PROYECTO FIN DE CARRERA

VoIP TESTER

INGENIERO DE TELECOMUNICACIÓN

Antonio Sánchez Navarro

CURSO 2010/2011

REALIZADOR POR:

Antonio Sánchez Navarro

DIRIGIDO POR:

Juan Manuel López Soler

Jorge Navarro Ortiz

DEPARTAMENTO:

Teoría de la señal, Telemática y Comunicaciones

Granada, Diciembre de 2010

D. Juan Manuel López Soler, Profesor y D. Jorge Navarro Ortiz, Titulares de Ingeniería de Telecomunicación del departamento de Teoría de la Señal, Telemática y Comunicaciones de la Universidad de Granada, como directores del Proyecto Fin de Carrera de Antonio Sánchez Navarro

Informan:

que el presente trabajo, titulado: “VoIPTester”

Ha sido realizado y redactado por el mencionado alumno bajo nuestra dirección, y con esta fecha autorizamos a su presentación.

Granada, a 10 de Diciembre de 2010

Fdo:

Los abajo firmantes autorizan a que la presente copia de Proyecto Fin de Carrera se ubique en la Biblioteca del Centro y/o departamento para ser libremente consultada por las personas que lo deseen.

Granada, a Diciembre de 2010

Antonio Sánchez Navarro DNI:

Juan Manuel López Soler DNI:

Jorge Navarro Ortiz DNI:

Granada, Diciembre de 2010

AGRADECIMIENTOS

En primer lugar quiero agradecer a mis padres su constante interés y apoyo, con más ganas que yo de que este proyecto saliera adelante.

En segundo lugar a mi hermano, por orientarme siempre que me perdía.

A Alba por aguantarme todas las mañanas en la biblioteca haciendo que las horas de trabajo pasen más rápido.

A Marta por su terapia semanal y por su mitones para que trabaje más a gusto.

A Nacho por sacarme de la cueva de vez en cuando para despejarme.

A Javier por su interés y gran ayuda con *GStreamer*.

A Juanjo por sus buenas aportaciones y dejarse la “pasta” en los documentos técnicos.

Y por supuesto no puedo olvidarme de Juanma, por su gestión, paciencia y trato cercano, que ha hecho muy llevadera y amena la realización de este proyecto.

A todos, Gracias.

Índice general

Índice de figuras

Índice de cuadros

Nomenclatura

ARPANET Advanced Research Projects Agency Network

CSV Comma-Separated Values

DDS Data Distribution Service

DNS Domain Name System

FTP File Transfer Protocol

HTTP Hypertext Transfer Protocol

ICMP Internet Control Message Protocol

IEPM Internet End-to-end Performance Monitoring

IETF Internet Engineering Task Force

iLBC Internet Low Bitrate Codec

IP Internet Protocol

ITU International Telecommunication Union

ITU-T Telecommunication Standardization Sector of ITU

JVM Java Virtual Machine

M-Lab Measurement Lab

MOS Mean Opinion Score

NTP Network Time Protocol

OMG Object Management Group

P2P Peer-to-Peer

PCM Pulse Code Modulation

PESQ Perceptual Evaluation of Speech Quality

PEVQ Perceptual Evaluation of Video Quality

QoS Quality of Service

RFC Request For Comments

RMI Remote Method Invocation

RTP Real Time Protocol

RTT Round Trip Time

SAP Session Announcement Protocol

SDP Session Description Protocol

SIP Session Initiation Protocol

SLA Service Level Agreement

SNTP Simple NTP

ST Stream Protocol

TCP Transmission Control Protocol

UDP User Datagram Protocol

UML Unified Modeling Language

URL Uniform Resource Locator

UTC Universal Time Coordinated

VoIP Voice over IP

WAV WAVeform audio file format

1. INTRODUCCIÓN

1.1. CONTEXTO Y DEFINICIÓN DEL PROBLEMA

Permitir la comunicación interpersonal en cualquier momento y desde cualquier lugar ha sido desde siempre uno de los principales objetivos de cualquier sistema de telecomunicaciones. Sin embargo con la aparición de nuevas tecnologías se han añadido otras necesidades, relacionadas con la interactividad entre personas y sistemas, y con el entretenimiento. Es por ello que las redes y servicios multimedia han cobrado gran relevancia en el sector de las telecomunicaciones.

En términos generales, las transmisiones de datos[1] a larga distancia se llevan a cabo a través de una red de nodos intermedios de conmutación. Una de las características de los nodos de conmutación es que el contenido de los datos no les incumbe; en su lugar, su objetivo es proporcionar el servicio de conmutación que traslade los datos de un nodo a otro, hasta alcanzar el destino final [1]. En concreto existen dos técnicas de conmutación:

1. **Conmutación de circuitos**: implica la existencia de un camino dedicado para el intercambio de información entre dos estaciones, formado por una secuencia de enlaces conectados entre los nodos de la red. La comunicación vía conmutación de circuitos requiere del establecimiento previo de la conexión, y del cierre de la misma tras el intercambio de datos.
2. **Conmutación de paquetes**: implica la división de la información en unidades llamadas paquetes, de tamaño definido y que contiene información de control adicional que necesita la red para enviar el paquete a través de ella y alcanzar el destino deseado. En cada nodo, el paquete se recibe, se almacena temporalmente y se envía al siguiente nodo. Sin embargo no existe un camino dedicado entre origen y destino, por lo que los paquetes pueden viajar por caminos diferentes.

La conmutación de circuitos puede llegar a ser bastante ineficiente, ya que la capacidad del canal se reserva permanentemente durante toda la duración de la conexión, incluso en el caso de que no se transfieran datos. Esta técnica fue originalmente diseñada para el tráfico de voz[2], siendo aún hoy día la responsable de la mayor parte del tráfico en estas redes.

Sin embargo, en una conexión de datos usuario/computador (por ejemplo, un usuario conectado a una base datos) el canal se encuentra desocupado la mayor parte del tiempo. Por tanto la técnica de conmutación de circuitos resulta ineficaz para este tipo de comunicaciones, y por ello la conmutación de paquetes es la alternativa más adecuada.

Dado que los datos transmitidos a través de una red de conmutación de paquetes pueden comprender cualquier tipo de información, en principio no existiría ningún inconveniente en

[1] Aquí se usa el término en un sentido muy general, para incluir voz, imágenes y vídeo, así como datos ordinarios (números y textos)

[2] Con voz se especifica el subtipo de audio con el que VoIPTester trabaja. Sin embargo a lo largo del proyecto este término se sustituirá por audio cuando no sea necesario diferenciar respecto al tipo de datos de VoIPTester.

establecer conexiones de voz vía conmutación de paquetes. Sin embargo la pregunta esencial es si, bajo qué circunstancias, el transporte mediante paquetes podrá soportar adecuadamente servicios de voz, como multiconferencia, que realizan intercambios de señales de voz con requisitos de tiempo real.

La posibilidad de crear tales servicios de voz mediante conmutación de paquetes ofrece tanto una oportunidad como un problema para su desarrollo. Por un lado, el desarrollo de un servicio de transporte de voz crea la oportunidad de unificar los servicios de voz y de datos en el mismo entorno, lo que permitiría explotar al máximo las características de las redes de conmutación de paquetes. Por otro lado, el problema principal es que no está clarificado si los servicios de voz que se desarrollen serán satisfactorios.

Para resolver este problema, los administradores de servicios de comunicaciones deben de ser capaces de evaluar las características operacionales de las redes de paquetes y determinar cómo los usuarios perciben la calidad de los servicios de voz considerados. De esta manera la provisión de servicios puede realizarse con un grado de calidad de servicio (QoS).

En este punto es importante entender de forma no ambigua el significado de QoS. Comúnmente, el concepto de calidad es entendido como “algo” que juzga cómo de bueno es un servicio. Sin embargo, en el contexto de las telecomunicaciones, la calidad se entiende con pluralidad como aquellos factores que determinan la valoración de un servicio, de acuerdo a una serie de atributos inherentes al mismo [2]. De este modo, en términos prácticos medir la calidad de servicio comprende realizar un conjunto de medidas de parámetros objetivos y subjetivos.

Otra implicación en la noción de calidad es que puede expresarse desde varios puntos de vista. La figura 1.1 muestra las posibles distinciones. Existen tres tipos, pero relacionados, de QoS, atendiendo a los factores que determinan si un cliente usará un servicio de telecomunicaciones:

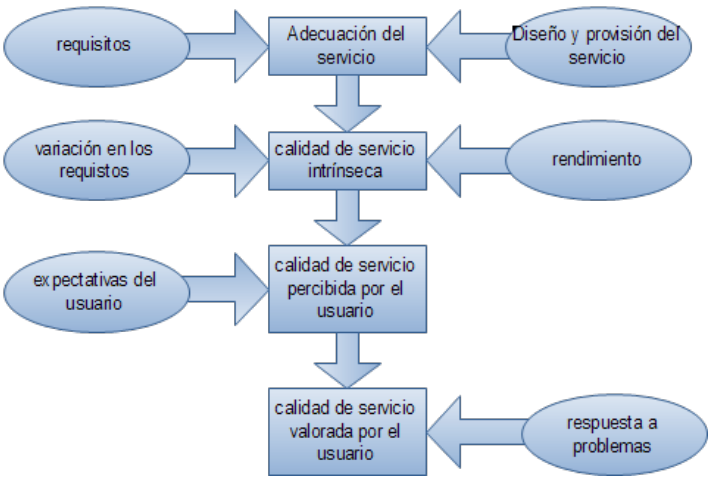


Figura 1.1.: Puntos de vista de QoS

1. **QoS intrínseca**: es la resultante tras evaluar mediante medidas el rendimiento del sistema implementado. Condicionada por la arquitectura y elementos de la red.
2. **QoS percibida**: es la resultante cuando el servicio se está utilizando, al tiempo que los usuarios experimentan los efectos de la QoS intrínseca y reaccionan ante los eventos que se producen en la comunicación. Condicionada por la experiencia del usuario.
3. **QoS valorada**: es la resultante cuando el cliente toma la determinación acerca de si la

calidad de servicio es lo suficientemente buena como para garantizar la continuidad del uso del servicio. Condicionada por la QoS percibida, pero añadiendo otros factores como la respuesta del proveedor de servicios ante problemas.

El presente proyecto expone el desarrollo y evaluación de un sistema evaluador de QoS. En concreto, el sistema permitirá la evaluación de QoS intrínseca y percibida, mediante la estimación de parámetros de calidad de servicio objetivos y subjetivos para un enlace de comunicaciones de una red de conmutación de paquetes.

1.2. MOTIVACIÓN

Cada servicio de telecomunicaciones impone sus restricciones para el correcto funcionamiento, y que varían de un servicio a otro. Supóngase que desea realizar varias consultas a una base datos. En principio no habría problema alguno en que el tiempo de respuesta de cada consulta fuese diferente, o bien que alguna respuesta se retrase demasiado. Esto, por contra, no es admisible en una comunicación por voz, en la que los datos que representan las señales de voz deben de entregarse al destino dentro de un límite temporal, y con la menor variación posible.

En definitiva, los servicios multimedia imponen unas restricciones de tiempo real, y que es necesario que se satisfagan para que la comunicación sea viable. Debido a la naturaleza de las redes de conmutación de paquetes descritas en la sección previa, este requisito no está asegurado.

Por tanto en el diseño de una red de conmutación de paquetes es necesario realizar una adecuación del servicio y obtener el nivel de QoS intrínseca que asegura una determinada QoS percibida. Este análisis se ha convertido en un requisito fundamental a la hora de desplegar servicios de voz, y en general servicios de telecomunicaciones, por parte de empresas proveedoras cuyo éxito y prestigio dependen de la calidad del servicio ofrecido.

Es por este motivo el interés de disponer de herramientas de monitorización de redes que ayuden a cumplir este requisito. Prueba de ello es la vasta cantidad de herramientas disponibles en la actualidad, tal y como se describe en la sección 2.2. Sin embargo el margen de análisis y evaluación de las redes es tan amplio que cada herramienta propone una solución distinta a las demás en cuanto a metodología de medida y funcionalidades ofrecidas.

Por tanto aportar nuevas herramientas contribuye a enriquecer el campo de evaluación y provisión de QoS, esencial en los servicios de telecomunicaciones que deben cumplir los contratos de servicios establecidos con los clientes: *Service Level Agreement* (SLA) [3].

1.3. OBJETIVOS

El principal objetivo es el desarrollo de una herramienta de evaluación de QoS en entornos *Voice Over IP* (VoIP). Se persigue demostrar la viabilidad del desarrollo de aplicaciones que estimen tanto QoS intrínseca como percibida, evaluando el producto final en aspectos como funcionalidad, facilidad de uso y reutilización.

Se pretende proporcionar una solución adaptable cuyo uso pueda encajar tanto en dominios de investigación como corporativos, aprovechando las características de configuración que ofrece

la herramienta.

Asimismo, es objetivo del proyecto que la presente memoria pueda servir como guía de referencia para el desarrollo posterior de aplicaciones similares sobre otros entornos, por ejemplo, sobre *middleware*. Para ello, a lo largo del desarrollo del proyecto se separarán los conceptos de estimación y evaluación de QoS respecto de la tecnología de red considerada.

1.4. RECURSOS

El desarrollo y explotación del presente proyecto no requiere recursos especiales. A continuación se presentan los recursos humanos, *hardware* y *software* que han sido dedicados durante su desarrollo.

Humanos

- D. Juan Manuel López Soler y D. Jorge Navarro Ortiz, profesores del Departamento de Teoría de la Señal, Telemática y Comunicaciones de la Universidad de Granada, como tutores del proyecto.
- Antonio Sánchez Navarro, alumno de la Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación de la Universidad de Granada, que ha sido el encargado de desarrollar el proyecto con la supervisión e indicaciones de los tutores.

Hardware

Como se indicó anteriormente, para la realización y explotación del proyecto no es necesario emplear *hardware* especializado. Concretamente, se ha utilizado:

- Ordenador portátil
- Diversos ordenadores personales para pruebas de funcionamiento y evaluación
- Enrutadores y conmutadores para la implementación de la red de pruebas

Software

El proyecto ha sido implementado mediante el lenguaje de programación Java [4], modelando el sistema mediante el estándar *Unified Modeling Language* (UML) [5] del *Object Management Group* (OMG) [6]. La elección de Java se debe, principalmente, a la amplia API que dispone para el desarrollo de cualquier sistema en general, y en cuanto a funcionalidades para sistemas distribuidos en particular. Además, debido a la tecnología inherente a este lenguaje, las aplicaciones desarrolladas pueden ejecutarse en cualquier sistema operativo.

Además han sido necesarios los siguientes recursos:

- Sistema operativo Windows XP y Windows 7
- Sistema operativo GNU/Linux Ubuntu 9.10
- Paquetes de ofimática
- Entorno de desarrollo Netbeans [7]
- Marco de trabajo de audio *GStreamer* [8]

1.5. FASES DE DESARROLLO

El desarrollo del proyecto se dividirá en distintas fases, organizadas de forma secuencial en el tiempo:

1. **Revisión del estado del arte**. En esta fase se realizará un estudio de la evolución de las redes multimedia, sus implementaciones, así como una caracterización de la funcionalidad de diferentes sistemas de monitorización de red existentes.
2. **Especificación y diseño de la plataforma**. Durante esta fase del proyecto se caracterizarán las funcionalidades que cabe esperar de la aplicación, materializadas en la especificación de la misma. El objetivo será integrar y proveer el conjunto de servicios y herramientas especificados de una forma compacta y unificada.
3. **Implementación**. Una vez concluida la especificación y el diseño, se implementará el código necesario para demostrar la viabilidad de la propuesta realizada.
4. **Evaluación**. Finalmente se evaluará el sistema diseñado y su implementación en un entorno real, verificando el cumplimiento de los requisitos especificados.
5. **Documentación**: Concurrente con las etapas anteriores, se genera la documentación pertinente al desarrollo del proyecto.

1.6. APORTACIONES Y LOGROS

Con el presente proyecto se ha conseguido contribuir al área de análisis de QoS en los sistemas de telecomunicaciones con una nueva herramienta. Al tratarse de un sistema de monitorización activo con control de las transmisiones a bajo nivel, es posible utilizarlo para realizar estudios en un amplio conjunto de condiciones y escenarios de medida.

Además, al adoptar una solución modular en la que el modelo de obtención de datos de QoS es independiente al resto de componentes del sistema, ha permitido cumplir con los objetivos de funcionalidad y reutilización. Es más:

- Separar el modelo de obtención de datos de QoS habilita su análisis y validez independientemente la metodología empleada para su utilización en componentes superiores. De esta manera es posible determinar si este modelo será adecuado para un sistema de red en particular sin necesidad de implementación y realización de pruebas
- Como consecuencia de lo anterior, es posible aplicar el modelo de obtención de datos de QoS a diversas arquitecturas de red objetivo, habilitando la realización de evaluaciones de su rendimiento tanto a nivel individual como comparativo.

A raíz de estas implicaciones, la solución aportada por la herramienta se pretender usar y publicar una vez que se realice una completa evaluación de la misma.

1.7. ESTRUCTURA DE LA MEMORIA

Para finalizar este primer capítulo de introducción, se muestra una breve descripción de cada uno de los capítulos en los que se ha estructurado la presente memoria.

Capítulo 2 Se realiza una revisión del estado del arte de los sistemas multimedia y de las herramientas de evaluación de QoS más relevantes.

Capítulo 3 Previo al diseño e implementación del sistema a desarrollar, para garantizar que se satisfacen todos los requerimientos, es necesario realizar un modelado de requisitos. En este capítulo se lleva a cabo dicho proceso.

Capítulo 4 Una vez fijados los requisitos del sistema, se presenta el diseño de la plataforma.

Capítulo 5 El presente proyecto no se ha limitado al diseño del sistema, sino que se ha llevado a cabo la implementación del mismo. En este capítulo se describen algunos detalles de la implementación que se consideran especialmente relevantes.

Capítulo 6 Finalmente se llevará a cabo una evaluación del proyecto realizado, mediante una estimación de costes y comparando si se satisfacen los requisitos y objetivos especificados.

2. ESTADO DEL ARTE

En este capítulo se estudiarán el estado actual y antecedentes de los temas abordados por el proyecto. Concretamente, en la primera sección se hará una breve reseña histórica acerca del origen de las redes multimedia, y en la segunda, se describirán algunas de las herramientas de análisis de calidad en VoIP más relevantes.

2.1. ANTECEDENTES DE LAS REDES MULTIMEDIA

Los primeros experimentos relacionados con el transporte de voz en la redes de paquetes se remonta a la década de los setenta, apareciendo el primer *Request for comments* (RFC) - *Network Voice Protocol* (NVP, RFC 741 [11]) - en 1977. El transporte de vídeo llegó más tarde, aunque la experiencia de *streaming* de audio/vídeo supera ya los diez años.

Primeros experimentos con paquetes de voz

Los primeros desarrolladores de NVP investigaban la transmisión de paquetes de voz sobre la *Advanced Research Projects Agency Network* (ARPANET) [12], la predecesora de Internet. ARPANET proporcionaba un servicio fiable, análogo al actual TCP/IP, pero tenía el inconveniente de introducir un alto retardo extremo a extremo. Por esta razón se desarrolló un servicio de paquetes “no controlados”, similar al actual UDP/IP, sobre el que se apoyó NVP.

Estos primeros experimentos estaban limitados a uno o dos canales (llamadas) de voz simultáneos, debido al bajo ancho de banda existente. Sin embargo en la década de los ochenta se creó la red de banda ancha de satélite a 3 $Mbps$, que habilitó no sólo aumentar el número de canales de voz sino incluir vídeo. Entonces se desarrolló el *Stream Protocol*, ST, que junto con una segunda versión de NVP (NVP-II) permitía ofrecer servicios de multiconferencia en redes de conmutación de paquetes. Más tarde, con la llegada de la red de banda ancha terrestre, se desarrollaría una segunda versión de ST, ST-II (RFC 1819 [13]), que permitía aumentar el número de asistentes en videoconferencias.

Audio y vídeo sobre Internet

ST y ST-II operaban paralelamente con IP, pero sólo tenían aplicación en redes de investigación o gubernamentales. Así que como alternativa, se consideró desplegar servicios de multiconferencia sobre IP, utilizando NVP-II sobre UDP/IP. En un encuentro de la *Internet Engineering Task Force* (IETF) [14] en 1992, se transmitió audio a través de Internet a veinte sitios repartidos en tres continentes mediante túneles *multicast*. En ese mismo encuentro se inició el desarrollo del *Real Time Protocol* (RTP, RFC 1889 [15]) , el protocolo actual utilizado en las redes multimedia y que ha favorecido su crecimiento. RTP no solo incluye facilidades para la entrega de datos multimedia, sino que también soporta gestión de miembros e información sobre la calidad de recepción de datos.

Además de RTP se han desarrollado otros protocolos para coordinar y controlar los *streams*

multimedia:

- *Session Announcement Protocol* (SAP, RFC 2974 [16]): permite avisar de la existencia de sesiones *multicast*, de manera que los equipos que reciben estos anuncios pueden conocer que encuentros y transmisiones se están produciendo en un instante dado.
- *Session Description Protocol* (SDP, RFC 2327 [17]): permite describir los parámetros de un sesión, indicando las direcciones de red, formato de compresión y paquetización, etc., que utilizarán los emisores y receptores. Estos informes se incluyen en los anuncios de sesión proporcionados por SAP.
- *Session Initiation Protocol* (SIP, RFC 3261 [18]): se desarrolló para ofrecer un servicio simple para encontrar participantes e iniciar una sesión *multicast*. Para ello incluye características de control y negociación de la sesión.

Esta breve reseña muestra no sólo la corta vida de las redes multimedia, sino también el continuo crecimiento que siguen teniendo. Las características de tiempo real que requiere el *streaming* multimedia generan la necesidad de implantar nuevos protocolos que ayuden a cumplir esos requisitos.

2.2. SISTEMAS DE MONITORIZACIÓN

El establecimiento una comunicación por voz y/o vídeo demanda una serie de recursos mínimos necesarios, y que sin estos no sería viable dicha comunicación. Por este motivo una gran cantidad de herramientas destinadas a evaluar el estado de los recursos de una red han sido desarrolladas.

El grupo de investigación *Internet End-to-end Performance Monitoring* (IEPM) [19] mantiene una lista con todas las herramientas disponibles hasta el momento. Sin embargo el análisis de redes es muy amplio y las herramientas tienden a especializarse según el área de aplicación. Por ello en esta sección se incluirán aquellas más relevantes relacionadas con la monitorización de redes VoIP.

ACE Live VoIP Monitoring

Aplicación desarrollada por OPNET [20], que permite monitorizar llamadas de voz en tiempo real. Además detecta problemas relacionados con la conexión VoIP, identificando que otras aplicaciones compiten con llamadas VoIP. Ofrece análisis subjetivo de la calidad de voz y muestra distintas métricas ofrecidas por RTP.

PathView

Se trata de una solución de *ApparentNetworks* [21] constituida por *software* y *hardware*. Por un lado una aplicación, *AppView*, se ofrece al cliente través de servidores propios del proveedor, accediendo mediante inicio de sesión. Esta aplicación, que muestra los resultados obtenidos, accede a un dispositivo, *PathView MicroAppliance*, que se conecta en los puntos de la red que se deseen monitorizar. El dispositivo se muestra en la figura 2.1.

El uso conjunto de ambos componentes permite analizar el tráfico entre dos puntos de una red. Es posible medir características como *jitter*, latencia y ancho de banda disponible. La ventaja

Figura 2.1.: Dispositivo *PathView MicroAppliance*

de ésta plataforma es la fácil implantación en las empresas, ya que no requiere de distribución de aplicaciones en todos los equipos que se deseen evaluar.

La herramienta utiliza protocolos estándares (ICMP y/o UDP) para transmitir paquetes a través del enlace configurado. Estos paquetes varían en tamaño y patrón de envío. Identifica los errores que afectan con más probabilidad a las comunicaciones: congestión, excesiva reordenación de paquetes, errores en los datos, etc.

La herramienta proporciona cuatro casos de uso:

- generación sintética de paquetes TCP/IP para realizar análisis sin afectar al rendimiento de la red
- análisis de conexiones VoIP reales que simulan hasta veinticinco llamadas activas
- análisis de conexiones web mediante protocolos *Hipertext Transfer Protocol* (HTTP), *File Transfer Protocol* (FTP) y *Domain Name System* (DNS)
- captura (*sniffing*) de paquetes

NetQoS VoIP Monitor

Es una herramienta de monitorización de llamadas de voz, desarrollada por CA [22], que proporciona un análisis continuo de la calidad subjetiva de una llamada, informando sobre problemas de rendimiento. Forma parte de una aplicación supervisora, *NetQoS Performance Center*, de manera que es posible monitorizar la calidad subjetiva de una llamada VoIP al mismo tiempo que se monitorizan otros parámetros de la red.

VQmon

Herramienta desarrollada por *Telchemy* [23] para medir el rendimiento de las comunicaciones multimedia en redes IP y sistemas de multiplexación por división en tiempo.

VQmon mide la distribución de las pérdidas de paquetes utilizando un modelo de Markov diseñado para mostrar dos principales estados en las llamadas: estado de ráfaga, donde hay suficientes pérdidas como para degradar la comunicación, y estado de *gap*, donde las pérdidas no afectan a la comunicación. La calidad de la llamada se calcula separadamente para cada estado y combinando un modelo perceptual que representa la reacción de las personas ante la variación temporal de la calidad de la llamada.

La herramienta permite comparar codecs caracterizando la tasa de información que generan. Además incorpora medidas de nivel de ruido, pérdidas de retorno y retardo extremo a extremo. En la figura 2.2 se muestra un ejemplo de los resultados que proporciona.

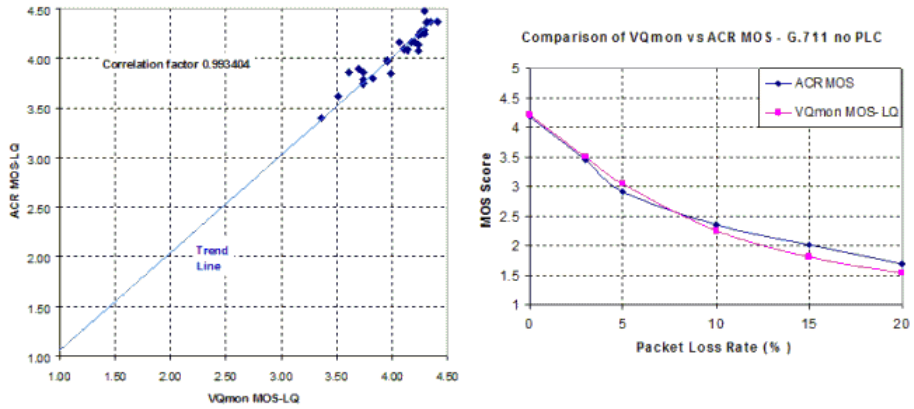


Figura 2.2.: Ejemplo de resultados de *VQmon*

IP Traffic Test & Measure

Herramienta de generación de datos para redes IP, desarrollada por ZTI [24]. Utiliza TCP, UDP o ICMP para realizar intercambios de correos, ficheros y *pings*. Pueden implementarse varias configuraciones de prueba utilizando más de dos computadores. La arquitectura se muestra en la figura 2.3.

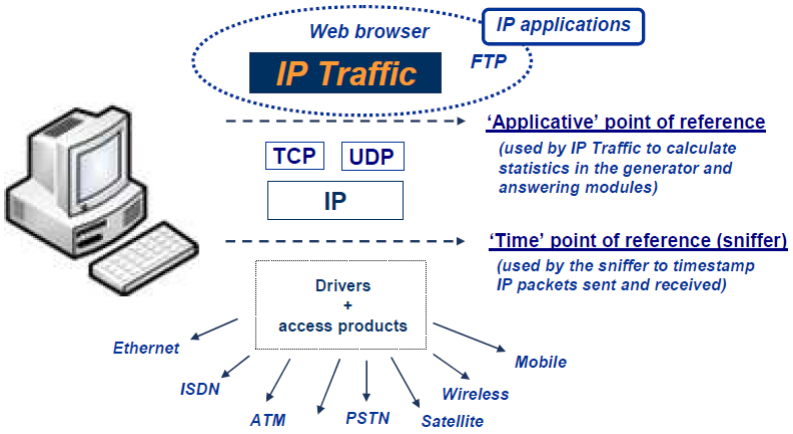


Figura 2.3.: Arquitectura de *IP Traffic Test & Measure*

La herramienta esta formada por cuatro módulos principales:

1. *IP Generator*: encargado de generar el tráfico IP en las conexiones activas
2. *IP Answering*: encargada de recibir el tráfico IP de las conexiones activas
3. *Traffic Sniffer*: captura tráfico a nivel de enlace, permitiendo realizar *timestamps* a los paquetes IP.
4. *Traffic Observer*: herramienta gráfica que muestra las estadísticas obtenidas en tiempo real

MCS My VoIP

Se trata de una herramienta de *Visualware* [25] que simula tráfico de voz sobre una conexión UDP o TCP, configurable por el usuario. Calcula la capacidad del enlace tanto en sentido

ascendente como descendente, realiza una simulación de una comunicación VoIP y obtiene medidas relacionadas con el retardo y las pérdidas. También permite realizar simulaciones de establecimiento de conexión mediante el protocolo SIP.

