

G. Procesamiento: Con la herramienta de análisis de tráfico Tshark, son extraídos los datos de las cabeceras de cada paquete RTP en el receptor, luego se procesan, filtrando los tiempos de llegada, marcas de tiempo, números de secuencia y los identificadores de la sesión de cada llamada. Los resultados desplegados por Tshark son ordenados y normalizados en formato JSON a través de un script de Python (Figura 36).

Figura 36. Esquema de la interacción de Tshark con Python.



La función analizar() comunica los parámetros con las heurísticas asociadas con el protocolo RTP y con expresiones regulares se depura la salida con los valores de las métricas y se añaden a un *array* de objetos JSON (ver Figura 37)

Figura 37. Código en Python con el método de extracción de métricas normalización de los datos.

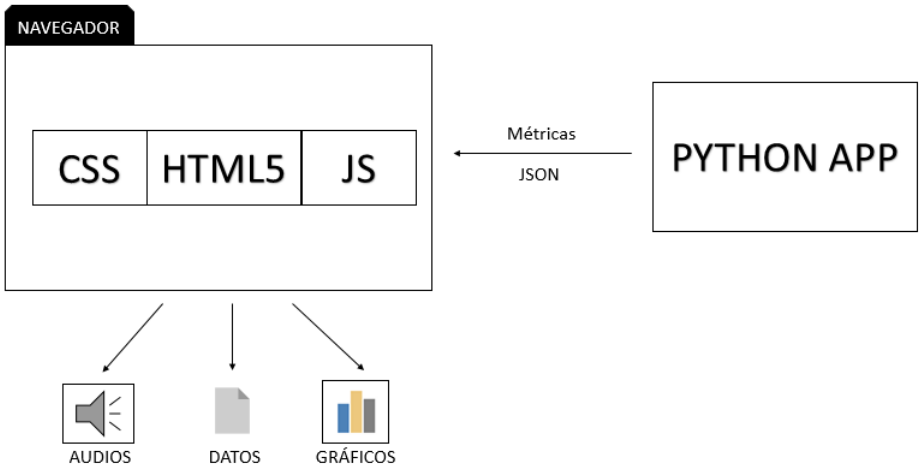
```
#ANALIZA EL TRAFICO CAPTURADO, EXTRAE METRICAS RTP
def analizar(self):
    try:
        comando    =    "tshark -r "
        filtro    =    " -o    rtp.heuristic_rtp:TRUE -nqz rtp,streams"
        salida = commands.getoutput(comando + self.pcap_recv + filtro).split('\n')
        i=0
        print "TSHARK", salida
        while True:
            if salida[i].find(self.ipd) < 0:
                salida.pop(i)
            else:
                i+=1
                if salida[i].find("================================")>=0:
                    salida.pop(i)
                    break

        datosOut=[]
        for linea in salida:
            linea = linea.strip()
            linea = re.sub(' {1,}',' ',linea).split(' ')
            tmp = {'ipd':linea[2],'port':linea[3],'paquetes':linea[8],'loss':linea[9], 'latencia':linea[11],'maxJitter':linea[12],'avgJitter':linea[13]}
            datosOut.append(json.dumps(tmp))
        return datosOut

    except:
        print "*** ERROR: En transmision.analizar(), verifique que se haya capturado trafico"
        return False
```

H. Resultados: Después de tener los datos ordenados y definidos en formato JSON, cada nodo envía todos los valores adquiridos durante la prueba por medio de Websockets, dichos datos son recibidos por el navegador Web y éste a su vez los válida. Posteriormente se despliegan los contenedores que estructuran la vista , asociando los resultados con los valores de las métricas de calidad de servicio, gráficos estadísticos y los audios de cada llamada gracias a la combinación de tecnologías como HTML5, CSS y JavaScript (ver Figura 38), con ello se facilita la interpretación de la información para los usuarios.

Figura 38. Esquema de envío y despliegue de los resultados de la prueba de calidad de servicio.



5.3 REQUISITOS Y ENTORNO DE PRUEBAS

Para evaluación de métricas de QoS para VoIP se tuvieron en cuenta las siguientes condiciones y ambiente de pruebas.

5.3.1 Software y características de las máquinas virtuales

iQos fue ejecutado en dos máquinas virtuales emulando una arquitectura cliente servidor con la siguiente configuración:

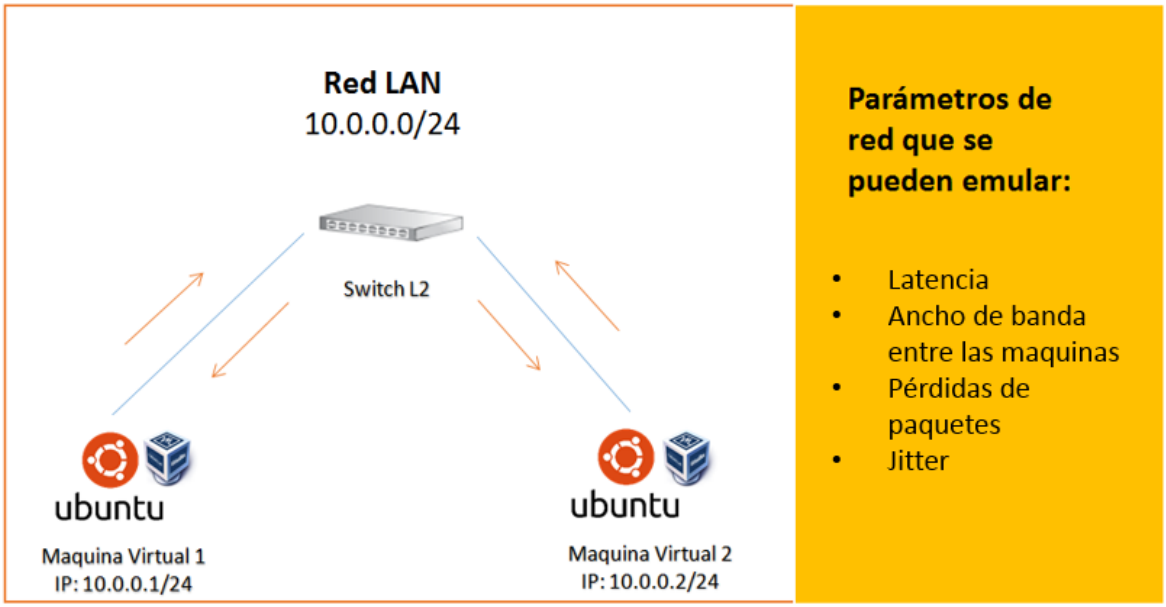
Tabla 6. Configuración de los equipos cliente y servidor en máquinas virtuales

Software máquina virtual	VirtualBox
Sistema operativo	Ubuntu Linux 14.04
Arquitectura	64 bits
Disco Duro	8 GB
Memoria RAM	64 bits
Tipo de Red	*Bridge*
NIC	1 Gbps

5.3.2 Topología y arquitectura de Red

Se configuró una red LAN con topología en estrella donde un switch capa 2 actuó como el punto de interconexión para las máquinas virtuales las cuales son visibles entre sí, en el segmento de red 10.0.0.0/24 como se muestra en la Figura 39.

