TRABAJO FIN DE GRADO

INGENIERÍA DE TECNOLOGÍAS DE TELECOMUNICACIÓN

Análisis del tráfico Skype

Evaluación objetiva y subjetiva para diferentes condiciones de red

Autor
César Senés Romo

Director
Jorge Navarro Ortiz

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

—

Granada, Septiembre de 2017

Análisis del tráfico Skype

Evaluación objetiva y subjetiva para diferentes condiciones de red.

Autor
César Senés Romo

Director
Jorge Navarro Ortiz

Análisis del tráfico Skype: Evaluación objetiva y subjetiva para diferentes condiciones de red

César Senés Romo

Palabras clave: *Skype*, llamada de voz, calidad de servicio (QoS), calidad de experiencia (QoE), análisis objetivo, análisis subjetivo

Resumen

En la actualidad, las comunicaciones de voz sobre IP cada vez han tomado más importancia, por lo que han aparecido numerosas aplicaciones para realizar este tipo de comunicación. Entre ellas destaca *Skype* con millones de usuarios en todo el mundo.

Por este motivo, este trabajo fin de grado se centra en el desarrollo de una herramienta capaz de evaluar, en diferentes condiciones de red, el funcionamiento de *Skype*. El estudio realizado se basa en el análisis de la calidad de servicio (QoS) y calidad de experiencia (QoE) durante una llamada de voz con *Skype*.

Para la realización de este estudio, es necesario obtener el tráfico generado por ambos extremos de la comunicación. Procesando esta información, es posible caracterizar de forma objetiva la llamada realizada (e.g. calculando *throughput*, paquetes perdidos, retardo y *jitter*). Por otro lado, es necesario almacenar el audio utilizado durante la llamadafigura en ambos extremos para su posterior procesamiento. Comparando el audio enviado con el recibido, se puede obtener una caracterización subjetiva de la llamada (e.g. mediante PESQ). Gracias a la herramienta desarrollada, se pueden introducir perturbaciones en la red para así observar cómo se comporta *Skype* en esas condiciones.

Finalmente, usando la aplicación desarrollada se han estudiado y presentado los resultados de varios escenarios. Estos escenarios se consideran relevantes para entender el funcionamiento de *Skype*, permitiendo ver cómo evolucionan la generación de tráfico y las estadísticas objetivas y subjetivas en función de diferentes condiciones de red. En particular, se ha analizado cómo afectan, por separado, las variaciones de *throughput*, del retardo y de las pérdidas de paquetes.

Analysisfigura of Skype traffic: objective and subjective evaluation for different network conditions

Cesar Senes Romo (student)

Keywords: *Skype*, voice call, quality of service, quality of experience, objective analysis, subjective analysis

Abstract

Nowadays, voice over IP communications have become increasingly important, so many applications have been developed for this type of communication. Among them, Skype is one of the most important with millions of users around the world.

For this reason, this degree thesis focuses on the development of a tool capable of evaluating, in different network conditions, the operation of Skype. The study is based on the analysis of the quality of service (QoS) and the quality of experience (QoE) during a voice call with Skype.

For this study, it is necessary to obtain the traffic generated by both ends of the communication. By processing this information, it is possible to extract objective statistics such as throughput, lost packets, delay and jitter. On the other hand, it is also necessary to store the audio used during the call at both ends. By comparing the original and the received signals, a subjective characterization of the call can be obtained. To emphasize that, during the call it is possible to introduce network impairments and thus to observe how Skype behaves in such network conditions.

Finally, by means of the developed application, the results of several scenarios have been studied and presented. These scenarios are considered to be relevant to understand the functioning of Skype, allowing us to see how traffic is generated and how objective and subjective statistics evolve depending on different network conditions. In particular, results show how different network impairments, in particular throughput, delay and packet losses, affect to these statistics.

Yo, **César Senés Romo**, alumno de la titulación Grado en Ingeniería de Tecnologías de Telecomunicación de la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación de la Universidad de Granada**, con DNI XXX, autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Grado en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

Fdo: César Senés Romo

Granada a 12 de Septiembre de 2017.

D. **Jorge Navarro Ortiz**, Profesor del Área de Ingeniería Telemática del Departamento de Teoría de la Señal, Telemática y Comunicaciones de la Universidad de Granada.

Informa:

Que el presente trabajo, titulado ***Análisis del tráfico Skype: Evaluación objetiva y subjetiva para diferentes condiciones de red***, ha sido realizado bajo su supervisión por **César Senés Romo**, y autorizo la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expide y firma el presente informe en Granada a 12 de Septiembre de 2017.

El director:

Jorge Navarro Ortiz

Agradecimientos

Quisiera gradecer a mi familia por el apoyo recibido en todo momento, a mi padre y a mi hermano por estar a mi lado siempre que los he necesitado y en especial a mi madre que también ha vivido intensamente este proyecto, preguntando todos los días y preocupándose por la evolución.

También quiero agradecer a mi tutor Jorge Navarro por el apoyo recibido en todo momento, sin duda sin su ayuda hubiera sido mucho más complicado conseguir lograr los objetivos de este proyecto.

Finalmente a Almudena que me ha ayudado con su compañía y apoyo a que estos últimos meses hayan sido mucho más agradables.

Índice general

Página

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

1.1. Contexto

VoIP (*voice over Internet Protocol*) [1], es un conjunto de especificaciones que convierte las señales de voz en paquetes, los cuales se transmiten a través de una red IP. Esto implica una gran ventaja frente a la telefonía convencional, que requiere de una red telefónica conmutada (RTC) o PSTN (*Public Switched Telephone Network*) para enviar de forma analógica las señales de voz a través de circuitos utilizables solo, por este servicio. Por este motivo, VoIP presenta como principal ventaja que no necesita mantener dos redes (voz y datos) y hace más sencillo cuestiones como la escalabilidad al compartir los enlaces entre diferentes comunicaciones, a diferencia de la conmutación de circuitos.

Por estas ventajas frente a la telefonía tradicional, han ido surgiendo numerosas aplicaciones basadas en VoIP. Estas aplicaciones son una alternativa real a la telefonía tradicional.

Este proyecto se va a centrar en *Skype* [2], una de las aplicaciones de este tipo más usadas. Los últimos datos facilitados por *Micrososft*, en Febrero de 2015, nos indican que esta aplicación la utilizan 300 millones de usuarios activos mensualmente [3], tanto para entorno empresarial como para uso doméstico.

Skype se basa en una infraestructura de *peer-to-peer*(P2P), con un único servidor de acceso centralizado (servidor de *login*), usando dos tipos de nodos:

- Nodos ordinarios: estos nodos (ordenadores, dispositivos móviles...) permiten realizar y recibir llamadas.
- Nodos de propósito especial o súper nodos: encargados de ayudar a los nodos ordinarios a conectarse con otros nodos dentro de la red *Skype*.

  El servidor de *login* es el encargado de la autenticación de los nodos antes de acceder a la red *Skype*. En la Figura1.1 [4] se muestra un

esquema de la red Skype.

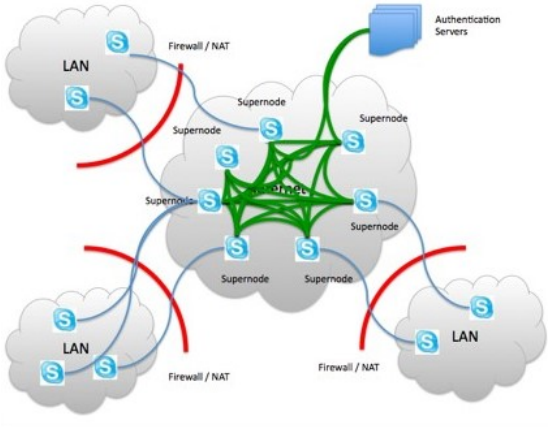


Figura 1.1: Esquema red *Skype* [4].

Una de las claves del éxito de *Skype* es el uso del códec SILK [5] (propietario y diseñado por *Skype*) que, como podemos ver en la Figura 1.2 [6] tiene un mejor comportamiento frente a la pérdida de paquetes, en comparación con otros códec usados por otras aplicaciones de VoIP.

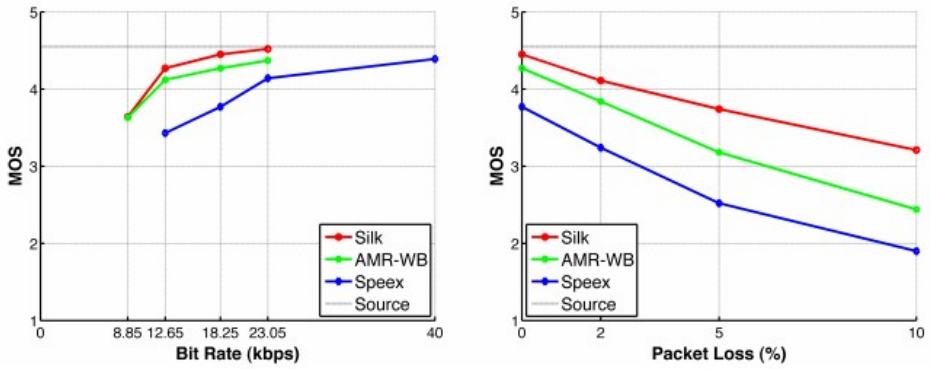


Figura 1.2: Comparación SILK con otros códec [6].

Este mejor comportamiento es debido a que usa un mecanismo para la corrección de errores denominada *Fordward Error Correction*(FEC) [7]. Este método se usa para la recuperación de paquetes perdidos y consiste en el envío de datos redundantes desde el origen hasta el destino, como se puede observar en la Figura 1.3.

Otro razón de la fortaleza del códec de audio SILK es la utilización de *bitrate* variable, es decir, es capaz de ajustar la tasa de bits y frecuencia de

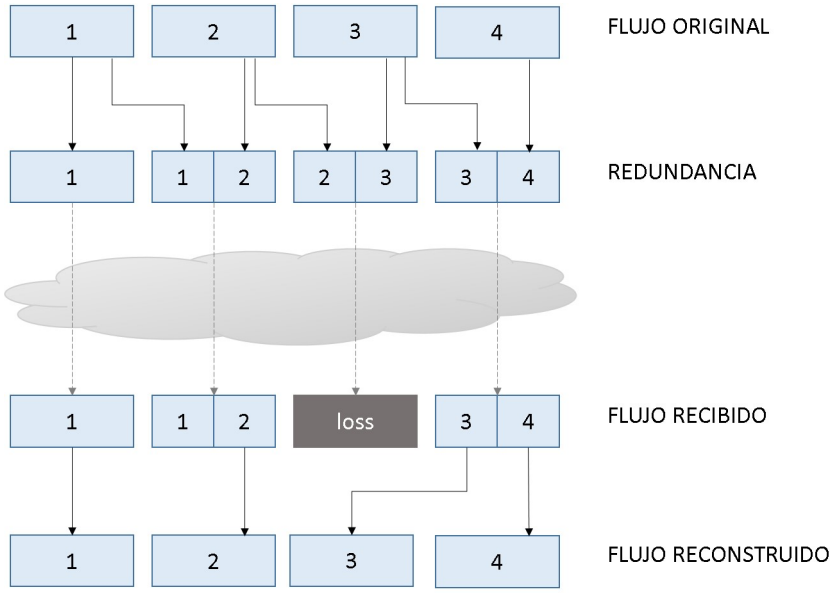


Figura 1.3: Método recuperación paquetes perdidos

muestreo dependiendo de las condiciones de la red. Como se muestra en el Cuadro 1.1 [6].