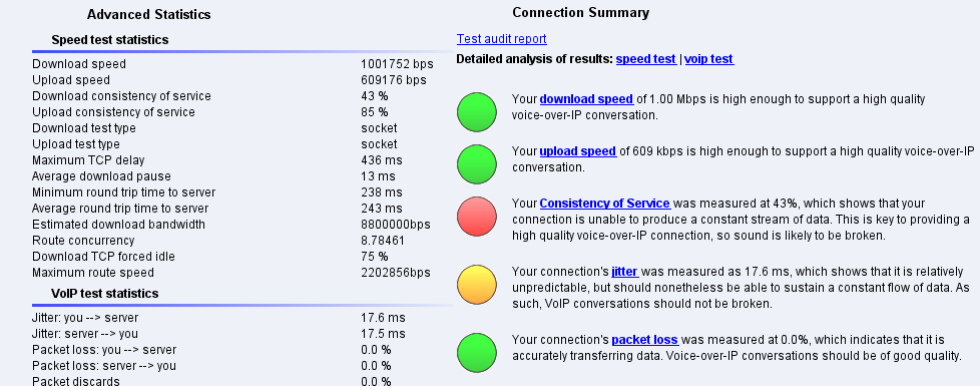


Figura 2.4.: Ejemplo de resultados de *MCS My VoIP*

Los resultados se obtienen dentro de un intervalo de tiempo que la herramienta elige. y los presenta mediante gráficos e informes de texto. También muestra un informe subjetivo acerca de la calidad de voz que se obtendría a partir de las medidas realizadas. En la figura 2.4 se muestra un ejemplo de los resultados de evaluación de una conexión TCP.

M-Lab

Measurement Lab (M-Lab) [26] es una plataforma abierta de servidores dedicados principalmente a los investigadores para que desplieguen sus herramientas de medidas en Internet y cuyo objetivo es potenciar la información disponible al público acerca del estado de sus conexiones. Este tipo de plataformas utilizan un vasto número de servidores de test localizado en situaciones estratégicas a lo largo del mundo. Así cada herramienta alojada tiene recursos dedicados para facilitar medidas precisas.

A día de hoy, M-Lab cuenta con seis herramientas disponibles, ejecutándose en cuarenta y cinco servidores en quince localidades alrededor del mundo. Brevemente, las herramientas actuales:

1. *Network Diagnosis Tool* (NDT): mide el rendimiento de TCP entre un servidor de M-Lab y un equipo cliente ejecutando la herramienta.
2. *Network Path and Application Diagnosis* (NPAD): mide el rendimiento extremo a extremo de TCP y obtiene información de las colas de enrutadores y conmutadores a lo largo de la ruta.
3. *Glasnost*: Emula un cliente *Peer-to-Peer* (P2P) y mide el rendimiento hacia un servidor central
4. *Pathload2*: utiliza UDP para estimar el ancho de banda disponible
5. *ShaperProbe*: Utiliza UDP para detectar si un usuario transmite bajo conformación de tráfico
6. *SideStream*: mide el rendimiento de TCP de manera análoga a NPAD

La intención de implantar M-Lab es que los investigadores utilicen los datos generados por estas herramientas para sus análisis sobre Internet, sin tener que desplegar otros sistemas.

3. ESPECIFICACIÓN DE REQUISITOS

En este capítulo se procede a comentar la especificación de los requisitos del sistema, identificando claramente las necesidades y comportamiento del mismo. Estos requisitos, además, se utilizarán como criterio de evaluación y verificación.

La especificación se va a llevar a cabo en tres secciones. En la primera se define el propósito y las limitaciones del sistema. En la segunda se describe de manera general las características y funciones del sistema junto con los factores influyentes. En la tercera se completa la especificación de manera consistente exponiendo los requisitos más detalladamente.

3.1. PROPÓSITO Y ALCANCE

La solución planteada consiste en una plataforma para automatizar la evaluación de la calidad de voz resultante en un enlace real. La herramienta debe ser máximamente configurable, tal que permita seleccionar el códec, parámetros de transmisión, y sentido de la comunicación, y como resultado debe caracterizar la calidad de VoIP resultante del enlace tanto con medidas objetivas como subjetivas.

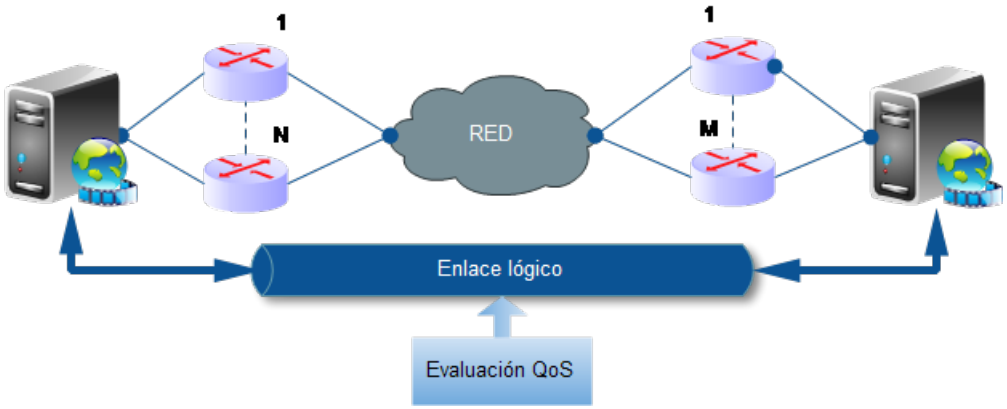


Figura 3.1.: Escenario de evaluación de QoS

Además, la herramienta debe permitir seleccionar las interfaces de red que delimitarán el enlace a evaluar. El escenario de actuación se muestra en la figura 3.1. Se observa como dos equipos pueden estar conectados a una red a través de múltiples interfaces, definiendo en los extremos de la comunicación un enlace lógico que podrá ser evaluado.

La principal meta es determinar la viabilidad para soportar una comunicación por voz en un enlace determinado, utilizando una técnica de codificación (códec) concreta. Lo que el sistema no pretende aportar es una comparación o *benchmark* de las distintas técnicas de codificación de voz existentes.

3.2. DESCRIPCIÓN GLOBAL

La plataforma VoIPTester debe ofrecer una evaluación de la calidad de servicio que sea fiable, es decir, la información que proporcione acerca del estado real del enlace sea válida, representativa y consistente. Para conseguir esto es importante considerar el entorno del sistema y sus funciones.

Perspectiva

La plataforma consistirá en una aplicación *software* que se despliega en equipos de propósito general. Así se deben de tener en cuenta las siguientes restricciones:

1. **Interfaz con el usuario**: será ofrecida mediante una interfaz gráfica compuesta por una ventana principal que contendrá las distintas opciones disponibles. Los distintos eventos se mostrarán al usuario mediante diálogos que contienen una breve descripción del mismo, mientras que las descripciones detalladas se incluirán en un archivo de *log*.
2. **Interfaces con el *hardware***: la aplicación debe interactuar con las interfaces físicas de red, permitiendo la configuración de las direcciones de red que definen los extremos del enlace de comunicación, de acuerdo a la arquitectura de red considerada. La configuración será posible sólo a nivel de red y de transporte.
3. **Interfaces con el *software***: el funcionamiento de la aplicación requiere de un sistema operativo compatible instalado en el equipo actuador. El sistema operativo debe actuar como punto de unión entre la aplicación y los interfaces *hardware* requeridos.
4. **Interfaces de comunicaciones**: la aplicación está enfocada a funcionar en redes con arquitectura TCP/IP y UDP/IP, por lo que los servicios a estos niveles deben ser proporcionados por los interfaces *software* requeridos.
5. **Operaciones**: la aplicación constará de dos modos de operaciones, cada uno aplicable al equipo de cada extremo. En un extremo (equipo servidor), la aplicación funcionará en un modo encargado de proporcionar la comunicación necesaria con el otro extremo (equipo cliente). Este último funcionará en el modo complementario, que comprende no sólo comunicación con el extremo anterior sino la interacción con el usuario y la proporción de los resultados.

Debido al entorno *software* donde la aplicación se implanta, es posible que otras aplicaciones compartan los mismos recursos *hardware*. Por tanto hay que tener en cuenta que la aplicación puede demandar recursos y que estos sean proporcionados con retraso, lo que provocaría una disminución drástica del rendimiento. Por ello se recomienda asegurar que el acceso a los recursos se produzca de manera correcta.

Funciones

1. Gestión de conexiones

    a) Conectar/registrar a un cliente en el servidor. Sólo un cliente registrado puede establecer una comunicación con el servidor y realizar peticiones.

    b) Desconectar a un cliente. Termina la comunicación con el servidor dejándolo accesible a otros clientes.

    c) Establecer extremos del enlace a evaluar. Definir y configurar las interfaces de red

por las que se transmitirá/recibirá información.

2. Configuración de la transmisión de información

   a) Seleccionar los parámetros de transmisión. Será posible generar tráfico con un perfil configurable por el usuario.

   b) Seleccionar la técnica de codificación de voz. La aplicación debe ofrecer al usuario la posibilidad de seleccionar el códec que se aplica a los datos de voz de prueba que se van a transmitir.

   c) Seleccionar el sentido de la comunicación. Debe ser posible evaluar la comunicación en ambos sentidos de la comunicación, es decir, de cliente a servidor y viceversa.

3. Configuración de los criterios de evaluación del enlace. Debe ser posible evaluar:

   a) **Retardo extremo a extremo**: representa el tiempo de tránsito en la red, es decir, el tiempo que transcurre desde que un paquete se envía en un extremo hasta que se recibe en el otro.

   b) ***Jitter***: variación del retardo extremo a extremo entre paquetes. Cada paquete puede viajar por una ruta distinta por lo que puede sufrir un retardo distinto. Esta diferencia de retardo afecta a la comunicación por voz dificultando la escucha.

   c) **Probabilidad de pérdidas**: número medio de paquetes que se pierden. Representa una medida general del estado de congestión de la red.

   d) **Distribución de pérdidas**: muestra bajo que patrón se producen las pérdidas de paquetes. Los sistemas robustos ante pérdidas deben conocer si las pérdidas son aleatorias o se producen en ráfagas.

   e) **Ancho de banda**: tasa de información a la que se transmiten los datos. Está limitada por la capacidad máxima del enlace.

   f) ***Perceptual Evaluation of Speech Quality*** **(PESQ)** : es un método para la evaluación de la calidad vocal de extremo a extremo de redes telefónicas de banda estrecha y codecs vocales. Está definido en la recomendación P.862 de la ITU-T [27]. Proporciona una evaluación subjetiva mediante una *Mean Opinion Score* (MOS).

   g) **Modelo-E**: modelo para evaluar los efectos combinados de las variaciones de diversos parámetros en la transmisión que afectan a la calidad de la conversación. Descrito en la recomendación G.107 de la ITU-T [28]. El resultado del modelo es un valor escalar de determinación de índice de calidad, R, que varía linealmente con la calidad global de la conversación y que representa una evaluación subjetiva.

4. Mostrar los resultados de la evaluación. Los resultados obtenidos deben de presentarse al usuario de una manera sencilla que permita fácilmente su interpretación. Esto se conseguirá mostrando un gráfico de puntos para cada parámetro seleccionado en la configuración de los criterios de evaluación.

Características del usuario

El usuario final de la aplicación debe tener un perfil de conocimientos sobre redes amplio, que incluya habilidades de configuración y mantenimiento de conexiones, así como de comprensión e interpretación de las métricas de evaluación de calidad de servicio ofrecidas por la aplicación.

Suposiciones y dependencias

La aplicación se implementará mediante el lenguaje de programación Java, lo que implica que el equipo subyacente debe de tener instalada la *Java Virtual Machine* (JVM). Por otra parte, la selección de la técnica de codificación de voz requiere de un marco de trabajo multimedia que proporcione la implementación de dichas técnicas. El marco de trabajo se integrará como un módulo de la aplicación, aunque el usuario podrá disponer de él previamente de manera ajena al sistema para otros propósitos, siendo válida su convivencia con la aplicación.

3.3. REQUISITOS ESPECÍFICOS

Esta sección describe los requisitos *software* con el nivel de detalle suficiente que permita diseñar un sistema que los satisfaga, y que facilite el diseño de las pruebas que ratifiquen que el sistema cumple con las necesidades impuestas. Esta dividida en dos secciones, atendiendo a los requisitos funcionales en una, y a los nos funcionales en la otra.

3.3.1. Requisitos funcionales

A continuación se describirán las distintas funciones que deben tener lugar, considerando una jerarquía funcional.

Subsistema de conexión/registro

Es el encargado de gestionar la conexión principal con el servidor y gestionar las operaciones accesibles a los clientes. El diagrama de caso de uso se muestra en la figura 3.2. Los dos actores del sistema son, por un lado `Cliente nodo`, que representa el usuario de la entidad cliente, y por otro, `Servidor de registro`, que representa el usuario de la entidad servidor. El nodo representa una entidad funcional que permite realizar distintas operaciones, tanto en el equipo cliente como en el servidor.

Descripción del caso de uso:

- registrar (cuadro 3.1)
- poseer nodo remoto (cuadro 3.2)
- comprobar actividad (cuadro 3.3)
- desconectar (cuadro 3.4)
- liberar nodo remoto (cuadro 3.5)

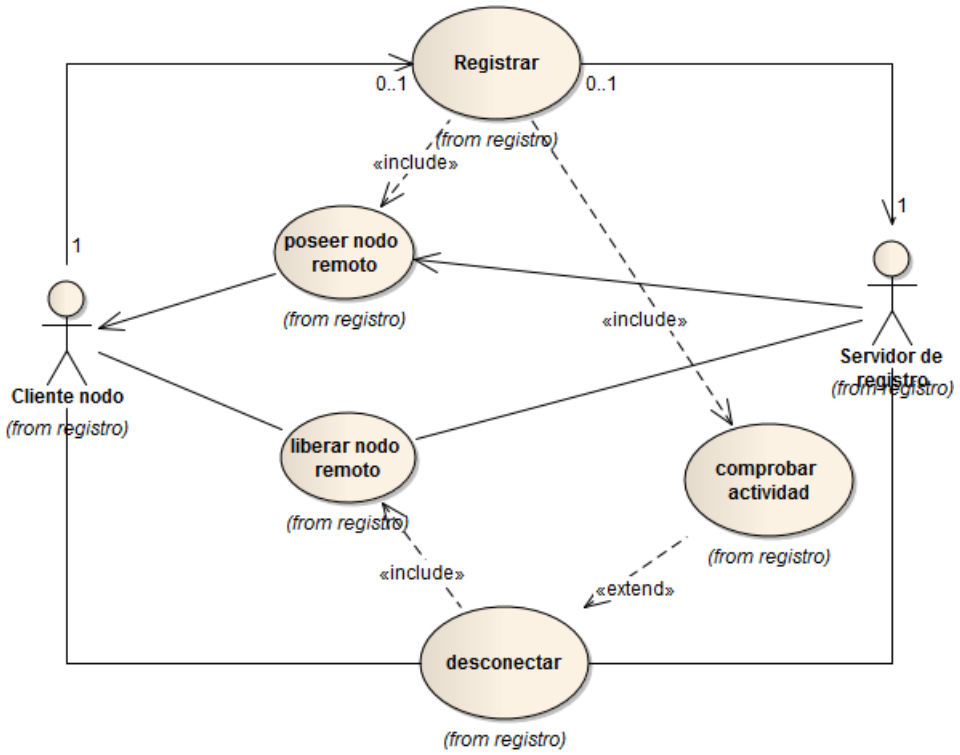


Figura 3.2.: Diagrama de casos de uso del subsistema de registro

Resumen	El cliente se conecta/registra en el servidor, obteniendo permiso para realizar operaciones remotas
Dependencias	
Actores	cliente nodo, servidor de registro
Precondiciones	El sistema de registro está iniciado y no contiene otro usuario registrado.
Postcondiciones	El usuario obtiene una referencia al nodo remoto que le permite interactuar con él
Curso normal	1. El cliente se registra en el servidor y éste crea un nodo remoto 2. El cliente obtiene la referencia al nodo remoto 3. El servidor inicia un temporizador que marca los instantes de comprobación de actividad del cliente

Cuadro 3.1.: Caso de uso registrar

Resumen	El cliente obtiene la referencia al nodo remoto
Dependencias	es incluido por registrar
Actores	cliente nodo, servidor de registro
Precondiciones	El cliente ha realizado una petición de registro
Postcondiciones	El cliente tiene la referencia remota y el servidor marca el cliente como registrado
Curso normal	1. El servidor envía al cliente la referencia al nodo remoto

Cuadro 3.2.: Caso de uso poseer nodo remoto

Resumen	Se inicia un proceso secundario de comprobación de actividad del cliente
Dependencias	es incluido por registrar
Actores	
Precondiciones	El cliente completa con éxito la operación de registro
Postcondiciones	La comprobación se mantiene mientras que el cliente esté registrado
Curso normal	1. El servidor inicia el proceso de comprobación de actividad

Cuadro 3.3.: Caso de uso comprobar actividad

Resumen	El cliente se desconecta del servidor, perdiendo el acceso a las operaciones remotas
Dependencias	es extendido por comprobar actividad
Actores	cliente nodo, servidor de registro
Precondiciones	El cliente completa con éxito el proceso de registro
Postcondiciones	El cliente pierde la referencia al nodo remoto y deja el registro libre
Curso normal	1. El cliente envía la petición de desconexión 2. El servidor libera el registro y se detiene el comprobador de actividad 3. El servidor elimina el nodo remoto y la referencia
Curso alternativo	1. El comprobador de actividad detecta al cliente como no disponible y realiza una desconexión forzada

Cuadro 3.4.: Caso de uso desconectar

Resumen	El cliente pierde/libera la referencia al nodo remoto
Dependencias	es incluido por desconectar
Actores	cliente nodo, servidor de registro
Precondiciones	El cliente inicia una petición de desconexión
Postcondiciones	El nodo remoto está eliminado y el cliente no dispone de la referencia
Curso normal	1. El servidor indica al cliente que ya no dispone de la referencia al nodo remoto

Cuadro 3.5.: Caso de uso liberar nodo remoto

Subsistema de operaciones

Es el encargado de ofrecer las distintas operaciones que el cliente puede realizar. El funcionamiento de las operaciones estará determinado por las herramientas proporcionadas por los paquetes `audio` y `qos`. El diagrama de caso de uso se muestra en la figura 3.3. Los actores son la entidad cliente y servidor.

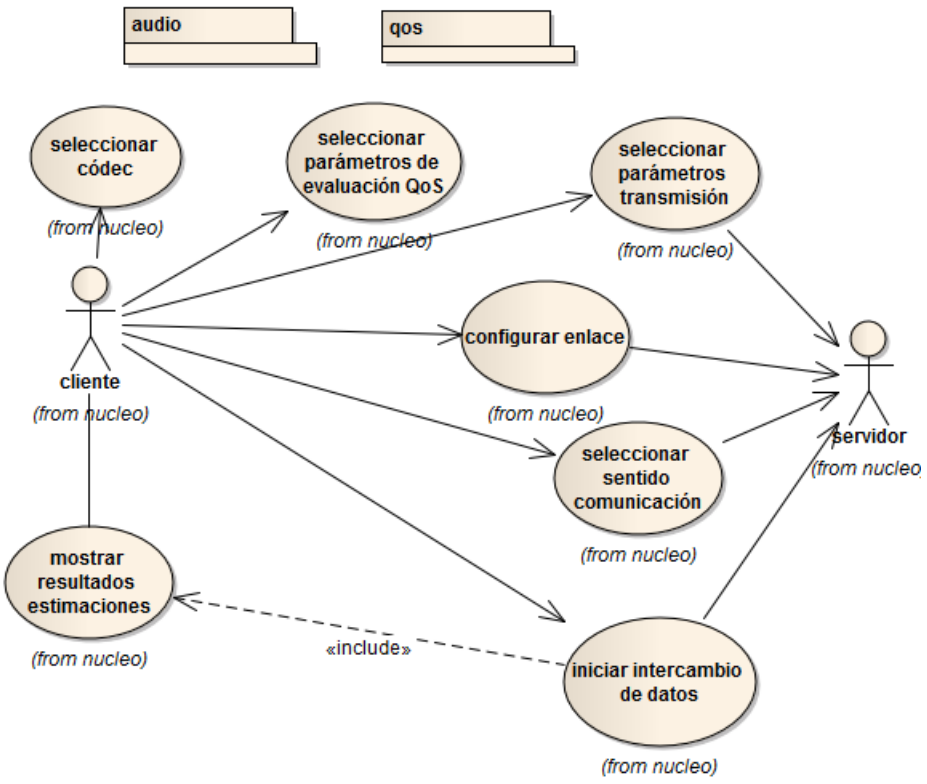


Figura 3.3.: Diagrama de casos de uso del subsistema de operaciones

Descripción del caso de uso:

- configurar enlace (Cuadro 3.6)
- seleccionar códec (Cuadro 3.7)
- seleccionar parámetros de evaluación de QoS (Cuadro 3.8)

- seleccionar parámetros de transmisión (Cuadro 3.9)
- seleccionar sentido de la comunicación (Cuadro 3.10)
- iniciar intercambio de datos (Cuadro 3.11)
- mostrar resultados estimaciones (Cuadro 3.12)

Resumen	El cliente define los extremos del enlace de comunicación a evaluar
Dependencias	
Actores	cliente, servidor
Precondiciones	
Postcondiciones	Se crean los servicios que atienden la transmisión y recepción de datos por las interfaces seleccionadas
Curso normal	1. El cliente selecciona las direcciones de red del equipo cliente y servidor

Cuadro 3.6.: Caso de uso configurar enlace

Resumen	El cliente elige la técnica de codificación que se aplicará a los datos de voz que se incluirán en el intercambio de información con el servidor
Dependencias	
Actores	cliente
Precondiciones	el enlace está configurado
Postcondiciones	
Curso normal	1. El cliente selecciona la técnica de una lista que contiene los codecs disponibles en el sistema

Cuadro 3.7.: Caso de uso seleccionar códec

Resumen	El cliente selecciona qué parámetros de QoS desea obtener para la evaluación del enlace
Dependencias	
Actores	cliente
Precondiciones	el enlace está configurado
Postcondiciones	
Curso normal	1. El cliente selecciona los parámetros de QoS que desee entre los disponibles en el sistema

Cuadro 3.8.: Caso de uso seleccionar parámetros de evaluación de QoS

Resumen	El cliente selecciona el perfil de tráfico de datos que intercambiará con el servidor
Dependencias	
Actores	cliente, servidor
Precondiciones	el enlace está configurado
Postcondiciones	
Curso normal	1. El cliente selecciona los parámetros de transmisión

Cuadro 3.9.: Caso de uso seleccionar parámetros de transmisión

Resumen	El cliente determina quién actúa como emisor o receptor. Existe dos sentidos: ascendente (cliente emisor, servidor receptor) y descendente (cliente receptor, servidor emisor)
Dependencias	
Actores	cliente, servidor
Precondiciones	el enlace está configurado
Postcondiciones	
Curso normal	1. El cliente selecciona el sentido

Cuadro 3.10.: Caso de uso seleccionar sentido de la comunicación

Resumen	El cliente indica que comience el proceso de intercambio de datos con el servidor
Dependencias	
Actores	cliente, servidor
Precondiciones	el enlace está configurado
Postcondiciones	El intercambio de información se produce sin errores y los resultados de la estimación deben de estar disponibles
Curso normal	1. El cliente indica el comienzo del intercambio, produciéndose en el sentido indicado 2. El intercambio finaliza cuando se transmiten todos los datos seleccionados

Cuadro 3.11.: Caso de uso iniciar intercambio de datos

Resumen	El cliente accede a los resultados de las estimaciones de los parámetros de QoS seleccionados
Dependencias	incluido en iniciar intercambio de datos
Actores	cliente
Precondiciones	Se produce un intercambio de datos previo
Postcondiciones	
Curso normal	1. El cliente selecciona mostrar los resultados

Cuadro 3.12.: Caso de uso mostrar resultados estimaciones

Subsistema de audio

Es el encargo de ofrecer el servicio de codificación y decodificación de voz. El diagrama de caso de uso se muestra en la figura 3.4. Un actor compone el sistema, que es una entidad funcional encargada de interactuar con el marco de trabajo multimedia.

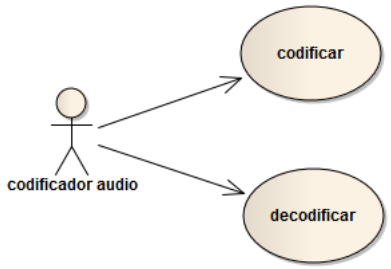


Figura 3.4.: Diagrama de casos de uso del subsistema de audio

Descripción del caso de uso:

- codificar (Cuadro 3.13)
- decodificar (Cuadro 3.14)

Resumen	El codificador toma datos de voz sin codificar y genera datos codificados según la técnica especificada
Dependencias	
Actores	codificador
Precondiciones	El códec aplicado debe estar disponible
Postcondiciones	Se genera el audio codificado
Curso normal	

Cuadro 3.13.: Caso de uso codificar

Resumen	El codificador toma datos de voz codificados y genera datos sin codificar según la técnica especificada
Dependencias	
Actores	codificador
Precondiciones	▪ Ocurre un proceso de codificación previo ▪ el códec debe de corresponderse con el aplicado en la codificación
Postcondiciones	Se genera el audio decodificado
Curso normal	

Cuadro 3.14.: Caso de uso decodificar

3.3.2. Requisitos no funcionales

Implementación

Para la implementación del sistema completo se utilizará el lenguaje de programación Java. El marco de trabajo multimedia será *GStreamer* y será necesario disponer de la librería para el lenguaje escogido.

Interfaz

La aplicación está preparada para funcionar en equipos de propósito general y acceder a los interfaces de red que dispongan.

Facilidad de uso

La aplicación será mostrada al usuario mediante una interfaz gráfica de usuario de ventanas, con controles de fácil manipulación y que aparecerán en distintos paneles según la función que realicen.

Fiabilidad

La aplicación debe manejar internamente los errores y excepciones que puedan producirse. Se incluirá un servicio de *logging* que registre los eventos que se producen en la aplicación y que permitan corregir rápidamente situaciones inesperadas.

Rendimiento

La interfaz al usuario no se bloqueará al realizar operaciones de largo tiempo de procesamiento. El rendimiento de la aplicación dependerá del *hardware* del equipo y del estado del sistema operativo subyacente. Por tanto se recomienda limitar o anular el número de aplicaciones ajenas a VoIPTester y disponer de *hardware* de última generación.

Soporte

El proceso de desarrollo de *software* será modelado mediante UML. Se proporcionará un sistema abierto que permita la inclusión de nuevos módulos que aporten nuevas funciones. Para

facilitar una posible reutilización del sistema, se proporcionará la documentación del código fuente en Javadoc.

Empaquetamiento

La aplicación no requiere instalación. Se proporcionarán los *bytecodes* generados tras la compilación del código fuente, además de los correspondientes a librerías externas. Podrán proporcionarse dos paquetes; uno básico que no incluya los ficheros binarios del marco de trabajo multimedia y otro paquete que sí los incluya. Al realizar la implementación en Java, se proporcionarán los paquetes tanto para sistemas *Windows* como *Unix*.

4. DISEÑO

En este capítulo, para resolver el problema planteado se aborda el diseño de la solución propuesta. El contenido queda estructurado en dos secciones principales. La primera describe un conjunto de librerías que proporcionan las funcionalidades principales del sistema. La segunda aborda el diseño de un núcleo principal que gestiona y provee acceso a las operaciones ofrecidas por las librerías.

El sistema especificado requiere de un intercambio de información entre dos nodos en una red. La forma en la que se produce este intercambio así como el tipo de información del mismo depende del rol que ocupen los nodos involucrados. Es necesario, por tanto, disponer de un sistema de comunicación entre los mismos, por lo que se adopta una arquitectura cliente/servidor para cumplir este requisito.

Otro de los aspectos a tener en cuenta es que el diseño de las librerías deber ser modular e independiente, aislando el servicio que ofrecen respecto a la forma de incluirlo en otros componentes. Así el núcleo no queda condicionado por el diseño de las librerías, existiendo múltiples alternativas para el diseño de éste según el entorno del sistema.

El diseño se va a realizar partiendo de los principales paquetes por los que el sistema estará compuesto. El diagrama de paquetes se muestra en la figura 4.1. Se definen cuatro paquetes principales:

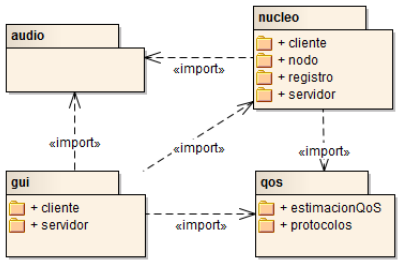


Figura 4.1.: Diagrama de paquetes del sistema

- `audio`: librería encargada de proporcionar la aplicación de distintas técnicas de codificación a los datos de voz. Tanto el diseño como las dependencias del paquete dependerá del marco de trabajo multimedia que se escoja.
- `qos`: librería que contiene el conjunto de herramientas que permiten evaluar un enlace de comunicaciones mediante la estimación de parámetros de calidad de servicio. Se asume como un módulo independiente que no condicione el diseño del resto del sistema.
- `nucleo`: módulo encargado de acceder, controlar y usar las librerías de audio y estimación para ofrecer el comportamiento final del sistema deseado.
- `gui`: proporciona la interfaz gráfica de usuario.

4.1. LIBRERÍAS

Para llevar a cabo la estimación de los parámetros de QoS se va a considerar el escenario de la figura 4.2. Se trata de un equipo emisor que envía un conjunto de paquetes de información a un equipo receptor; durante este intercambio, ambos equipos obtienen una marca de tiempo por cada paquete enviado y recibido, respectivamente.