Figura 39. Esquema de conexiones de red y virtualización.



Durante la ejecución de las pruebas se estableció que la dirección 10.0.0.1/24 actuara como el servidor y la dirección 10.0.0.2/24 como el cliente.

5.3.3 Requisitos de hardware

En caso de ejecutar iQos de forma nativa se recomienda tener en consideración los siguientes requisitos mínimos de hardware:

- Procesador Intel Celeron de 700 MHz.
- Memoria RAM de 512 MB
- 50 MB de espacio en disco para el núcleo del programa, se recomienda por lo menos 500 MB para almacenar datos durante las pruebas que superen las 20 llamadas simultáneas.
- Acceso a internet cuando se requiera hacer uso del campo “Texto de prueba” (ver sección 6.3 Formulario de configuración inicial) lo cual necesita conectarse al servicio de *Text To Speech* de Google.
- Para el despliegue de la interfaz web se requiere una pantalla VGA con resolución mínima de 1024 x 768 pixeles.

5.3.4 Dependencias

Aunque iQos posee un instalador por línea de comandos almacenado en un script, donde se relacionan todas las dependencias (ver Anexo B), se pueden instalar por separado los siguientes paquetes de software:

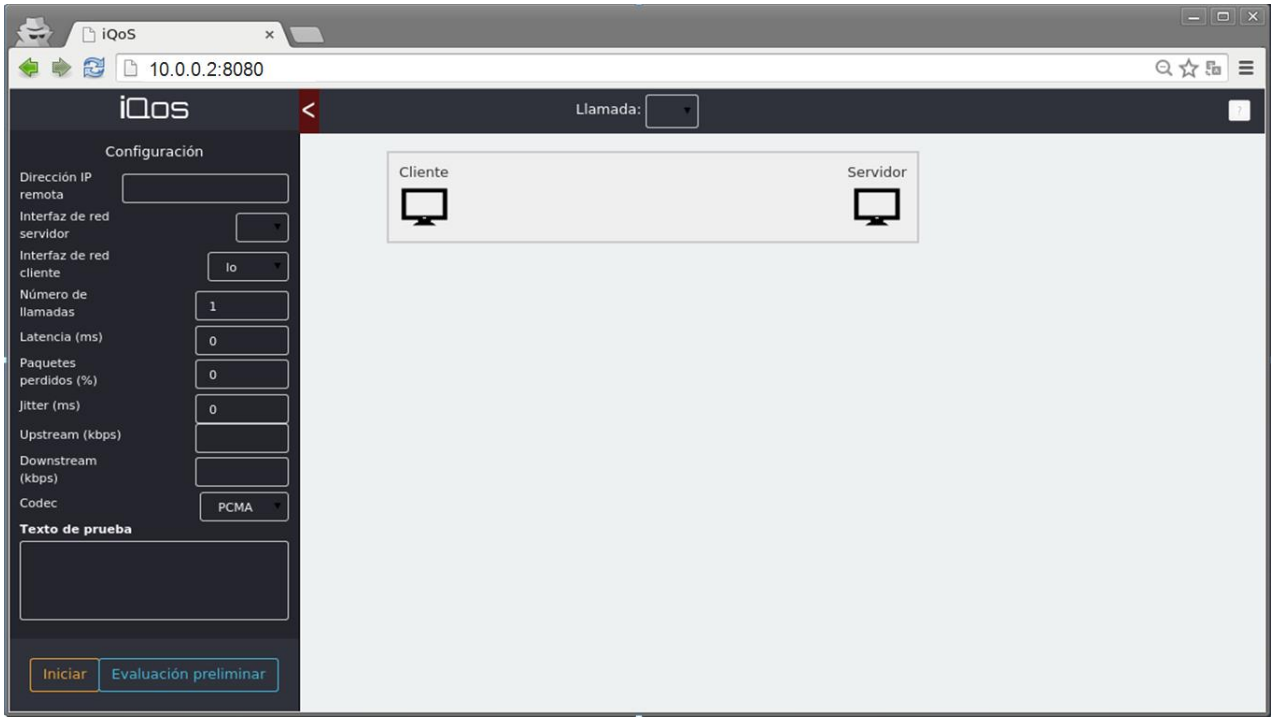
- Python 2.7.6:
  - shapy
  - Flask
  - SocketIO_Client
  - json
- Tratamiento de audios:
  - gstreamer
  - sox
  - pico2wave
- Tratamiento de paquetes:
  - tc
  - iptables
  - rtpbreak
  - tshark
  - tcpdump
  - iperf

5. RESULTADOS

6.1 INTERFAZ GRÁFICA

iQos posee una interfaz gráfica Web, a la que se puede acceder digitando la dirección IP del equipo cliente en el puerto 8080, para el siguiente ejemplo se accede por medio de la barra de URL a la dirección 10.0.0.2:8080, esto despliega la siguiente vista (Figura 40):