Modo	Bite Rate(Kbps)	Sample Rate(KHz)
Banda estrecha	6-20	8
Banda media	7-25	8,12
Banda ancha	3-30	8,12,16
Banda súper ancha	12-40	8,12,16,24

Cuadro 1.1: Modos operación SILK

El protocolo UDP [8] es el encargado de encaminar los paquetes generados por *Skype* por la red IP. Este protocolo es no orientado a conexión, por lo que no garantiza cuando llegará el paquete e incluso es posible que este no llegue. Además, es utilizado ampliamente por aplicaciones multimedia, ya que la velocidad de transferencia es el requisito más importante. Sin embargo, no es capaz de garantizar unas prestaciones mínimas a los diferentes servicios.

Para caracterizar objetivamente el rendimiento de una llamada VoIP, tenemos que evaluar como se ven afectados los paquetes enviados por el estado de la red.

Para el caso de una llamada, los principales parámetros son los siguientes:

- Latencia: se define como la suma de los retardos producidos por la demora en la propagación y transmisión de paquetes dentro de la red.

El valor máximo para que la llamada no se vea deteriorada es de 150 ms [9].

- *Jitter*: se define como la variación del retardo sufrido por los paquetes recibidos. Causada por congestión de red, por la pérdida de sincronización o por las diferentes rutas seguidas por los distintos paquetes para llegar al destino. El efecto de este parámetro depende principalmente del método de transporte usado.

- Paquetes perdidos: se define como el fallo en llegar al destino de un paquete o varios. El máximo valor admitido para VoIP, es dependiente del códec usado por la aplicación. Por tanto, cuanto mayor sea la compresión del códec, menor será la tolerancia de la llamada a paquetes perdidos.

Con el análisis de estos parámetros es posible determinar de manera objetiva si la aplicación estudiada cumple con los requerimientos críticos especificados para el correcto funcionamiento de una determinada aplicación. A este proceso se le denomina **calidad de servicio** (QoS, *quality of service*).

Por otro lado, no siempre el resultado obtenido del estudio de la QoS coincide con la experiencia del usuario al utilizar la aplicación. Por ello, se introduce el concepto de **calidad de experiencia** (QoE, *quality of experience*). Este concepto es puramente subjetivo y nos indica el nivel de satisfacción de un usuario al usar la aplicación. Generalmente, para medir el nivel de QoE se usa la escala MOS (*Mean Opinion Score*) [10], que es una medida que toma valores desde 1 hasta 5, como muestra el Cuadro 1.2.

MOS	Calidad experimentada
5	Excelente
4	Buena
3	Media
2	Baja
1	Muy Baja

Cuadro 1.2: Escala MOS

1.2. Motivación

En general, los servicios multimedia y, en particular, los servicios de VoIP, son aplicaciones a tiempo real, por lo que son sensibles tanto a retardos como a pérdida de paquetes, como se ha comentado en el apartado 1.1. Cuando estos servicios se transmiten a través de Internet no se pueden garantizar estos requisitos, debido a que fue diseñada inicialmente como una red *best-effort* y no incorpora mecanismos para garantizar QoS.

Dado que *Skype* es la aplicación de VoIP más popular, resulta interesante la implementación de una aplicación que permita estudiar detalladamente el funcionamiento de una llamada realizada a través de esta plataforma, así como la robustez de su códec en diferentes escenarios. Esto permitirá estudiar cómo afectan las diferentes condiciones de red tanto en los parámetros de QoS como de QoE en el usuario final.

Existen numerosas aplicaciones que monitorizan servicios VoIP pero. Sin embargo específicamente de *Skype*, el número es más reducido, debido a que el protocolo que usa *Skype* para las llamadas es cerrado y privativo. Por este motivo, las aplicaciones que analizan llamadas VoIP genéricas no suelen funcionar con *Skype*.

Por tanto, ya que *Skype* es una de las aplicaciones de VoIP más usada a nivel mundial, en este trabajo fin de grado se plantea el desarrollo de una aplicación que sea funcional y capaz de evaluar la calidad de servicio tanto objetiva como subjetiva de las llamadas a través de *Skype*.

1.3. Objetivos

El principal objetivo de este proyecto es el diseño de una aplicación cliente-servidor que permita evaluar de una forma objetiva y subjetiva la calidad de las llamadas a través de *Skype*.

Para este fin, durante la llamada, la aplicación captura los paquetes UDP generados por la llamada tanto en el cliente como en el servidor. Además se obtendrán las señales de audio generadas durante la llamada, tanto la original como la degradada por el efecto de la red. Con estos datos y mediante un post-procesado se calcularán las principales estadísticas, que se representarán gráficamente para caracterizar la calidad de la llamada. Específicamente las estadísticas que se pretenden conseguir son:

- Estadísticas objetivas (QoS): la tasa de paquetes perdidos, el retardo y el *throughtput*.
- Estadísticas subjetivas (QoE): el MOS usando diferentes métodos. El primer método de referencia completa, PESQ (*Perceptual Evaluation of Speech Quality*) [11], en el que se comparan la señal de original y la señal degradada durante la llamada. Y otro método sin referencia, el e-model [12], que usando parámetros objetivos, como son el retardo o la tasa de paquetes perdidos, permite obtener un valor en la escala MOS 1.2.

En resumen, el objetivo es reunir la máxima información posible de una llamada de *Skype* para poder caracterizarla lo más detalladamente posible y así poder realizar un análisis de cómo le afectan las diferentes condiciones de red.

1.4. Estructura de la memoria

En esta sección se va hacer una breve descripción del contenido de cada capítulo, los cuales forman este trabajo fin de grado. El objetivo de este apartado es el de tener una idea global de la estructura de este documento. La memoria de este trabajo fin de grado se divide en los siguientes capítulos:

- **Introducción**: En este capítulo 1 se indica el motivo por el cual se ha elegido este proyecto. A continuación se detalla el problema que se pretende resolver. Finalmente se enumeran los objetivos para conseguir resolver el problema planteado.
- **Estado del arte**: Este capítulo 2, describe algunas aplicaciones existentes con las que se resuelve parcialmente el problema tratado en este proyecto o problemas similares. También se recogen estudios realizados sobre el tema que trata este proyecto.
- **Planificación y costes**: En el capítulo 3, se detallan todos los recursos usados para el desarrollo de este trabajo. También se especifica y detalla la duración de cada fase de desarrollo en las que se ha dividido este proyecto. Finalmente se desglosa el coste del proyecto.
- **Especificación de requisitos**: En el capítulo 4 se describe de qué modo se va realizar la aplicación para conseguir alcanzar los objetivos planteados. Por otro lado se especifican los requisitos tanto funcionales como no funcionales que se deben aplicar en el diseño de la aplicación para cumplir con los objetivos.
- **Descripción de herramientas utilizadas**: En este capítulo 5 se realiza una descripción de las aplicaciones o recursos usados durante la implementación de la aplicación.
- **Implementación**: En este capítulo 6 se describe detalladamente la arquitectura usada por la aplicación. Finalmente se detalla paso por paso cada una de las funciones implementadas en la aplicación, para conseguir que esta cumpla con todos los requerimientos.
- **Realización de simulaciones y resultados obtenidos**: en el capítulo 7 se muestra los resultados así como un análisis detallado para varias simulaciones representativas .
- **Conclusiones y líneas futuras**: Finalmente en este capítulo 8 se recogen las valoraciones personales del autor así como posibles maneras de mejorar o completar este trabajo.

Capítulo 2

Estado del arte

En este capítulo se revisan algunas aplicaciones existentes que realizan un análisis de la calidad de servicio en comunicaciones VoIP.

2.1. Voip Tester

Voip Tester [13] es una aplicación diseñada para evaluar la calidad de voz sobre redes IP, diseñada e implementada durante un Proyecto Fin de Carrera realizado en la Universidad de Granada.

VoIP Tester está desarrollado en *JAVA* con una arquitectura del tipo cliente-servidor. Posee un sistema de paquetes totalmente configurable, ya que parámetros como el tamaño de los paquetes o tiempo entre paquetes son especificados por el cliente.

Para que las estadísticas obtenidas sean aplicables a una red real, requiere un estudio previo del tráfico VoIP sobre esa red. Después hay que ajustar los parámetros del flujo VoIP, ya que *VoIP Tester* no trabaja con flujos reales de VoIP.

El principal motivo por el cual se ha desarrollado este Trabajo Fin de Grado es porque *VoIP Tester* no funciona con *Skype*. Esto es consecuencia de que VoIP Tester solo funciona con flujos VoIP de tamaño de paquetes fijos y el caso de *Skype* usa tamaño de paquetes que varían, entre otros motivos, con las condiciones de red.

2.2. Información Técnica Skype

En el programa *Skype* mientras se está realizando una llamada, se puede consultar en tiempo real los principales parámetros objetivos (paquetes perdidos, *jitter*, ancho de banda) del estado de la red durante la llamada, como se muestra en la Figura 2.1. Con esta información se puede realizar un estudio de la QoS de la llamada. Sin embargo, no se obtiene información acerca de la QoE, un punto que se trata y resuelve en este proyecto.

Esta opción se puede encontrar en el menú *Llamada-Información Técnica*, de dicho programa.

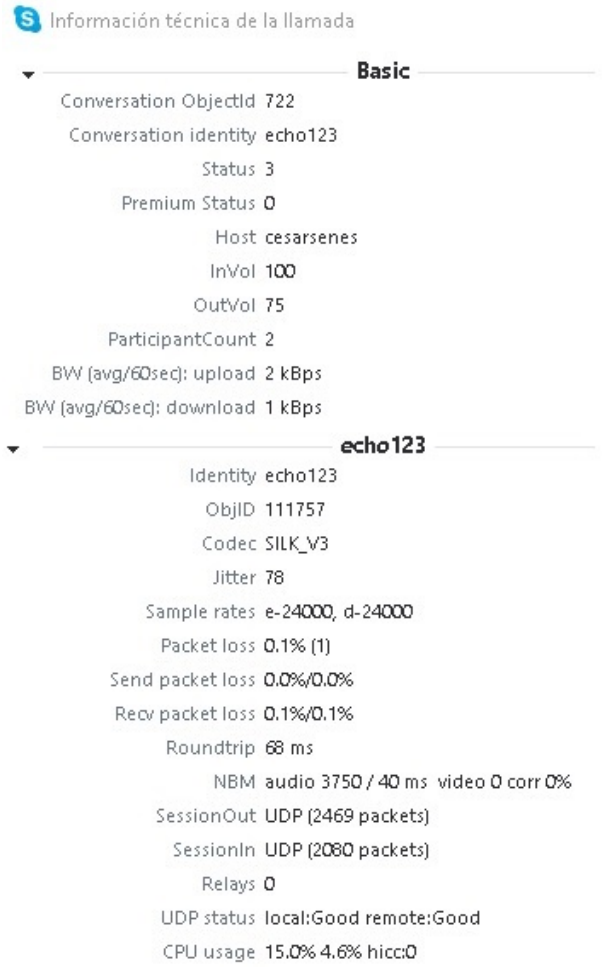


Figura 2.1: Información técnica *Skype*

En resumen, con esta opción del programa podemos saber en tiempo real el estado de la red y cómo influye en el tráfico generado por *Skype*. Como aspecto negativo, estas estadísticas no pueden ser recogidas de manera automática para un posterior análisis más detallado.

2.3. PRTG Network Monitor

PRTG Network Monitor [14] es un programa que se encarga de monitorizar el tráfico y filtrarlo según los diferentes protocolos, sabiendo así el tipo de servicios que está utilizando cada equipo dentro de la red ya que, este programa se ejecuta en un equipo dentro de una determinada red. So-

porta un gran número de protocolos y permite almacenar la información recolectada para visualizar datos históricos y realizar un análisis posterior.