A partir de este procedimiento hay que tener en cuenta tres elementos principales que intervienen en el proceso de estimación de parámetros de QoS adoptado:

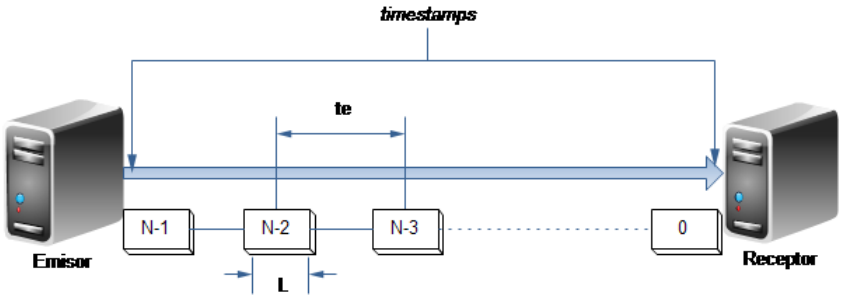


Figura 4.2.: Procedimiento de marcas de tiempo

1. **Variables de entrada**. Determinan de qué manera se produce el intercambio de información en la red así como el contenido de ésta. Quedan definidas por el número total de paquetes a enviar, N, la longitud de cada paquete, L, y el intervalo de tiempo que transcurre entre dos envíos consecutivos, t_e.
2. **Conjunto de medidas experimentales obtenidas del proceso**. Cada equipo involucrado en la comunicación debe de generar una marca de tiempo o *timestamp* que indique el tiempo de envío o recepción de cada paquete, según corresponda. Estas marcas de tiempo, junto con la información que transportan los paquetes, constituyen el conjunto de medidas experimentales que se utilizarán en el procedimiento de estimación.
3. **Función estimadora**. Constituye el algoritmo que, tomando las variables de entrada y las medidas experimentales, proporciona el parámetro estimado. La definición de esta función depende del parámetro que se estime.

Una aspecto a tener en cuenta en el procedimiento de la figura 4.2 es el sentido de envío de paquetes, que determina el sentido de la comunicación que será evaluado. Además el intercambio de información debe de producirse sin mecanismos de control de errores o de congestión, ya que es la forma habitual de transportar voz. Así pues, el transporte se realizará mediante UDP.

A continuación se detallan los distintos componentes que contiene la librería desarrollada y que proporcionan la funcionalidad necesaria para el objetivo del proyecto: medir la calidad entre dos puntos en la red.

4.1.1. LIBRERÍA DE ESTIMACIÓN DE QoS

Esta sección describe la librería que permite aplicar el procedimiento de marcas de tiempo introducido. Se estructura en base a cuatro subsecciones. La primera define el formato de

los paquetes de información; la segunda define los parámetros de transmisión y cómo se representan; en la tercera se explican los detalles del mecanismo de intercambio de paquetes; y finalmente, en la cuarta, se indica el formato de representación de los parámetros estimables así como de las funciones estimadoras.

4.1.1.1. Formato de los mensajes

El procedimiento de actuación de la figura 4.2 exhibe que la unidad de información de transporte es el paquete, considerado como un conjunto de bytes cuyo formato es genérico. Sin embargo conviene definir cómo se organizan estos bytes y qué datos representan. El tipo de datos puede variar desde tipos primitivos (enteros, por ejemplo) hasta otras unidades de información superiores (ficheros, por ejemplo). Por tanto se define el mensaje como unidad de información de la aplicación, cuya representación se muestra en la figura 4.3.

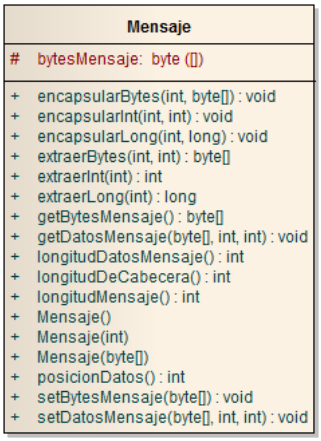


Figura 4.3.: Clase `Mensaje`

Un mensaje no es más que una secuencia de bytes, ordenados según el criterio *big-endian*, con la que pueden realizarse distintas operaciones básicas, como encapsular y extraer tipos primitivos o una subsecuencia de bytes. Esta clase padre representa un mensaje en el que la secuencia de bytes representa un campo de datos, y no se considera ninguna cabecera extra.

Sin embargo hay que valorar la posibilidad de incluir una cabecera formada por una serie de campos que permite poner de acuerdo a ambas partes de la comunicación. El formato de la cabecera varía según el protocolo considerado, por lo que basta extender la clase `Mensaje` añadiendo los campos adicionales según corresponda. Así para el procedimiento de marcas de tiempo (figura 4.2) es necesario numerar cada mensaje para conocer el orden de envío y esto se consigue añadiendo una cabecera con un campo que identifique numéricamente a cada mensaje, tal y como se muestra en la figura 4.4.

La cabecera consta simplemente de un entero de cuatro bytes que determina el orden en el que se envían los mensajes. Este identificador es imprescindible en el receptor para conocer a qué mensaje hay que asignar el *timestamp* obtenido tras una recepción.

La clase que representa este tipo de mensaje se muestra en la figura 4.5. Se trata un mensaje cuya longitud total es igual a la indicada por los datos más la longitud de la cabecera extra.

El encapsulamiento de estos mensajes para el protocolo de transporte es tal que existe una relación uno a uno entre mensaje y paquete UDP. Como UDP dispone de dieciséis bits para

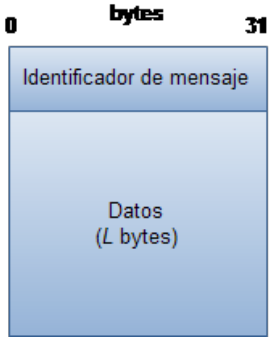


Figura 4.4.: Formato de los mensajes del protocolo de timestamps

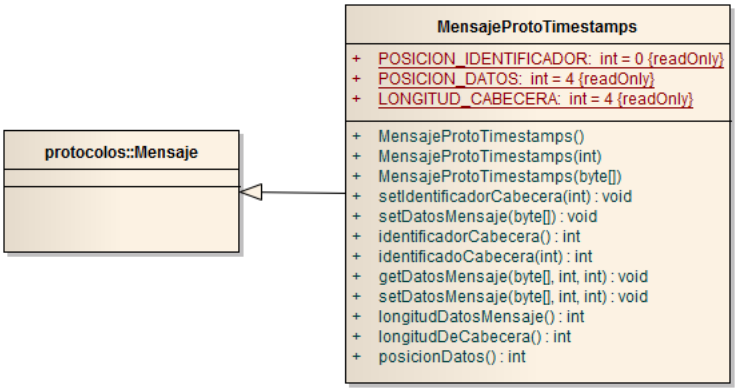


Figura 4.5.: Clase `MensajeProtoTimestamps`

indicar la longitud en bytes del paquete, y su cabecera tiene una longitud de ocho bytes, el formato del mensaje empleado permite albergar una carga útil de:

$$L_{max} = 2^{16} - H_{UDP} - H_{protoTimestamps} = 65536 - 8 - 4 = 65524\, bytes \tag{4.1}$$

Luego este parámetro junto con el tiempo mínimo entre envíos definirá el ancho de banda máximo de emisión, y por tanto, el ancho de banda máximo que se podrá estimar.

4.1.1.2. Protocolo de marcas de tiempo

El siguiente paso es describir con más detalle el procedimiento indicado en el escenario de la figura 4.2 especificando qué información adicional y mecanismos deben considerarse. La implementación del protocolo de marcas de tiempo debe llevarse a cabo en la clase `ProtocoloTimestamps`, mostrada en la figura 4.6. La clase demanda funcionalidades de temporización (descritas en la sección 4.1.3), utilizar mensajes para representar los datos intercambiados, y disponer de un *socket* no orientado a conexión.

La clase `ProtocoloTimestamps` es la que se encarga de manejar el intercambio de mensajes entre emisor y receptor mediante un `DatagramSocket` (*socket* no orientado a conexión). Junto a este intercambio se genera un vector con las marcas de tiempo de envío o recepción, según corresponda.

En la figura 4.6 se observa que se ha optado por realizar una sola clase que represente una

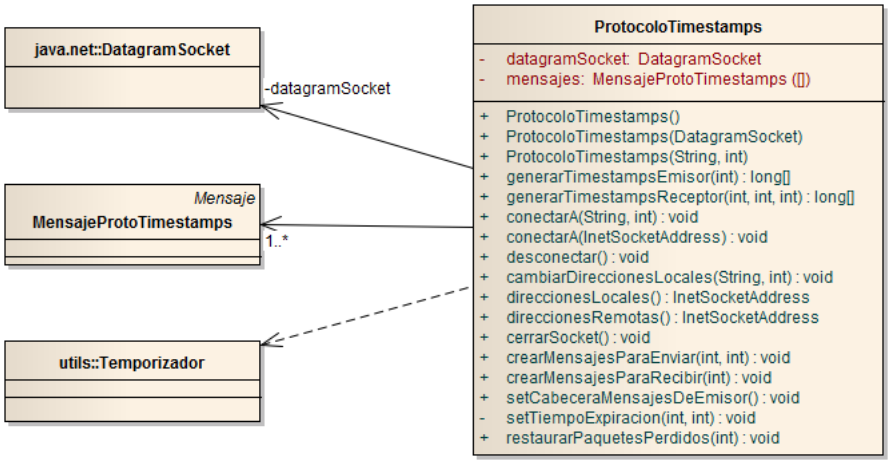


Figura 4.6.: Clase `ProtocoloTimestamps`

aplicación que puede actuar tanto de emisor como de receptor, independientemente de si se ejecuta en el equipo cliente o en el servidor. Esto es debido a que ambas entidades pueden realizar las mismas operaciones.

Tanto cliente como servidor pueden actuar como emisor o receptor, con el fin de poder considerar ambos sentidos en la comunicación. Para conseguir esto es necesario que ambas entidades puedan cambiar de rol dentro del procedimiento de *timestamps*. Así, con un mismo programa que contenga las operaciones de un emisor y un receptor es suficiente para implementar el protocolo completo.

Para comprender este diseño hay que pensar en que cada objeto de la clase `ProtocoloTimestamps` tendrá, entre otros, un atributo de tipo `DatagramSocket`. Este proporciona un servicio UDP que permite:

- Envío de mensajes en modo emisor.
- Recepción de mensajes en modo receptor.

Un objeto `ProtocoloTimestamps` puede funcionar en cualquiera de estos modos, pero no en ambos al mismo tiempo. Además, la entidad que actúe como receptora debe estar ejecutándose antes de recibir el primer mensaje. De lo contrario se perderían mensajes debido al mal funcionamiento del sistema, no siendo la causa las condiciones de la red. Por tanto el usuario de la librería tiene que disponer de su propio protocolo de gestión de peticiones para configurar adecuadamente los modos de actuación de cada entidad, así como el acceso a las marcas de tiempo generadas en cada entidad.

A continuación se explica el comportamiento que se desea en cada uno de los modos de operación. Para ello se incluye un diagrama de secuencia que describe la distintas operaciones que se llevan a cabo en cada modo.

Por un lado, la figura 4.7 presenta la operación en modo emisor, en la que por cada paquete UDP (que contiene un mensaje) enviado se genera una marca de tiempo, cortesía de la clase `Temporizador` (ver sección 4.1.3). Por tanto se generan N marcas de tiempo.

Por otro lado, la operación en el modo complementario, es decir el modo receptor, se presenta en la figura 4.8. En este caso se genera una marca de tiempo por cada paquete UDP recibido.

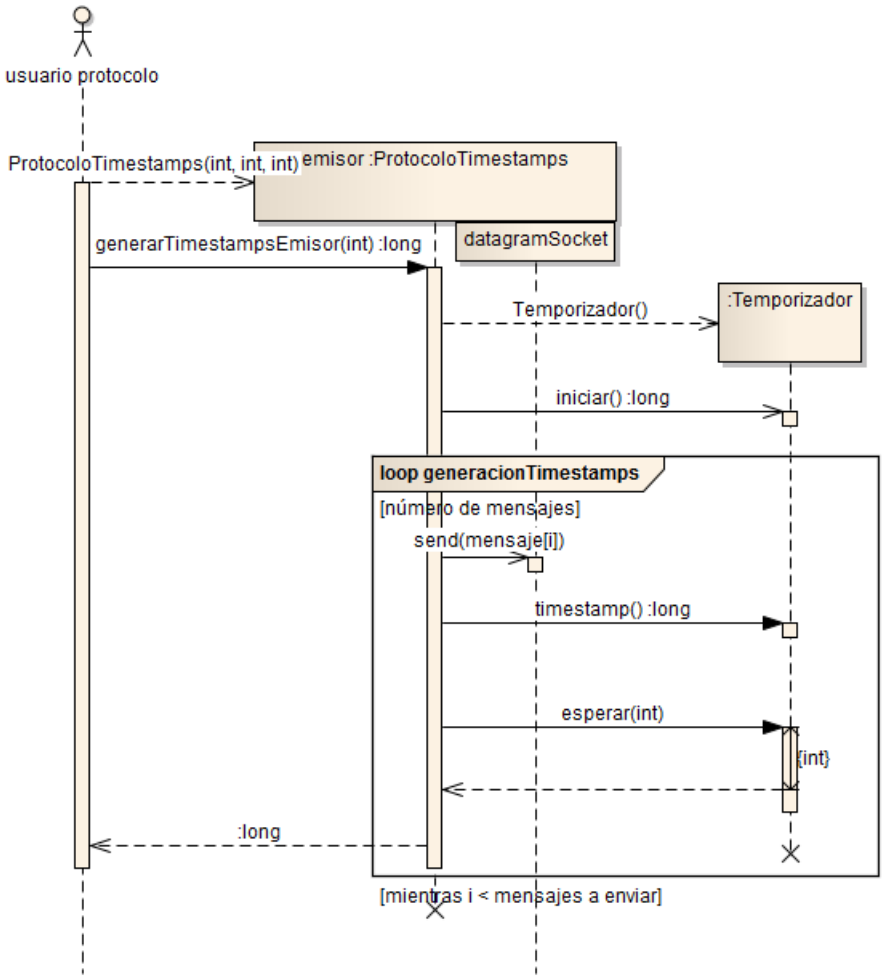


Figura 4.7.: Generación de timestamps en modo emisor

Los paquetes pueden llegar desordenados, pero la información de la cabecera del mensaje contenido en el paquete UDP permite crear correctamente la correspondencia entre mensaje y marca de tiempo asociada. En este caso, debido a las posibles pérdidas, el número de marcas de tiempo generadas puede ser menor o igual que N.

En este punto queda claro que, si un paquete se pierde, no existirá un *timestamp* asociado. Sin embargo, además de los paquetes perdidos están los paquetes con errores. El tratamiento correcto sería poder distinguir entre pérdidas y errores. Para ello sería necesario realizar la suma de comprobación que proporciona UDP, pero esta comprobación la realiza el sistema operativo subyacente descartando aquellos paquetes erróneos. Por tanto, desde el punto de vista de la aplicación estos paquetes no se recibirán, obligando a ésta a interpretarlos como pérdidas.

No obstante, debido a la naturaleza de la suma de comprobación [29] de UDP, puede ocurrir que un paquete haya sido corrompido y que la suma de comprobación sea correcta. En este caso el paquete sí se entrega a la aplicación y es responsabilidad de ésta determinar la validez de los datos. Dada esta posibilidad, en el procedimiento de generación de marcas de tiempo en modo receptor mostrado en la figura 4.8 se observa una comprobación de la cabecera de los mensajes. Si esta cabecera presenta errores se lanzará una excepción de

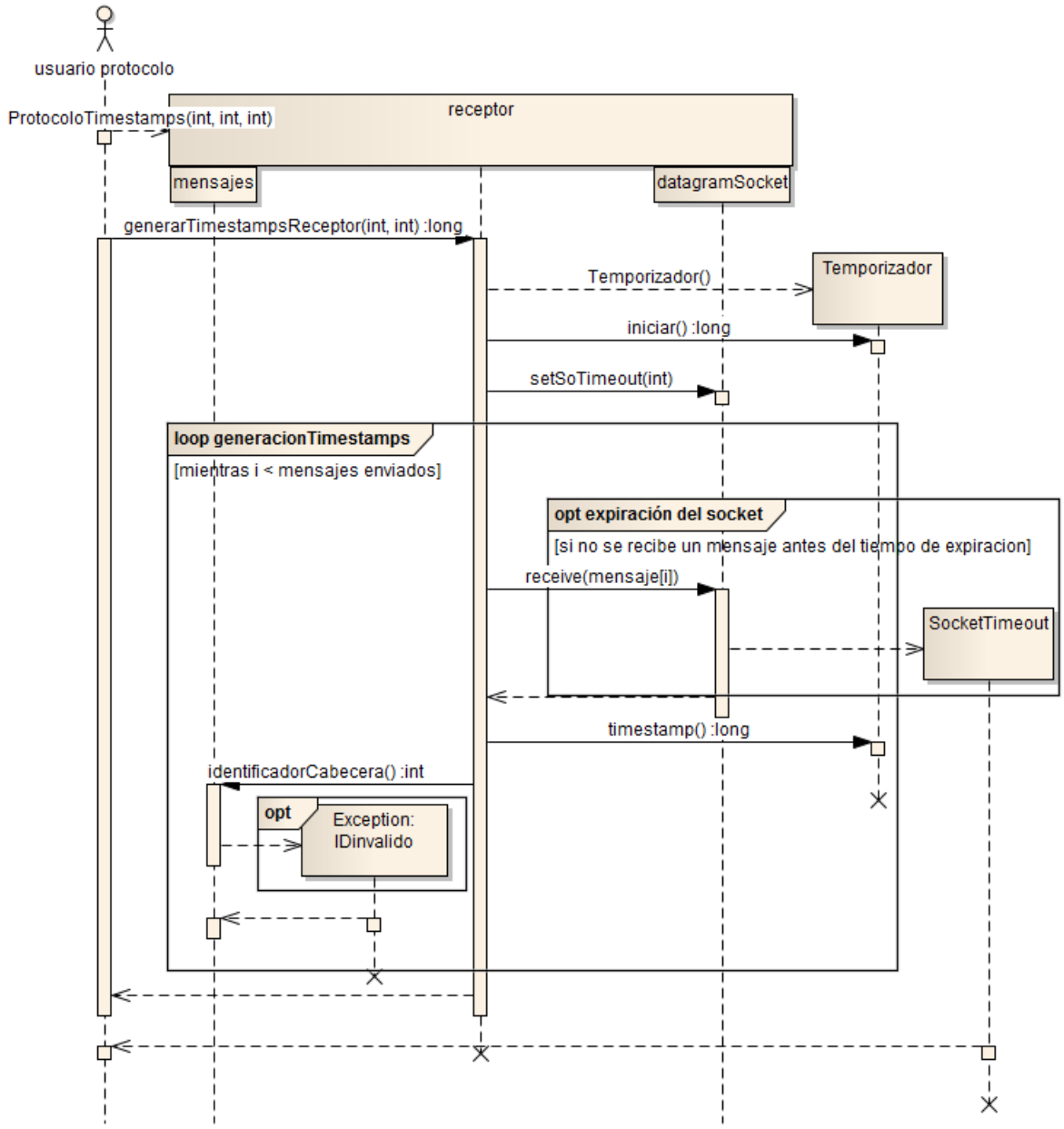


Figura 4.8.: Generación de timestamps en modo receptor

tipo `IdCabeceraInvalidoException`, y descartará la marca de tiempo correspondiente a este mensaje. Aun así, el procedimiento de comprobación de la aplicación no es sumamente fiable, y pueden darse situaciones en las que la cabecera sea válida pero no los datos. En cualquier caso, esta casuística tiene una probabilidad de ocurrencia baja por lo que sus efectos en las posteriores estimaciones pueden considerarse despreciables.

Por estas razones VoIPTester interpreta pérdidas y errores de la misma manera. Esto no es del todo correcto pero en sí ambas situaciones provocan los mismos efectos. Si bien un paquete se pierde o contiene errores, la aplicación receptora notará la ausencia de los datos de ese paquete. Por tanto a efectos prácticos, una pérdida o un paquete erróneo se corresponde con una pérdida de datos, y esto es lo que VoIPTester analiza.

Otra de las diferencias respecto al emisor es la existencia de un tiempo de expiración. En principio, el receptor espera la recepción de N paquetes, siendo N el número de paquetes enviados. Sin embargo dada la posibilidad de que se pierdan uno o más paquetes, es necesario contemplar un periodo de expiración para no que se produzca espera infinita y que debe ser lo suficientemente grande como para dar margen suficiente para que llegue el resto de paquetes.

Se pueden dar dos situaciones:

- Si el retardo de propagación es mayor que el tiempo entre envíos de paquetes, entonces en el emisor el paquete $x + 1$ se genera antes de que el receptor obtenga el paquete x.
- Si el retardo de propagación es menor que el tiempo entre envíos entonces el receptor obtiene el paquete x antes de que el emisor genere el paquete $x + 1$.

Luego el periodo de expiración finalizará si:

- Uno o más paquetes se pierden
- Si el siguiente paquete a recibir se retrasa demasiado
- Si hay una ráfaga de paquetes perdidos excesivamente larga (lo que se podría interpretar como fallo de conexión).

Por tanto para fijar la duración del periodo de expiración hay que tener en cuenta la ráfaga máxima que se puede tolerar, N_{Rmax}. Por tanto el tiempo de expiración se fija de acuerdo a la ecuación 4.2:

$$t_{exp} = RTT + N_{Rmax} \cdot t_e \tag{4.2}$$

donde RTT es el tiempo de ida y vuelta entre emisor y receptor, y t_e el tiempo entre envíos de paquetes.

Así, después de cada intercambio completo de mensajes se generan dos vectores de marcas de tiempo que indican el instante de transmisión y recepción de los mensajes enviados y recibidos, respectivamente. En la figura 4.9 se muestra el formato de estos vectores, donde Te_x y Tr_x son los *timestamps* asociados al paquete número x enviado y recibido, respectivamente.

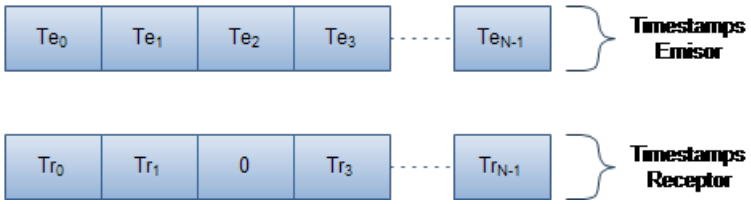


Figura 4.9.: Formato del vector de *timestamps*

En el caso de que un mensaje no se reciba, el *timestamp* correspondiente dentro del vector será nulo.

Por último hay que aclarar que la asignación de los modos del protocolo en los extremos determina el sentido de la comunicación para la cual se va a realizar la estimación de los parámetros de QoS. Y como en un enlace de comunicaciones hay dos sentidos, es necesario establecer un convenio que permita identificar ambos:

- **enlace ascendente**: de cliente a servidor. El cliente participa en modo emisor y el servidor en receptor.
- **enlace descendente**: de servidor a cliente. El cliente participa en modo receptor y el servidor en emisor.

Un detalle importante es que las marcas de tiempo se obtienen a nivel de transporte, es decir, la marca de tiempo no se genera cuando el paquete sale o entra por la interfaz de red física, sino cuando el paquete se envía o se recibe por el servicio de transporte correspondiente. En principio tiene más sentido considerar las medidas a este nivel, ya que en la realidad la comunicación por VoIP se realiza entre dos aplicaciones (que utilizan servicios de transporte) y por tanto el camino directo lógico que siguen los paquetes es el existente entre aplicación y aplicación.

4.1.1.3. Parámetros de transmisión

Del protocolo de marcas de tiempo descrito en la sección anterior puede deducirse que el tiempo total empleado en la transmisión es $T = N \cdot t_e$. Este tiempo puede ser lo suficientemente grande como para que las condiciones de la red varíen, y como consecuencia, las métricas asociadas [30].

Así pues la idea es poder "muestrear" la red y realizar las estimaciones de los parámetros de QoS sobre cada muestra, obteniéndose la variación de estos en función del tiempo. Esto se consigue introduciendo el concepto de intervalo, según se muestra en la figura 4.10.

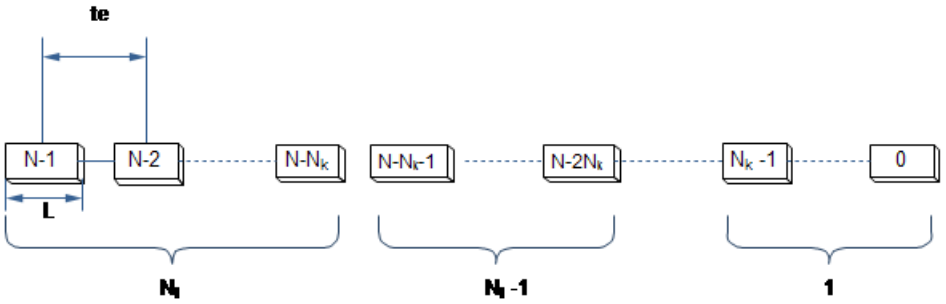


Figura 4.10.: Concepto de intervalo de mensajes

Un intervalo k está formado por un conjunto de N_k, $N_k = N_M = cte \; \forall \, k \in [0, N_I - 1]$ mensajes en el que sólo se considera la información relativa a estos para la estimación de los parámetros de calidad de servicio. Por tanto se efectúan un total de N_I estimaciones, una por cada intervalo de tiempo $N_k \cdot t_e$. De la figura 4.10 se desprende que el número total de mensajes enviados es $N = N_k \cdot N_I$, con N_I el número de intervalos a considerar.

Así que, para representar estos parámetros de transmisión, se diseña la clase mostrada en la figura 4.11. La responsabilidad de esta clase simplemente es proporcionar la encapsulación de los parámetros de transmisión y mantener las relaciones entre los mismos (por ejemplo el número total de mensajes o el número de bytes por intervalo).

Los datos miembro de la clase `ParametrosTransmision` son:

- `tiempoEntreEnvios`: t_e
- `longitudDatos`: L

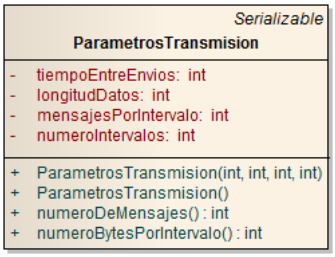


Figura 4.11.: Clase `ParametrosTransmision`

- `mensajesPorIntervalo`: N_k
- `numeroIntervalos`: N_I

Al definir estos parámetros de transmisión, de manera indirecta se incluyen los conceptos de velocidad de transmisión, R_T, y cantidad total de información a transmitir, L_T. Ambas magnitudes pueden obtenerse a partir de los parámetros de transmisión aplicando las expresiones 4.3 y 4.4, respectivamente.

$$L_T = N_I \cdot N_k \cdot L \tag{4.3}$$

$$R_T = L/t_e \tag{4.4}$$

4.1.1.4. Gestión de los datos que transportan los mensajes

La información que contienen los mensajes se trata como una cadena de bytes. Por ello hay que habilitar la transformación de una cadena de bytes a una secuencia de mensajes y viceversa. En la figura 4.12 se muestra la clase encargada de ello.

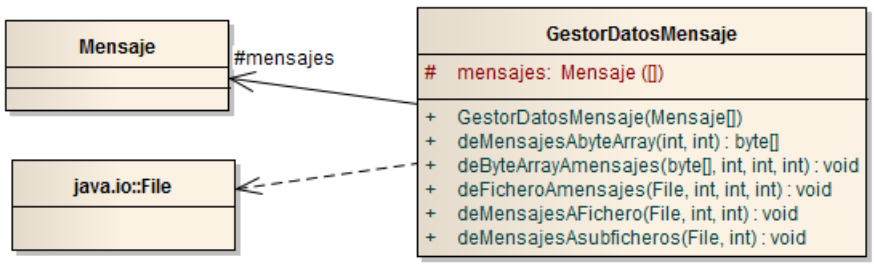
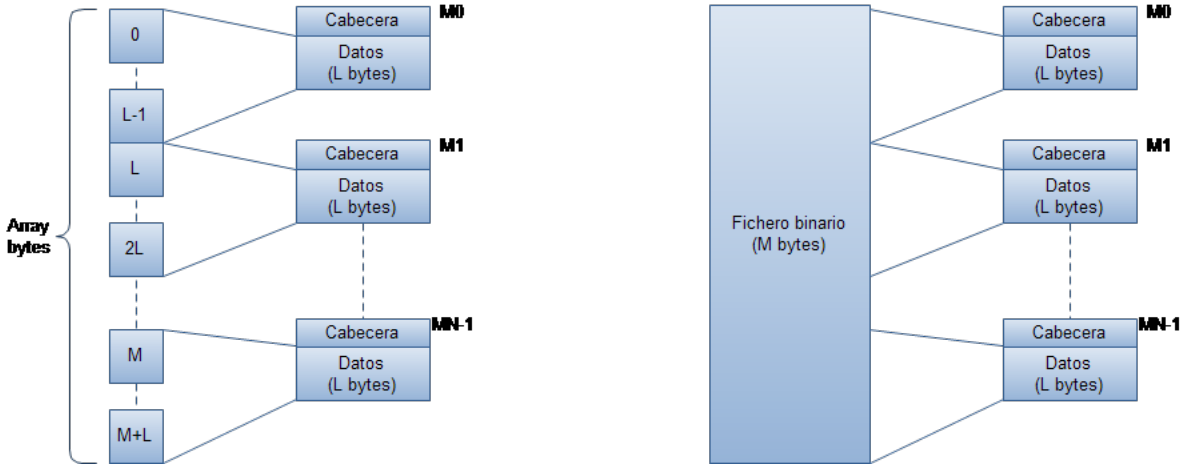


Figura 4.12.: Clase `GestorDatosMensaje`

La clase `GestorDatosMensaje` tiene como dato miembro la secuencia de mensajes a tratar. Es una clase genérica que permite mapear una cadena de bytes o un fichero binario a un conjunto de mensajes, o bien obtener los datos de los mensajes y devolverlos como una cadena de bytes o un fichero binario. Las posibles conversiones se muestran en la figura 4.13.