Figura 40. Interface inicial de iQos



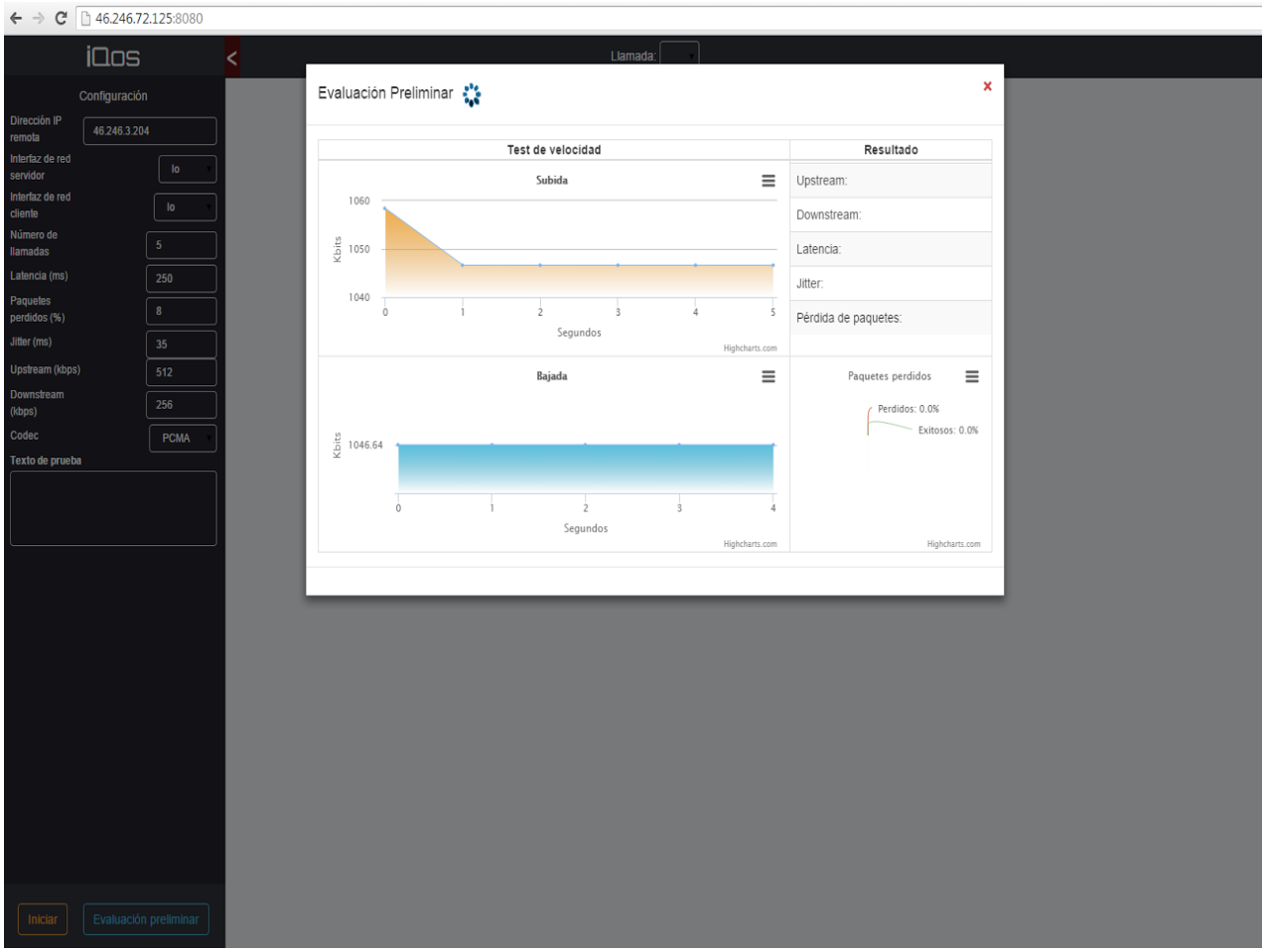
En la página principal se configura la ejecución de pruebas de calidad de servicio, cada campo del formulario y elemento de la GUI es explicado en la sección 6.4 y 6.5. iQos permite realizar una prueba preliminar donde se evalúa el estado real del enlace antes de llevar a cabo una prueba con emulación de parámetros en la red.

6.2 PRUEBA PRELIMINAR SIN EMULACIÓN

La prueba de evaluación preliminar consiste en realizar una comunicación entre el cliente y el servidor con el fin de analizar el desempeño real de la red de datos. Finalizada la prueba, se mostrará en una ventana modal la información del ancho de banda, *jitter*, latencia y pérdida de paquetes (ver Figura 41). Este tipo de prueba es importante debido a que el usuario puede verificar previamente los parámetros reales en la red antes de

llevar a cabo una prueba con emulación, lo que permitirá hacer interpretaciones adecuadas de los resultados.

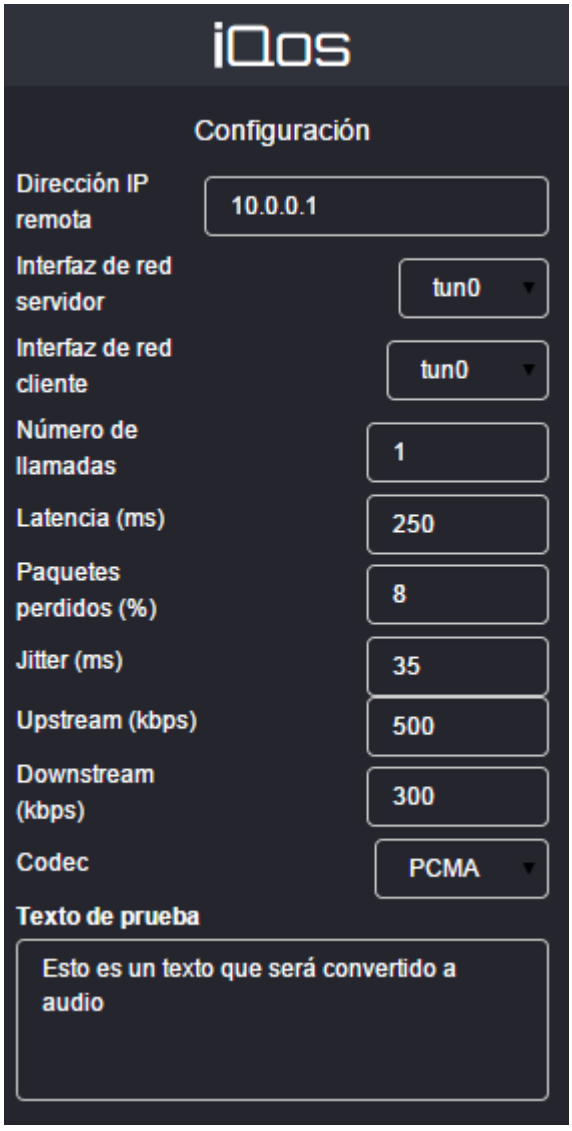
Figura 41. Ventana modal de la prueba de evaluación preliminar.



6.3. FORMULARIO DE CONFIGURACIÓN INICIAL

Como se mencionó en la etapa A, se debe diligenciar un formulario que contiene los campos necesarios para dar inicio a la prueba de emulación, cada campo se explica en la sección 5.1.3.