En principio, esta aplicación esta destinada para la gestión y administración de redes, aunque también posee un sensor destinado para el estudio de la QoS específico para el tráfico VoIP.

Permite conocer, tanto para dos puntos finales así como para nodos intermedios, las siguientes estadísticas:

- *Jitter* en ms de acuerdo al RFC 3550.
- *Packet delay variation* (PDV) en ms de acuerdo al RFC 3393.
- Paquetes perdidos en %.
- Paquetes que llegan desordenados en %.
- Paquetes duplicados en %.

Con estas estadísticas es posible caracterizar la QoS de la red. Dado que se puede medir entre nodos intermedios de la red, se puede saber de manera precisa dónde se produce el problema. Por el contrario, como puntos débiles destaca que este *software* no recoge información acerca de la QoE, por lo que sería necesario el uso de otras aplicaciones para hacer un estudio completo. Por último, hay que indicar que esta aplicación es de pago.

2.4. VoIP and Network Quality Manager

Esta aplicación [15] se basa en la tecnología IP SLA de *Cisco* [16], que nos permite simular datos de tráfico VoIP para probar la red entre enrutadores de esta compañía. Se pueden obtener datos objetivos de la red, como son el *jitter*, latencia o paquetes perdidos. También es capaz de obtener un valor de QoE como es el MOS.

Como puntos positivos podemos indicar que realiza un estudio completo de la llamada de VoIP es decir, tanto objetiva (QoS) como subjetivamente (QoE).

Por otro lado, es necesario realizar la simulación en equipos conectados por un router *Cisco*. Dado que este proyecto se centra en el estudio de una llamada de *Skype*, y esta debe llegar a los servidores externos de *Skype*, *VoIP and Network Quality Manage* no sería una aplicación útil para este análisis. Al igual que en el caso anterior, esta aplicación es de pago.

2.5. VoIP Test ONSIP

Esta herramienta *online* realiza un test que mide los indicadores de rendimiento clave para la calidad de una comunicación VoIP. En concreto, realiza

pruebas desde el navegador obteniendo la velocidad de subida y bajada, *jitter* y latencia. Está diseñada con el objetivo de realizar un estudio previo sobre una red y así poder valorar una futura implantación de un sistema de VoIP.

Como punto positivo, hay que destacar que este test se realiza en menos de 60 segundos,pero no muestra ninguna estadística que nos permita valorar la QoE.

2.6. Artículo publicado en *Multimedia Tools and Applications*

En el artículo titulado *Subjective MOS model and simplified E-model enhancement for Skype associated with packet loss effects: a case using conversation-like tests with Thai users* [17] se proponen dos modelos matemáticos usados para estimar la calidad de una llamada de VoIP usando la aplicación *Skype*. El primer modelo se ha calculado realizando entrevistas a usuarios que han realizado llamadas en diferentes condiciones de red (diferente número de paquetes perdidos) y se ha recogido el nivel de satisfacción de estos. El segundo modelo ha calculado el E-model, usando el modelo simplificado.

Este artículo realiza un estudio similar al realizado en este proyecto. Sin embargo, no permite configurar todos los escenarios y parámetros que se contemplan en este trabajo. Además, este proyecto permite obtener estas medidas y otras no incluidas en el artículo, como son el PESQ y otras acerca de la calidad de servicio de la comunicación como son el retardo, *jitter* o el *throughtput*.

2.7. Estudio sobre calidad de servicio (QoS) para comunicaciones multimedia usando Skype

En este artículo publicado en 2007 [18], se estudia el tráfico generado por la aplicación *Skype*, usando dos tipos de experimentos para capturar las trazas del tráfico generado por la llamada. En primer lugar, se usan herramientas como *Ethereal* [19] (analizador de protocolos, hoy en día denominado WireShark) para un posterior análisis con aplicaciones ofimáticas (hojas de calculo). En segundo lugar, se observan las cabeceras de los paquetes capturados y se determinan así los parámetros de QoS.

Como aspecto negativo, se puede indicar que, a diferencia del presente proyecto, este artículo se centra en la QoS, obviando la QoE. Además, requiere de un posprocesado manual para obtenerlas estadísticas, a diferencia de la aplicación desarrollada en este trabajo fin de grado, que lo realiza de forma automática y transparente al usuario. Por último, se centra en el servicio de videoconferencia y no en el de voz.

2.8. Conclusión

Tras realizar un análisis de las aplicaciones y estudios que, al igual que este proyecto, se centran en la caracterización de llamadas realizadas mediante tecnología de VoIP, podemos concluir que ninguna de ellas cumple los objetivos propuestos en este trabajo. Resumiendo estos objetivos son, realizar un análisis tanto de la calidad de servicio como de experiencia durante una llamada de *Skype* con diferentes condiciones de red.

Capítulo 3

Planificación y costes

3.1. Recursos

3.1.1. Humanos

- D. Jorge Navarro Ortiz profesor del Departamento de Teoría de la Señal, Telemática y Comunicaciones de la Universidad de Granada, tutor del proyecto.

- César Senés Romo, alumno del Grado en Ingeniería de Tecnologías de Telecomunicación en la Universidad de Granada, autor del proyecto.

3.1.2. Hardware

- Ordenador personal.

3.1.3. Software

En general, se ha intentado utilizar herramientas gratuitas siempre que ha sido posible.

- Sistema Operativo: son necesarios dos licencias de *Microsoft Windows 7*. Esto es debido a que la aplicación se basa en una estructura de cliente-servidor y por tanto, cada equipo necesita un sistema operativo independiente. Se recomienda utilizar *MS Windows 7* porque ha sido el utilizado durante las pruebas realizadas y, por tanto, está verificado su correcto funcionamiento. No obstante, la aplicación debe ser capaz de funcionar en otras versiones de *MS Windows*.

- Herramienta de virtualización *Oracle VirtualBox* [20]. Se ha utilizado para poder implementar, en un único equipo, tanto el cliente como el servidor.

- Entorno de desarrollo *Netbeans 8.1* [21], usado para escribir, compilar y ejecutar el proyecto, el cual está desarrollado con el lenguaje de programación JAVA versión 8.
- Librería de JAVA jfreechart-1.0.19 [22], para representar los resultados obtenidos de manera gráfica.
- Compilador Python 2.7.1 [23]
- Programa para realizar llamadas de VoIP, *Skype*.
- Programa *Pamela for Skype* [24]. Este programa es el encargado de grabar la llamada en formato WAV para su posterior análisis.
- Analizador de protocolos *Wireshark* [19], nos permite capturar todos los paquetes generados durante la llamada.
- Emulador de red *Network Emulatorfor Windows Toolkit*. Encargado de simular las condiciones de red durante la llamada.
- Herramienta para trabajar con *Skype* a través de linea de comandos, *Clisk 2.5.1* [25]
- *VB-CABLE Virtual Audio Device*. Con esta herramienta es posible conectar el audio entre dos aplicaciones. En nuestro caso, hacemos que el sonido que sale desde el reproductor vaya directamente a *Skype*.
- Reproductor multimedia *VLC*, necesario para reproducir un audio de test, durante la llamada.
- ITU-T P.862 Versión 1.2 software, para calcular el valor PESQ.

3.2. Fases de desarrollo

En el diagrama de *Gantt* mostrado en la Figura 3.1 se puede observar la distribución temporal de las seis diferentes fases de desarrollo del proyecto.

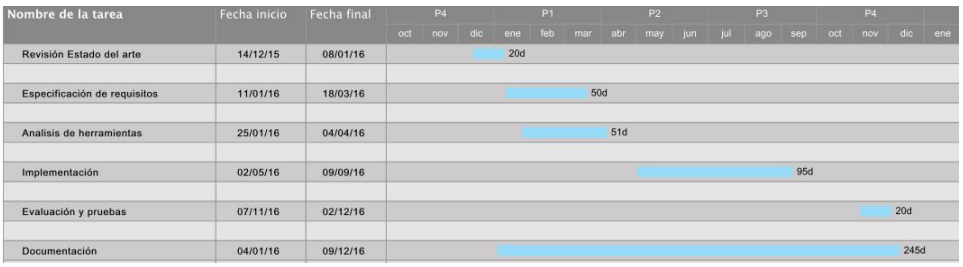


Figura 3.1: Planificación temporal del proyecto

3.2.1. Revisión del estado del arte

Estudio de las aplicaciones existentes en el mercado, relacionadas con el problema que se está tratando, con el fin de justificar la necesidad de realizar este proyecto.

3.2.2. Especificación de requisitos

Análisis detallado del problema para especificar las funcionalidades y limitaciones que va a tener esta solución.

3.2.3. Implementación

Teniendo en cuenta lo comentado en este Capítulo 3, se desarrolla la aplicación en el lenguaje de programación elegido, en este caso *JAVA*.

3.2.4. Evaluación y pruebas

Tras finalizar el desarrollo de la aplicación se procede a estudiar algunos casos significativos con el fin de asegurar que la aplicación funciona correctamente, así como obtener resultados y evaluarlos, con el objetivo de conseguir una caracterización del funcionamiento de códec usado por *Skype* (SILK).

3.2.5. Documentación

Redacción del informe donde se especifica cómo se ha desarrollado este proyecto. Se ha llevado a cabo de forma paralela al resto de tareas.

3.3. Estimación de Costes

3.3.1. Recursos Humanos

En el Cuadro 3.1 se puede observar el tiempo empleado en cada uno de los diferentes bloques del proyecto.

Para el coste se ha considerado una jornada laboral de ocho horas a un coste de 20 euros la hora, sin contar fines de semanas y festivos. Hay que destacar que las fases de *especificación de requisitos* 4 y *descripción de herramientas utilizadas* 5 se realizaron de forma conjunta en gran parte, por lo que aparecen los días de trabajo desglosados en días completos y media jornada. En la Figura 3.2 queda reflejado el coste de cada una de las fases del proyecto.

Fase	Duración (días,media jornada)
Revisión del estado del arte	20
Especificación de requisitos	10 , 40
Estudio de herramientas	11 , 40
Implementación	95
Evaluación y pruebas	20
Total	196

Cuadro 3.1: Coste Temporal del proyecto

3.3.2. Herramientas

Siempre que ha sido posible se ha usado software gratuito, con el objetivo de minimizar el coste del proyecto. En el Cuadro 3.2 se recoge el coste de los recursos usados.

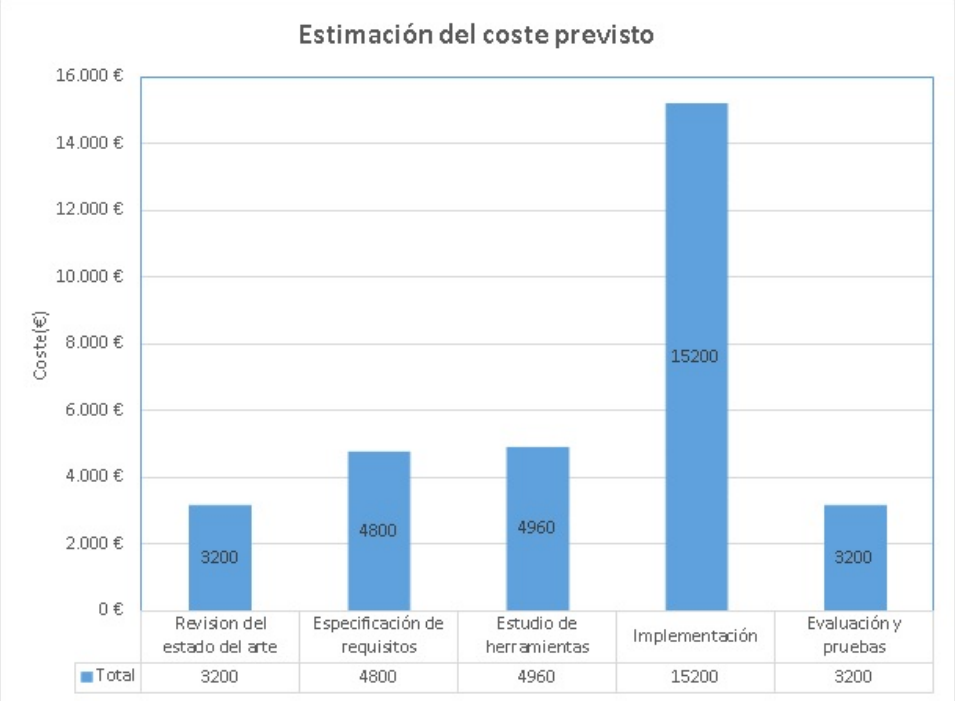


Figura 3.2: Coste asociado a cada fase del proyecto

3.4. Presupuesto

En el Cuadro 3.3 se especifica el presupuesto de este proyecto.