El tratamiento con cadenas de bytes comprende dos operaciones (figura 4.13a):



(a) Conversión entre bytes y mensajes

(b) Conversión entre fichero y mensajes

Figura 4.13.: Gestión de los datos de los mensajes

1. Conversión de $M + L$ bytes a un conjunto de N mensajes de longitud L, con $N = \frac{M}{L} + 1$.
2. Conversión de N de mensajes de longitud L a un *array* de bytes de longitud $M + L$, con $M = L \cdot (N - 1)$

El tratamiento con ficheros comprende dos operaciones (figura 4.13b):

1. Conversión de un fichero binario (o parte de este) de M bytes a una secuencia de N mensajes de L bytes.
2. Conversión de una secuencia de N mensajes de L bytes a un fichero de longitud $M = N \cdot L$ bytes.

El número de mensajes y la longitud de los datos albergados determinan la cantidad total de datos que se transmiten y se reciben.

4.1.1.5. Representación de parámetros de QoS y funciones estimadoras.

Una vez que se dispone de las medidas experimentales de tiempo, de los datos enviados y recibidos, y de los parámetros de transmisión asociados, es posible estimar una serie de parámetros de calidad de servicio.

Por un lado se pretende que el formato de representación para los distintos parámetros de QoS estimables sea el mismo, aunque cada uno disponga de sus propias particularidades. Por otra parte, se desea que la obtención de estos parámetros sea transparente para el usuario de la librería, es decir, que no necesite conocer de qué manera funcionan los estimadores.

Por tanto se van a definir dos responsabilidades: la representación y la obtención de los parámetros de QoS.

Al fin y al cabo, un parámetro de calidad de servicio es una magnitud física, y como tal queda representado por un valor numérico y las unidades asociadas. Pero una medida de magnitud aislada no proporciona información acerca de las condiciones en las que se realizó la medida. Así pues, para la representación de una medida de calidad de servicio se requiere:

1. Definir un formato de representación numérica.
2. Relacionar el parámetro con las condiciones de medida asociadas, es decir con las variables de entrada.

A continuación se describe como se consigue cada uno de estos requisitos.

Dado que al final la estimación consiste en la obtención de un valor numérico, este se va a representar por números decimales en coma flotante (tipo `float`), teniendo así mayor precisión en el caso de tener que realizar divisiones o cambios de unidades. Este valor numérico debe ir acompañado de las unidades de medida correspondientes.

También hay que diferenciar los distintos tipos de parámetros de QoS que ofrece VoIPTester:

- **Parámetros orientados a paquete**: son aquellos en los que se obtiene un valor asociado a cada paquete enviado/recibido. Es el caso del retardo, *jitter* y ancho de banda.
- **Parámetros orientados a intervalo**: son aquellos en los que se obtiene un solo valor a partir de un conjunto de N_k mensajes. Es el caso de probabilidad y distribución de pérdidas y MOS.

Por tanto, existirán parámetros con N_k valores por intervalo, y otros con un solo valor por intervalo. La idea es definir una clase base genérica que englobe las características comunes y extender ésta de acuerdo las características propias de cada parámetro.

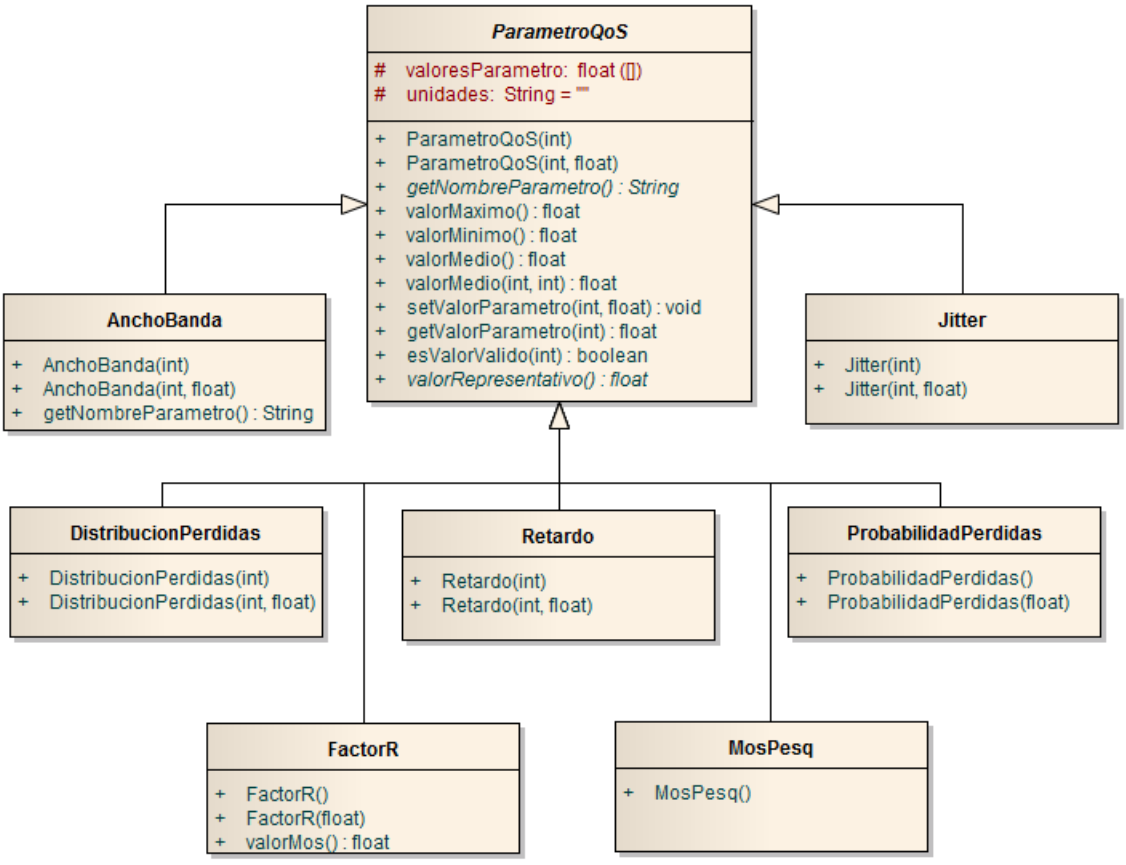


Figura 4.14.: Clase `ParametroQoS` y la jerarquía establecida

La clase `ParametroQoS`, mostrada en la figura 4.14, contiene el conjunto de valores numéricos que representan la magnitud y las unidades correspondientes. También define algunas operaciones útiles que pueden hacerse con estos valores, como son la media aritmética o el cómputo de los valores máximo y mínimo.

Cada instancia de `ParametroQoS` representa los valores estimados en un intervalo concreto. Por ello puede ser útil obtener un valor representativo en ese intervalo (por ejemplo el valor medio). Este valor dependerá del tipo de parámetro por lo que se define un método abstracto para que cada parámetro elija el valor mas adecuado. Así esta superclase abstracta establece un punto común que será útil para determinar la manera en que se obtienen los distintos parámetros de QoS.

A continuación se detalla cada uno de los parámetros y sus características:

`AnchoBanda`

> El concepto de ancho de banda hace referencia a la velocidad de transmisión/recepción de datos que soporta un enlace, es decir, a su capacidad. Teniendo en cuenta las características de transmisión mencionadas en la sección 4.2, en principio por cada paquete de datos enviado/recibido en un tiempo determinado sería posible obtener una estimación de ancho de banda. Por ello este parámetro se considera orientado a paquete. Así cada elemento del vector i contiene el valor de ancho de banda estimado tras la recepción del paquete i. Sin embargo, cuantos más paquetes estén implicados mejor será la estimación, por lo que el valor más representativo se corresponde con el valor estimado a partir del mayor número de paquetes recibidos.

`Retardo`

> Tiempo que transcurre desde que se envía el paquete en un extremo hasta que se recibe en el otro. Cada elemento i del vector contiene el retardo que sufre el paquete i, siempre que el paquete se haya recibido correctamente.

`Jitter`

> Variación del retardo. Cada elemento i del vector contiene la diferencia de retardo entre el paquete i y el $i-1$.

`ProbabilidadPerdidas`

> Probabilidad de que se pierda un paquete. El vector contiene sólo un elemento que es el valor de la probabilidad de pérdidas estimada en un intervalo.

`DistribucionPerdidas`

> No sólo es importante conocer la probabilidad de pérdidas sino también su distribución, es decir, con qué frecuencia y con qué longitud se producen ráfagas de paquetes perdidos. Así el elemento i del vector contiene el número de veces que se producen ráfagas de tamaño $i+1$.

`FactorR`

> El vector contiene un solo elemento correspondiente a una estimación del factor de determinación de índices de transmisión, R, obtenido a partir del modelo E descrito en la recomendación G.107 de la ITU-T [28]. Debido a sus condiciones de cálculo, es posible obtener una estimación del factor R por cada intervalo de duración $D = duracion(N_k)$, con N_k el número de mensajes por intervalo.

`MosPesq`

El vector contiene un solo elemento correspondiente a una estimación de la nota media de opinión, obtenida a partir del test PESQ descrito en la recomendación P.862 de la ITU-T [27]. Debido a sus condiciones de cálculo, es posible obtener una estimación del valor MOS por cada intervalo de duración $D = duracion(N_k)$, con N_k el número de mensajes por intervalo.

Llegado a este punto, el formato de representación para los parámetros de QoS está definido. Ya sólo queda asociar los distintos parámetros con las variables de entrada (que fijan las condiciones en las que se obtienen las medidas experimentales). Para ello se define la **medida** de un parámetro de QoS como una asociación entre las variables de entrada y los valores del parámetro estimados. De esta manera es conveniente distinguir entre dos tipos de medidas:

- **Medidas de calidad de servicio objetivas**: los parámetros de QoS se obtienen a partir de los parámetros de transmisión y de las marcas de tiempo obtenidas para los mensajes, siendo irrelevantes los datos que transportan. Por ello pueden asociarse simplemente con los parámetros de transmisión.
- **Medidas de calidad de servicio subjetivas**: los parámetros de QoS se obtienen a partir de los parámetros de transmisión y de las marcas de tiempo obtenidas para los mensajes, pero también a partir de los datos que transportan. Por ello también deben de asociarse con el el tipo de datos que se utilizó. Dado que se trabaja con datos de voz, el tipo de datos queda representado unívocamente por el códec empleado.

Además de esta clasificación, es posible que ciertas medidas requieran variables de entrada adicionales. Así, siguiendo la misma filosofía de diseño de la figura 4.14 se define un punto común a todos los tipos de medidas de QoS dejando que cada una añada las características propias.

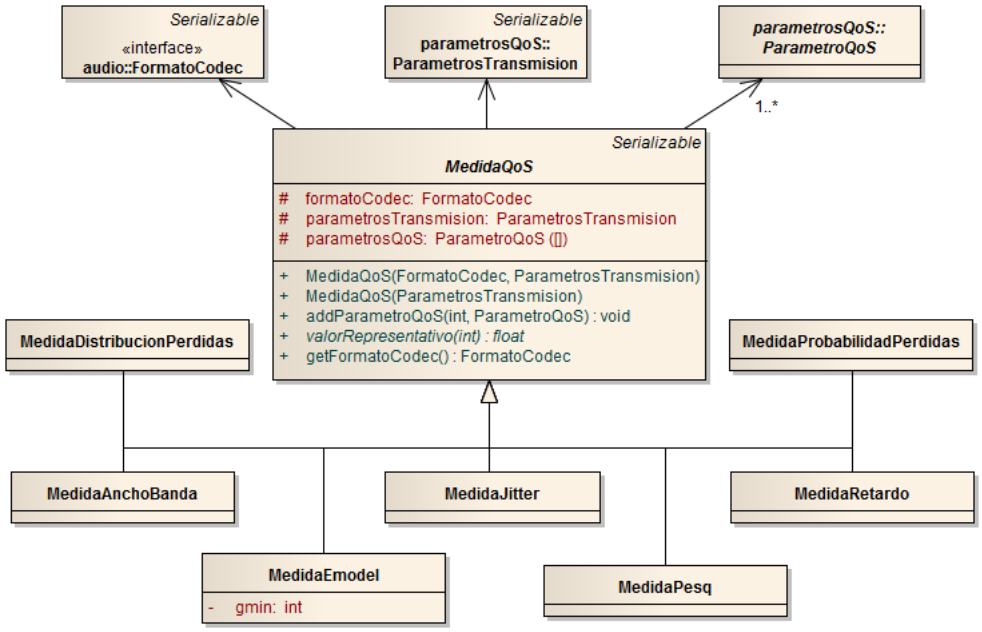


Figura 4.15.: Clase `MedidaQoS`

La clase abstracta `MedidaQoS` (figura 4.15) establece dicho punto común. En él se indican cuáles son las variables de entrada comunes a todos los parámetros y qué operaciones deben de poder realizarse con cada uno. Cada medida que extienda de la clase principal debe imponer sus restricciones.

Por ejemplo, una medida de retardo puede crearse teniendo en cuenta sólo los parámetros de transmisión o, además, el tipo de datos de audio intercambiados. Sin embargo una medida de PESQ sólo puede crearse teniendo en cuenta tanto los parámetros de transmisión como el tipo de datos, ya que para obtener esta medida es necesaria incluir voz en los datos de los mensajes.

También se observa que `MedidaQoS` contiene un vector de `ParametrosQoS`. Cada elemento del vector se corresponde con la representación del parámetro obtenido en un intervalo. Por tanto existirán tantos elementos en el vector como número de intervalos indiquen los parámetros de transmisión.

Otra de las funciones que se ofrece es la obtención de un valor representativo del parámetro en cada intervalo. Esto permite realizar un procesado de los valores de los parámetros dependiendo del parámetro en sí. Por ejemplo, en la especificación del protocolo RTP (RFC 1889 [15]) se expone la aplicación de un filtro de suavizado para la obtención de un valor óptimo de *jitter*. Este tipo de procesado puede aplicarse a cada medida de manera independiente sin más que implementar el método `valorRepresentativo()`.

Una vez que ya está disponible la representación de las medidas de QoS queda desarrollar la forma en que se realizan las estimaciones, es decir, cómo se definen las funciones estimadoras. Para ello se introduce un servicio de estimación de QoS que toma las variables de entrada y las medidas experimentales, y devuelve la medida estimada, tal y como se muestra en la figura 4.16.

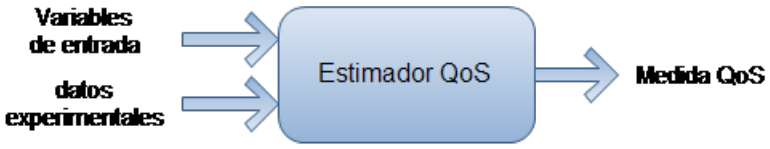


Figura 4.16.: Estimador de QoS

El elemento estimador de QoS también va a depender de la medida que trate de estimar, por lo que, de nuevo, se establece un punto común a todas las posibles medidas. La clase que representa a este elemento se muestra en la figura 4.17. Cada `EstimadorQoS` contiene la medida de QoS que debe estimar. El conjunto de variables y datos experimentales que necesite para obtener la medida dependerá del tipo de estimación.

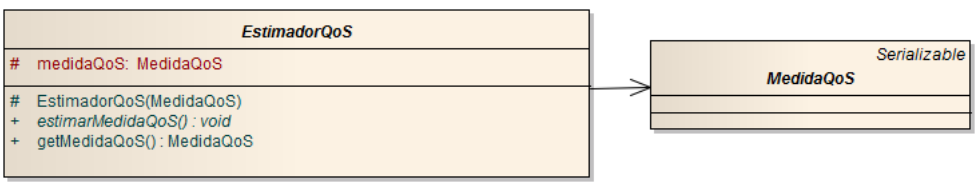


Figura 4.17.: Clase `EstimadorQoS`

La idea de crear una clase abstracta que represente una medida de QoS es debida a que la

obtención de cada parámetro requiere de un algoritmo diferente. Así las distintas clases que hereden de esta superclase deberán implementar el método abstracto `estimarMedidaQoS()` con el algoritmo correspondiente y devolver el objeto `MedidaQoS` obtenido.

4.1.2. LIBRERÍA DE SINCRONIZACIÓN TEMPORAL

Considérese el escenario en el que se desea medir el retardo extremo a extremo. Para ello se requiere información de temporización con la misma referencia temporal, es decir, debe existir una sincronización entre los relojes de los equipos. La figura 4.18 exhibe el escenario planteado.

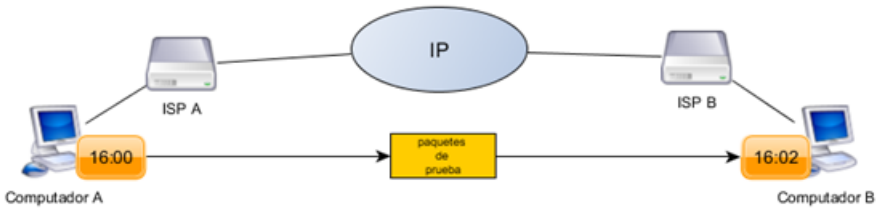


Figura 4.18.: Desajuste del *offset* en un sistema distribuido

Un computador A envía un computador B un paquete de prueba que incluye el instante de tiempo en el que se envía, $16:00$. Transcurrido un retardo de propagación T, B recibe dicho paquete en el instante $16:02+T$ y examina la marca temporal incluida. A continuación calcula el retardo extremo a extremo T_e según la expresión 4.5.

$$T_e = 16:02 + T - 16:00 = 2minutos + T \tag{4.5}$$

Lo que ocurre es que aparece un error en la estimación debido a que, en el instante en el que A generó la marca temporal, el reloj de B marcaba una hora distinta al de A, o mejor dicho, existía un sesgo entre ambos relojes.

Luego para reducir el error introducido, es necesaria la estimación del sesgo (*offset*), por lo que el procedimiento de sincronización debe de limitarse a intercambiar la información necesaria sólo para obtener dicho parámetro.

En primera instancia se consideró utilizar *Simple Network Time Protocol* (SNTP, RFC 2030 [31]). Por tanto se realizó una implementación y una serie de pruebas evaluadoras. Sin embargo dos razones provocan que recurrir a este servicio sea un inconveniente:

- La precisión varía considerablemente dependiendo de la localización de los equipos y de la sobrecarga del servidor SNTP en el momento de la petición.
- Utilizar SNTP requiere que el procedimiento de sincronización esté condicionado por la disponibilidad y correcto funcionamiento de una entidad ajena al sistema VoIPTester. Por tanto si dicha entidad falla, el procedimiento de sincronización no podrá realizarse.

Por estas razones es necesario pensar en otra alternativa para la sincronización. Sin embargo una comunicación entre los equipos sigue siendo indispensable, por lo que de nuevo una una arquitectura cliente/servidor es adecuada. Así, siguiendo el mismo criterio de diseño del protocolo de marcas de tiempo (sección 4.1.1.2), el procedimiento de sincronización debe quedar

implementado en una clase llamada `ProtocoloSincronizacion`, y que permita funcionar tanto en modo cliente como en servidor.

El protocolo de sincronización genera la información necesaria sobre la que se aplica un procesamiento para obtener el sesgo. De nuevo, siguiendo el mismo criterio empleado en la estimación de los parámetros de QoS (sección 4.1.1.5), se va a diseñar una clase que represente los parámetros de sincronización estimados y otra encargada de obtenerlos. El diagrama se muestra en la figura 4.19.

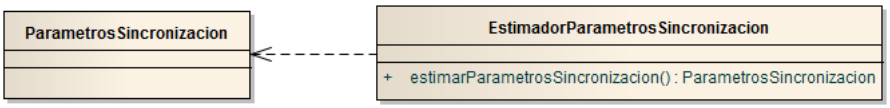


Figura 4.19.: Diagrama de clases para la obtención del *offset*

La clase `ParametrosSincronizacion` contiene los parámetros estimados (sólo es necesario el sesgo) mientras que `EstimadorParametrosSincronizacion` aplica el algoritmo para la obtención de estos. Este algoritmo dependerá del procedimiento de sincronización elegido, y que se describe en la sección 5.1.2.

4.1.3. LIBRERÍA DE GENERACIÓN DE MARCAS DE TIEMPO

Una de las funcionalidades más importantes que requiere VoIPTester es la generación de marcas de tiempo, ya que estas constituyen una parte de las medidas experimentales consideradas para las estimaciones de la mayoría de los parámetros de QoS. Estas marcas de tiempo deben tener un formato común en ambas entidades de la comunicación además de la mayor resolución posible, ya que de ella dependerá directamente la resolución de los valores de las medidas de QoS que se obtengan.

En cualquier caso, la generación de *timestamps* es responsabilidad de la clase `Temporizador`, que se muestra en la figura 4.20. Esta debe proporcionar un *timestamp* como un dato de tipo `long`, permitiendo así generar el mayor rango posible de valores.

Figura 4.20.: Clase `Temporizador`

4.1.4. LIBRERÍA DE PROCESAMIENTO DE VOZ

4.1.4.1. Propiedades de flujos de voz

Como ya se indicó en la sección 4.1.1, es necesario manejar y procesar datos de voz. La voz (y en general el audio digital) tiene unas características especiales que hacen que no pueda considerarse como datos binarios sin más, ya que se trata de datos multimedia.

Los datos de audio pueden considerarse como un flujo de bytes, pero con unas propiedades asociadas que indican como debe de tratarse. Por tanto el primer paso es definir una clase que represente estas propiedades, tal y como se muestra en la figura 4.21.

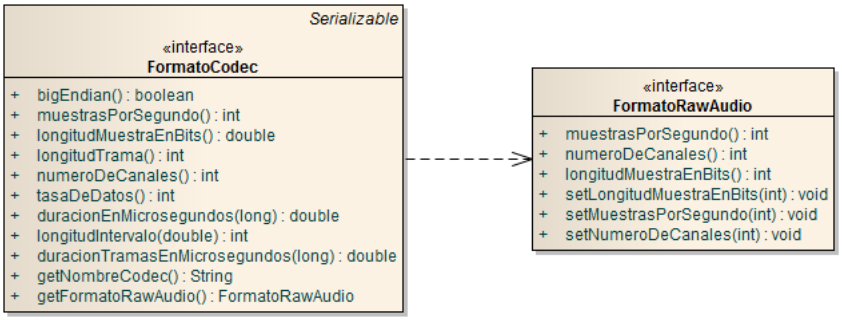


Figura 4.21.: Interfaz `FormatoCodec`

Se ha definido la interfaz `FormatoCodec` que establece qué propiedades deben de identificarse. Entre estas se tiene:

- Técnica de codificación: cómo se codifican/decodifican los datos de audio
- Número de canales: uno para sonido monofónico, dos para estereofónico, etc.
- Frecuencia de muestreo
- Número de bits por muestra
- Longitud de tramas: mínima unidad de información para todos los canales
- Tasa de tramas
- Orden de los bits: *big-endian* o *little-endian*
- Formato para el audio sin compresión (*raw* audio)

La manera de obtener estas propiedades dependerá del marco de trabajo que se utilice para procesar el audio. Además de estas propiedades es posible obtener otras propiedades implícitas. Sea F_s la frecuencia de muestreo de la señal y S_s el tamaño de la muestra en bits, entonces la tasa de datos por canal viene dada por:

$$R_s = \frac{F_s \cdot S_s}{8} \quad [bytes/s] \tag{4.6}$$

por lo que una secuencia de L $bytes$ de audio representa una señal con un canal de duración:

$$D = \frac{L}{R_s} \quad [s] \tag{4.7}$$

y una señal con un canal de duración $D\ segundos$ equivale a una secuencia de bytes de longitud:

$$L = R_s \cdot D \quad [bytes] \tag{4.8}$$

Las expresiones 4.6, 4.7 y 4.8 determinan las equivalencias entre tiempo, tramas y longitud de datos, que permiten relacionar correctamente los parámetros de transmisión (sección 4.1.1.3) con los datos de voz que se van a transportar mediante el protocolo de marcas de tiempo (sección 4.1.1.2).

4.1.4.2. Tratamiento de flujos de voz

La mayoría de las técnicas de codificación pueden encontrarse en diferentes marcos de trabajo multimedia. Sin embargo la elección de un marco u otro debe ser independiente del procesado de datos de voz que realiza el sistema.

En concreto, el flujo de procesado que se debe seguir se muestra en la figura 4.22. Simplemente se trata de aplicar un algoritmo de codificación a una señal de voz referencia (que representa una señal original sin degradación), enviar los datos resultantes mediante el protocolo de marcas de tiempo, y aplicar el algoritmo de decodificación correspondiente para obtener la señal de audio recibida (que representa una señal con posible degradación).

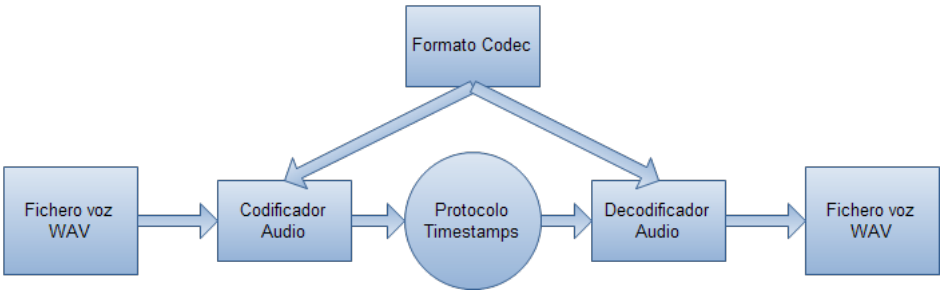


Figura 4.22.: Flujo de tratamiento de voz

Los datos de cada señal de voz se encuentran en un fichero *Wave Audio File Format* (WAV). Este tipo de contenedor permite almacenar voz sin en formato *Pulse Code Modulation* (PCM), y que se requiere para realizar el test PESQ a ambas señales.

Para conseguir el comportamiento de la figura 4.22 se va a diseñar una interfaz que permita realizar las operaciones de codificación y decodificación. El diagrama de clases se muestra en la figura 4.23. Hay que considerar que los distintos codificadores de voz generan información en paquetes de bytes de distinto o igual tamaño, llamados *buffers*, que contienen las muestras de voz comprimidas y, dependiendo de la técnica, información adicional necesaria para la decodificación. La clase `BuffersAudio` representa estas unidades de información.

Brevemente, el objetivo de cada operación:

- **Codificar**: toma el archivo de audio original en formato WAV, y devuelve el conjunto de *buffers* que contienen el audio codificado según la técnica indicada.
- **Decodificar**: toma un conjunto de *buffers* con audio codificado y devuelve un archivo de audio en formato WAV con el resultado de la decodificación. En este punto es importante señalar que el decodificador descartará los *buffers* corruptos.

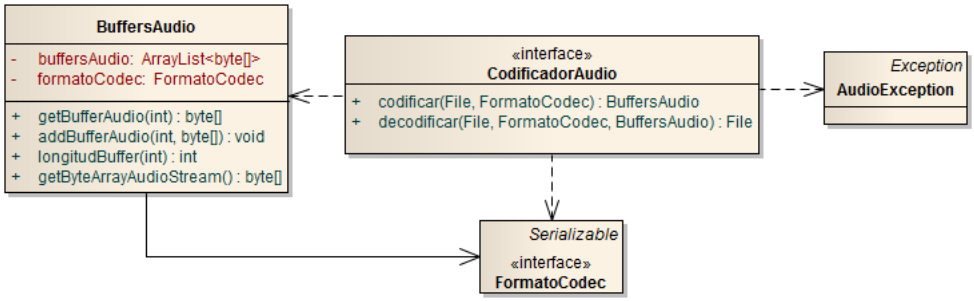


Figura 4.23.: Interfaz `CodificadorAudio`

La interfaz `CodificadorAudio` permite dejar libre la elección del marco de trabajo que se quiera utilizar para aplicar distintas técnicas de codificación.

4.1.4.3. Gestión de codecs

Una vez que ya es posible tratar con datos de voz, queda una última tarea que acometer concerniente a la gestión de los distintos codecs que pueden aplicarse. La gestión incluye seleccionar aquellos codecs que sean válidos para voz. Como cada tipo de codificación queda representada por un objeto de tipo `FormatoCodec`, el gestor de codecs debe de conocer y proporcionar el objeto correspondiente a cada códec. Para ello se crea una clase, cuyo diagrama se muestra en la figura 4.24.

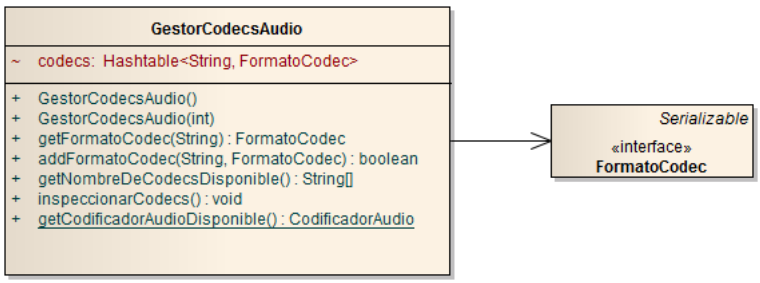


Figura 4.24.: Clase `GestorCodecsAudio`

La responsabilidad de la clase es, simplemente, mantener una correspondencia entre el nombre de un códec concreto y el objeto que lo representa. Así la clave para acceder a un códec es simplemente indicar su nombre representativo.

4.2. NÚCLEO

El conjunto de librerías descritas hasta ahora forman parte de un paquete que contiene los servicios que permiten medir una serie parámetros de calidad de servicio. Pero estos servicios hay que gestionarlos y controlarlos. Es como si de una caja de herramientas se tratase. Por sí sola no sirve de mucho si no existe un operario que pueda acceder a ella y sepa utilizar las

herramientas que contiene de una manera adecuada. Así el objetivo de diseño en esta sección es disponer de un operario. Un controlador de las librerías.