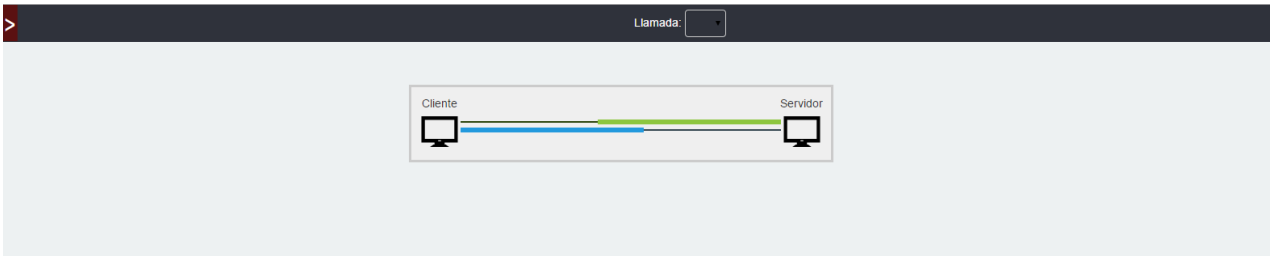
Figura 42. Panel de prueba con emulación.



6.4 TRANSCURSO DE LA PRUEBA

Durante el transcurso de la prueba se muestran gráficamente dos imágenes que representan los nodos cliente y servidor junto a una barra de progreso indicando el estado y las etapas en curso de la prueba (ver Figura 43). Es aquí donde se sincronizan las llamadas telefónicas para que sean transmitidas simultáneamente por medio del uso de websockets que actúa como un protocolo de señalización.

Figura 43. Indicador de progreso de la prueba con emulación.



6.5. RESULTADOS Y GENERACIÓN DE ESTADÍSTICAS

Al finalizar se ordenan las llamadas VoIP con los datos obtenidos en una tabla. En la parte inferior se puede reproducir cada audio enviado y recibido, antes y después de decodificar, con lo cual se puede contrastar su calidad (ver Figura 44). La tabla despliega valores numéricos de las métricas obtenidas para cada llamada tanto en el cliente como en el servidor, ya que es importante tener en cuenta el sentido del flujo de los datos. Como se mencionó anteriormente es posible reproducir cada uno de los audios por medio de unos controles *play, pause y stop*, además se permite la descarga de los audios. Cabe anotar que por cada llamada se generan seis archivos, 3 en el servidor y 3 en el cliente. El primero corresponde al audio generado antes de codificar a G.711 y que posee la mayor calidad, el segundo cuando se codifica a G.711 y se encapsula en el paquete RTP, y el tercero se crea extrayendo el payload del paquete en el momento que éste arriba al destino.

Figura 44. Vista de los resultados de cada llamada

CONCLUSIONES

El principal aporte de este proyecto es la construcción de una herramienta de software llamada iQos, portable, gráfica y simple de usar, que emula llamadas de VoIP en una red de datos y las evalúa mediante métricas de calidad de servicio, enmarcadas dentro de estándares abiertos; esto hace posible tener un punto de referencia para validar los resultados y la hace útil en entornos corporativos y domésticos.

Una fase crítica del proyecto consistió en integrar diversas herramientas de software libre, tecnologías y estándares en un solo aplicativo, por ello fue indispensable el diseño de un diagrama de arquitectura para tener una perspectiva global de los módulos requeridos por el programa, donde se relaciona cada componente que debe ser integrado, y posteriormente se definieron un conjunto de pruebas aisladas basadas en la metodología de desarrollo ágil TDD, esto permitió hacer avances incrementales para depurar errores o anomalías que afectaban el funcionamiento adecuado del software.

Se evidenció que durante la ejecución de las pruebas con llamadas simultáneas, el orden de los paquetes en el momento de arribar al destino era incierto, esto representaba un inconveniente al momento de desplegar los resultados debido a que no se podía identificar cada llamada ni asociarles los valores correspondientes de las métricas; por lo anterior se debió implementar un algoritmo que inspeccionara cada paquete, lo agrupara en la llamada a la que pertenecía para que coincidiera el audio y los valores de las métricas. Teniendo en cuenta que el tamaño de la captura de los paquetes puede ser considerable a medida que el *payload* y el número de llamadas sea mayor, se optó por convertir los campos más importantes de los paquetes RTP a formato JSON para transmitirlos por *websockets* al navegador WEB.

Al implementar una arquitectura cliente-servidor para realizar llamadas simultáneas basadas en RTP, se deben considerar métodos de sincronización precisos para la transmisión de los paquetes, debido a que no se están usando protocolos de señalización convencionales como SIP o H.323. En el caso particular del proyecto, se llevó cabo una sincronización con *websockets* entre el navegador web y el *backend* para no omitir datos en el momento de la transmisión y recepción de las llamadas, logrando evitar una distorsión en la lectura de las métricas de QoS evaluadas en este proyecto.

iQos proporciona valores cuantitativos de las métricas de calidad de servicio, estos valores deben ser analizados e interpretados por los usuarios para dar las valoraciones de los resultados, debido a que el software no emite juicios o recomendaciones después de realizadas las pruebas.

RECOMENDACIONES

En este proyecto se consideró implementar G.711 como codec de audio y RTP como protocolo de transporte de los datos, pero como se mencionó en capítulos precedentes VoIP está compuesto de un extenso conjunto de protocolos y codecs que pueden ser abarcados como una mejora de este software.

Se sugiere se pueda en ocasiones futuras adicionar funcionalidades de detección de fallas en llamadas, y que en respuesta a éstas se sugieran cambios de políticas o configuraciones para un adecuado funcionamiento de VoIP en la red de datos.

Debido a que las pruebas de iQos fueron llevadas a cabo en una red LAN, se recomienda extenderlas a diferentes tipos de redes como WAN, VPN o Wireless con el fin de verificar el comportamiento en entornos de producción.

iQos evalúa de forma cuantitativa las métricas de calidad de servicio calculando éstas por medio de técnicas que indican el estado de la red, un complemento al software podría ser la adición de métodos subjetivos y cualitativos para medir la calidad de la experiencia.

GLOSARIO

Ancho de banda: Capacidad de transferencia de datos entre dos host durante una unidad de tiempo.

Backend: Es la parte de un sistema de software que procesa los datos enviados y emite una respuesta al frontend.

BPMN: Notación para el modelado de procesos de negocio.

Buffer: Memoria temporal que almacena de datos utilizado para mitigar las variaciones en los flujos de transmisión.