Recurso	Coste
Odenador personal + Windows 10	800€
2 licencias de Windows 7	270€
Entorno de desarrollo	0€
Paquetes de ofimática y otros	0€

Cuadro 3.2: Coste herramientas usadas

Recurso	Coste(/euro)
Desarrollador(196 dias)	31360 €
Odenador personal + Windows 10	900€
2 licencias de Windows 7	270€
Entorno de desarrollo	0€
Paquetes de ofimática y otros	0€
Total	32409.99

Cuadro 3.3: Presupuesto

Capítulo 4

Especificación de requisitos

En este capítulo se describe la especificación de las necesidades que con este proyecto se pretenden resolver. Este estudio se va a dividir en dos secciones:

- En la primera se define el propósito por el cual se ha implementado este sistema.
- En la segunda se exponen los requisitos tanto funcionales como no funcionales los cuales deben ser implementados en la fase de desarrollo.

4.1. Propósito

La herramienta desarrollada consiste en una aplicación configurable. Esta aplicación permite obtener de manera automática la evaluación de la calidad de voz, tanto objetivamente como subjetivamente, de una llamada realizada con Skype. Con el objetivo de observar y estudiar como se comporta la llamada en diferentes condiciones de red. Lo cual es interesante debido a que Skype no se comporta igual siempre, ya que usa códec adaptativo, como se explica en la sección 1.1.

En el esquema mostrado en la Figura 4.1, se observa el escenario de operación de la aplicación desarrollada. Se aprecian dos ordenadores conectados a Internet, los cuales establecen una comunicación de voz mediante Skype, la cual será analizada en profundidad.

Destacar que se pretende que la aplicación desarrollada sea lo más adaptable posible a las nuevas tecnologías. Con el objetivo de que la aplicación sea capaz de funcionar sobre nuevas tecnologías de red, versiones de sistemas operativos o cambios en políticas de Skype. Lo cual debido a la constante evolución del sector de las TIC (Tecnologías de la Información y la Comunicación), se puede considerar interesante estudiar como afectan estas evoluciones o cambios sin la necesidad de realizar cambios en la aplicación.

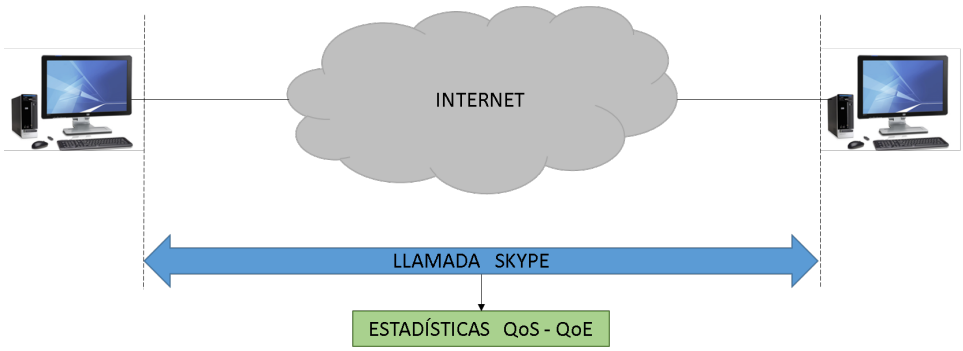


Figura 4.1: Escenario de operación de la aplicación desarrollada

El principal objetivo, es el de estudiar la robustez del códec *SILK* [5] en diferentes escenarios de red, para ello es necesario estudiar los siguientes parámetros:

- **Retardo extremo a extremo:** tiempo que transcurre desde que un paquete se envía en un extremo hasta que se recibe en el otro extremo.
- **Jitter:** variación del retardo extremo a extremo entre paquetes. Este parámetro afecta de forma significativa a las comunicaciones de VoIP.
- **Probabilidad de pérdidas:** número medio de paquetes perdidos durante una simulación.
- **Throughput:** tasa de paquetes que se transmiten por unidad de tiempo.
- **Percentual Evaluation Speech Quality (PESQ):** es un método para la evaluación de la calidad vocal, extremo a extremo, de redes telefónicas de banda estrecha y codecs vocales. Está definido en la recomendación P.862 [26] de la ITU-T. Proporciona una evaluación subjetiva mediante una *Mean Opinion Score*(MOS) [10].
- **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 [12] de la ITU-T. 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.

4.2. Requisitos

Esta sección es determinante debido a que en la siguiente fase de desarrollo es necesario tener muy presente lo descrito aquí. Ya que si durante la

fase de diseño se consigue implementar todos estos requisitos se puede considerar el trabajo realizado como exitoso, sin la necesidad de realizar grandes remodelaciones con las contras que esto representa.

4.2.1. Requisitos funcionales

En este apartado se enumeran los requisitos básicos funcionales que se deben implementar en el diseño de la aplicación, para cumplir con los objetivos definidos en la sección 1.3

- **Establecimiento y finalización de una llamada:**

  La aplicación debe ser capaz tanto de realizar una llamada de Skype como de finalizarla, sin usar la interfaz propia de Skype.

- **Responder llamada:**

  Es necesario que la aplicación sea capaz de responder una llamada realizada con Skype en un momento determinado.

- **Envío de audio durante llamada:**

  La herramienta desarrollada debe tener la capacidad de enviar un determinado audio, en este caso en formato *WAV*, durante la llamada.

- **Configuración del enlace de red:**

  Es importante que desde la aplicación sea posible especificar, las características del enlace de red. Los parámetros que nos resultan más significativos, es decir tienen más efecto en la simulación son el *throughout*, latencia y tasa de paquetes perdidos.

- **Grabación de la llamada:**

  Con el objetivo de obtener una comparación entre el audio original y obtenido de la simulación (normalmente degradada) es necesario que la aplicación tenga una función que grabe, en el formato deseado, la llamada.

- **Captura del tráfico generado por la llamada:**

  Durante la simulación es necesario que en ambos equipos se este monitorizando el tráfico generado por la llamada, para un posterior análisis y estudio.

- **Análisis datos:**

  Con todos los datos obtenidos es necesario su procesado para obtener los parámetros, tanto objetivos como subjetivos, que caracterizan de forma detallada la comunicación realizada con Skype.

- **Representación gráficamente de los resultados:**

  Finalmente con el objetivo de mostrar de la forma más clara posible los parámetros obtenidos, será necesario que la aplicación desarrollada sea capaz de mostrar estos en gráficas.

4.2.2. Requisitos no funcionales

Ahora se muestran los requisitos no funcionales, los cuales nos especifican algunas características interesantes del sistema implementado.

- **Software y compatibilidad**

  Para la implementación se ha utilizado el lenguaje de programación Java, lo cual requiere que en los equipos que se ejecute la aplicación deben tener instalada una versión de Java. Por otro lado indicar que la aplicación se ha desarrollado para que funcione correctamente en el sistema operativo *Windows 7*, aunque no debería presentar ningún inconveniente en otras versiones de *Windows*, ya sean mas recientes o posteriores.

- **Interfaz gráfica**

  La interfaz debe ser lo más sencilla e intuitiva posible, con el objetivo de facilitar al usuario la configuración de los parámetros necesarios para la simulación.

- **Gestión de fallos**

  La aplicación durante la simulación va mostrando en un terminal, paso a paso mensajes de que va haciendo incluyendo mensajes de error con una breve descripción. De este modo el usuario es capaz de localizar e identificar los posibles errores. Se implementa de manera ante un posible error la aplicación no quede bloqueada y pueda volver a un estado en el que pueda continuar funcionando.

- **Rendimiento**

  El rendimiento de la aplicación dependerá de las especificaciones del equipo en el que se ejecuta. Por lo que se recomienda que cuando se realice una simulación el equipo no este ejecutando otras aplicaciones ajenas a este proyecto, que puedan limitar el rendimiento de equipo.

- **Red de comunicación**

  Al realizar una simulación las pérdidas del canal vienen descritas según los parámetros introducidos por el usuario en la aplicación. Por tanto se recomienda para un correcto funcionamiento que la conexión entre cliente y servidor sea de la mayor calidad posible (a ser posible usar cable ethernet, evitar redes Wi-Fi). Con el objeto de una interpretación más real de los resultados.

- **Usuario**

  El usuario final de la aplicación debe poseer unos conocimientos medios-altos sobre redes. Así como comprensión de las estadísticas mostradas por la aplicación, que caracterizan la calidad de servicio de la conversación VoIP.

Capítulo 5

Descripción de herramientas utilizadas

En esta sección se detallan las especificaciones y características de las aplicaciones o recursos externos usados en la aplicación desarrollada. Estas herramientas se han usado con el objetivo de conseguir algunas de las funcionalidades requeridas para este proyecto, definidas en la sección 4.2.1.

Destacar que siempre que ha sido posible se ha intentando usar software libre, es decir, que no sea necesario el pago de una licencia para su uso, con el objetivo de mantener los costes del proyecto lo más bajo posible. Además de los motivos económicos, en la búsqueda de estas aplicaciones se ha tenido muy en cuenta que se pudieran usar mediante línea de comandos. Ésto se debe a que si poseen esta característica, se pueden ejecutar mediante procesamiento por lotes desde el programa principal de JAVA. La principal característica del procesamiento por lotes es que su ejecución no precisa ningún tipo de interacción con el usuario, consiguiendo así que su uso sea transparente para el usuario final.

5.1. *Clisk*

Clisk (*Command-Line Interface for Skype*) [25], es una herramienta basada en línea de comandos para trabajar con *Skype*. Su objetivo inicial era proporcionar la función de respuesta automática a una determina llamada, pero *Clisk* evolucionó y pasó a ofrecer la mayoría de funciones de *Skype* desde una línea de comandos. Sin embargo con la última actualización de *Skype*, la mayoría de funciones dejaron de ir, esto se ha producido debido a que la API ha dejado de estar disponible. Indicar que en un primer momento para este proyecto estaba planeado usar esta aplicación para la grabación y envío de audios durante la llamada. Debido a lo comentado anteriormente ha sido necesario buscar alternativas que se comentarán más adelante en esta sección.

Por lo tanto, esta aplicación ha sido usada en el proyecto desarrollado para contestar y colgar la llamada con estos comandos:

- **ans:** Responde una llamada entrante.
- **ha:** Finaliza una llamada en curso.

Esta herramienta está diseñada para funcionar en cualquier sistema operativo que pueda soportar tanto *Skype* como *Python*, versión 2.7 . Sólo en las plataformas *Windows* y *MacOS* ha sido probado, aunque *Clisk* también debería funcionar en *Linux* sin problema.