El conjunto de servicios de control y acceso a las librerías constituye el núcleo del programa. Si bien el control representa al operario que maneja la caja de herramientas (operaciones que se pueden realizar), el acceso representa la manera en que el usuario final puede disponer del operario (petición de realizar una operación). Por tanto la construcción del núcleo se va abordar desde la provisión y sustento de cada uno de estos servicios.

Esta sección se divide en tres subsecciones principales. En la primera de diseña el controlador de las librerías; en la segunda se describe un sistema de registro que acceder a dicho controlador; en la tercera se exponen otros elementos del núcleo necesarios para el correcto funcionamiento del sistema.

4.2.1. CONTROL DE LAS OPERACIONES.

Dentro del conjunto de herramientas, la llave maestra sería el protocolo de marcas de tiempo. Dentro de éste existen dos sentidos en la comunicación, tal y como se describe en la sección 4.1.1.2. Esto implica que los dos extremos que intervienen en el protocolo deben de intercambiarse los papeles dependiendo del sentido de la comunicación que se esté considerando. Por consiguiente, tanto el equipo cliente como el servidor deben de poder realizar las mismas operaciones.

Sin embargo sigue existiendo un arquitectura cliente/servidor en la que el cliente realiza las peticiones, es decir, elije las operaciones que desea realizar. Luego el cliente es el encargado de elegir qué operación y dónde se realiza. Para conseguir esto se va a utilizar una arquitectura basada en llamadas a métodos en objetos remotos.

El primer paso consiste en definir una interfaz que ofrezca las distintas operaciones que se pueden realizar en cada extremo, es decir, las operaciones remotas. La interfaz define el conjunto de operaciones comunes en ambos extremos de la comunicación, es decir, en cada nodo, y se implementará utilizando las herramientas que proporcionan las librerías. En la interfaz, `NodoRemoto`, se pueden considerar principalmente tres grupos de operaciones:

- Operaciones involucradas en la configuración y uso de los *sockets* utilizados en el protocolo de marcas de tiempo y sincronización. Es necesario ajustar los parámetros de comunicación en cada nodo y realizar las peticiones oportunas.
- Operaciones involucradas en el tratamiento de los datos de los mensajes intercambiados. Debe ser posible introducir voz en los mensajes, de tal manera que su posterior recuperación permita obtener medidas subjetivas de QoS.
- Operaciones involucradas con la estimación de parámetros. Una de las ideas que se pretende conseguir es que sólo el cliente sea el encargado de realizar las estimaciones a partir de las variables y medidas experimentales. Una parte de esta información se genera en el servidor, por lo que debe ser posible entregarla al cliente para que pueda realizar las estimaciones. De esta manera se simplifica el programa servidor en cuanto a recursos necesarios, ya que no necesita disponer ni de estimadores ni de un marco de trabajo multimedia.

Una vez definidas todas las operaciones, estas deben implementarse en una clase. En la figura 4.25 se muestra el diagrama de clases.

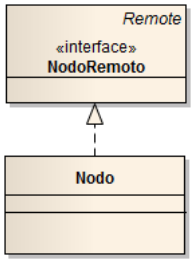


Figura 4.25.: Diseño del nodo remoto

La implementación se llevará a cabo en la clase `Nodo`. Una instancia de la misma puede constituir un objeto remoto que accesible a través de la red, y que tendrá un *Uniform Resource Locator* (URL) asociado.

En este punto hay que aclarar que el servicio de nodo remoto va a ser ofrecido por el equipo servidor. El cliente realizará las peticiones ya que este es el usuario final. Por tanto es útil disponer de una clase que controle tanto las operaciones locales como remotas. Esa es la tarea de la clase `ControlNodos`, que se muestra en la figura 4.26.

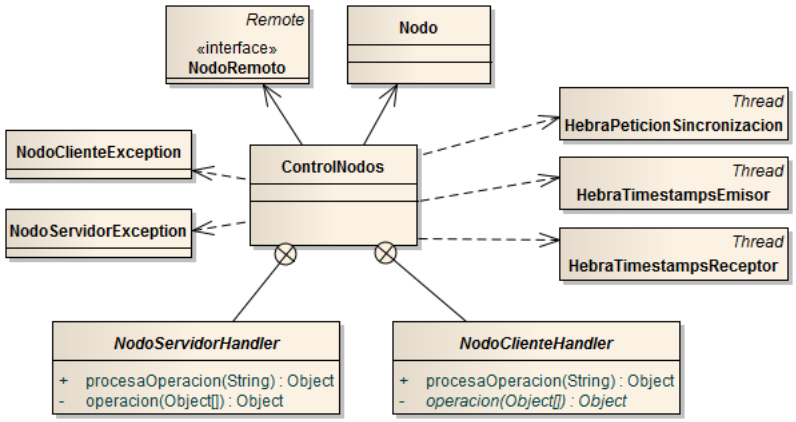


Figura 4.26.: Clase `ControlNodos`

`ControlNodos` coordina las operaciones entre ambos equipos, teniendo en cuenta el sentido de la comunicación que se quiera evaluar. Además resuelve el problema relacionado con las peticiones bloqueantes en el cliente. Un cliente que realiza una petición bloqueante o síncrona debe mantenerse a la espera de recibir la respuesta y, por tanto, el proceso que ejecuta la petición permanece bloqueado. Así, cualquier llamada a un método en un objeto remoto es de carácter bloqueante.

El problema aparece cuando tras realizar una petición bloqueante se desee retomar el control para realizar otras operaciones. Considérese el escenario en el que se desea obtener el conjunto de *timestamps* en sentido ascendente. Esta operación requiere que el servidor actúe como receptor, es decir, que permanezca a la espera de recibir paquetes; el cliente, por otro lado, tiene que actuar como emisor enviando paquetes de acuerdo a los parámetros de transmisión. El bloqueo se produce en el cliente al realizar la llamada en el objeto remoto del servidor, que en esencia consiste en que el *socket* no orientado a conexión de éste se ponga a la espera de recibir N_T mensajes (ver figura 4.8 de la sección 4.1.1.2). Pero justo después de

realizar esta llamada es necesario poder invocar otra, para que el cliente envíe N_T mensajes al servidor. Luego cada llamada debe de ejecutarse en una hebra distinta.

En la figura 4.27 se muestra un diagrama de colaboración que representa la obtención de los timestamps en sentido ascendente. En el diagrama aparecen tres grupos de operaciones:

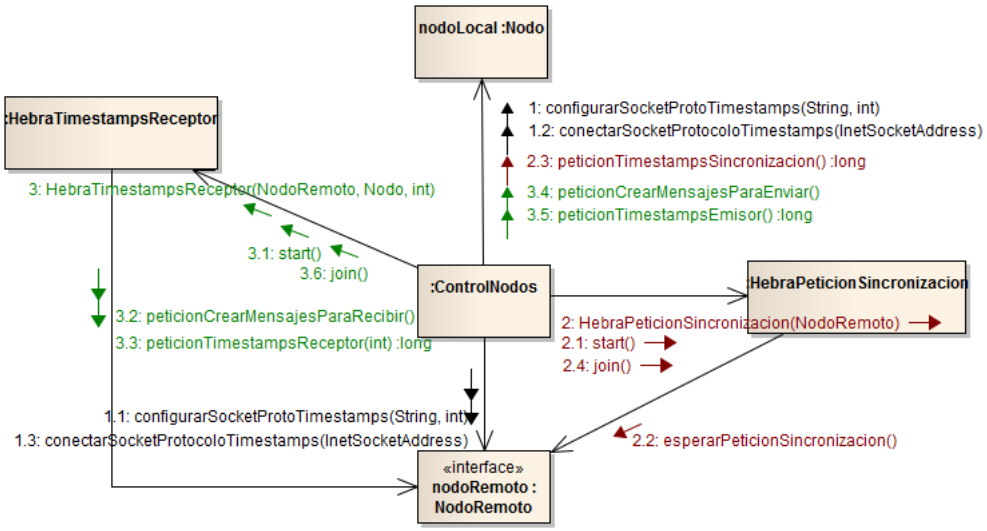


Figura 4.27.: Diagrama de colaboración para obtener los *timestamps uplink*

- **Grupo 1**: configuración y conexión de los *sockets* del protocolo de marcas de tiempo y sincronización. La conexión hace referencia a que ambos *sockets* conozcan la información necesaria (dirección y puerto) para intercambiar mensajes con el extremo.
- **Grupo 2**: procedimiento de sincronización, necesario para estimar el sesgo. Una instancia de `HebraPeticionSincronizacion` controla la petición al servidor.
- **Grupo 3**: procedimiento de generación de marcas de tiempo. Una instancia de `HebraTimestampsReceptor` controla petición al servidor.

Otros elementos que aparecen en el diagrama de clases de la figura 4.26 son dos clases abstractas anidadas, `NodoServidorHandler` y `NodoClienteHandler`. El objetivo de estas es manejar adecuadamente las operaciones, y permitir la diferenciación de una operación en el equipo servidor de otra en el equipo cliente, respectivamente.

Después del procedimiento completo, la instancia de `ControlNodos` contiene las marcas de tiempo tanto del emisor como del receptor, a las que se le puede aplicar el factor de corrección de sesgo y ser utilizadas para estimar los parámetros de QoS. Estos mismos criterios se aplican al caso de obtener las medidas en sentido descendente.

Mapeo de voz en los mensajes

Uno de los puntos clave es el transporte de voz entre los nodos, lo que permite realizar una evaluación perceptual de QoS. Sin embargo, debido a las especificaciones y diseños planteados hasta ahora, hay que contemplar lo siguiente:

- La responsabilidad del procesamiento de voz se delega al cliente, liberando de esta tarea al servidor. El sistema tiene que disponer de un marco de trabajo multimedia que contenga el conjunto de librerías que permiten aplicar las técnicas de codificación de

voz. Pero disponer de estas librerías tanto para el programa cliente como para el servidor significa aumentar considerablemente la complejidad y los recursos. Así dos escenarios son posibles, atendiendo al procesamiento indicado en la sección 4.1.4:

- Enlace ascendente: el cliente aplica la codificación y puede mapear los datos de voz obtenidos directamente en los mensajes del protocolo de timestamps.
- Enlace descendente: el servidor no dispone de codificador y por tanto es el cliente el que aplica la codificación. Los datos de voz codificados deben de enviarse al servidor mediante una llamada al objeto remoto para que éste pueda mapearlos en los mensajes del protocolo de marcas de tiempo.

- Los datos de voz se introducen en los mensajes para su envío, y se extraen de ellos tras su recepción. Este mapeo debe realizarse según los parámetros de transmisión de la sección 4.1.1.3, donde la longitud de los mensajes y la longitud del intervalo determinan este proceso.

Estas implicaciones junto con el diseño de la librería de voz de la sección 4.1.4.2, determinan la manera en la que el sistema transporta los datos de voz codificada. El proceso de mapeo se muestra en la figura 4.28. En ésta se manifiesta que los datos de voz incluidos en cada intervalo son los mismos. De esta manera la evaluación perceptual de QoS en cada intervalo se realiza en igualdad de condiciones.

Más detalladamente, el codificador de audio debe generar N_B *buffers* de voz correspondientes a la duración de un intervalo de N_k mensajes. La cadena de bytes resultante tendrá una longitud $N_s = N_k \cdot L$, y se mapea mediante el gestor de datos (ver sección 4.1.1.4) en cada uno de los N_I intervalos.

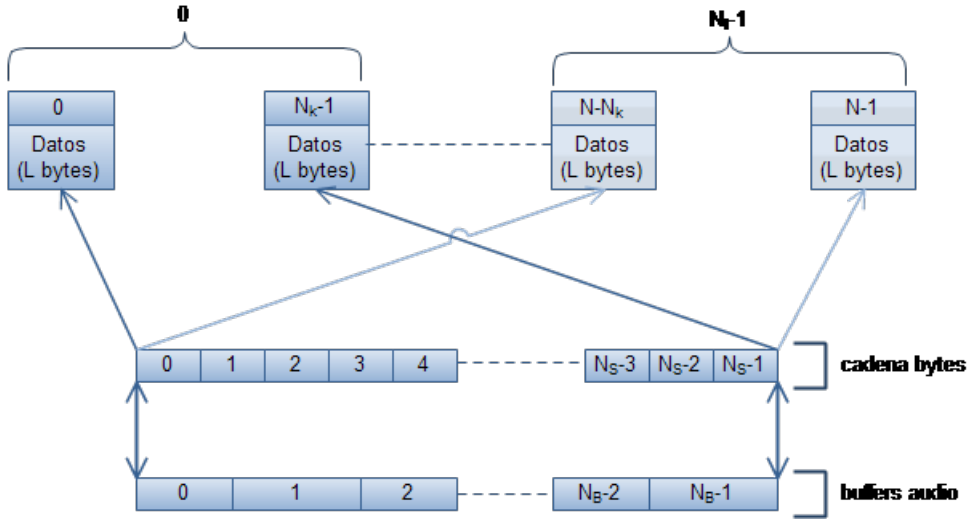


Figura 4.28.: Mapeo de voz en los mensajes

El proceso complementario, es decir, la extracción de voz de los mensajes, requiere recuperar un conjunto de N_B *buffers* por cada intervalo y realizar la decodificación a cada uno para que se genere una señal de voz potencialmente degradada. Tanto la inserción como extracción de datos son también operaciones remotas, por lo que el paso de cadena de bytes a mensajes y viceversa deberán de implementarse en la clase `Nodo`.

Por otra parte, ya que el procesado de voz debe hacerse en el cliente, la clase `ControlAudio` toma esta responsabilidad, cuyo diagrama se muestra en la figura 4.29. La clase ofrece dos

operaciones principales:

- comprimir: toma como argumentos la técnica de codificación y la duración total de audio a comprimir, y devuelve la cadena de bytes resultante que se mapeará en los mensajes del protocolo de *timestamps*.
- descomprimir: toma como argumentos la técnica de codificación y las cadenas de bytes obtenidas tras el proceso de extracción de datos de los mensajes del protocolo de *timestamps*, y devuelve un conjunto de archivos de voz en formato WAV; uno por cada cadena de bytes (intervalo) decodificada.

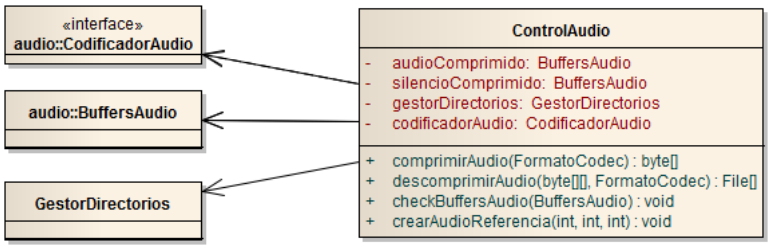


Figura 4.29.: Clase `ControlAudio`

También hay que tener en cuenta el detalle de los *buffers* de voz corruptos (debido a las pérdidas de los mensajes), por lo que es necesario disponer de un método que compruebe su validez. Cada *buffer* detectado como corrupto será sustituido por un *buffer*, de igual duración, que contiene ausencia de voz (silencio). Así se simulan los efectos de las pérdidas en una conversación de voz en tiempo real. Esta es la tarea del método `checkBuffersAudio(BuffersAudio)`.

Una última consideración es acerca de la fuente de voz a la que se aplica la compresión. VoIPTester proporcionará voz a partir de un archivo en formato WAV con unas propiedades que el usuario podrá modificar. En la tabla 4.1 se muestran estas propiedades y los posibles valores que pueden tomar. El parámetro D_{max} es la duración máxima del archivo de voz original.

Propiedad	Valores
Frecuencia de muestreo (Hz)	8000,16000
Numero de canales	1,2
Duración (segundos)	[1,D_{max}]
Bits/muestra	16

Cuadro 4.1.: Propiedades de la fuente de voz

La creación de las señales de voz fuente a partir de la señal de voz original será tarea de la operación `crearAudioReferencia(int,int,int)`. La ubicación tanto del archivo original como de las fuentes generadas la proporcionará la clase `GestorDirectorios` (sección 4.2.3).

4.2.2. ACCESO A LAS OPERACIONES

Una vez disponible el operario con las herramientas, queda proporcionar el acceso a los servicios que ofrece. El servidor proporciona un operario que debe ser compartido por todos los clientes, y estos pueden acceder a él obteniendo una referencia remota de tipo `NodoRemoto`.

Sin embargo aparece un inconveniente cuando dos clientes quieren utilizar el mismo servidor al mismo tiempo. Si esto ocurre, al iniciar el procedimiento de generación de marcas de tiempo,

los mensajes intercambiados con el equipo servidor estarían interfiriendo entre sí en la parte común de ambas rutas. En la figura 4.30 se muestra este posible escenario. Puede verse como en el tramo final se mezclan los mensajes del emisor A (mensajes rojos) con los mensajes del emisor B (mensajes verdes) y con el tráfico existente ajeno a la aplicación o *cross-traffic* (mensajes blancos).

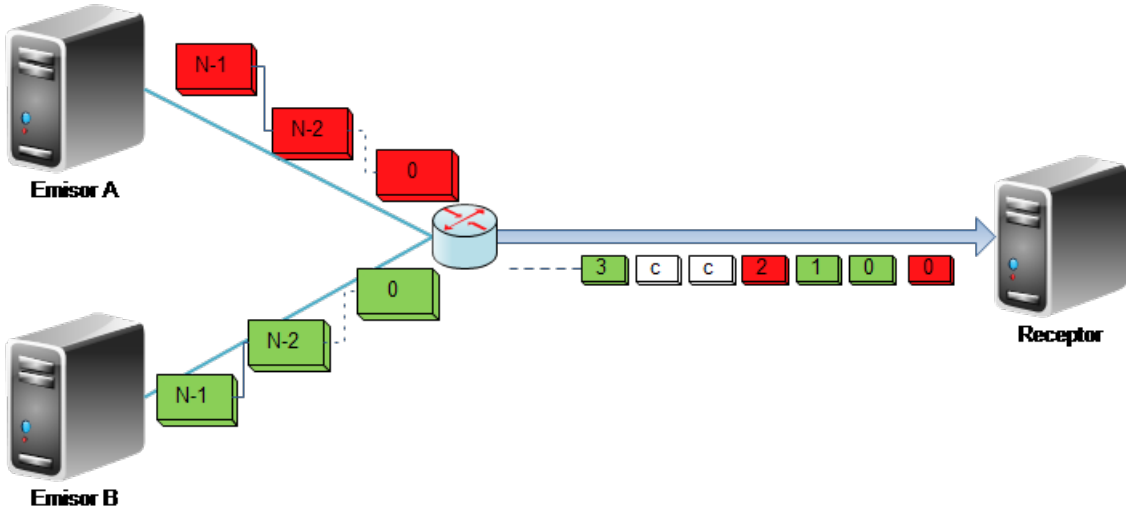


Figura 4.30.: Ejemplo interferencia de datos entre dos clientes

El problema es que la evaluación de un enlace cliente-servidor debería depender únicamente de las capacidades de todos los enlaces de la ruta y del tráfico ajeno existente. Sin embargo, al estar los dos clientes evaluando al mismo tiempo, el tráfico que genera el emisor A interfiere con el tráfico del emisor B (y viceversa), y por tanto las marcas de tiempo generadas para un cliente estarán condicionadas por el tráfico del otro cliente.

Así, para evitar esta posible interferencia se va a restringir el acceso al servidor de manera que sólo un cliente pueda obtener los servicios del operario e un instante determinado. Para conseguir esto se implanta un sistema de registro en el servidor, en el que un cliente pide obtener el nodo remoto, es decir, se registra, y ningún otro puede obtenerlo hasta que el cliente registrado lo libere, es decir, se desconecte.

En la figura 3.2 de la sección 3.2 se muestra un diagrama de caso de uso en el que se identifican las operaciones que pueden realizar las entidades involucradas. En la tabla 4.2 se describe brevemente las acciones de cada entidad.

Este sistema de registro hace que el cliente pueda permanecer en varios estados. Este proceso se muestra en el diagrama de estados de la figura 4.31.

Tres estados describen el funcionamiento del sistema de registro:

- **cliente no registrado**: El cliente no posee un nodo remoto y por tanto no puede estimar parámetros de QoS. Debe registrarse en un servidor que esté libre.
- **cliente registrado activo**: El cliente se registra y obtiene el nodo remoto que ofrece el servidor. Existe un temporizador de manera que si durante un periodo de tiempo fijado el cliente no notifica actividad al servidor, el temporizador expira y se comprueba la disponibilidad del cliente.
- **cliente registrado no activo**: Cuando el temporizador expira, el servidor debe comprobar la disponibilidad del cliente. De no ser así lo eliminará del registro, dando posibilidad

Intención del cliente	Responsabilidad del servidor
Registrarse en el servidor	Comprobar que no existe ningún otro cliente registrado, y en caso afirmativo registrar el nuevo cliente, devolviendo a este una referencia remota de un objeto tipo Nodo. En este punto, el servidor debe de comprobar de manera continua la disponibilidad del cliente, y en caso negativo desconectar al cliente y liberar el nodo asociado.
Utilizar los servicios del protocolo de timestamps a través del nodo local y el remoto	Impedir que otro cliente se registre exitosamente
Desconectarse del servidor	Eliminar al cliente del registro y liberar el nodo que poseía el cliente

Cuadro 4.2.: Acciones de cada entidad en el sistema de registro

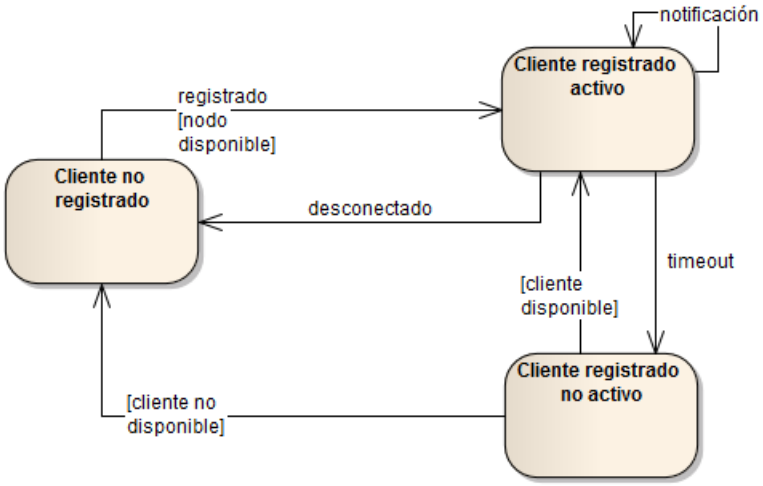


Figura 4.31.: Diagrama estados del cliente en el sistema de registro

a que otro cliente pueda acceder.

Por tanto, dos entidades están involucradas en el sistema de registro y con ello son necesarias dos interfaces, una por cada entidad. La interfaz `ServidorRegistro` ofrecerá las operaciones de registro, mientras que la interfaz `ClienteNodo` ofrecerá las operaciones necesarias para que el servidor de registro pueda enviarle la referencia del nodo remoto.

El proceso de registro debe ser sencillo, tal y como se muestra en la figura 4.32. Este proceso consiste, básicamente, en que el cliente envíe la petición de registro al servidor, este compruebe la disponibilidad, y en caso afirmativo devolver la referencia del nodo remoto. En caso negativo deberá informar al cliente de la denegación de servicio.

4.2.3. OTROS COMPONENTES DEL NÚCLEO

Los elementos de control y acceso de las operaciones descritos en las secciones 4.2.1 y 4.2.2 constituyen la estructura crítica del núcleo. Externamente a esta estructura se han considerado otros elementos importantes que ofrecen servicios complementarios para el funcionamiento

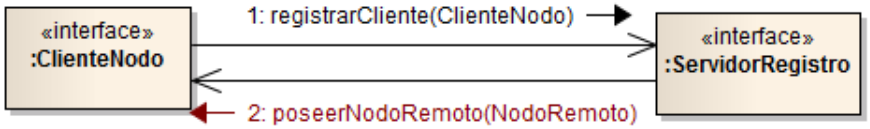


Figura 4.32.: Diagrama de colaboración para el proceso de registro

global del sistema.

Los servicios adicionales se constituyen por la gestión de los parámetros de transmisión (ver sección 4.1.1.3), que proporciona el soporte para su correcta selección, la gestión de directorios, que mantiene y localiza la estructura de directorios del sistema, y por último, la gestión de los componentes del sistema de registro.

Gestión parámetros de transmisión

Hasta ahora no se ha comentado ningún tipo de restricción acerca de los límites inferior y superior de los parámetros de transmisión, así de como la relación entre ellos. Sin embargo hay que establecer un intervalo de selección acorde a las características del sistema, tarea asignada a la clase `GestorParametrosTransmision`:

- tiempo entre envíos: el intervalo de selección debe de ser coherente con las capacidades de los enlaces actuales. La tasa de transmisión resultante debe situarse entre KB/s y GB/s.
- longitud de mensaje: está limitado por el servicio de transporte subyacente, es decir, UDP.
- número de mensajes y número de intervalos: no hay factores externos que restrinjan el intervalo de selección, pero aún así es conveniente limitar estos parámetros para evitar situaciones de sobrecarga o denegación del servicio.

También hay que tener en cuenta el tipo de datos que se van a transportar a la hora de imponer restricciones. El sistema ofrece el transporte de voz, y por tanto la selección de los parámetros debe de ser acorde con este tipo de información. Así pues, cuando en la transmisión se incluya voz habrá que considerar otros límites para los parámetros de transmisión así como relaciones entre ellos.

Los datos de audio digital se organizan en tramas (*frames*). Una trama F contiene los datos para todos los canales de audio (mono, estéreo, etc) en un instante de tiempo, constituyendo así la mínima unidad de información de audio. Luego los mensajes deben contener un número entero de tramas.

Los datos de audio tienen asociada una tasa de información o *bitrate*, medida en $bits/s$, que puede ser constante o variable según la técnica de codificación. Sin embargo VoIPTester sólo ofrece transmisión a *bitrate* constante. En cualquier caso, la tasa de información en un momento dado debe mantenerse, y por ello, la relación entre longitud de mensaje y tiempo entre envíos es constante.

La clase `GestorParametrosAudio` es la encargada de gestionar los parámetros de transmisión cuando se van a transmitir datos de voz. El diagrama completo de los gestores aparece en la figura 4.33

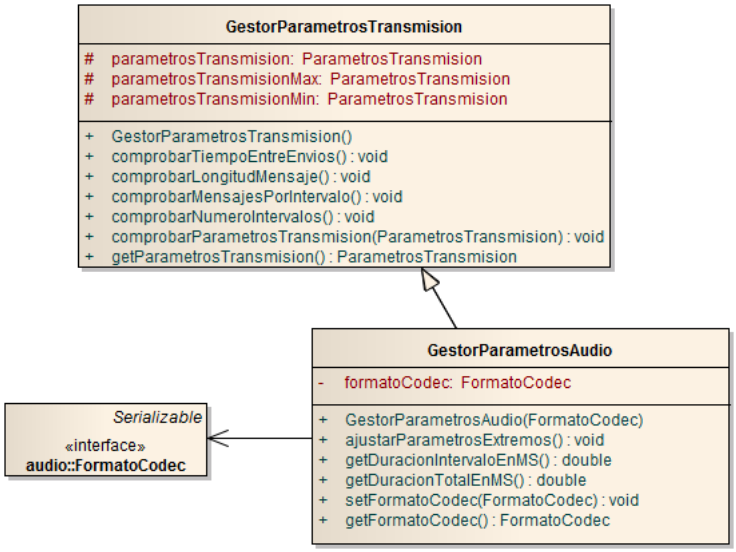


Figura 4.33.: Diagrama de clases para la gestión de los parámetros de transmisión

La fuente de voz del sistema proviene de un archivo que tiene una duración limitada. Por tanto el número de mensajes por intervalo debe ser tal que la duración de voz que incluya no supere la del archivo. En la tabla 4.3 se resume las restricciones de los parámetros dependiendo del tipo de datos que se transporte.

Parámetros de transmisión	**Intervalo**		**Cuanto**	
	Datos genéricos	Datos de voz	Datos genéricos	Datos de voz
t_e	Tal que la tasa de transmisión resultante: $R \in [KB/s, GB/s]$	Tal que la relación con la longitud del mensaje sea igual al *bitrate*	$1\ [\mu s]$	$Duración(F)\ [\mu s]$
L	Limitada por UDP y las cabeceras adicionales	Limitada por UDP y las cabeceras adicionales	$1\ [B]$	$Longitud(F)\ [B]$
N_k	Tal que no se produzcan situaciones de sobrecarga y menor que 2^{32} (identificador de mensaje)	Tal que la duración equivalente sea inferior a la duración del archivo de voz fuente y menor que 2^{32} (identificador de mensaje)	$1\ [mensaje]$	$1\ [mensaje]$
N_I	Tal que no se produzcan situaciones de sobrecarga	Tal que no se produzcan situaciones de sobrecarga	$1\ [intervalo]$	$1\ [intervalo]$

Cuadro 4.3.: Restricciones de los parámetros de transmisión

Gestión de directorios

El sistema incluye una estructura de directorios que contiene diversos recursos necesarios (por ejemplo el archivo de voz original) para el funcionamiento del sistema. Algunas clases tiene que acceder a estos recursos pero no tienen la responsabilidad de conocer dónde se encuentran. Por tanto esta responsabilidad se delega a la clase `GestorDirectorios`.

Gestión del sistema de registro

La distintas operaciones del sistema de registro deben de coordinarse para que los procesos de registro, desconexión y consulta se realicen adecuadamente. Así las clases `ControlCliente` y `ControlServidor` se encarga de controlar las operaciones del cliente y del servidor, respectivamente.