Downstream: Ancho de banda disponible para la recepción de datos

FIFO: Primero en entrar, primero en salir, corresponde a una estructura de datos de colas.

Framework: Una arquitectura de sofware que incluye artefactos o módulos especializados.

Frontend: Es el componente donde se ingresan los datos en un sistema de software.

Host: Una entidad en Internet capaz de emitir y recibir paquetes IP.

HTML: Es un lenguaje de etiquetado (*Hyper Text Markup Language*) para la *World Wide Web*, tiene como objetivo permitir la creación, estructuración y presentación de contenidos en lo que se conoce hoy en día como páginas web.

IETF: Internet Engineering Task Force

ITU: International Telecommunication Union (En español Unión Internacional de las Telecomunicaciones)

Jitter: Variación de la latencia

JSON: *JavaScript Object Notation* es un formato de objetos de javascript usado como estándar abierto para el envío de datos en servicios web.

LAN: Red de área local.

Métrica*:* Unidad de medida de carácter cuantitativo, deben ser concretas y bien delimitadas y se obtienen a través un método acordado o específico.

Paquete: Unidad básica de transporte de información en redes IP

Pérdida de paquetes: No recepción, con respecto al destinatario del paquete, de un paquete en un tiempo razonable

Latencia: Tiempo que toma un paquete en viajar de un origen a su destino

Linux: Sistema operativo abierto, basado en Unix.

QoS: Conjunto de tecnologías y técnicas aplicadas a las redes de datos, cuyo objetivo es intentar garantizar el buen desempeño de la red para ofrecer servicios

RAD: El *Rapid Application Development* o Desarrollo rápido de aplicaciones es un proceso de creación de software.

REST: Es una técnica de arquitectura de software para crear servicios web escalables.

Router: Una entidad capaz de recibir paquetes IP y encaminarlos hacia su destino.

RTP: Es un protocolo para la transmisión de datos en tiempo real y adoptado como estándar en internet desarrollado por la IETF.

Software libre: Software que se puede copiar, estudiar, redistribuir y modificar sin restricciones.

Servicio Web: Son sistemas de software diseñados para soportar una interacción máquina a máquina sobre una red de datos, se compone de protocolos y estándares usados para intercambiar datos entre aplicaciones.

TCP: *Transmission Control Protocol,* protocolo de transporte sobre IP orientado a conexión

TDD: *Test-Driven Development*, es una metodología RAD que se orienta en pruebas continuas que depuran fallos o errores en cada una de las etapas.

Timestamp: Secuencia de caracteres que denotan el tiempo en que ocurrió un evento.

UDP: *User Datagram Protocol*, protocolo de transporte sobre IP.

Unix: Es un sistema operativo multiusuario y multitarea desarrollado en 1969 por AT&T y ahora propiedad de Novell.

Upstream: Ancho de banda disponible para el envío de datos

VoIP: Voz sobre IP es una tecnología que permite la transmisión de voz en tiempo real utilizando una red de datos conmutada por paquetes.

WAN: Es una red de telecomunicaciones de área extendida en lugares de ámbito regional, nacional o internacional.

ANEXO A. DIAGRAMA DE GANTT

Nº	ACTIVIDADES	SEMANA																										
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
1	Definición de requerimientos y funcionalidades del software																											
2	Diseñar un diagrama de arquitectura																											
3	Diseño de la interfaz gráfica																											
4	Relacionar los elementos de la vista con las funcionalidades del software																											
5	Codificar archivos de audio mp3 a G711																											
6	Manipular paquetes RTP con el módulo Scapy.																											
7	Definir las métricas de QoS																											
8	Definir herramientas de software para aplicar las métricas																											
9	Emular condiciones latencia, jitter, perdida de paquetes y ancho de banda																											
10	Integrar herramientas con *Backend*																											
11	Realizar las pruebas de funcionalidad del *Backend* con las herramientas																											
12	Integración del *Backend* y *Frontend*																											
13	Desplegar resultados estadísticos de la evaluación de métricas QoS																											
14	Definir vista de los resultados																											
15	Depuración de errores y validaciones																											
16	Prueba final de integración y conexión cliente-servidor																											
17	Elaboración del manual de uso de la aplicación																											

ANEXO B. MANUAL DE INSTALACIÓN IQOS

A continuación se describen el conjunto de comandos necesarios para realizar la instalación del software iQos en sistemas GNU/LINUX.

1. Extracción de archivos comprimidos

iQos se distribuye como un solo archivo comprimido en formato de extensión "*tar.gz"*, este contiene el código de la aplicación y todos los componentes. Primero se debe descomprimir el archivo *"iQos.tar.gz"*

```
$tar zxvf iqos.tar.gz
```

2. Instalación de dependencias

- **Arquitecturas i386 y x64**

```
$sudo apt-get update
$sudo apt-get install -y zlib1g-dev libreadline-dev libssl-dev lame sox libsox-fmt-mp3 libnet1 libnet1-dev libpcap0.8 libpcap0.8-dev gstreamer0.10-* wondershaper tshark iperf rename libttspico0 libttspico-utils libttspico-data mp3wrap
```

- **Arquitectura ARM**

```
$sudo apt-get update
$sudo apt-get install -y zlib1g-dev libreadline-dev libssl-dev
tcpdump iptables libpopt-dev lame sox libsox-fmt-mp3 libnet1 libnet1-dev
libpcap0.8 libpcap0.8-dev gstreamer0.10-* wondershaper tshark iperf rename
mp3wrap libttspico0 libttspico-utils libttspico-data
$sudo dpkg --install {path-iqos}/bin/pico2wave.deb
```

3. Instalación de Python 2.7.6

```
$wget https://www.python.org/ftp/python/2.7.6/Python-2.7.6.tar.xz
$tar Jxvf Python-2.7.6.tar.xz
$cd Python-2.7.6
```