Para iniciar *Clisk* en *Windows*, una vez instalada la versión 2.7 de *Python*, sólo se tiene que ejecutar *clisk.py*. Aparecerá una consola en la cual se permite escribir comandos. La primera vez que se ejecuta *Clisk* se mostrará en *Skype* un mensaje "en espera de autorización". Es necesario aceptar esa notificación en *Skype* con el fin de hacer que *Clisk* funcione correctamente.

5.2. Implementación en C de UIT-T P.862

El método objetivo descrito en esta recomendación se conoce por "evaluación de la calidad vocal por percepción"(PESQ, *Perceptual Evaluation of Speech Quality*) o UIT-T P.862 [11], preparada por la Comisión de Estudio 12 (2001-2004) del UIT-T la cual fue aprobada por el procedimiento de la Resolución 1 de la AMNT el 23 de febrero de 2001 y proporciona una implementación de referencia escrita en lenguaje de programación C.

El método PESQ mide la calidad vocal mediante la ecualización de la función de transferencia, la alineación de tiempo y un nuevo algoritmo para promediar distorsiones en función del tiempo. La validación de PESQ incluía un número de experimentos que probaban específicamente su calidad de funcionamiento para combinaciones de factores, tales como filtrado, retardo variable, distorsiones de codificación y errores de canal.

Esta recomendación describe un método objetivo para predecir la calidad subjetiva de una comunicación.

En la aplicación principal esta implementación de UIT-T P.862, es usada una vez terminada la simulación y es ejecutada pasándole como argumentos dos audios. Por un lado, el audio original enviado y por otro lado, el audio grabado en el servidor, el cual será una copia degradada del audio original debido al efecto del canal sobre la llamada. Finalmente, se obtiene un valor entre 1.0 y 4.5 con el cual podemos evaluar la calidad de la comunicación.

5.3. Network Emulator for Windows Toolkit

Network Emulator for Windows Toolkit (NEWT) un emulador basado en software el cual permite emular el comportamiento de redes tanto cableadas e inalámbricas utilizando un enlace físico fiable. Una gran variedad

de atributos de red son configurables, incluyendo el tiempo de ida y vuelta a través de la red (latencia), la cantidad de ancho de banda disponible, el comportamiento de la cola, pérdida de paquetes, la reordenación de paquetes y propagaciones. Además proporciona flexibilidad en el filtrado de los paquetes de red basado en direcciones IP o protocolos como TCP, UDP e ICMP.

El motivo por el cual se ha elegido este emulador de red para la realización de la aplicación principal es que puede ejecutarse mediante líneas de comandos.

Para su ejecución, mediante un archivo de procesamiento por bloques, es necesario pasarle como atributo un archivo XML, el cual describe el comportamiento del canal de comunicación, ya que incluye el valor para los atributos nombrados anteriormente. Además es necesario especificar un parámetro que determina el tiempo durante el cual el emulador va a estar ejecutándose.

5.4. Wireshark

Wireshark [19] es un analizador de paquetes de red. Este tipo de programas capturan los paquetes de red y muestran los datos de cada uno de ellos lo más detalladamente posible.

Para la aplicación principal se ha usado *Tshark* [27] una versión de *Wireshark* ejecutable desde una línea de comandos, con las mismas funcionalidades que *Wireshark* .

Tshark es ejecutado tanto en el cliente como en el servidor con el objetivo de capturar, para un posterior análisis, todos los paquetes generados por la conversación realizada por *Skype*.

5.5. VLC

VLC [28] es un reproductor multiplataforma gratuito y de código abierto que reproduce la mayoría de archivos multimedia y varios protocolos de emisión. Se ha elegido este reproductor ya que se puede configurar de manera, que por defecto tome como salida de audio un dispositivo determinado. En este caso un dispositivo virtual creado por *Virtual Audio Cable*.

5.6. Voicemeeter Virtual Audio

Con la instalación de *Voicemeeter Virtual Audio* [29], aparecen en la lista de dispositivos de audio de tu equipo un nuevo dispositivo de grabación y otro de reproducción. Toda señal procedente del dispositivo virtual de entrada va al dispositivo de salida directamente. En este proyecto se utiliza esta herramienta para conectar el audio introducido en la simulación, reproducido con VLC, con *Skype*.

5.7. Pamela

Pamela for Skype [24] es una herramienta que permite grabar una llamada realizada con la aplicación *Skype*. Se ha elegido esta aplicación debido a que permite seleccionar el formato en el cual va a guardar el audio, lo cual es necesario para este proyecto, ya que es necesario tener en el mismo formato tanto el audio enviado como el grabado para su comparación. Por otro lado, ha de saberse que esta aplicación permite realizar algunas operaciones mediante línea de comandos, lo que facilita el uso de Pamela desde la aplicación desarrollada.

Esta aplicación no es 100 % gratuita pero la versión básica usada si lo es. Además esta versión limita el tiempo máximo de simulación a 15 minutos, ya que este es el tiempo máximo que permite grabar. Sin embargo, en el caso de que el usuario desee realizar simulaciones más largas deberá obtener una cuenta *proffessional* o *bussiness*, lo que conlleva un desembolso económico.

5.8. SoX

SoX (*Sound eXchange*) [30] es una aplicación multiplataforma (*Windows, Linux, MacOS X*, entre otros) que funciona con una interfaz de línea de comandos. Esta herramienta es capaz de convertir archivos de audio de un formato a otro y de aplicar diferentes efectos de sonido a estos archivos, así como reproducir y grabar audios. Para este proyecto, del gran número de funcionalidades que nos ofrece este programa, se ha utilizado la opción de edición: ***trim***, con esta opción podemos dividir, un archivo de audio con formato WAV, en varios fragmentos.

Capítulo 6

Implementación

En este capítulo se van a describir las características referentes al diseño y a la implementación de la aplicación que se va a desarrollar. Con el fin de cumplir con las especificaciones definidas teóricamente a lo largo del Capítulo 4.

Primeramente se va a especificar detalladamente la arquitectura elegida, describiendo todos los elementos que la forman.

Finalmente se detalla el diseño final y la implementación de la aplicación desarrollada. Esta explicación se va a realizar, especificando por orden cronológico y detalladamente cada una de las funciones de la herramienta desarrollada.

6.1. Arquitectura

La arquitectura del sistema diseñado se muestra en la Figura 6.1. Después de estudiar las necesidades y funcionalidades que debe poseer este sistema, se ha decidido que esta arquitectura es la más adecuada para el proyecto.

Esta arquitectura constará de dos ordenadores, donde uno actuará como cliente y el otro como servidor.

El ordenador que actúa como cliente, en un primer lugar se encarga de la interacción con el usuario para que este, ayudado por una interfaz gráfica, establezca los principales eventos que van a ir produciéndose durante la simulación. Y finalmente después de que termine la llamada, se encarga del procesado de los datos recogidos, mostrando los resultados al usuario.

Por otro lado, el equipo que actúa como servidor se encarga de establecer la llamada entre los dos equipos, grabar la conversación e indicar al emulador de red que parámetros de la red debe modificar. Indicar que en ambos equipos durante la simulación se está capturando el tráfico generado por la conversación, con el fin de recoger la mayor información posible para su análisis.

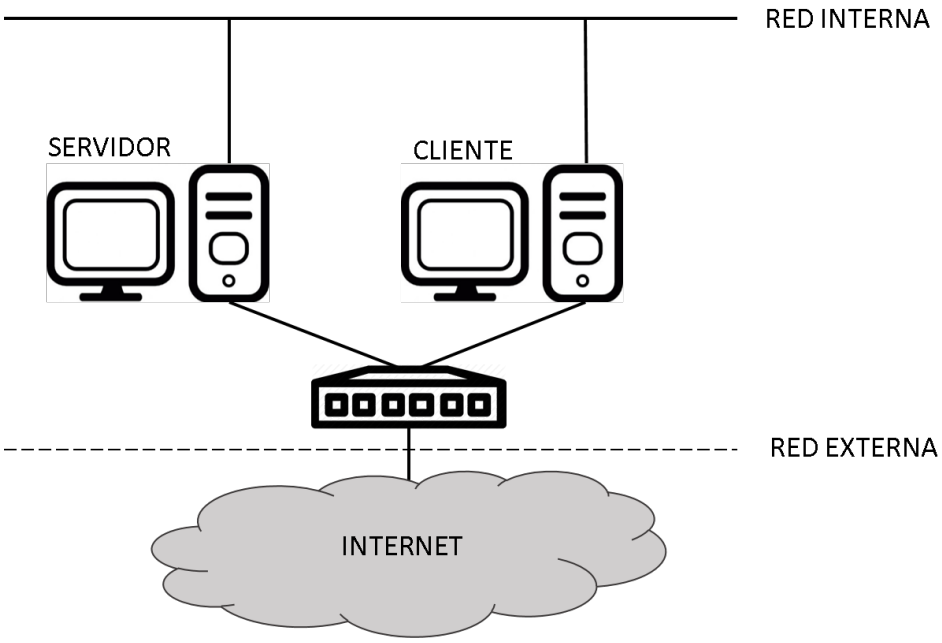


Figura 6.1: Arquitectura del sistema diseñado

En la Figura 6.2 se muestran un esquema en el cual se detalla secuencialmente como funciona la aplicación, con el fin de obtener una idea global de como iteractúan cliente y servidor.

A continuación se presenta una breve explicación de este diagrama:

1. **Inicio Sesión en el cliente:** el usuario que actúa como cliente introduce en la aplicación diseñada un usuario y una contraseña, que este registrado previamente en Skype.

2. **Inicio Sesión en el servidor:** al igual que en el cliente, es necesario introducir un usuario y contraseña válidos, es decir registrados en *Skype*.

3. **Envío datos para la simulación al servidor:** desde el cliente se envía al servidor la información referente a la simulación. La información enviada principalmente es el audio que se reproduce durante la llamada y los perfiles de trafico. Estos perfiles definen los parámetros introducidos al emulador de red. Y por último, el usuario de *Skype*, usado por el cliente, con el objeto de que el servidor pueda realizar la llamada. Esta información ha sido introducida por el cliente usando la interfaz gráfica de la aplicación creada.

4. **Inicio de la llamada y captura de información en el servidor:** usando el usuario recibido en el paso anterior, el servidor realiza una

llamada, a través de la aplicación *Skype*. Por otro lado, se comienza capturar, usando un analizador de protocolos, el tráfico generado durante la llamada. También simultáneamente se graba el audio de la llamada. Esta información será indispensable para el análisis y muestra de resultados final.

5. **Respuesta a la llamada entrante y captura de información en el cliente:** en este punto el cliente responde la llamada realizada por el servidor. Y del mismo modo que en el punto anterior se comienza a capturar el tráfico generado durante la llamada.

6. **Pérdidas en el enlace:** desde el servidor, se ejecuta el emulador de red. Éste irá variando los parámetros que caracterizan la red, dependiendo de lo introducido por el cliente al inicio de la simulación.

7. **Reproducción del audio de test durante la llamada:** una vez establecida la llamada desde el lado del cliente (sin perdidas), se reproduce el audio elegido al principio.

8. **Fin de la llamada y envío de los datos recogidos en el servidor:** una vez que todos los eventos de la simulación se han completado, el servidor finaliza la llamada. Por último el servidor envía tanto la captura del tráfico y la grabación de la llamada al cliente.

9. **Análisis de los datos de los datos recogidos:** Finalmente, en el cliente se procede a analizar los resultados obtenidos. Para ello, se comparan las capturas del servidor (con pérdidas, introducidas por el emulador de red) y las del cliente (sin pérdidas). Con esta información es posible realizar un estudio de la calidad de servicio objetiva. Mientras que para obtener la calidad de servicio subjetiva, es necesario comparar el audio del test original con el grabado en el servidor (con pérdidas).