Llegado hasta aquí el núcleo queda completamente finalizado. Compuesto principalmente por dos programas, uno para la entidad cliente y otro para la entidad servidor, el enlace de transporte que se quiera evaluar queda determinando por la ubicación de los equipos donde se ejecutan estos programas. Finalmente sólo queda generar una interfaz gráfica de usuario que se apoye en este núcleo, y que ofrezca al usuario controlar tanto las conexiones como todos los parámetros involucrados en la estimación de parámetros de QoS.

5. IMPLEMENTACIÓN

En este capítulo se procede a la implementación del sistema de acuerdo a los criterios de diseño presentados en el capítulo 4. El presente capítulo se estructura en base a dos secciones principales:

1. Implementación de librerías, donde se incluye: algoritmos de las funciones estimadoras de los parámetros de QoS, cómo se generan las marcas de tiempo, el procedimiento de sincronización y el soporte del codificador de voz.
2. Implementación del núcleo, donde se describe la tecnología de invocación remota empleada, cómo se implementan los servicios remotos de acuerdo a esta, y la incorporación de un servicio de *loggin* que registre en tiempo real las distintas operaciones que realiza el sistema.

5.1. LIBRERÍAS

5.1.1. ALGORITMOS DE LOS ESTIMADORES

En la sección 4.1.1 se indicó el formato de representación de las medidas de QoS y de las funciones estimadoras. En esta sección se procede a la implementación de las funciones estimadoras y de las medidas de QoS devueltas por cada uno de los estimadores.

Esta sección se divide de acuerdo a la diferenciación de las medidas de QoS realizada en la sección 4.1.1.5. Por tanto la implementación se emprende para las medidas objetivas en primer lugar, y para las medida subjetivas en segundo. Si bien esto no representa ninguna ventaja adicional, se realiza así para que el desarrollo sea coherente a las implicaciones teóricas de los conceptos de calidad de servicio y sus recursos relacionados.

5.1.1.1. Medidas objetivas

El diagrama de clases completo de los estimadores de medidas objetivas se muestra en la figura 5.1. Cada clase hija tiene como objetivo estimar el parámetro para el cual ha sido creada, tomando las variables y medidas experimentales adicionales que necesite. El objeto devuelto tras la medida es uno de los subtipos de `MedidaQoS` mostrados en la figura 4.15, dependiendo del tipo de medida que represente.

A continuación se detallan los algoritmos que se emplean en cada una de los distintos estimadores:

`EstimadorRetardo`

Obtiene el retardo de propagación en el sentido de la comunicación que sufre cada cada paquete n, en cada intervalo $k \in [0, N_I - 1]$. Sólo se utiliza la información de las marcas de

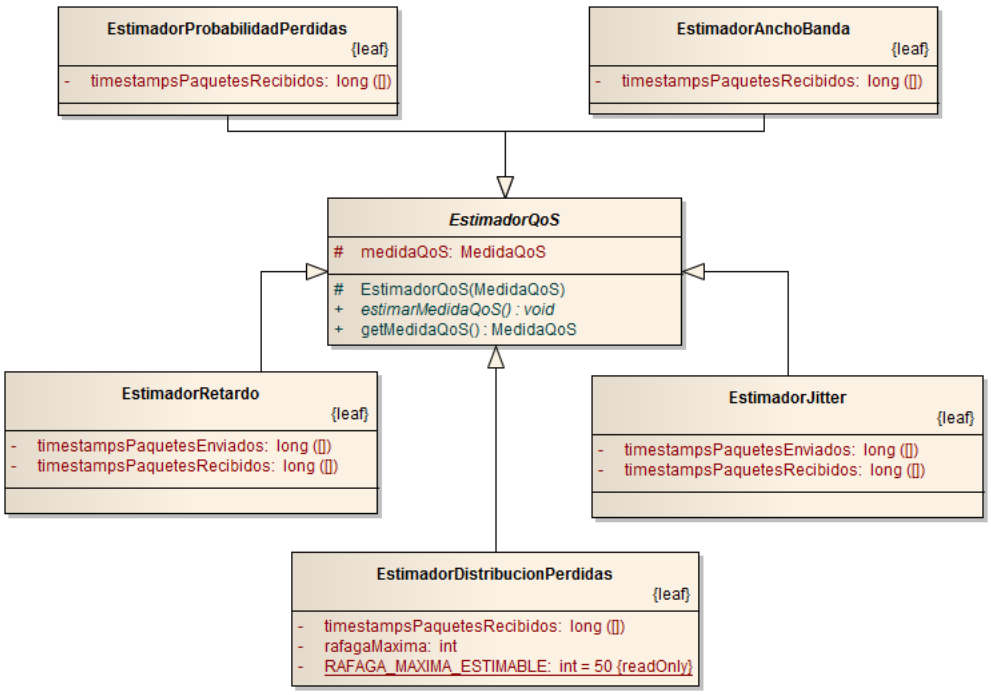


Figura 5.1.: Diagrama de clases de las estimaciones de QoS objetivas

tiempo:

$$R_k(n) = T_r(n) - T_e(n) \qquad \forall n \in [k \cdot N_k, k \cdot (N_k + 1)] \tag{5.1}$$

La ecuación 5.1, donde $T_r(n)$ y $T_e(n)$ son los instantes de recepción y envío del paquete n respectivamente, exige que los *timestamps* de emisor y receptor tengan la misma referencia temporal. De lo contrario aparecerá un término aditivo debido al desajuste. Asumiendo que $T'_e = T_e + offset$, entonces la medida del retardo viene dado por la ecuación 5.2:

$$R'_k(n) = T_r(n) - T'_e(n) = T_r(n) - T_e(n) - offset_{local} \qquad \forall n \in [k \cdot N_k, k \cdot (N_k + 1)] \tag{5.2}$$

donde aparece un error debido al *offset* existente entre los relojes de los equipos emisor y receptor.

`EstimadorJitter`

Obtiene el *jitter* o variación del retardo en el sentido de la comunicación que sufre cada paquete n, en cada intervalo $k \in [0, N_I - 1]$. Para su estimación se necesita el tiempo de recepción del paquete anterior, por lo que no puede calcularse para el primer paquete. En cualquier caso, por cada paquete n puede obtenerse un valor de *jitter,* $D_k(n)$, según la ecuación 5.3:

$$D_k(n) = (T_r(n) - T_e(n)) - (T_r(n-1) - T_e(n-1)) \qquad \forall n \in [k \cdot N_k + 1, k \cdot (N_k + 1)] \tag{5.3}$$

De igual modo, si un paquete n se pierde, no será posible calcular el *jitter* asociado al paquete $n + 1$.

`EstimadorProbabilidadPerdidas`

Obtiene la probabilidad de pérdidas en el sentido de la comunicación en cada intervalo $k \in [0, N_I - 1]$. Puede estimarse solo partir de las marcas de tiempo, de manera que una ausencia de marca implica que el paquete no se ha recibido. La ecuación 5.4 permite estimar la probabilidad de pérdidas considerando el número de paquetes que se envían/reciben:

$$P(k) = 1 - \frac{N_{recibidos}(k)}{N_k} \qquad \forall k \in [0, N_I - 1] \tag{5.4}$$

donde:

$P(k)$: probabilidad de pérdidas observada en el intervalo k

$N_{recibidos}(k)$: número de paquetes recibidos en el intervalo k

N_k: número de paquetes enviados en el intervalo k

`EstimadorAnchoBanda`

Antes de describir el algoritmo utilizado en la obtención de este parámetro conviene describir una serie de implicaciones. En primer lugar hay que distinguir dos posibles estimaciones de ancho de banda:

- **Ancho de banda esperado**: capacidad del canal que se obtiene en condiciones de canal libre (sin congestión ni pérdidas), acorde a los parámetros de transmisión. Está limitado por la capacidad máxima del canal.
- **Ancho de banda disponible**: capacidad del canal disponible en un momento determinado, debido a situaciones de congestión y pérdidas. Es siempre menor o igual que el ancho de banda máximo, y puede ser menor que el ancho de banda esperado.

El segundo de ellos requiere un análisis especial del estado de la red y existen numerosas aplicaciones destinadas a ello [32]. Por tanto el objetivo de este estimador es obtener el ancho de banda esperado, sin considerar las condiciones de la red.

La idea para obtener el ancho de banda es determinar la relación entre la cantidad de datos que se reciben y el tiempo empleado. Sin embargo para obtener esta relación hay que pensar en el comportamiento de una red de paquetes y las situaciones que pueden producirse. La figura 5.2 muestra dos posibles escenarios de un proceso de obtención de *timestamps* de receptor tras el envío de cinco paquetes.

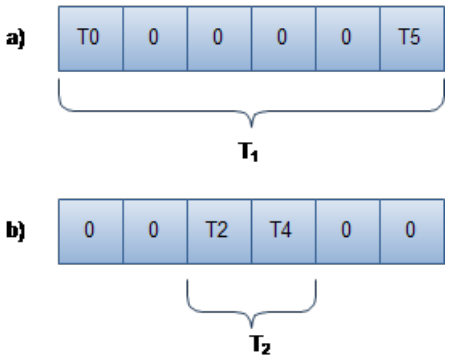


Figura 5.2.: Dos posibles vectores de *timestamps* de receptor

En ambos casos, la cantidad de datos recibida es la misma, pero no lo es el tiempo total. En el caso a), sólo están disponibles los *timestamps* primero y último, ya que el resto de paquetes intermedios se perdieron. En el caso b) también hay disponibles dos *timestamps*, pero estos son consecutivos. Sea L la longitud de los paquetes y T el tiempo total transcurrido durante la recepción, si se aplica $AB = 5 \cdot L/T$, al ser $T_2 < T_1$ el valor de ancho de banda obtenido en el primer caso será menor que en el segundo.

Teniendo en cuenta estos casos, se van a establecer dos restricciones para la estimación:

- Para cuantificar los datos que se reciben es necesario hacerlo con paquetes de datos consecutivos. Esta es la manera de no tener en cuenta las condiciones de la red en cuanto a congestión y pérdidas.
- Debido al *jitter* existente entre paquetes, pueden obtenerse valores dispares de ancho de banda dependiendo de esta diferencia de retardo. Por tanto cuanto más paquetes consecutivos se consideren menos influirá el efecto del *jitter*.

De esta forma el estimador obtiene el ancho de banda esperado en el sentido de la comunicación por cada paquete consecutivo recibido n, en cada intervalo $k \in [0, N_I - 1]$. Se requieren, al menos, dos paquetes consecutivos para realizar una estimación. Si n_0 es el primer paquete de una ráfaga de N_R paquetes consecutivos, por cada paquete n recibido puede obtenerse un valor de ancho de banda según la ecuación 5.5:

$$AB_k(n) = \frac{(N_R(n) - 1) \cdot L}{T_r(n) - T_e(n_0)} \qquad \forall n \in [k \cdot N_k + 1, k \cdot (N_k + 1)] \tag{5.5}$$

donde

$AB(n)$: ancho de banda observado en el instante de recepción del paquete n.

$N_R(n)$: número de paquetes consecutivos recibidos en el instante de recepción del paquete n.

L: longitud en bytes de los paquetes

En caso de que un paquete n se pierda, entonces se considera fin de ráfaga.

`EstimadorDistribucionPerdidas`

Obtiene la frecuencia de repetición de aparición de ráfagas de paquetes perdidos. Fijado un tamaño máximo de ráfaga N_{Rmax}, se analiza la secuencia de N_k *timestamps* del receptor en cada intervalo $k \in [0, N_I - 1]$, buscando secuencias de ceros de tamaños comprendidos en $[1, N_{Rmax}]$, con $N_{Rmax} \leq N_k$. El código que busca estas secuencias se muestra en el algoritmo 5.3.

Se trata de una función recursiva que toma como argumentos el vector de *timestamps*, la posición inicial dentro del vector donde comienza la búsqueda, y la posición final donde se detiene. El resultado que devuelve es el tamaño de la primera ráfaga encontrada en el subconjunto de *timestamps*.

Los parámetros de QoS que representan la medida son objetos de tipo `DistribucionPerdidas`, que contienen un vector de valores donde el índice i representa un tamaño de ráfaga $N_R(i) = i + 1$. Por tanto el contenido del vector viene dado por la ecuación 5.6:

$$F(N_R(i)) = n(i) \qquad \forall i \in [0, N_{Rmax} - 1] \tag{5.6}$$

```
int buscarRafaga(
                int posicionInicial,
                int posicionFinal,
                long[] timestampsPaquetesRecibidos
                ) {

        if (posicionInicial >= timestampsPaquetesRecibidos.length) {
                return 0;
        } else if (posicionInicial > posicionFinal) {
                return 0;
        } else if (timestampsPaquetesRecibidos[posicionInicial] != 0) {
                return 0;
        }

        return buscarRafaga(
                        posicionInicial+1,
                        posicionFinal,
                        timestampsPaquetesRecibidos
                        ) + 1;
}
```

Figura 5.3.: Algoritmo de búsqueda de ráfagas de paquetes perdidos

donde $n(i)$ representa el número de veces que aparecen ráfagas de tamaño $N_R(i)$.

5.1.1.2. Medidas subjetivas

Los parámetros de QoS subjetiva que el sistema ofrece para su estimación son la nota MOS del PESQ [27] y el factor R del Modelo E [28]. Las correspondientes clases estimadoras se muestran en la figura 5.4. Como ocurre para las medidas objetivas, estas clases también heredan de `EstimadorQoS`. Cada una necesita una serie de recursos adicionales debido a la gran diferencia entre un tipo de medida y otra,

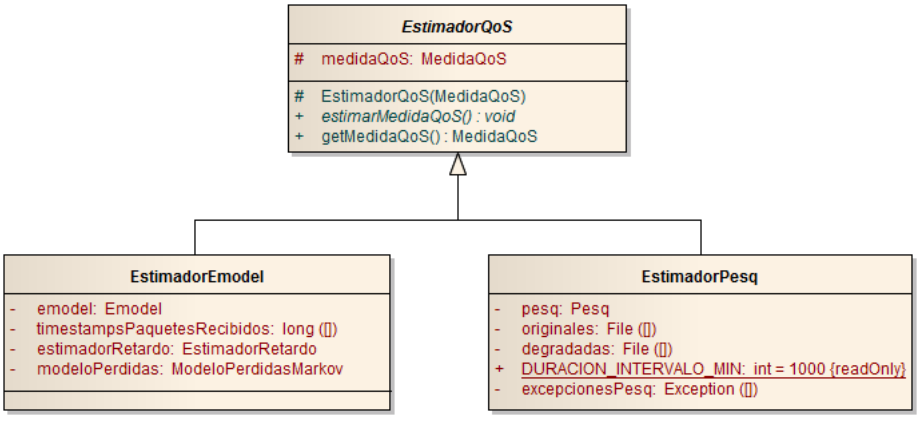


Figura 5.4.: Diagrama de clases de las estimaciones de QoS subjetivas

Obtención del PESQ

El test PESQ toma como entradas dos señales de voz: una que represente la señal original y otra que represente la señal recibida (con una potencial degradación debido a la pérdida de paquetes). Como puede ser posible realizar tantas estimaciones como intervalos indiquen los parámetros de transmisión, la clase estimadora, mostrada en la figura 5.5, debe tener acceso a estas señales.

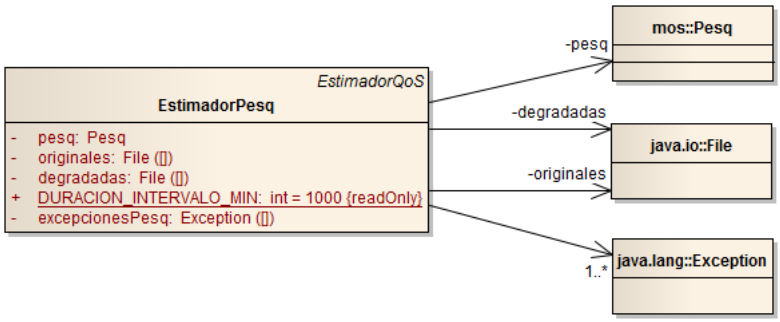


Figura 5.5.: Clase `EstimadorPesq`

La ejecución del test es responsabilidad de la clase `Pesq`, mostrada en la figura 5.6. En concreto, la clase utiliza un archivo ejecutable obtenido tras la compilación del código fuente en C que se incluye como anexo en dicha recomendación.

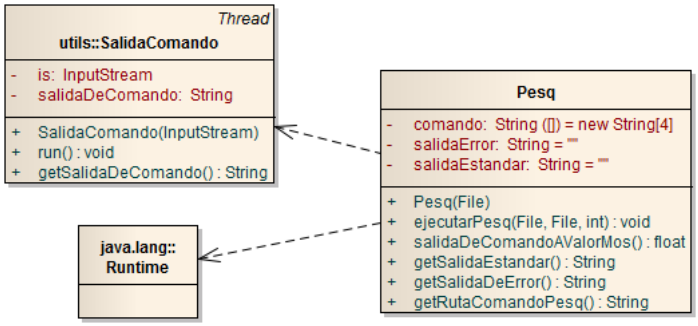


Figura 5.6.: Clase `Pesq`

El archivo ejecutable puede utilizarse como un comando, que muestra por la salida estándar la nota MOS obtenida tras la realización del test. La sintaxis del comando viene dada por:

```
pesq [ref] [deg] [fs]
```

donde

- `pesq`: nombre del archivo ejecutable
- `[ref]`: ruta de la señal de voz referencia (enviada)
- `[deg]`: ruta de la señal de voz degradada (recibida)
- `[fs]`: frecuencia de muestreo de las señales de voz

La duración de las señales comparadas debe ser la misma, y como mínimo de un segundo. Esto debe de tenerse en cuenta como una restricción en el intervalo de selección de los

parámetros de transmisión.

Esta alternativa para realizar el test PESQ permite reutilizar el programa proporcionado en la correspondiente recomendación, teniendo garantizada de antemano la validez de los resultados que proporciona.

Como se pretende que el sistema sea multiplataforma, basta con compilar el código fuente para las distintas plataformas que se soporte y elegir el ejecutable obtenido según corresponda.

Obtención del factor R

El proceso consiste en estimar el factor R del modelo E en cada intervalo $k \in [0, N_I - 1]$ teniendo en cuenta el retardo, la probabilidad de pérdidas y el tamaño promedio de las ráfagas.

$$Factor_R(k) = Factor_R(R_m(k), P(k), raf(k)) \qquad \forall k \in k \in [0, N_I - 1] \tag{5.7}$$

En concreto, el factor R puede representarse como una función según la ecuación 5.7, donde $R_m(k)$ es el retardo medio, $P(k)$ la probabilidad de pérdidas y $raf(k)$ es el parámetro de ráfaga, en el intervalo k. Así por cada intervalo, los parámetros del modelo E se fijan de la siguiente manera:

- $T(k) = T_a(k) = R_m(k)$, $Tr(k) = 2 \cdot R_m(k)$
- $ppl(k) = P(k)$
- $burstr(k) = raf(k)$

El resto de parámetros del modelo E se fijan a los valores por defecto. El retardo medio puede obtenerse mediante el correspondiente estimador mostrado en la sección 5.1.1.1. La probabilidad de pérdidas y el parámetro de ráfaga $burstr$ se obtienen a partir de un modelo de pérdidas de Markov de cuatro estados, como el que se muestra en la figura 5.7. Los estados que forman el modelo son los siguientes:

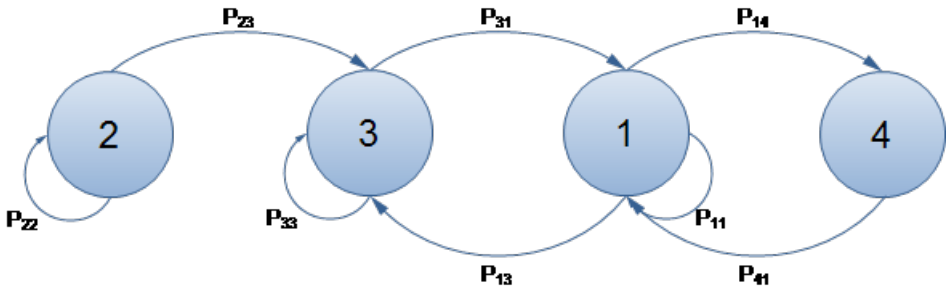


Figura 5.7.: Modelo de pérdidas de Markov con cuatro estados

- **Estado 1**: Paquete recibido correctamente (*gap*). El estado de gap se corresponde con la recepción continua y correcta de paquetes. El parámetro g_{min} establece el número mínimo de paquetes que deben de recibirse correctamente entre dos pérdidas para considerar estado de *gap*.
- **Estado 2**: Paquete recibido dentro de una ráfaga de paquetes perdidos.
- **Estado 3**: Paquete perdido dentro de una ráfaga de paquetes perdidos.
- **Estado 4**: Paquete perdido dentro del *gap*. Se corresponde con pérdida aislada.

Una descripción más detallada y un algoritmo eficiente para la implementación puede encontrarse en la especificación técnica ETSI TS 101 329-5 [36].

Finalmente la clase `EstimadorEmodel` queda tal y como se indica en la figura 5.8. Puede observarse que la clase depende de:

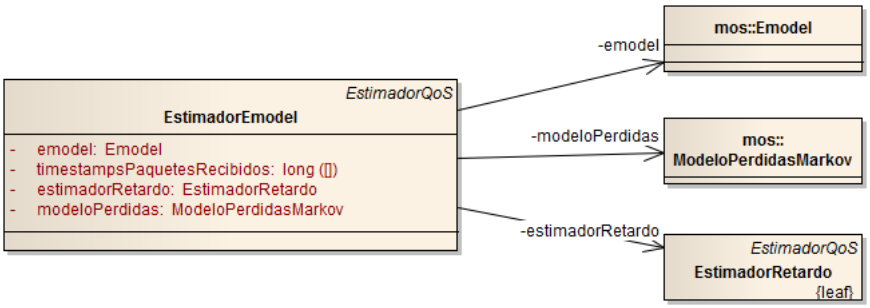


Figura 5.8.: Clase `EstimadorEmodel`

- `EstimadorRetardo`: Permite obtener el retardo medio en cada intervalo.
- `ModeloPerdidasMarkov`: Implementación del modelo de pérdidas de Markov de cuatro estados. Aplicado en cada intervalo.
- `Emodel`: Clase que implementa el algoritmo de cálculo del factor R aplicando paso a paso las expresiones del modelo E.

Con las clases mostradas en esta sección quedan definidas las herramientas necesarias para estimar los parámetros de QoS que debe proporcionar el sistema. En el caso de querer un nuevo tipo de parámetro, bastaría con añadir una nueva clase que herede de `ParametroQoS` y `MedidaQoS` para su representación, y de `EstimadorQoS` para realizar el algoritmo correspondiente.

5.1.2. MECANISMO DE SINCRONIZACIÓN

Para emprender el procedimiento de sincronización justificado en la sección 4.1.2, se ha escogido un mecanismo basado en el utilizado en SNTP, como puede verse en la figura 5.9. Lo que se pretende es obtener el sesgo del equipo cliente respecto al servidor, por lo que puede evitarse recurrir a un servidor NTP ajeno al sistema.

El proceso que se sigue es el siguiente:

1. El cliente marca el instante $T1'$ en el que envía una petición al servidor.
2. El servidor marca el instante $T2$ en el que recibe la petición.
3. El servidor marca el instante $T3$ en el que envía la respuesta.
4. El cliente marca el instante $T4'$ en el que recibe la respuesta.

Se considera que el cliente tiene un sesgo que indica la cantidad de tiempo que su reloj está adelantado o retrasado respecto del reloj del servidor, es decir $T1' = T1 + O$, donde O es el *offset*. Así, para estimar el *offset* se calcula en primer lugar los retardos extremo a extremo, según las ecuaciones 5.8 y 5.9:

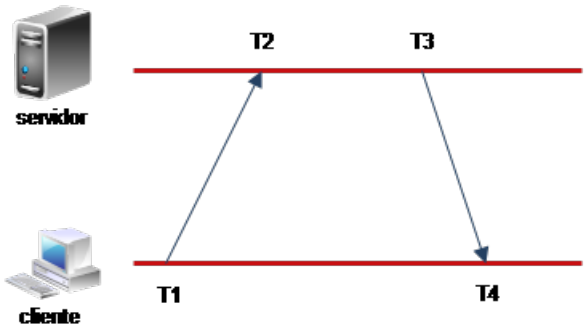


Figura 5.9.: Mecanismo de estimación del sesgo mediante *timestamps*

$$R'_1 = T2 - T1' = T2 - T1 - O \tag{5.8}$$

$$R'_2 = T4' - T3 = T4 - T3 + O \tag{5.9}$$

y la estimación del *offset* O_e viene dada por la ecuación 5.10:

$$O_e = \frac{R'_2 - R'_1}{2} = \frac{R_2 - R_1}{2} - O = e_r - O \simeq -O \tag{5.10}$$

donde aparece un error debido a la diferencia de los retardos en un sentido y en otro. Pero si se considera que ambos retardos son prácticamente iguales, el error introducido puede despreciarse. Por tanto la exactitud de este método depende de las condiciones de la red y de la diferencia de retardo que pueda existir debido a rutas o tiempos de espera en diferentes puntos de la red en cada sentido.

Si se analiza la ecuación 5.10 puede deducirse que el signo del *offset* depende de si el reloj del cliente está adelantado o retrasado, respecto al reloj del servidor. Llamando *offset* local al que sufre el reloj del equipo que lo estima, $t_1(t)$ al tiempo que marca el equipo local en el instante t y $t_2(t)$ al tiempo que marca el equipo remoto en el mismo instante t, deben cumplirse las condiciones de la expresión 5.11:

$$offset_{local} \begin{cases} < 0 & t_1 > t_2 \\ > 0 & t_1 < t_2 \end{cases} \tag{5.11}$$

Estas condiciones deben de tenerse en cuenta a la hora de corregir los *timestamps*.

Además del *offset*, a partir del procedimiento de la figura 5.9 también se puede estimar también el tiempo de ida y vuelta, RTT, que viene dado por la ecuación 5.12:

$$RTT = R'_2 + R'_1 = R_2 + R_1 \tag{5.12}$$

El tiempo de ida y vuelta proporciona información acerca de las condiciones de retardo del enlace entre los equipos, y sirve como parámetro para la determinación del tiempo de expiración de los sockets del protocolo de *timestamps* (ver sección 4.1.1.2).

La implementación del mecanismo se realiza de acuerdo a los criterios de diseño de la sección 4.1.2. El diagrama de clases completo se muestra en la figura 5.10. La clase `ProtocoloSincronizacion` es la responsable de de realizar el procedimiento de sincronización, utilizando un *socket* no orientado a conexión (`DatagramSocket`) para el intercambio de mensajes.

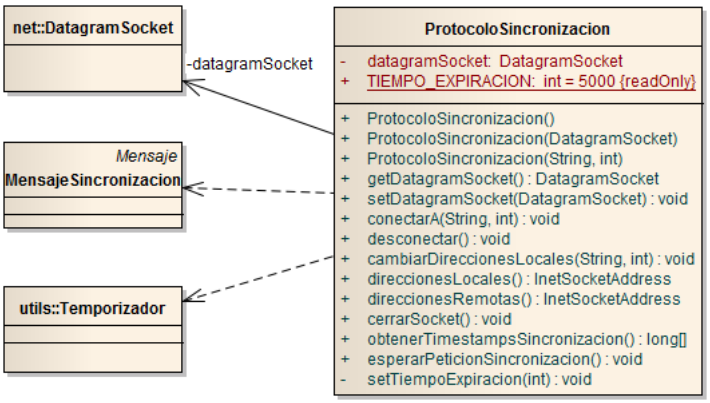


Figura 5.10.: Diagrama de clases de sincronización en tiempo

Brevemente, la tarea de cada clase:

`ProtocoloSincronizacion`

Lleva a cabo el procedimiento planteado en la figura 5.9. Los modos de operación serían los siguientes:

- **Servidor**: espera la recepción de un mensaje de sincronización, realiza los *timestamps* correspondientes, los encapsula en el mismo mensaje y lo devuelve al cliente.
- **Cliente**: envía un mensaje de sincronización y espera la respuesta del servidor, realizando los *timestamps* correspondientes.

El formato del vector que contiene las marcas de tiempo generadas se muestra en la figura 5.11.

Figura 5.11.: Formato del vector de *timestamps* de sincronización

`Temporizador`

Descrita en la sección 4.7. Lo importante es tener en cuenta que los *timestamps* son datos de tipo `long`.

`MensajeSincronización`

Define el formato de los datos de los paquetes que se intercambian. En esencia, lo importante es que el tamaño de los mensajes empleados para la obtención de las marcas de tiempo sea el mismo en ambos sentidos para que no se agrande la diferencia de retardo que pueda existir. La implementación se muestra en la figura 5.12.

Un *timestamp* es un entero de tipo `long`, cuya longitud es de ocho bytes. Luego en total hay que transmitir dieciséis bytes, correspondientes a $T1$ y $T2$. Cuando el cliente inicia la petición, estos bytes son cero.

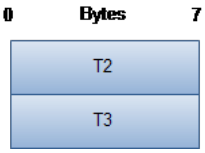


Figura 5.12.: Formato de los mensajes para la sincronización en tiempo

En cuanto a la función estimadora, basta implementar la clase que la represente (ver sección4.1.2). El diagrama de clases se muestra en la figura 5.13, donde intervienen:

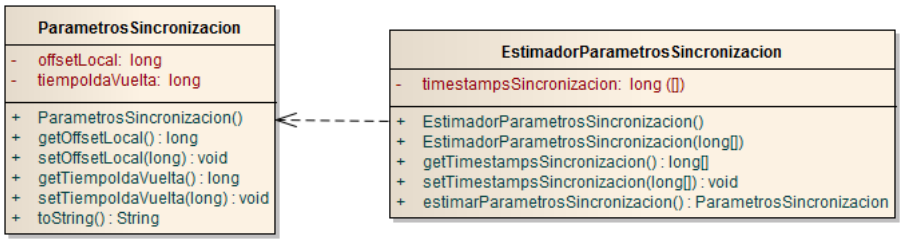


Figura 5.13.: Diagrama de clases para estimación de parámetros de sincronización

`ParametrosSincronizacion`

Representación del *offset* local y del tiempo de ida y vuelta.