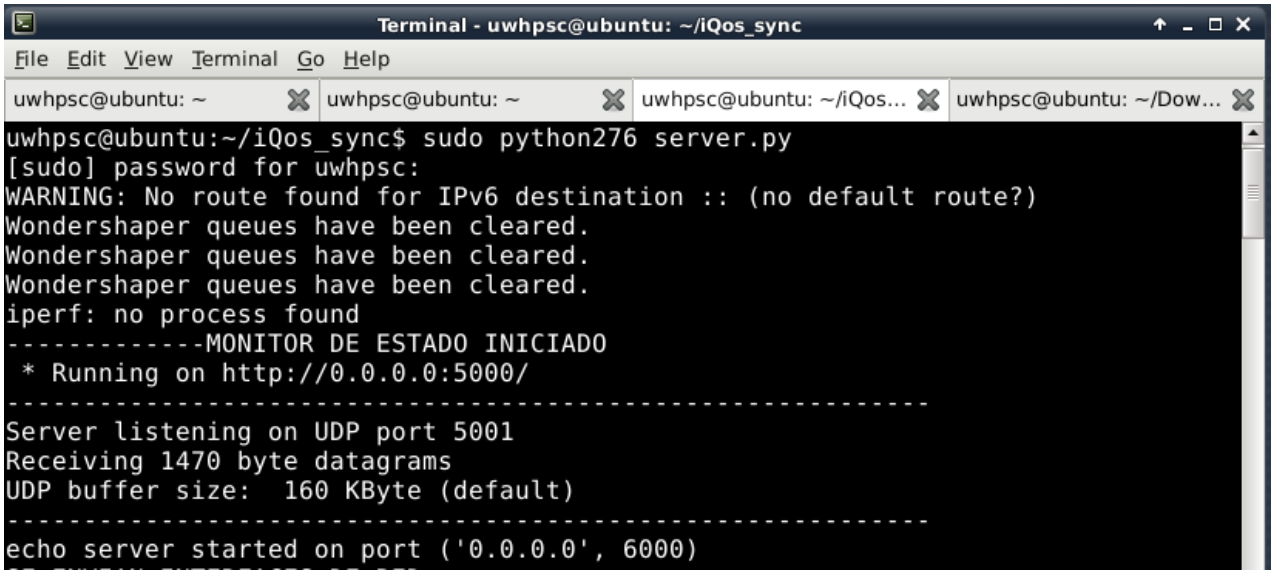
```
$./configure
$sudo make
$sudo make install
$sudo ln -s $PWD/python /usr/bin/pythonero
```

4. Instalación del gestor de paquetes PIP y módulos de Python

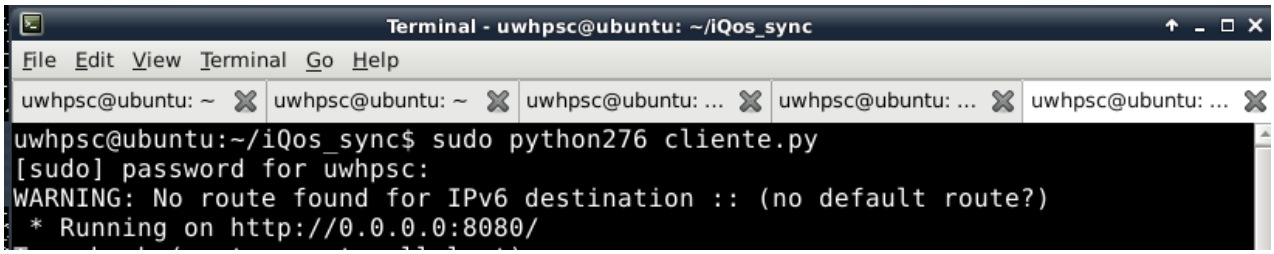
```
$wget https://bootstrap.pypa.io/get-pip.py
$sudo pythonero get-pip.py
$sudo ./pip install socketIO_client netifaces scapy flask flask-socketio shapy flask-cors
```

ANEXO C. EJECUCIÓN DE iQos

Después de instalada la aplicación basta con que el usuario escoja los roles de cliente y servidor, teniendo en cuenta que iQos funciona bajo este tipo de arquitectura, por lo tanto se tiene que ejecutar en cada host el respectivo archivo según el rol. Para el nodo que actúa como servidor, se debe ir a la carpeta del proyecto y abrir una terminal, escribir el comando *"sudo python276 server.py"* como se detalla a continuación:



Esto brindará los permisos de "superusuario" necesarios para abrir los sockets y garantizar el funcionamiento correcto del software. Para el nodo cliente se abre la carpeta en la ruta del proyecto y se escribe en una terminal de *comandos "sudo python276 cliente.py"* como se aprecia a continuación:



Con lo anterior iQos debería estar instalado y ejecutando en los dos nodos seleccionados.

ANEXO D. DOCUMENTO DE DEFINICIÓN DE PRUEBAS

Este proyecto se sustenta en una metodología de desarrollo ágil (RAD) llamada desarrollo dirigido por pruebas (TDD), la cual no contempla las primeras etapas del ciclo de vida de desarrollo de software, queriendo decir que no se tendrá una inicialización de proyecto y documentación formal respecto al análisis de requerimientos del software.

Glosario

Con el fin de integrar al lector con el presente documento, se definen los siguientes conceptos que serán tratados durante el escrito.

TDD: Desarrollo dirigido por pruebas.
PRF: Prueba de requerimiento funcional.
RNF: Requerimiento no funcional.
BPMN: Notación para el modelado de procesos de negocio.
VOIP: Voz sobre IP.
RTP: Protocolo de transmisión de datos en tiempo real.
JSON: Formato ligero para el intercambio de datos.

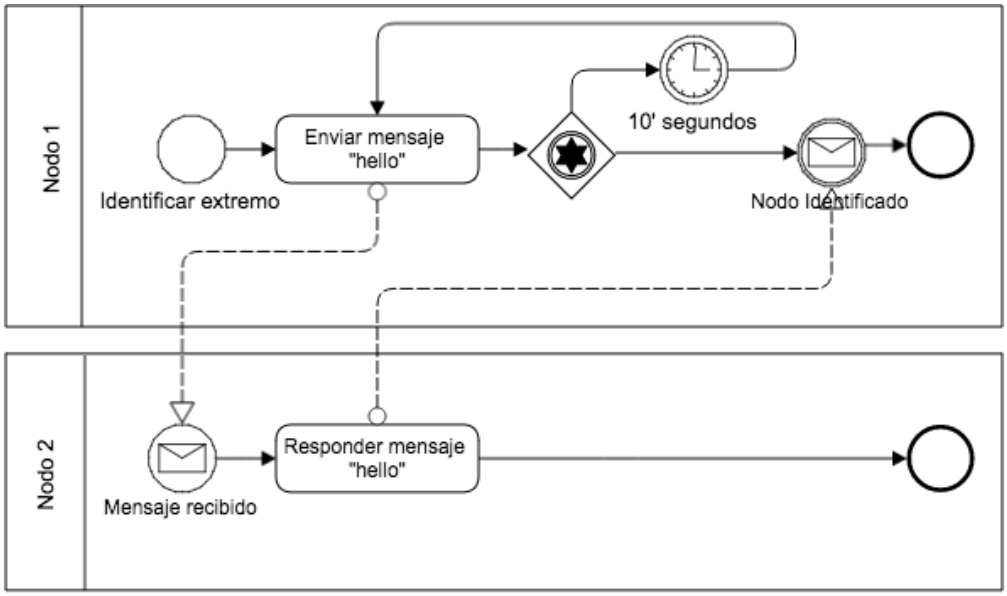
Introducción

En este documento se describe las funcionalidades del software con sus respectivos casos de prueba delimitando el alcance de sus funciones por medio de salidas que el usuario espera obtener a partir de entradas justas o erróneas, para así poder ajustar y re factorizar el código permitiendo mejorar su calidad y tener entregas en poco tiempo.