Continuando con la arquitectura, como se puede observar en la Figura 6.1 se muestran dos redes distintas denominadas *red externa* y *red interna*. El objetivo por el que se utilizan dos redes diferentes es disponer de una red sin perdidas ni otras restricciones (no como la que sufre los efectos del emulador de red) para que el cliente y el servidor sean capaces de comunicarse de manera segura. Destacar que la *red externa* necesita conexión a Internet, debido a que por esta red se realiza la llamada con *Skype* y, por tanto, se necesita que haya comunicación con los nodos de *Skype*.

Destacar que con el objeto de que este sistema sea portable y se pueda utilizar sin necesidad de realizar ningún tipo de configuración de red, el sistema se ha implementado usando máquinas virtuales. Más concretamente se ha usado el programa *VirtualBox*, mediante el cual se han creado dos equipos con sistema operativo *Windows 7*. Uno de ellos actúa como cliente

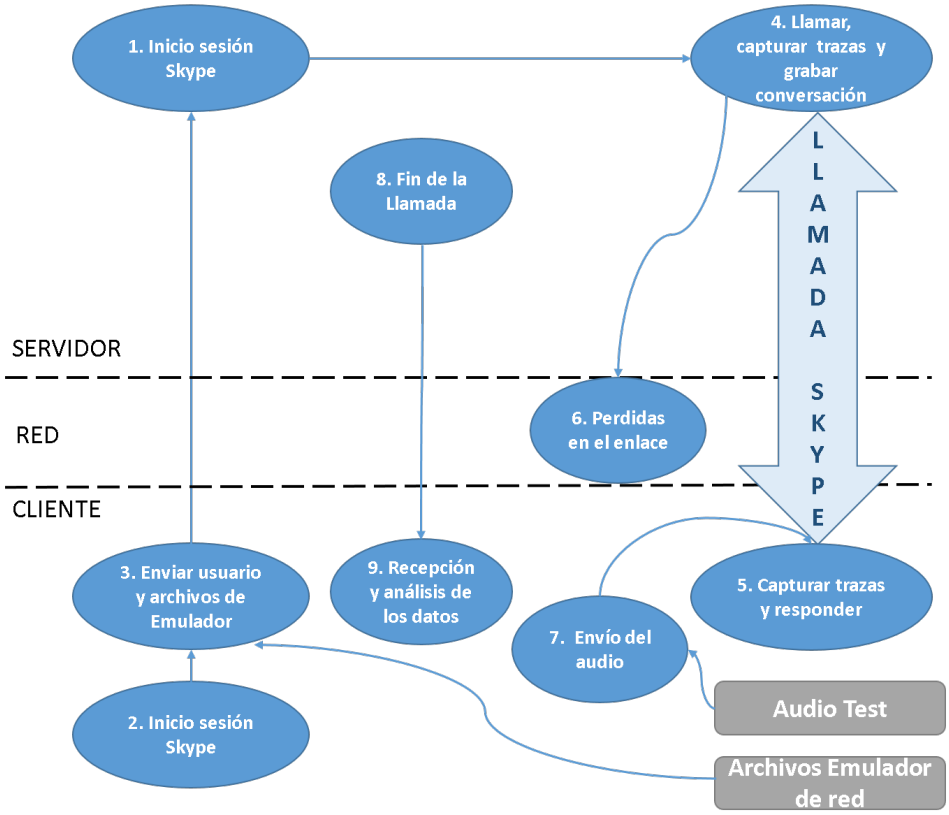


Figura 6.2: Esquema de funcionamiento de la aplicación

mientras que el otro actúa como servidor. Cada una de estas dos máquinas se le han habilitado dos adaptadores de red (cada uno para una red diferente) de la siguiente manera:

- **Red NAT** (*Network Address Translation*), como se muestra en la Figura 6.3, los equipos dentro de una red de este tipo pueden comunicarse entre ellos y además tener acceso a Internet. Para ello es necesario crear una red NAT, siguiendo estos pasos:

  Archivo - Preferencias - Red - Redes NAT - Agregar, sólo hay que especificar el nombre y la dirección de la red, en este caso como nombre hemos usado *NatNetwork* y como dirección 10.10.210.0/24.

  Por último, indicar que para asociar que una determinada máquina virtual pertenece a este tipo de red, al activar el adaptador se selecciona la opción *Red NAT* y como nombre se le pone el creado en el paso anterior.

- **Red Interna**, con esta función conseguimos tener comunicación solo entre las máquinas virtuales. Para su activación basta con habilitar

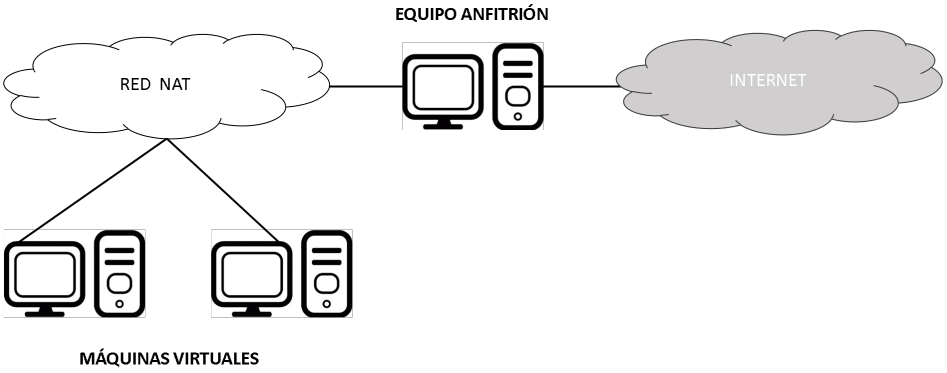


Figura 6.3: Esquema de una red NAT

el adaptador en modo red interna, indicar que en ambos equipos el nombre de la red debe ser el mismo.

Por último, indicar que para conseguir enviar un audio desde el cliente al servidor a través de Skype, como se muestra en la Figura 6.4.

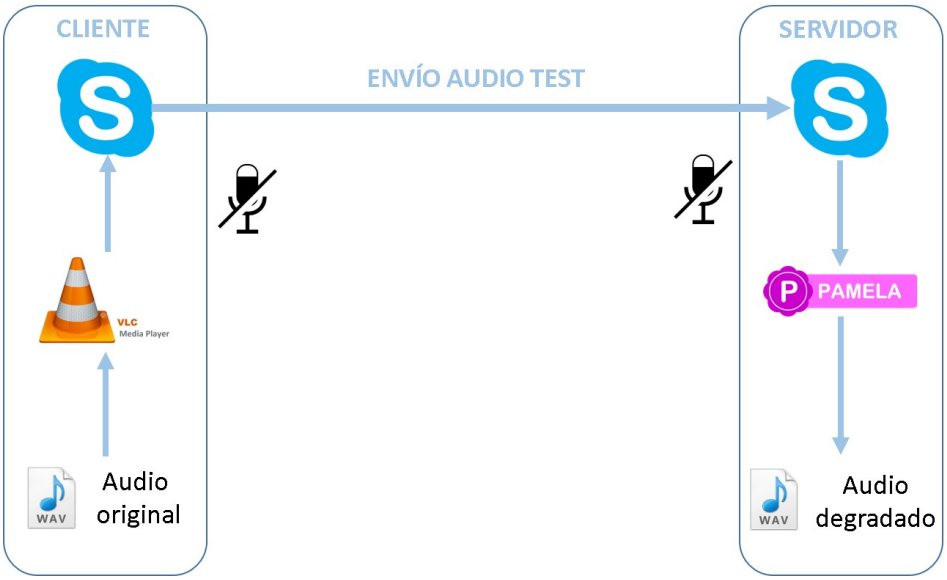


Figura 6.4: Esquema de la configuración del sistema de audio.

Es necesario realizar una serie de modificaciones en ambas máquinas virtuales. Estas modificaciones así como los programas de terceros usados, se detallan paso a paso a continuación:

1. En primer lugar es necesario la instalación del programa *Voicemeeter Virtual Audio*, especificado en la sección 5.6. Al instalar este *software*

conseguimos una entrada y una salida de audio, ambas virtuales. En la Figura 6.5, se muestran los listados de dispositivos de audio, tanto de grabación como de reproducción, instalados en el equipo. Se puede observar destacado en rojo los dos dispositivos virtuales creados al instalar *Voicemeeter Virtual Audio*.

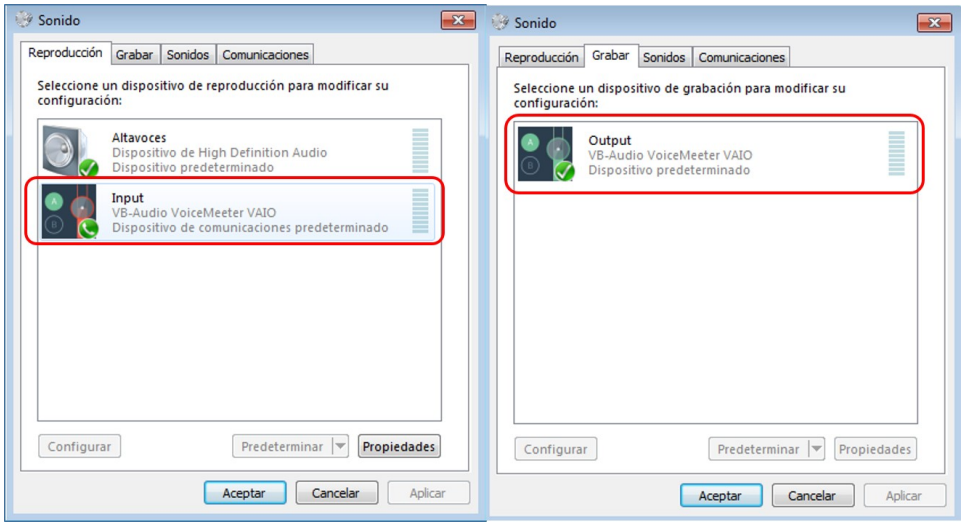


Figura 6.5: Dispositivos tanto de grabación como de reproducción instalados en el equipo que actúa como cliente.

2. A continuación para reproducir el audio original es necesario un reproductor multimedia, en este caso el usado es VLC, descrito en la sección 5.5. Para que el audio original se reproduzca durante la llamada de Skype es necesario conectar la salida de audio de VLC con la entrada de audio de Skype. Para ello en primer lugar, se configura VLC para que el dispositivo por el cual va a reproducir el audio sea el dispositivo virtual de entrada creado por la aplicación *Voicemeeter Virtual Audio*, como se muestra, destacado en rojo, en la Figura 6.6.