`EstimadorParametrosSincronizacion`

Aplica la expresión 5.10 para realizar el calculo del *offset* y la expresión 5.12 para obtener el tiempo de ida y vuelta.

5.1.3. MARCAS DE TIEMPO

Para la generación de las marcas de tiempo descritas en la sección 4.1.3 existen dos recursos disponibles:

- **Fecha actual**. En la plataforma Java, ésta se representa como el *Universal Time Coordinated* (UTC) transcurrido en milisegundos desde el uno de enero de 1970. Puede obtenerse llamando a la función `System.currentTimeInMillis()`, que devuelve el valor como tipo `long`. La granularidad de actualización de la fecha depende de la resolución del reloj del procesador y del sistema operativo. Habitualmente este periodo de actualización es del orden unidades de milisegundos, que es mucho mayor que el periodo de oscilación del reloj del procesador.
- **Contador de *ticks* de reloj**. Los sistemas operativos suelen proporcionar la información de los *ticks* del reloj como el tiempo transcurrido desde una referencia arbitraria pero con una resolución mucho mayor. En Java esta información se obtiene llamando al método `System.nanoTime()`, que devuelve el valor del contador en nanosegundos como tipo `long`. Sin embargo la resolución real del contador depende de la resolución del contador *hardware* del sistema.

Por tanto tomando la fecha como punto inicial de partida y la información del contador del reloj, es posible generar *timestamps* en formato UTC con precisión de hasta un nanosegundo. Para ello se implementa la clase `Temporizador`, donde las dos operaciones principales son `iniciar()` y `timestamp()`. Con la primera se establece una referencia inicial y se obtiene el valor actual del contador del reloj, correspondiente al instante t_0. Esta operación es obligatoria antes de poder realizar cualquier otra operación. Después, para obtener el *timestamp* en un instante t se llama a la segunda función, que aplica la ecuación 5.13:

$$timestamp(t) = tiempo_{UTC}(t_0) + (contador(t) - contador(t_0)) \tag{5.13}$$

Luego el *timestamp* generado es un dato de tipo `long` que representa el tiempo UTC transcurrido en nanosegundos desde del uno de enero de 1970.

5.1.4. MARCO DE TRABAJO DE AUDIO

Existen varias alternativas a la hora de trabajar con datos multimedia. En Java hay dos soluciones nativas posibles, que son *Java Media Framework* [33] y *Java Sound API* [34]. Estas dos alternativas tienen muy buena integración con la JVM, pero sin embargo el soporte de codecs que ofrecen es muy reducido, y por tanto no satisfacen los requisitos del sistema.

Por consiguiente hay que buscar otra alternativa, y ésta es *GStreamer* [8]. Este marco de trabajo multimedia incluye un amplio soporte de codecs (*plug-ins*) y un conjunto de librerías de funciones que facilitan la programación multimedia, además de tener licencia de código abierto. *GStreamer* es nativo en lenguaje C, pero existe un *binding* para Java que será incluido parte de los recursos de VoIPTester.

Por tanto ya es posible implementar el codificador de audio descrito en la sección 4.1.4, utilizando la API de *GStreamer*. Para ello se redefinen las clases `CodificadorAudio`, `FormatoCodec` y `FormatoRawAudio`, tal y como se muestra en la figura 5.14.

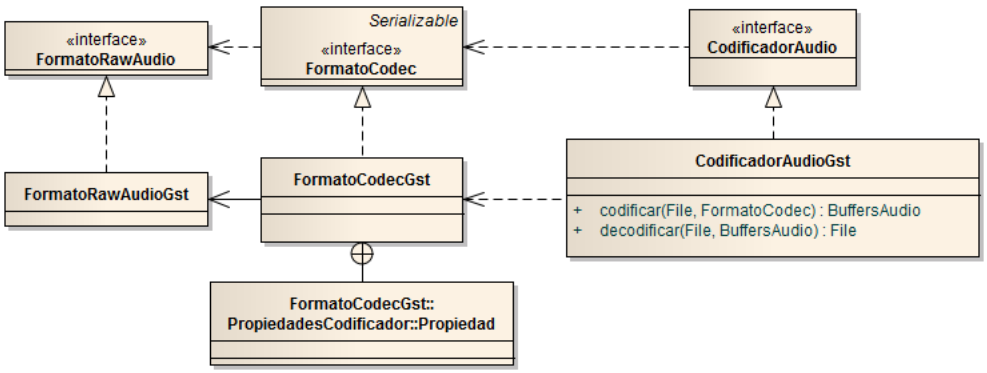


Figura 5.14.: Implementación del codificador de audio con *GStreamer*

Cada técnica de codificación tiene unas características propias y por tanto distintos parámetros ajustables. Es por ello que la clase `FormatoCodecGst` contiene una clase anidada, `PropiedadesCodificador`, que contiene las distintos parámetros (propiedades en *GStreamer)* que se pueden ajustar en el codificador. La selección del códec junto con el ajuste de sus parámetros la podrá realizar el usuario final de la aplicación

5.2. NÚCLEO

5.2.1. TECNOLOGÍA DE INVOCACIÓN REMOTA

En la sección 4.2 se impuso que el servicio de operaciones que ofrece el núcleo del sistema está basado en invocaciones a métodos remotos. Dado que el sistema se implementa en Java, existen varias alternativas. Pero una de ellas es la más adecuada por integración y sencillez: *Remote Method Invocation* (RMI).

Por tanto atendiendo al diseño de los servicios remotos, las interfaces que se encargan de ofrecerlos deberán heredar del interfaz `Remote` (tal y como se especifica en RMI).

Por un lado, la clase responsable del servicio de nodo remoto, según se indicó en 4.2.2, se muestra en la figura 5.15.

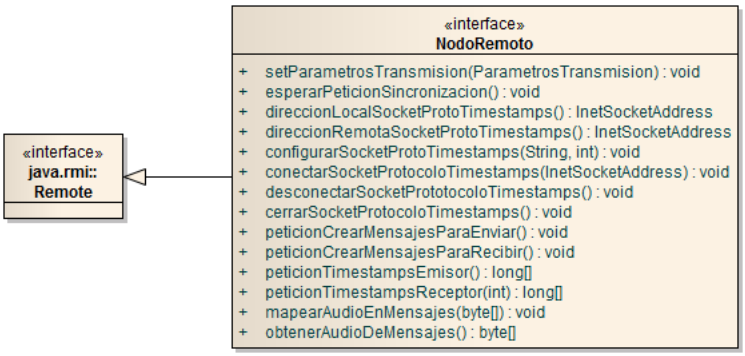


Figura 5.15.: Clase `NodoRemoto`

La interfaz `NodoRemoto` define el conjunto de operaciones remotas que estarán accesibles para el cliente, todas ellas relacionadas con el protocolo de *timestamps* y sincronización.

Por otra parte, el servicio de registro necesita de los elementos remotos, tal y como se indicó en 4.2.1. En la figura 5.16 se muestra el diagrama de clases definido.

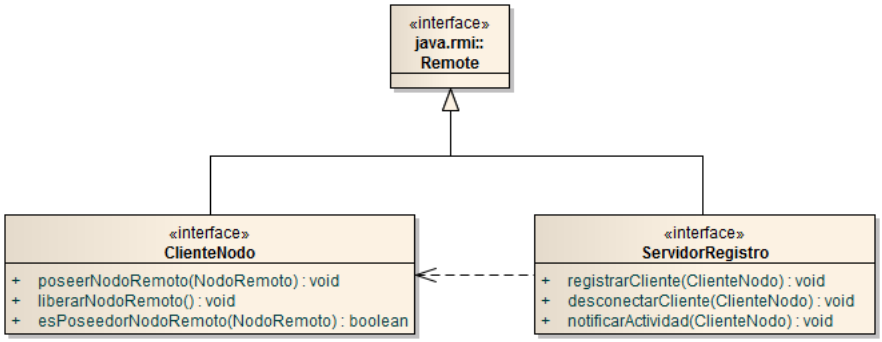


Figura 5.16.: Interfaces `ServidorRegistro` y `ClienteNodo`

El servicio de registro consta de dos interfaces, una para la entidad cliente y otra para la entidad servidor. Para un cliente dado, el servidor permite registrar, desconectar y notificar actividad. Por ello el cliente también debe ofrecer un servicio remoto, ya que el servidor debe de devolver las llamadas de éste. Es decir hay una interacción entre ambas entidades.

5.2.2. IMPLEMENTACIÓN DE LOS SERVICIOS REMOTOS

Partiendo de las definiciones de las interfaces de la sección 5.2.1, es tarea ahora de implementarlas teniendo en cuenta las librerías disponibles.

En primer lugar hay que implementar el conjunto de operaciones disponibles en cada nodo, o sea, implementar la clase `Nodo`. El diagrama de clases se muestra en la figura 5.17. Puede verse como se utilizan diversos elementos de las librerías (sección 4.1).

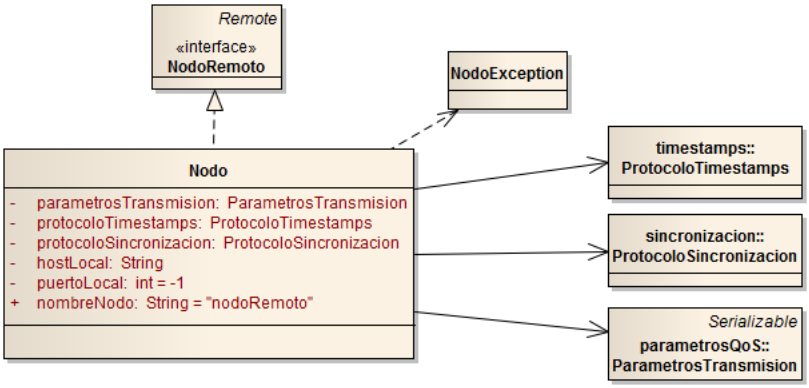


Figura 5.17.: Clase `Nodo`

Uno de los detalles a comentar es la creación de un nuevo tipo de excepción, `NodoException`, que permite abstraer de los distintos tipos de excepciones que puedan lanzar las operaciones realizadas por elementos de otras librerías.

Una instancia de la clase `Nodo` puede ser un objeto remoto, y como tal tendrá una dirección de recurso asociada. Esta dirección es una URL con formato:

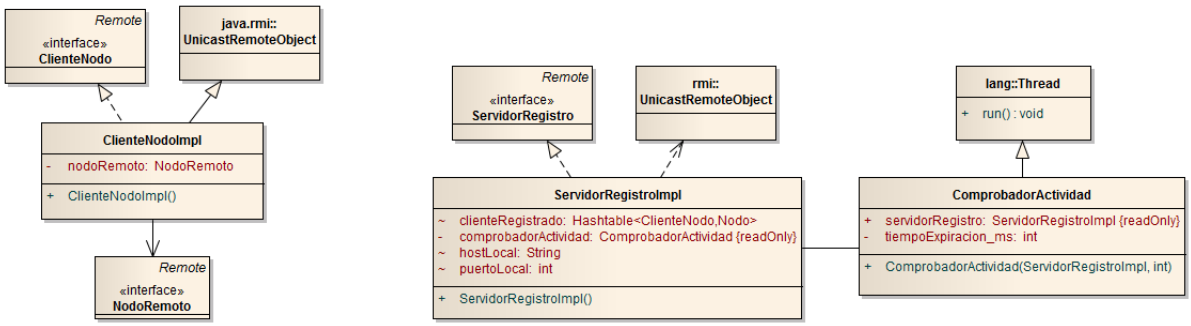
```
rmi://hostLocal:puertoLocal/nombreNodo
```

donde `hostLocal` , `puertoLocal` y `nombreNodo` son datos miembro de la clase `Nodo`, y que son configurables por el administrador del equipo servidor.

En segundo lugar, la implementación de las interfaces `ClienteNodo` y `ServidorRegistro`, correspondientes al servicio de registro, se realiza en las clases `ClienteNodoImpl` y `ServidorRegistroImpl`, respectivamente. En la figura 5.18 se muestra el diagrama de clases para cada una.

En la figura 5.18a se observa que `ClienteNodoImpl` hereda de `UnicastRemoteObject`, por lo que cada objeto que se instancie estará disponible en el registro RMI. Por defecto, el servicio remoto se ofrece a través de un puerto anónimo.

En cuanto al servidor, a figura 5.18b pone de manifiesto como `ServidorRegistroImpl` utiliza `UnicastRemoteObject`, por lo que requiere exportar el objeto al registro RMI explícitamente. Para ello es necesario especificar una URL, la cual tiene el mismo formato que el indicado para la clase `Nodo`.



(a) Clase `ClienteNodoImpl` (b) Clase `ServidorRegistroImpl` y `ComprobadorActividad`

Figura 5.18.: Diagrama de clases de la implementación del sistema de registro

El servidor de registro presenta un nuevo elemento asociado: la clase `ComprobadorActividad`. Ésta es responsable de controlar la actividad y estado del cliente, y debe de realizarse en una hebra adicional. Así el control es concurrente a las operaciones de registro, desconexión y notificación.

Finalmente, tanto la entidad cliente como servidor comparten un nuevo tipo de excepción: `RegistroException`. Esta permite representar los distintos errores que puedan producirse en las operaciones del servicio de registro.

5.2.3. SERVICIO DE *LOGGING*

Las aplicaciones *software* en general, y las aplicaciones de redes en particular, suelen incluir un servicio de registro de eventos durante el tiempo de ejecución. Estos registros contienen información relacionada con la ocurrencia de un evento.

Los registros tienen un formato estándar, legible por las personas, y que permiten corrobar el correcto funcionamiento del sistema así como detectar posibles *bugs*.

La generación de registros o *logs* puede realizarse a cualquier nivel de la aplicación, es decir, desde un nivel muy bajo que comprenda microperaciones hasta un nivel alto relacionado con las funciones generales que realiza. Seleccionar este nivel es lo recomendado por dos razones:

1. Realizar registros a muy bajo nivel puede implicar una alta carga de procesamiento
2. La comprensión de la información de los registros es más sencilla en tanto en cuanto el nivel es alto

Por tanto VoIPTester va a incluir el servicio de *logging* a un nivel alto. Exactamente, se implementa un servicio por cada aplicación entidad. Por un lado, se realizarán registros en el equipo servidor relacionados con las operaciones de conexión y desconexión de clientes. Por otra parte, en cliente se realizarán registros relacionados con las operaciones de conexión y desconexión, además de las operaciones relacionadas con las funciones del sistema (ajustar parámetros de transmisión, mapear datos, etc).

Java ofrece un paquete, `java.util.loggin`, que contiene herramientas de generación de *logs*. Por tanto se aprovecharán estas herramientas para incluir un objeto de tipo `Logger`, encargado de generar los registros, en las clases:

- Servidor: `ServidorRegistroImpl` y `ControlServidor`

- Cliente: `ClienteNodoImpl`, `ControlCliente`, `ControlNodos`

6. EVALUACIÓN

En este capítulo se procede a realizar la evaluación del proyecto con la intención de realizar una valoración objetiva sobre el resultado alcanzado. Para ello, en la primera sección se llevará a cabo una estimación de costes partiendo del tiempo y recursos empleados en el proyecto; en la segunda sección se procederá a la verificación de los requisitos del sistema; y finalmente en la tercera sección se describirán las conclusiones del proyecto así como las líneas de trabajo futuras.

6.1. ESTIMACIÓN DE COSTES

Para realizar una estimación de costes se va a tomar como punto de inicio un diagrama de *Gantt* que contiene la temporización de las distintas fases de desarrollo del proyecto. El diagrama se muestra en la figura 6.1. En el cuadro 6.1 se resume el tiempo dedicado a cada tarea.

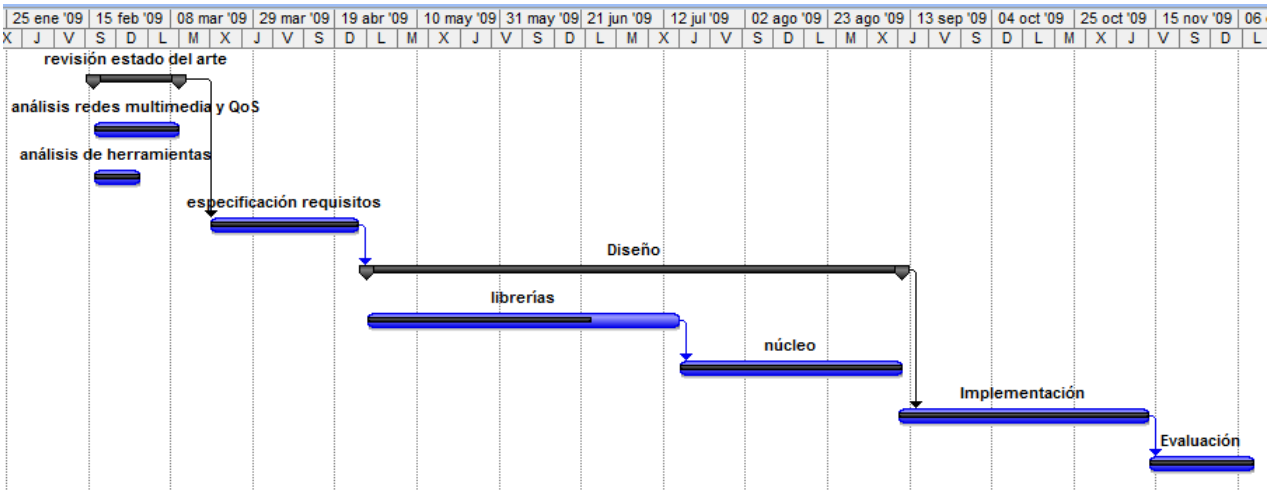


Figura 6.1.: Diagrama de Gantt

Fase	Temporización (días)
Revisión estado del arte	16
Especificación requisitos	28
Diseño	99
Implementación	46
Evaluación	19

Cuadro 6.1.: Temporización de las tareas

La fase de documentación no se ha incluido explícitamente ya que, al ser concurrente al resto

de tareas, se ha incluido como parte de estas. La estimación de los costes puede obtenerse directamente a partir del diagrama de Gantt teniendo en cuenta los recursos mostrados en el cuadro 6.2.

Recurso	coste
Desarrollador	20 €/hora (8 horas/día)
Ordenador portátil + Windows 7	950 €
Ordenador personal + Windows XP	1000 €
Librería GStreamer	0 €
Entorno de desarrollo	0 €
Sistema operativo GNU/Linux	0 €
Sistemas de evaluación	Desconocido (coste a cargo de UGR)
Paquetes de ofimática y otros	0 € (herramientas libres)

Cuadro 6.2.: Coste de los recursos asignados en el proyecto

A partir de los recursos y de la temporización se puede obtener el coste asociado a cada fase de desarrollo. La figura 6.2 indica el resumen de costes.

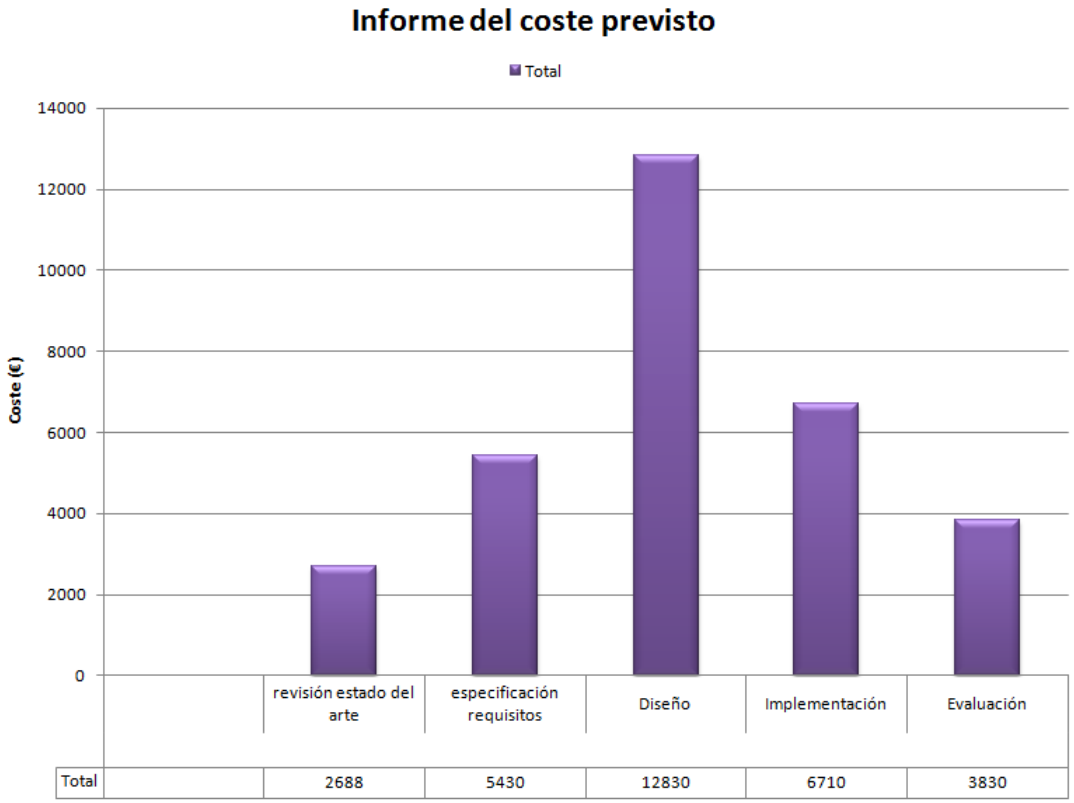


Figura 6.2.: Resumen de costes del proyecto

Hay que considerar que el coste se ve reducido significativamente por el uso de herramientas libres siempre que haya sido posible. En el diagrama de costes puede verse que las fases de diseño e implementación contienen mayor coste asociado, ya que son las de mayor duración.

6.2. VERIFICACIÓN DE REQUISITOS

La verificación de requisitos se va a realizar tomando como referencia la especificación realizada en la sección 3.3, e indicando las funcionalidades conseguidas.

Verificación de los requisitos de conectividad

El sistema ofrece un servicio de registro que permite a los clientes acceder a las operaciones disponibles en el servidor. El servicio de registro queda identificado por una dirección de red y puerto, configurables por el administrador del equipo.

El enlace a evaluar queda delimitado por la dirección y puerto de los *sockets* origen y destino. Estos parámetros pueden fijarse directamente en el programa cliente.

Verificación de los requisitos de operaciones

El sistema permite elegir el perfil de tráfico a generar seleccionando el tamaño, número y tiempo entre envíos de los paquetes a transmitir, así como el sentido de la comunicación.

El sistema permite elegir qué parámetros de QoS se quieren estimar. Los resultados se muestran mediante un gráfico de puntos para cada medida de QoS obtenida. También es posible visualizar parámetros estadísticos como los valores máximo, mínimo y medio. También se ofrece la posibilidad de exportar los datos generados a un fichero de texto en formato *Comma-Separated Values* (CSV).

Verificación de los requisitos de audio

El sistema permite seleccionar de una lista la técnica de codificación que se aplicará a los datos de voz a transmitir. Para cada técnica, además, se pueden ajustar las propiedades que estén disponibles (que depende de la implementación del marco de trabajo multimedia).

También se ha añadido la funcionalidad de seleccionar la fuente de voz original que representa los datos de voz que serán codificados.

El sistema se presenta mediante una interfaz gráfica basada en ventanas. Si bien el uso de los programas es sencillo e intuitivo, se ha añadido un tutorial de uso en el apéndice A que describe los pasos necesarios para la puesta en marcha del sistema e ilustra el funcionamiento mediante un ejemplo real.

6.3. CONCLUSIONES Y TRABAJO FUTURO

El trabajo actual ha conseguido cumplir los objetivos propuestos con las siguientes conclusiones:

- Realizar un sistema evaluador de QoS fiable y preciso es complejo, no por el modelo de obtención de parámetros y diseño de la arquitectura, sino por la intervención de factores externos, como son el *hardware* y *software* adicionales, que afectan directamente al rendimiento de la herramienta. Así mismo existen diversas metodologías de estimación de QoS, y elegir una u otra depende de la arquitectura y objetivos del sistema.

- Dada la arquitectura modular del sistema desarrollado, el código generado es altamente reutilizable, por lo que se pueden integrar nuevas funcionalidades, como son la evaluación de QoS subjetiva para vídeo. Así puede ser posible utilizar este proyecto como punto de partida para nuevos proyectos relacionados con la transmisión de vídeo, lo que podría proporcionar una solución única para la monitorización en redes multimedia.
- Utilizar componentes externos al proyecto (como por ejemplo el marco de trabajo multimedia) en lugar de realizar implementaciones nuevas es algo prácticamente obligatorio. De lo contrario los costes y la temporización se incrementarían considerablemente, lo que va en contra del ciclo de vida de desarrollo de un proyecto. De aquí que se comprenda y valore el requisito de hacer un sistema, módulo o librería reutilizable.
- La realización de un proyecto de este tipo enriquece al equipo desarrollador tanto profesional como personalmente. Reaccionar ante situaciones inesperadas, dificultades que aparecen en el camino y resolución de problemas crean un grado de experiencia. También contribuye a la capacidad de detectar los puntos fuertes y débiles de la solución llevada a cabo, lo que significa información aplicable en proyectos posteriores.

Aunque el sistema implementado es completamente funcional, podría ser mejorado en los siguientes aspectos:

- Añadir otras arquitecturas de red. La herramienta presentada permite la evaluación extremo a extremo de un enlace. Sin embargo puede ser de gran interés considerar redes *multicast*, propias en los servicios de multiconferencia. Para ello pueden considerarse nuevos protocolos de marcas de tiempo que utilicen *sockets multicast*, o bien, utilizar *middleware* como *Data Distribution Service* (DDS) [10].
- Añadir nuevos parámetros de QoS. Un ejemplo podría ser la inclusión de datos vídeo, pudiendo realizar evaluaciones subjetivas como el *Perceptual Evaluation Video Quality* (PEVQ) [37]. Esta mejora junto con la anterior proporcionaría una solución completa y potente para el análisis del *streaming* multimedia.
- Aumentar el soporte de códecs. Si bien el soporte ofrecido por *GStreamer* es bastante amplio, existen aún más codecs disponibles que no incluye como por ejemplo *Internet Low Bit Rate Codec* (iLBC, RFC 3951 [38]). Además también puede considerarse el uso de códecs de *bitrate* variable, aumentando aun más la capacidad de análisis de QoS.
- Incluir *streaming* en tiempo real. Una posible mejora es incluir transmisión continua de datos multimedia, y realizar evaluaciones en tiempo real. Con esta funcionalidad, la herramienta podría utilizarse para monitorizar, por ejemplo, multiconferencias en vivo.

Bibliografía

[1] William Stallings: *Comunicaciones y redes de computadores*. Prentice Hall, 5ª ed, 1997.

[2] William C. Hardy: *Mesaurement and evaluation of Teleccomunications quality of service*. Wiley, 2001.

[3] *An architecture for differentiated services*, IETF, RFC 2475, 1998. Disponible en: http://www.ietf.org/rfc/rfc2475.txt

[4] Java: *write once run everywhere*. Página web: http://www.java.com/es/

[5] OMG *Unified Modeling Language*. Página web: http://www.uml.org/

[6] Object Management Group: *we set the standar*. Página web del grupo: http://www.omg.org/

[7] *Netbeans IDE*. Página web: http://netbeans.org/

[8] G*Streamer: Open Source Multimedia Framework*. Página web del proyecto: http://www.gstreamer.net/

[9] OMG *Workshop on Real-time, Embedded and Enterprise-Scale Time-Critical Systems*. Página web: http://www.omg.org/news/meetings/realtime2011/index.htm

[10] OMG *Data Distribution Service: The Open, Multiplatform, Interoperable Publish-Subscribe Middleware Standard*. Página web: http://portals.omg.org/dds/

[11] *Specifications for the Network Voice Protocol*, Network Working Group, RFC 741, 1976. Disponible en: http://tools.ietf.org/html/rfc741

[12] Michael Hauben. *The history of ARPANET*. Disponible en: http://www.dei.isep.ipp.pt/∼acc/docs/arpa.html

[13] *Internet Stream Protocol (ST-II)*, Network Working Group, RFC 1819, 1995. Disponible en: http://tools.ietf.org/html/rfc1819

[14] IETF (*The Internet Engineering Task Force*). Página web del grupo: http://www.ietf.org/

[15] *Real Time Protocol: A Transport Protocol for Real-Time Applications*, IETF, RFC 1889, 1996. Disponible en: http://www.ietf.org/rfc/rfc3550.txt

[16] *SAP: Session Announcement Protocol*, Network Working Group, RFC 2974, 2000. Disponible en: http://www.ietf.org/rfc/rfc2974.txt

[17] *SDP: Session Description Protocol*, Network Working Group, RFC 2327, 2006. Disponible en: http://tools.ietf.org/html/rfc4566

[18] *SIP: Session Initiation Protocol*, Network Working Group, RFC 3261, 2002. Disponible en: http://tools.ietf.org/html/rfc3261

[19] I*EPM (Internet End-to-end Performance Monitoring).* Página web del grupo: http://www-iepm.slac.stanford.edu/

[20] OPNET *Ace Live Monitoring*. Página web: http://www.opnet.com/solutions/application_performance/appre xpert/voip.html

[21] Apparent Networks *PathView*. Página web: http://www.apparentnetworks.com/products/pathview/

[22] CA *NetQoS*. Página web: http://www.ca.com/us/netqos

[23] Telchemy *VQmon*. *Embedded Call Quality Monitoring Technology for IP Phones and IP/TDM Gateways*. Página web: http://www.telchemy.com/vqmonep.html

[24] ZTI I*P Traffic - Test & Measure*. Página web: http://www.zti-telecom.com/EN/IPTraffic_TM_KeyFeatures.html

[25] Visualware *MCS My VoIP Test*. Página web: http://www.myvoipmcs.com/index.html

[26] *M-Lab*. Página web: http://www.measurementlab.net/

[27] Rec.P.862: *Evaluación de la calidad vocal por percepción: Un método objetivo para la evaluación de la calidad vocal de extremo a extremo de redes telefónicas de banda estrecha y códecs vocales*. UIT-T , 2001. Disponible en: http://www.itu.int/rec/T-REC-P.862/en

[28] Rec. G.107: T*he E-model: a computational model for use in transmission planning*. UIT-T , 2009. Disponible en: http://www.itu.int/itudoc/itu-t/aap/sg12aap/history/g107/g107.html

[29] *UDP: User Datagram Protocol*, IETF, RFC 768, 1980. Disponible en: http://www.ietf.org/rfc/rfc0768.txt

[30] Colin Perkins: *RTP: Audio and Video for the Internet.* Addison Wesley, 2003

[31] *SNTP: Simple Network Time Protocol*, IETF, RCF 2030, 1996. Disponible en: http://tools.ietf.org/html/rfc2030

[32] Cesar D. Guerrero, Miguel A. Labrador: *On the applicability of available bandwidth estimation techniques and tools*. Computer Communications, 2010.