Pruebas de requerimientos funcionales

ID	PRF1
Nombre	Identificar el estado de extremos
Descripción	El software será capaz de identificar los nodos que van participar en la prueba con el fin de conocer su disponibilidad.

BPMN



Pruebas (TDD)

Entrada	Salida

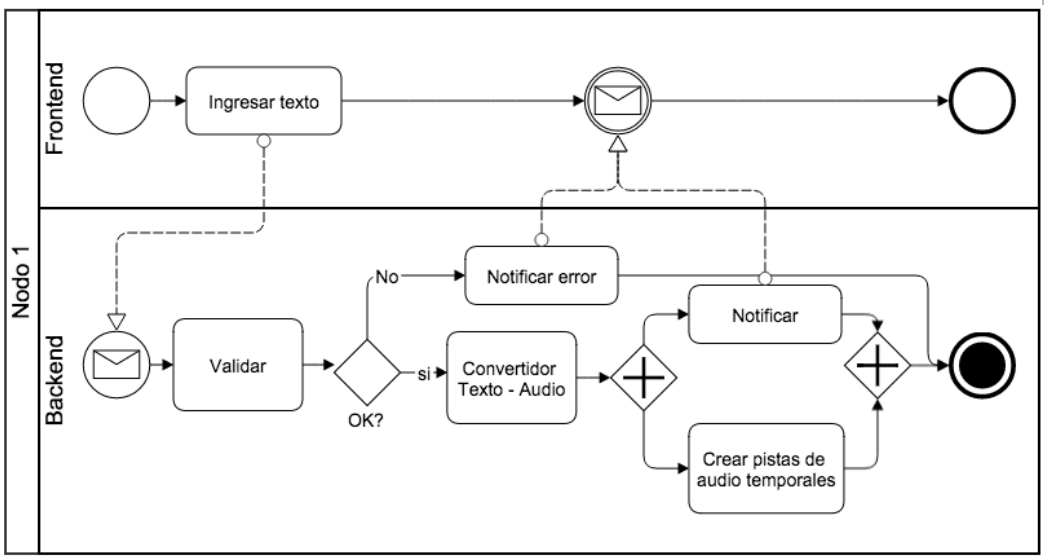
{
id : ‘nodo123’,
ip : ‘127.0.0.1’,
interface : ‘en1’
}
	{
	estado: ‘on’,
	nodoExtremo: {
	id : ‘nodo123’,
	ip : ‘127.0.0.1’
	},
	info:{
	estado: ‘error’,
	data:’Prueba localhost’
	}
	}

{
id : undefined,
ip : undefined,
interface : undefined
}
	{
	estado: ‘off’,
	info:{
	estado: ‘error’,
	data:’no detectado’
	}
	}

{
id : ‘nodo002’,
ip : ‘192.168.2.2’,
interface : ‘eth0’
}
	{
	estado: ‘on’,
	nodoExtremo: {
	id : ‘nodo123’,
	ip : ‘192.168.2.2’
	},
	info:{
	estado: ‘ok’,
	data:’detectado’
	}
	}

{	{

	id : 2134,
	ip : '192.168.2.1',
	interface : 'eth0'
	}
		estado: 'on',
		info:{
		estado: 'error',
		data:'id_invalido'
		}
		}

ID	PRF2
Nombre	Convertir texto en audios
Descripción	El usuario podrá ingresar textos que serán convertidos en pistas de audio para su posterior uso (PRF3).
BPMN	
Prueba (TDD)	

Entrada	Salida

{
texto: 'Prueba de sonido número: ',
tipo: 'server',
increMax:3
}
	{
	info:{
	estado: 'ok',
	data:'Audio creado'
	},
	path:['tmp_01.mp3',
	'tmp_02.mp3'
	'tmp_03.mp3'
	]
	}

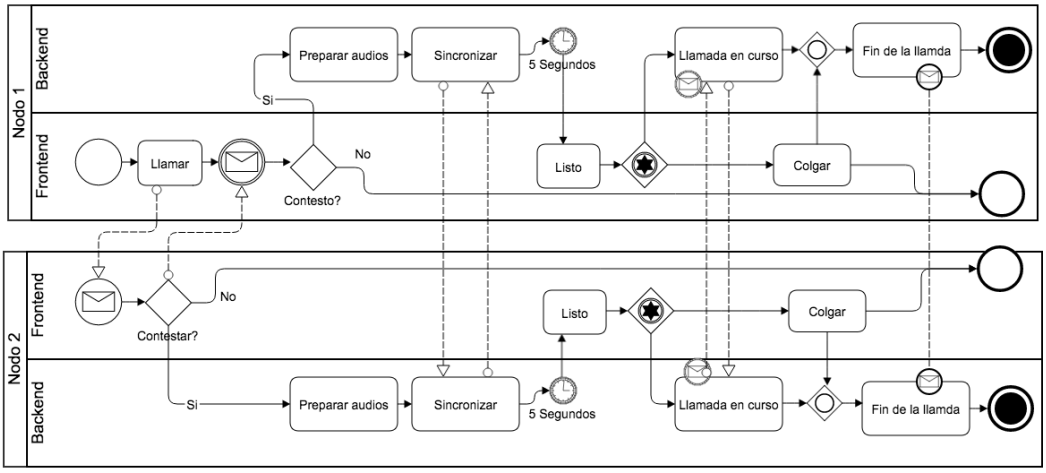
{
texto: 'Hola bin müde',
tipo: 'client',
increMax:1
	{
	info:{
	estado: 'ok',
	data:'Audio creado'

}	}, path:[‘tmp_01.mp3’] }

{
texto: ‘Libertad’,
tipo: ‘’,
increMax:1
}
	{
	info:{
	estado: ‘error’,
	data:’Error en el tipo’
	},
	path:[]
	}