   Para realizar este cambio en la configuración de VLC se deben seguir estos pasos:

   Ejecutas VLC - en la barra de menú *Herramientas - Preferencias - Audio - Dispositivo - Input(VB-VoiceMeeter VAIO)*

3. Ahora es necesario modificar la configuración de Skype, para conseguir que la entrada de audio sea a través dispositivo de salida virtual(Output(VB-VoiceMeeter VAIO)). Para realizar este cambio en la configuración de Skype es necesario seguir estos pasos:

   Ejecutas Skype - en la barra de menú *Herramientas - Opciones - Configuración de sonido - Entrada de audio - Output(VB-*

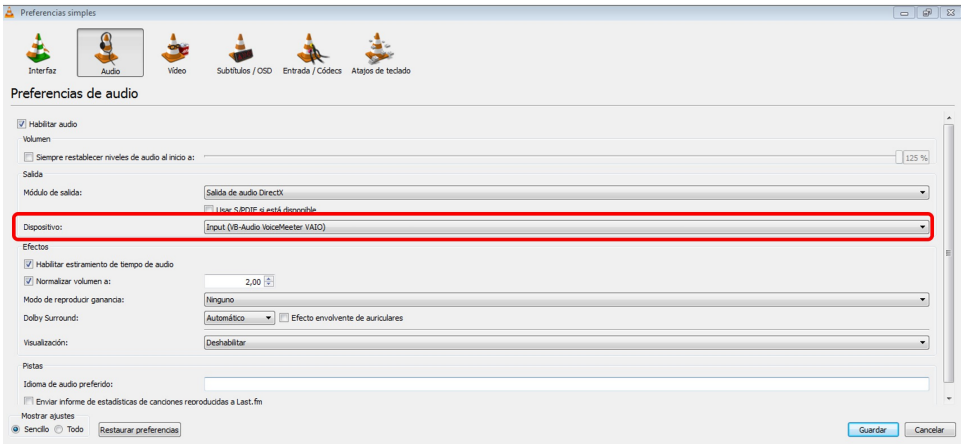


Figura 6.6: Selección dispositivo de salida de VLC.

VoiceMeeter VAIO)

En la Figura 6.7, se muestra la ventana de configuración de sonido de Skype. Se resalta la entrada de audio que debe tomar, para estar conectada con VLC.

4. Llegados a este punto, con las configuraciones realizadas hasta ahora. Al realizar una llamada de Skype y reproducir un audio en el cliente, este llega al servidor. Por tanto únicamente se necesita grabar este sonido en el servidor para su posterior comparación con el original. Para realizar esta grabación es necesario. En primer lugar desactivar el micrófono del equipo servidor, ya que este puede introducir ruido o voz que no nos interesa en nuestro análisis. Y por último para la grabación se necesita un programa llamado *Pamela*, descrito en la sección 5.7. Este programa debe estar en ejecución durante la llamada de Skype y automáticamente graba la llamada. Únicamente es necesario cambiar la configuración de *Pamela* para que el formato del archivo grabado coincida con el audio original. Para ello:

   Ejecutas Pamela - en la barra de menú *Herramientas - Opciones - Advanced - Sonido - Recording Format - Built-In WAVE Plugin(wav)*

   Finalmente pulsamos en el botón *Configure codec* y se seleccionan las opciones como se muestran en la Figura 6.8. Se han seleccionado estas características del codec, ya que los audios seleccionado para la realización del test y recomendados por la ITU, presentan estas características.

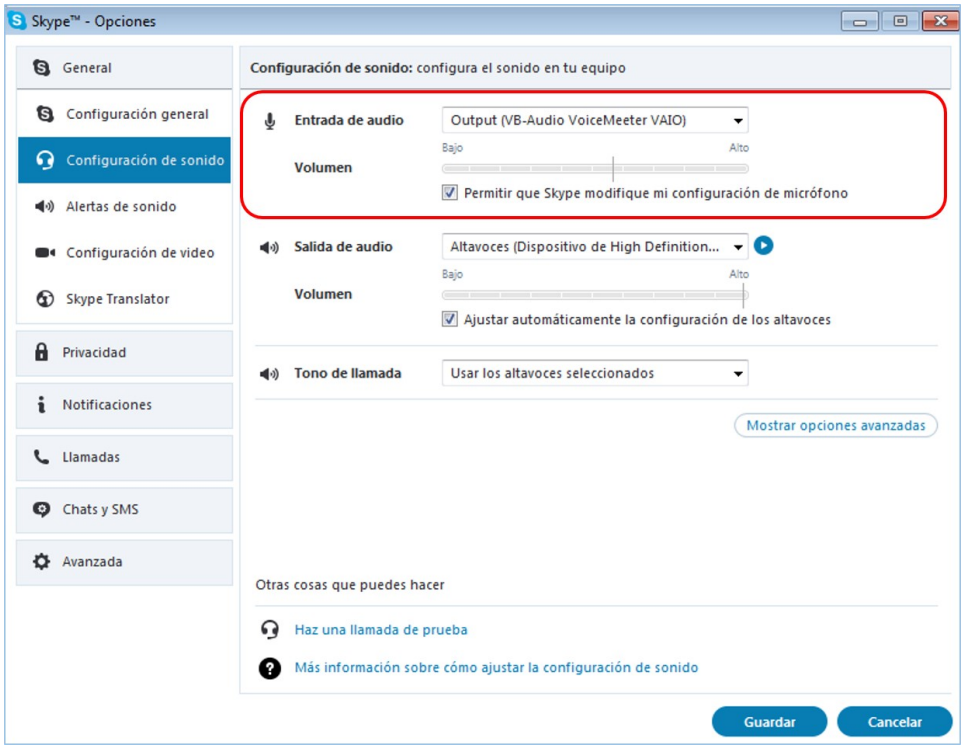


Figura 6.7: Configuración en *Skype* de la entrada de audio .

6.2. Diseño e implementación

En esta sección se van a describir todos los aspectos relacionados con el desarrollo del sistema. La metodología seguida para describir la funcionalidad del sistema final desarrollado, va a ser ir especificando de manera secuencial las operaciones que en cada momento el programa va realizando, tanto en el cliente como en el servidor. Haciendo hincapié en secciones, en las funcionalidades más determinantes para el correcto funcionamiento de la aplicación.

Destacar que en la figura6.9, se muestra la estructura de carpetas usada tanto en cliente como servidor.

Se ha decidido que la mejor opción para la estructura de carpetas, es que dentro del directorio raíz (C://) se cree un directorio llamado *TestSkype*, donde se ubican todos los archivos necesarios para la realización de la simulación, así como todos los resultados obtenidos. Estos archivos se encuentran ordenados en subcarpetas. A continuación se detalla el contenido de estas subcarpetas, en ambos equipos:

Servidor

audios: En esta carpeta se guarda el audio grabado en el servidor durante

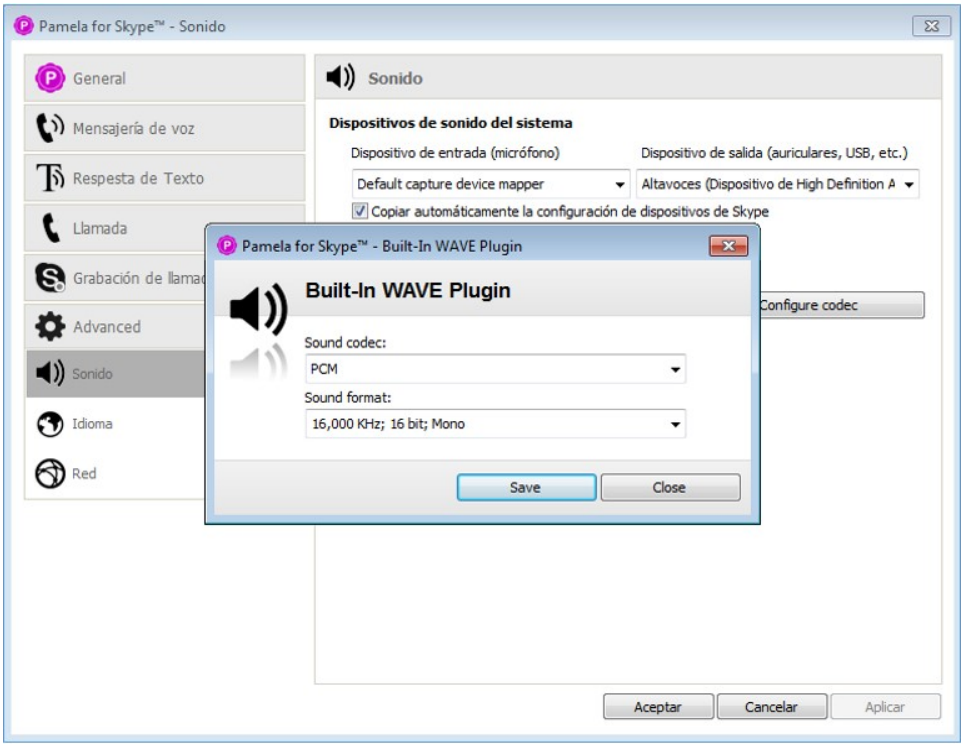


Figura 6.8: Especificación del formato del audio grabado.

la simulación.

bat: Contiene todos los archivos batch, la ejecución de los cuales es necesaria para la realización de la simulación.

Capturas: Contiene la captura realizada con *WireShark* durante la simulación.

Clisk: Contiene la herramienta *Clisk*. La cual a través de una consola situada en este directorio, realiza operaciones propias de *Skype* con comandos.

perfiles: Contiene los archivos *xml* que definen unas determinadas condiciones de un enlace de red. Estos archivos son usados durante la simulación por el emulador de red (NEWT, *Network Emulator for Windows Toolkit*).

Servidor: Dentro de este directorio se encuentra el proyecto generado por *Netbeans*. Es decir, todas las clases y archivos JAVA que describen el programa que se ejecuta en el servidor.

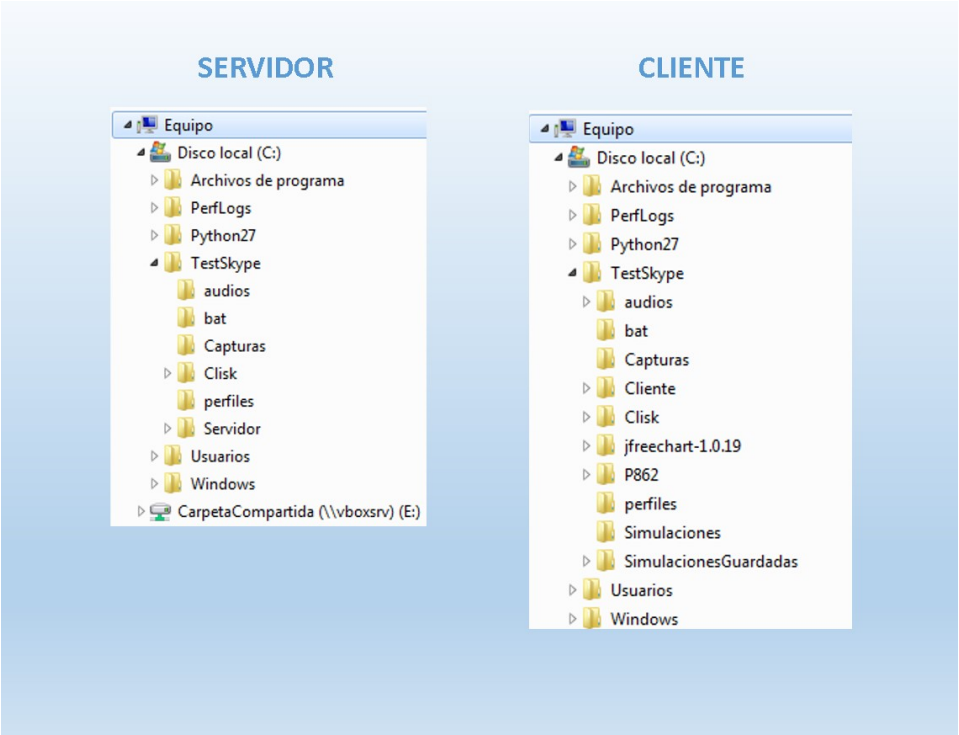


Figura 6.9: Estructura de las carpetas en el cliente y el servidor.

Cliente

audios: Aquí se guarda todo lo relacionado con audios de la simulación, como son el audio original y el degradado.

bat: Al igual que en el servidor en esta carpeta se guardan los archivos batch, que se ejecutan durante la simulación en el cliente.

Capturas: Contiene la captura de trafico generada por la simulación en el cliente y en el servidor.