[33] *Java Media Framework API*. Página web del proyecto: http://www.oracle.com/technetwork/java/javase/tech/index-jsp-140239.html

[34] *Java Sound API*. Página web del proyecto: http://www.oracle.com/technetwork/java/index-139508.html

[35] *Java RMI: Remote Method Invocation*. Página web del proyecto: http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136424.html

[36] ETSI TS 101 329-5 V1.1.1: *Technology Compliance Specification; Part 5: Quality of Service (QoS) measurement methodologies*. 2000

[37] Rec. J. 247: *Perceptual Evaluation of Video Quality*. UIT-T , 2008. Disponible en: http://www.itu.int/rec/T-REC-J.247/en

[38] *Internet Low Bitrate Codec*, IETF, RFC 3951, 2004. Disponible en: http://tools.ietf.org/html/rfc3951

A. TUTORIAL DE USO

A.1. INSTALACIÓN Y DEPENDENCIAS

Aunque la aplicación aquí presentada no requiere de instalación, para su correcto funcionamiento es necesario resolver algunas dependencias. En concreto, sólo existe una dependencia para el programa cliente. Éste requiere tener instalado el marco de trabajo multimedia *GStreamer*.

El resto de librerías ajenas al sistema se incluyen dentro del mismo, por lo que su dependencia es transparente para el usuario.

Por último, es necesario tener instalada en el sistema la JVM tanto en el equipo cliente como servidor.

A.2. EJECUCIÓN

Una vez resueltas las dependencias basta con situarse en el directorio raíz de la aplicación y ejecutar el archivo `.jar` incluido para cada programa.

Programa cliente

Ejecutar el archivo `voiptester_client.jar` pinchando dos veces con el ratón. Alternativamente, desde una línea de comandos habría que escribir:

```
java -jar voiptester_client.jar
```

Programa servidor

Ejecutar el archivo `voiptester_server.jar` pinchando dos veces con el ratón. Alternativamente, desde una línea de comandos habría que escribir:

```
java -jar voiptester_server.jar
```

A.3. DESCRIPCIÓN DE LA INTERFAZ DE USUARIO

La descripción de la interfaz de usuario se va a atender para cada uno de los programas que componen VoIPTester. Para cada uno se hará una breve descripción de la interfaz, y se indicarán los pasos necesarios para realizar un proceso de estimación utilizando la herramienta en un escenario real.

A.3.1. Programa servidor

La interfaz de usuario está compuesta por dos paneles. Una barra superior de pestañas proporciona el acceso a cada uno:

1. Panel de configuración: permite ajustar la dirección de red y puerto por la que el servidor atenderá las peticiones. También permite el ajuste del periodo de actividad del cliente
2. Panel de *logging*: Muestra en tiempo real los distintos eventos de conexión que se van produciendo.

Para poner en marcha el servidor basta con seleccionar el interfaz de red y el puerto que desee para la espera de peticiones, y pulsar el botón de conexión . Una vez se inicie el servidor correctamente se habilitará el control de selección del periodo de actividad del cliente mientras que los controles de configuración de red se deshabilitan.

El estado del panel una vez esté el servidor funcionando se muestra en la figura A.1.

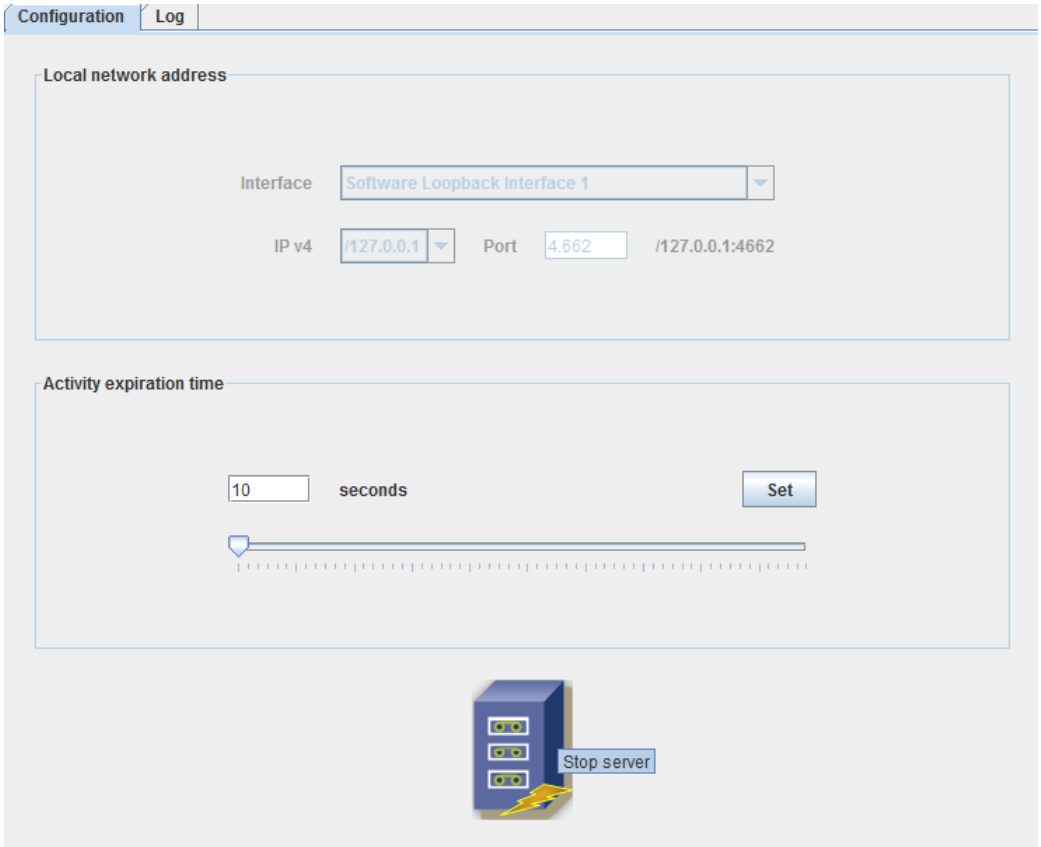


Figura A.1.: Panel de configuración de servidor (iniciado)

Para detener el servidor basta con pulsar .

Si un cliente se registra en el servidor, se mostrará la información en el panel de *loggin.* La figura A.2 muestra un ejemplo donde el servidor se inicia en la dirección `127.0.0.1:4662` y un cliente con dirección `192.168.1.114` se registra.

El panel de *loggin* también va indicando cuando se cumple el periodo de actividad del cliente.

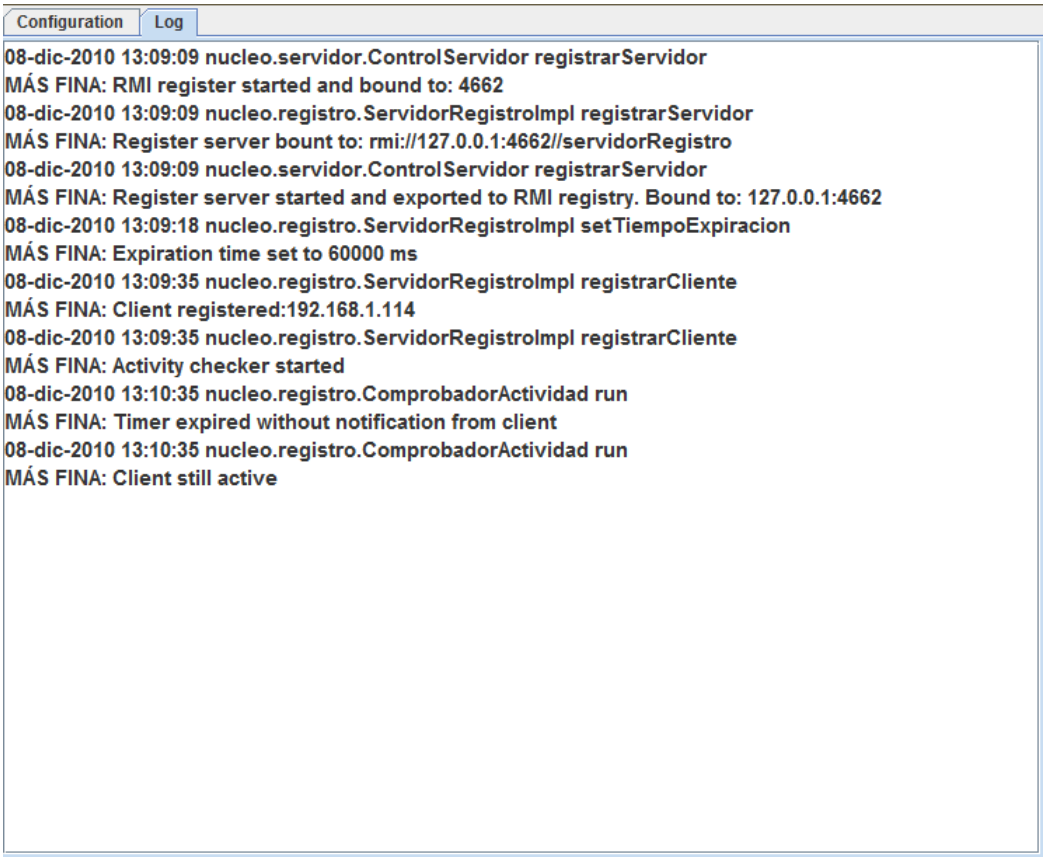


Figura A.2.: Panel de *loggin* de servidor

A.3.2. Programa cliente

La interfaz de usuario está compuesta por cuatro paneles. Una barra lateral situada a la izquierda formada por cuatro pestañas proporciona el acceso a cada panel:

1. Panel de conexiones  : permite la conexión al servidor y la configuración del enlace a evaluar
2. Panel de configuración  : permite seleccionar todos los parámetros relacionados con la estimación (parámetros de QoS, códec, parámetros de transmisión y sentido del enlace)
3. Panel de operaciones  : permite iniciar proceso de intercambio de información y de obtención de las estimaciones
4. Panel de resultados  : permite visualizar los resultados de las estimaciones

Para describir más detalladamente el funcionamiento del programa se va a ir describiendo paso a paso la puesta en marcha inicial de todos los controles para obtener un conjunto de medidas de QoS.

Primero hay que realizar el proceso de registro. El subpanel de conexión al servidor de registro se muestra en la figura A.3. Éste permite indicar la dirección y puerto del servidor, e iniciar la conexión/desconexión mediante un simple botón.

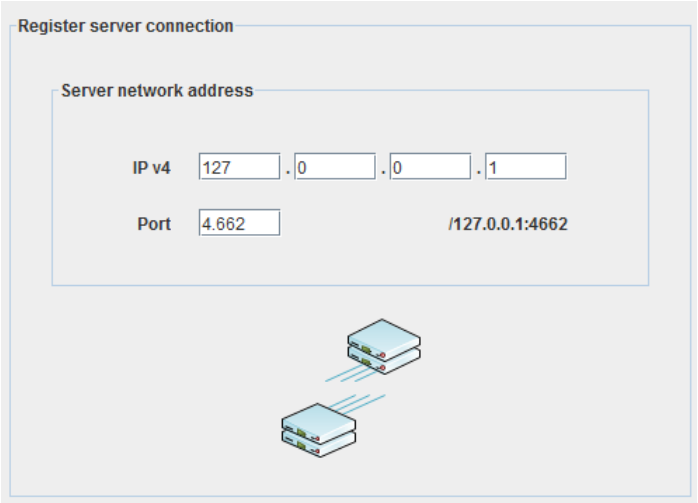


Figura A.3.: Subpanel de conexión al servidor de registro

Para conectar pulse  . Si se produce el registro correctamente, se habilitará el subpanel de configuración del enlace a evaluar. Este se muestra en la figura A.4.

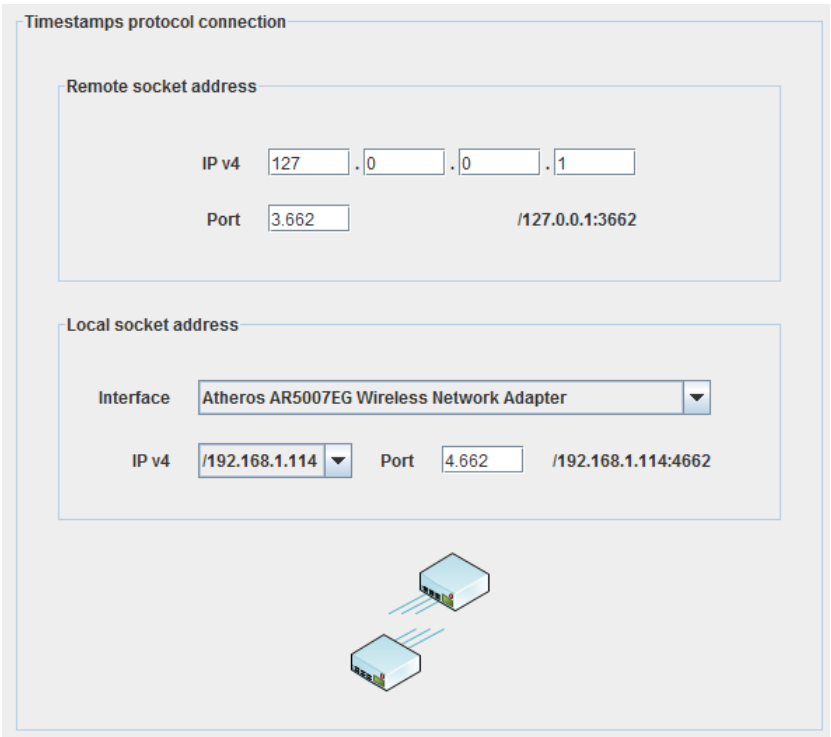


Figura A.4.: Subpanel de configuración del enlace a evaluar

Este subpanel permite seleccionar las dirección y puerto de los extremos del enlace a evaluar. Una vez seleccionados estos valores pulse  para realizar la conexión de los *sockets*. Si

ésta se produce con éxito, el panel de conexiones tendrá el aspecto de la figura A.5. Puede observarse que la pestaña de acceso al panel de configuración se habilita.

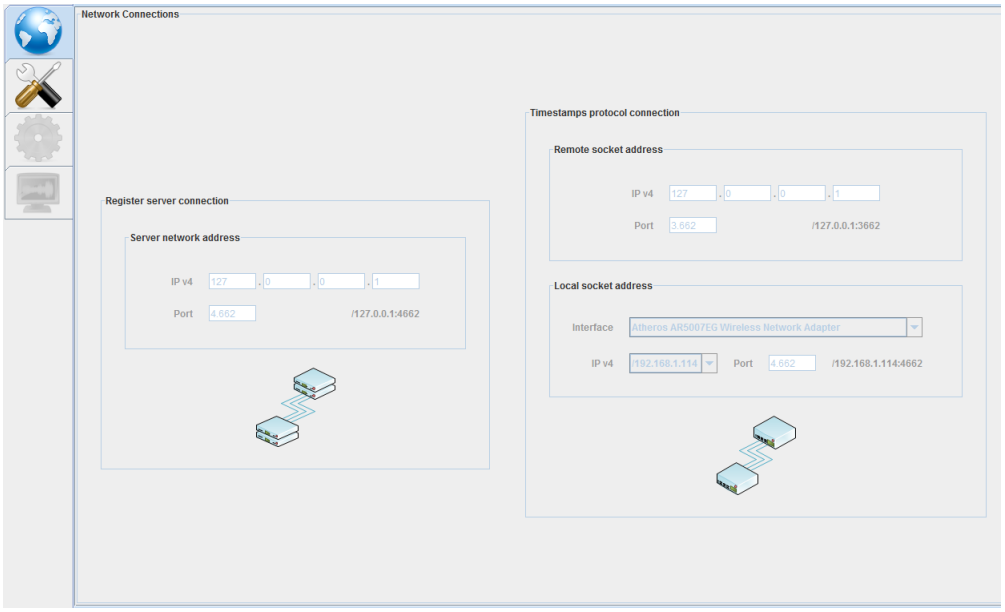


Figura A.5.: Panel de conexiones (conexiones establecidas)

A continuación hay que fijar todos los parámetros necesarios para la estimación. El panel de configuración se muestra en la figura A.6.

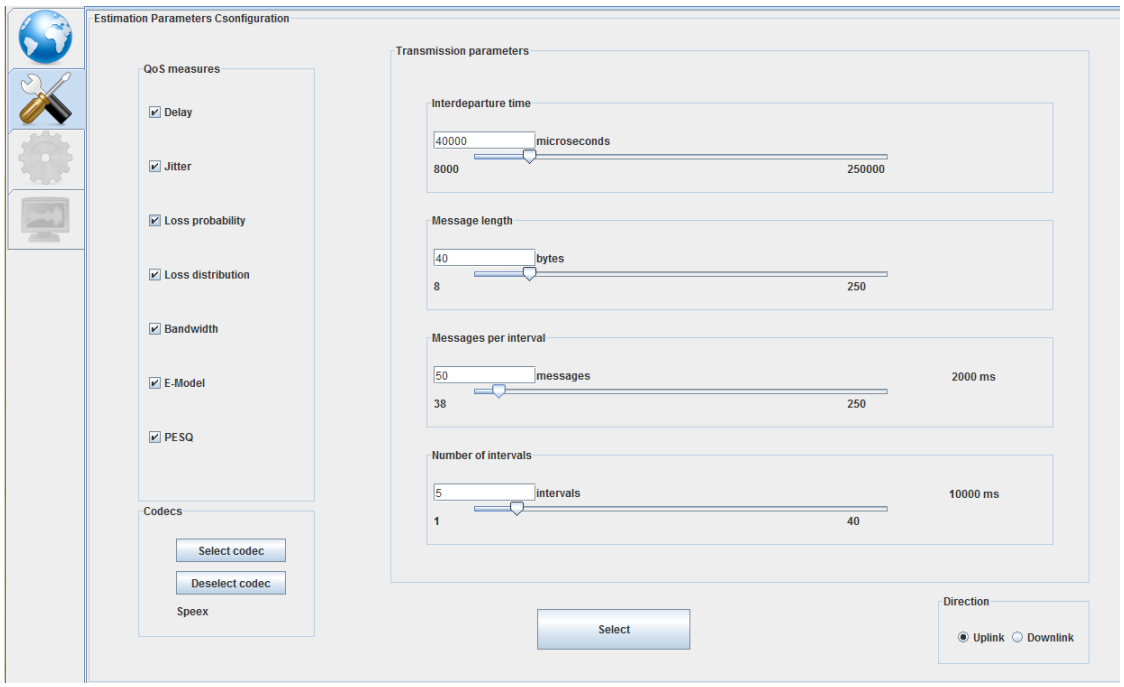


Figura A.6.: Panel de configuración

Pueden verse cuatro subpaneles:

1. Subpanel de selección de medidas (*QoS measures*): Permite elegir que parametros de QoS se quieren estimar. En el ejemplo se han seleccionado todos los parámetros disponibles.
2. Subpanel de selección de los parámetros de transmisión (*Transmission parameters*): Permite ajustar el perfil de tráfico a generar. En el ejemplo se han seleccionado los valores del cuadro A.1.

Interdeparture time	$40000\,\mu s$
Message length	$40\,bytes$
Messages per interval	50
Number of intervals	5

Cuadro A.1.: Parámetros de transmisión seleccionados

3. Subpanel de selección de dirección (*Direction*): Permite seleccionar el sentido de envío de la comunicación (ascendente o descendente). En el ejemplo se ha ajustado al sentido ascendente.
4. Subpanel de selección de códec (*Codecs*). Permite ajustar la técnica de codificación que se aplicará a los datos de voz a transmitir. Si se pulsa el botón aparecerá un diálogo como el de la figura A.7.

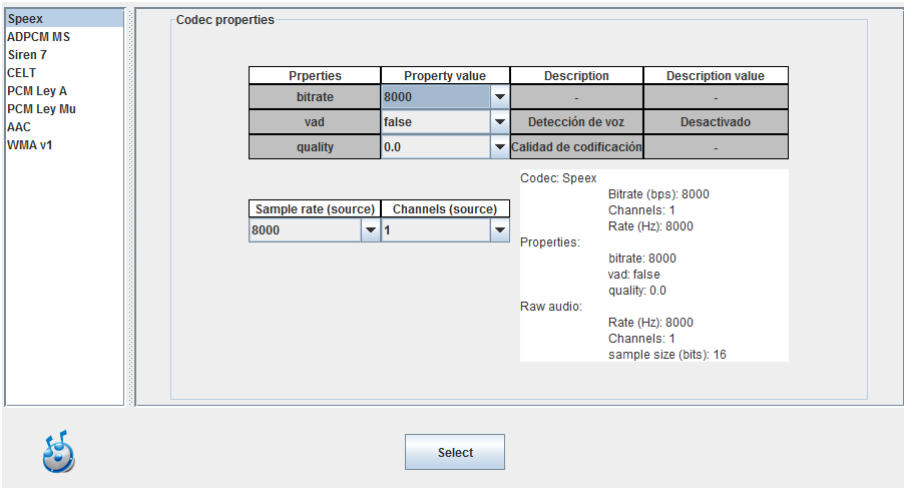


Figura A.7.: Diálogo de selección de códec

El diálogo esta dividido en dos secciones. Una para seleccionar un códec entre los disponibles, y otra con las propiedades ajustables de cada códec. Pulsando es posible elegir el archivo de audio original que servirá como fuente de audio de voz a la cual se le aplicará el códec.

Una vez estén ajustados todos los parámetros pulse el botón para confirmar. Una vez hecho esto el panel de operaciones se habilita y es posible iniciar el proceso de estimación. La figura A.8 muestra un ejemplo.

Puede verse que el panel contiene el botón iniciar/detener y una barra de progreso. Además incluye un área donde se van mostrando todas las operaciones que se realizan.

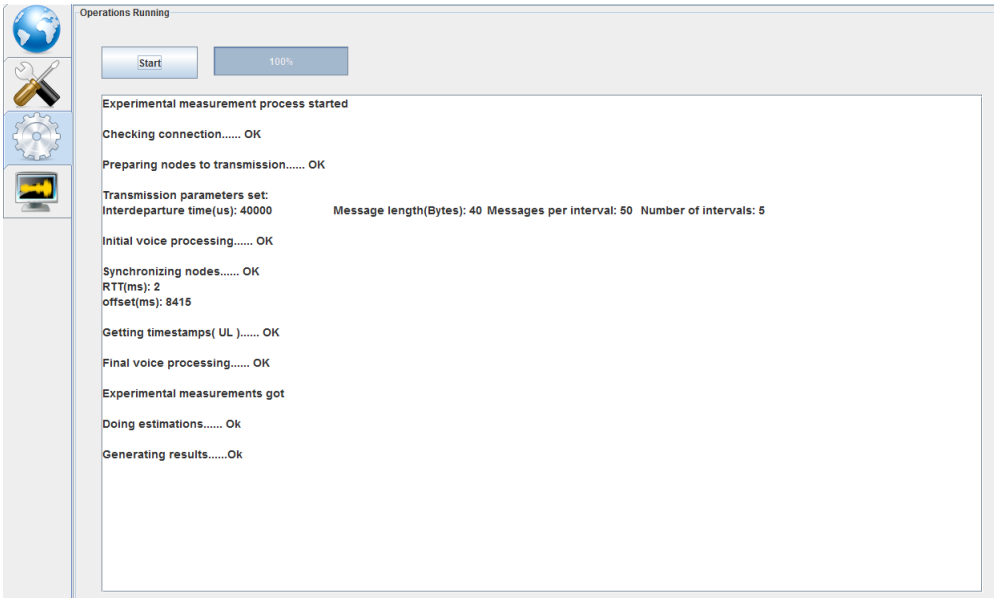


Figura A.8.: Panel operaciones

Si el proceso finaliza exitosamente entonces se habilitará el panel de resultados. La figura A.9 muestra el aspecto del panel.

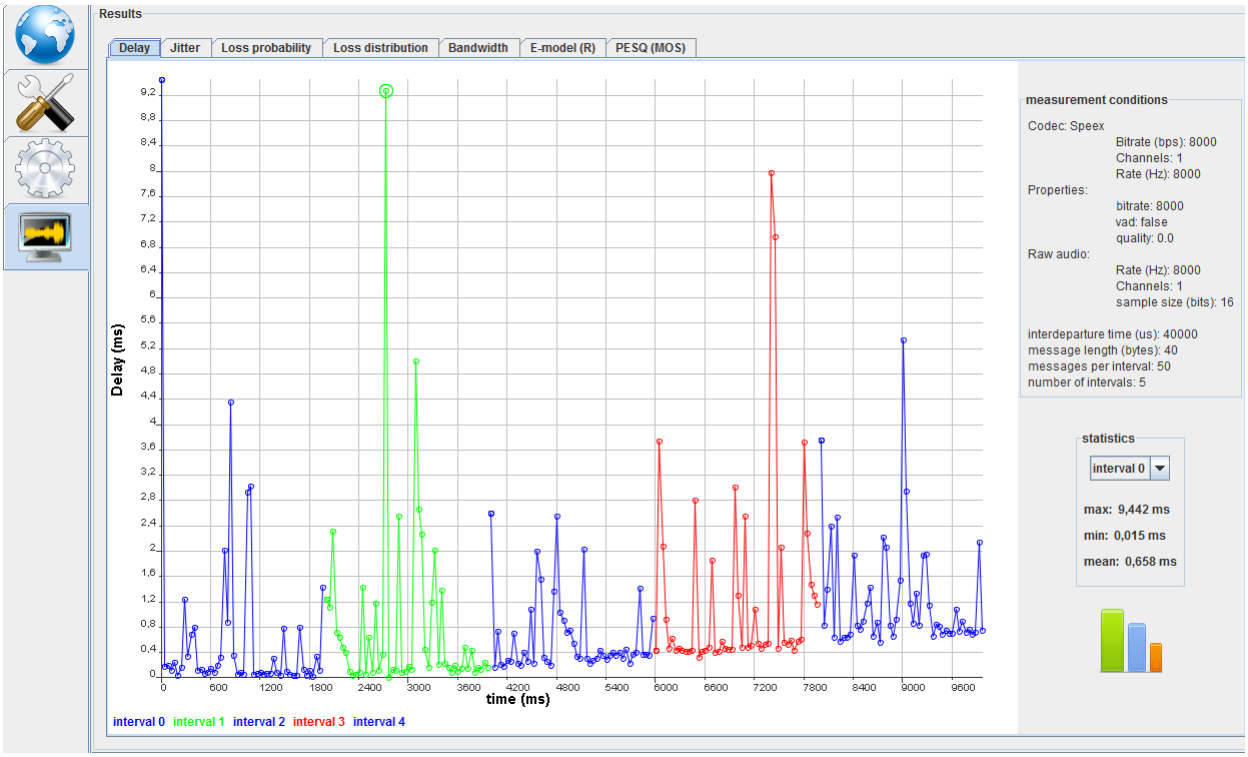


Figura A.9.: Panel de resultados

El panel contiene:

1. Una barra de pestañas en la parte superior para acceder a los resultados de cada medida. En el ejemplo de la figura A.9 se muestra la medida de retardo extremo a extremo

(*delay*).

2. Gráfico de puntos para mostrar los valores del parámetro de QoS obtenidos. El área del gráfico contiene varias opciones de presentación, como ajustar el color de los puntos para cada intervalo, seleccionar el color del *grid*, color de fondo, etc.
3. Área de resumen de las condiciones de realización de la medida, que muestra los parámetros de transmisión y códec ajustados.
4. Subpanel de estadísticas. Muestra los valores máximo, mínimo y medio para cada intervalo. Los datos generados (condiciones, valores del gráfico y estadísticas) se pueden exportar a un archivo CSV pulsando  .

ACTA DE VALORACIÓN

El tribunal constituido para la evaluación del PFC titulado:

"VoIPTester"

Realizado por el alumno Antonio Sánchez Navarro

Y dirigido por: Juan Manuel López Soler y Jorge Navarro Ortiz

Ha resuelto asignarle la calificación de:

- ☐ SOBRESALIENTE (9 - 10 puntos)
- ☐ NOTABLE (7 – 8.9 puntos)
- ☐ APROBADO (5 – 6.9 puntos)
- ☐ SUSPENSO

Con la nota[1]: ☐ puntos

El Presidente: _____

El Secretario: _____

El Vocal: _____

Granada, a 20 de Diciembre de 2010

[1] Solamente con un decimal