{
texto: null,
tipo: ‘server’,
increMax:3
}
	{
	info:{
	estado: ‘ok’,
	data:’Texto predeterminado’
	},
	path:[‘tmp_01.mp3’,
	‘tmp_02.mp3’,
	‘tmp_03.mp3’]
	}

{
texto: ‘Prueba de sonido’,
tipo: ‘client’,
increMax:0
}
	{
	info:{
	estado: ‘error’,
	data:’Número de audios igual o menor a cero’
	},
	path:[]
	}

ID	PRF3
Nombre	Recrear llamada VoIP
Descripción	El software permitirá recrear llamadas de VoIP utilizando archivos mp3 generados en (PRF2).
BPMN	

Prueba (TDD)

Entrada	Salida

{
bufferRTP:[],
estado: ‘timbrar’,
idOrigen:’nodo001’
}
	{
	info:{
	estado: ‘ok’,
	data:’timbrando’
	},
	idOrigen:’nodo002’
	}

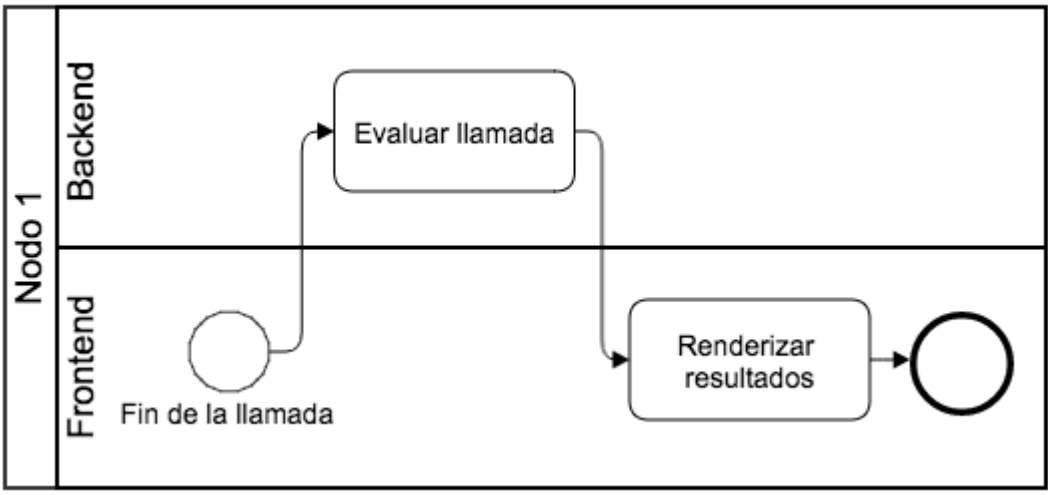
{
bufferRTP:[],
estado: ‘sincronizando’,
idOrigen:’nodo001’,
currenTime:21:12:43
}
	{
	info:{
	estado: ‘ok’,
	data:’sincronizando’
	},
	idOrigen:’nodo002’,
	currenTime_D:21:12:43
	}

{
bufferRTP:[],
estado: ‘hablando’,
idOrigen:’nodo001’,
timeOut:12:34:00
}
	{
	info:{
	estado: ‘error’,
	data:’Error buffer index -1’
	},
	idOrigen:’nodo001’
	}

{
bufferRTP:[A3FF..0011],
estado: ‘hablando’,
idOrigen:’nodo001’,
timeOut:12:34:00
}
	{
	info:{
	estado: ‘ok’,
	data:’hablando’
	},
	bufferRTP_in:[A3FF..0011],
	stream://nodo001.mp3
	timeIn:12:34:10
	}

{
bufferRTP:[A3FF..0011],
estado: ‘hablando’,
idOrigen:’nodo001’,
timeOut:12:34:00
}
	{
	info:{
	estado: ‘error’,
	data:’Error tiempos no sincronizados’
	},
	bufferRTP_in:[A3FF..0011],
	timeIn:11:34:10
	}

{
bufferRTP:[],
estado: ‘colgar’,
idOrigen:’nodo001’
}
	{
	info:{
	estado: ‘ok’,
	data:’colgado’
	},
	idOrigen:’nodo001’
	}

ID	PRF4
Nombre	Desplegar resultados estadísticos
Descripción	Al terminar la llamada el software permitirá mostrar los resultados estadísticos en términos de pérdida de paquetes, latencia y Jitter por medio de diagramas como polígonos de frecuencia y tortas.
BPMN	

Prueba (TDD)

Entrada	Salida

{
duracion:01:22:91,
ArrayBuffer[
{
time:00:11:32,
data:[AFF00..]
},
{
time:00:11:33,
data:[CFB61..]
}
],
evaluar:'perdida',
tipo:'torta'
}
	{
	info:{
	estado: 'ok',
	data:'torta paquetes perdidos'
	},
	loss:10%,
	ok:90%
	}

{
duracion:01:22:91,
ArrayBuffer[],
evaluar:'perdida',
tipo:'poligono'
}
	{
	info:{
	estado: 'error',
	data:'buffer data not found '
	}
	}

{
duracion:01:22:91,
ArrayBuffer[
	{
	info:{
	estado: 'ok',

{
time:00:11:32,
data:[AFF00..]
},
{
time:00:11:33,
data:[CFB61..]
}
],
evaluar:'perdida',
tipo:'poligono'
}
	data:'poligono latencia'
	},
	salida:{
	[{time:00:11:32,estado:0},
	{time:00:11:33,estado:1},
	]
	}
	}

{
duracion:null,
ArrayBuffer[],
evaluar:null,
tipo:null
}
	{
	info:{
	estado: 'error',
	data:'error general '
	}
	}

{
duracion:01:22:91,
ArrayBuffer[
{
time:00:11:32,
data:[AFF00..]
},
{
time:00:11:33,
data:[CFB61..]
}
],
evaluar:'jitter',
tipo:null
}
	{
	info:{
	estado: 'error',
	data:'No se especificó el tipo gráfico'
	}
	}

Requerimientos no funcionales

ID	Título	Descripción
RNF1	Diseño de una interfaz intuitiva	El software contará con una interfaz amigable y de fácil uso para el usuario.
RNF2	Evaluación preliminar	Se podrá conocer el estado actual del enlace entre los dos nodos.
RNF3	Calibrar Enlace	Se podrá emular un ambiente real con características específicas en el canal de comunicación.