Cliente: Al igual que en el servidor contiene el proyecto JAVA del programa que se ejecuta en el cliente.

Clisk: Al igual que en el servidor contiene la herramienta *Clisk*.

jfreechart-1.0.19: Es una librería necesaria en el proyecto JAVA para la representación gráfica de los resultados obtenidos de la simulación.

P862: Contiene la implementación realizada por la ITU de PESQ. Se ejecuta con diferentes parámetros para obtener este resultado subjetivo.

perfiles: Contiene los archivos *xml* que definen unas determinadas condiciones de red. Introducidos por el usuario y durante la simulación enviadas al servidor.

simulaciones: al introducir los datos de una simulación antes de iniciar esta, la aplicación indica si desea guardar esta información. Si indica sí aquí se guarda en un archivo txt. Al inicializar una nueva simulación es posible cargar esta información.

simulacionesGuardadas: Con el objetivo de volver a visualizar los resultados de una simulación, sin necesidad de volver a realizar la simulación al terminar ésta. Aquí se guardan los archivos con los resultados. Indicar que las diferentes simulaciones se guardan en carpetas independientes y el nombre de esta carpeta es la fecha y hora en la que se hizo la simulación.

Con esta estructura de carpetas es posible instalar la aplicación en cualquier otro equipo, ya que el programa va a saber encontrar todos los archivos necesarios para la simulación. Por otro lado de esta forma se tiene de una forma ordenada acceso a todos los archivos generados por la aplicación.

En primer lugar, es fundamental la introducción de los datos de la simulación por parte del cliente. Por ello la herramienta diseñada consta de una interfaz gráfica, de manera que la introducción de estos datos sea lo mas sencilla e intuitiva posible. En la sección 6.11 se especifica detalladamente este proceso.

Para la comunicación entre el cliente y el servidor es necesario la especificación de un sistema mediante el cual se permita enviar mensajes entre ambos lados. Con el objeto de que en cada momento cada uno de los extremos se encuentre en coordinación con el otro. Estos métodos quedan especificados detalladamente en la sección 6.2.2.

El sistema de comunicación descrito en 6.2.2 es fundamental para el correcto funcionamiento del protocolo de comunicación diseñado. El cual simplemente consta del intercambio de una serie de mensajes, como se muestra en la Figura 6.10.

A continuación se describe cada uno de los mensajes y lo que implica en cada una de los extremos su llegada:

- En primer lugar antes de enviar o recibir mensaje alguno, el cliente lanza el proceso de sincronización de los relojes de ambos equipos. Este proceso, especificado detalladamente en la sección 6.2.3, es fundamental ya que, si los equipos no están sincronizados, la comparación entre las estadísticas recogidas en ambos extremos no es útil.

- El primer mensaje enviado del cliente al servidor contiene información necesaria para el correcto funcionamiento de la simulación. Se envía la

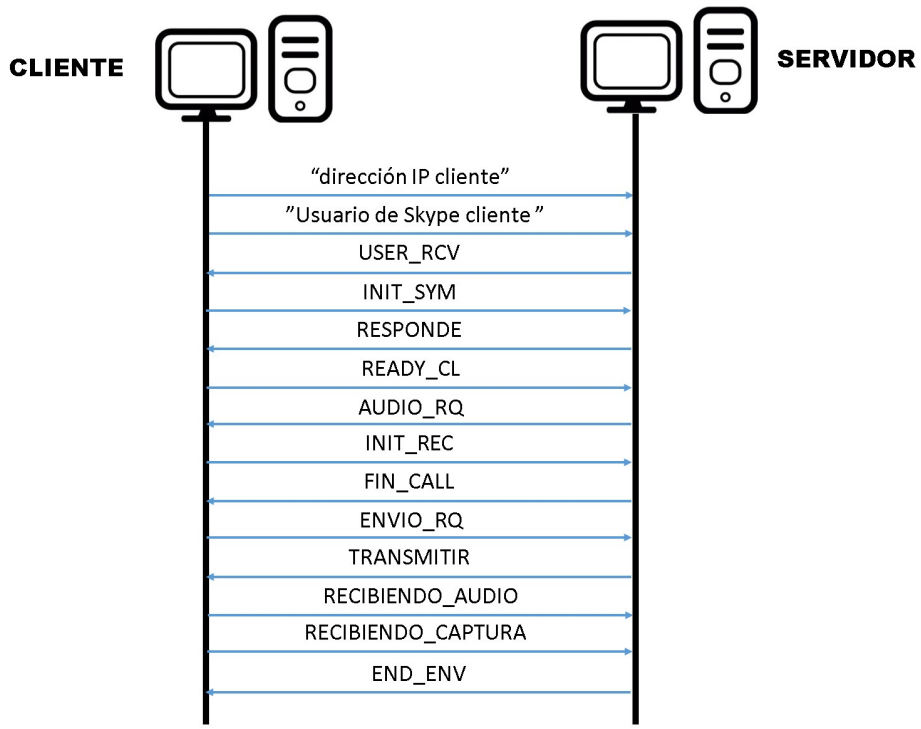


Figura 6.10: Esquema de funcionamiento de la aplicación

dirección IP de la red interna del cliente al servidor. Esta información se usa para construir los *Socket* usados en los métodos que intercambian información desde un extremo a otro. Con el envío de este mensaje y la correcta recepción en el servidor, se inicia un proceso encargado de enviar del cliente al servidor los datos de la simulación(perfiles de tráfico y los instantes de tiempo en los que se inicia cada perfil). Este proceso se detalla en la sección 6.2.4. Por último indicar que al finalizar el proceso anteriormente descrito, se ejecuta desde servidor un archivo *batch*. Este archivo contiene las ordenes necesarias para ejecutar desde linea de comandos el programa *Pamela*:

```
1 cd C:\Program Files\Pamela
2 start pamela.exe
```

Este programa al establecer una llamada con *Skype* graba automáticamente la llamada

Los archivos *batch* son archivos de texto con extensión *bat* que contiene una serie de comandos que pueden ser ejecutados en sistemas, como en nuestro caso *Windows*.

Para le ejecución de este tipo de archivos desde JAVA es necesaria la clase `Runtime` [31]. Esta clase permite la interactuación entre el sistema

operativo en el que se ejecuta y la propia aplicación. Esta clase posee un método denominado `exec(String command)`. Este método ejecuta un comando (pasado como argumento, *command*) propio del sistema, en este caso de *Windows7*.

Usando el método `exec(String command)` se ha implementado, en el código tanto del servidor como en la del cliente, un método. Con el objetivo de facilitar y simplificar la ejecución de archivos *batch* a lo largo de toda la simulación se ha implementado un método. Este método llamado `runBAT(String url)`, es un método de tipo *void*, es decir no devuelve nada. Y se le pasa como argumento *String url*, es decir una cadena de caracteres. La cual indica la ruta donde se encuentra exactamente el *batch* que se desea ejecutar. Por tanto el método se compone únicamente de esta linea:

```
1 Runtime.getRuntime().exec("cmd.exe " + url)
```

En primer lugar se necesita obtener el objeto *Runtime* asociado a la aplicación y sobre ese objeto se llama al método `exec`. Destacar que para la la correcta ejecución de los archivos *batch* es necesario indicar que se ejecute usando la consola de Windows. Para ello concatenamos el texto `cmd.exe` a la ruta de el archivo.

- El siguiente mensaje igual que el anterior manda información necesaria para la simulación al servidor, en este caso se envía el usuario de Skype usado por el cliente. Es necesario que el servidor conozca este usuario, ya que es este el que realiza la llamada.

- `USER_RCV`, este mensaje se genera en el servidor al comprobar que el usuario de *Skype* enviado anteriormente, es valido. Una vez realizada esta comprobación, el mensaje es enviado al cliente con el objeto de informar que el usuario es válido y puede continuar la ejecución de la simulación.

- `INIT_SYM`, la recepción de este mensaje en el servidor indica que el cliente esta preparado para recibir la llamada de Skype. Por tanto el servidor usando el método, `runBAT(String url)`, ejecuta el archivo que realiza una llamada. El archivo batch se compone de los siguientes instrucciones:

```
1 cd "C:\Program Files (x86)\Skype\Phone"
2 start Skype.exe /callto:%1
```

En primer lugar es necesario cambiar del directorio actual al directorio en el cual se instaló *Skype*. Para ello se usa el comando de la consola de *windows* `cd [destino]`.

Y por último siguiendo la información obtenida de la página web de soporte de Skype [32] se lanza el comando `/callto:[usuarioSkype]`. En el archivo *batch* el usuario de *Skype* es sustituido por %1. Esta es la forma que se usa en este tipo de archivos para introducir variables. Por tanto en la *url*, que se le pasa como parámetro al método `runBAT(String url)`, es necesario concatenar el usuario de Skype del cliente (este usuario fue recibido anteriormente).

- `RESPONDE`, tras enviar este mensaje, el servidor debe comenzar a capturar el tráfico generado por la llamada de *Skype*. Para realizar esta captura es necesario ejecutar un archivo *batch*, como en los casos anteriores con el método `runBAT(String url)`. Este *batch* esta compuesto por los siguientes comandos:

```
1 cd C:\Program Files\Wireshark
2 tshark.exe -i externa -w C:\TestSkype\Capturas\servidor.pcap
      -f "udp src net 10.10.210.4 and dst net 10.10.210.3"
```

En primer lugar es necesario ir al directorio en el que se encuentra instalado *WireShark*. Para ello se usa el comando de la consola de *windows* `cd [destino]`.

En segundo lugar es necesario ejecutar *TShark*, detallado en 5.4, con una serie de condiciones. Con el fin de capturar únicamente el tráfico que genera la llamada de *Skype* en un sentido, del cliente al servidor. Estas condiciones se exponen a continuación:

- `-i [interfaz]`: Se introduce la interfaz de la cual se va a realizar la captura. En este caso se introduce `externa`, este es el nombre de la tarjeta, que se ha configurado en la maquina virtual para que tenga acceso a Internet.
- `-w [archivo salida]`: En este caso se indica la ruta en la cual se va a guardar la captura. Por lo explicado anteriormente sobre la estructura de carpetas, esta es `C:\TestSkype\Capturas \servidor.pcap`
- `-f [filtro de captura]`: En primer lugar se le indica que filtre por protocolo, indicándole que solo capture los paquetes *udp*. Finalmente se filtra por dirección IP de origen *src net 10.10.210.4* que corresponde a la del cliente. También es necesario filtrar por la IP de destino que corresponde con la del servido, *dst net 10.10.210.3*.

Al recibir en el cliente el mensaje `RESPONDE`, en este también se debe comenzar a capturar el tráfico generado por *Skype*. Para ello se realiza igual que en el servidor. La única diferencia es que en el archivo

batch, cuando se ejecuta *TShark* el parámetro `-w [archivo salida]` es `C:\TestSkype\Capturas\cliente.pcap`.

Tras comenzar la captura de trazas, el cliente debe responder la llamada entrante. Para ello es necesario la utilización de la herramienta *Clisk*, especificada detalladamente en el Capítulo 5. El uso de esta herramienta se hace ejecutando con el método `runBAT(String url)` un archivo *batch*. Este archivo contiene la siguiente información:

```
1 cd C:\TestSkype\Clisk
2 start clisk.py ans
```

En primer lugar es necesario, ir al directorio en el cual se encuentran todos los archivos de la herramienta.

Finalmente se ejecuta *clisk* con el parámetro `ans`, con lo que se consigue responder la llamada.

- `READY_CL`, este mensaje recibido en el servidor únicamente sirve para informar al servidor que el cliente a conseguido responder la llamada con éxito.

- `AUDIO_RQ`, al recibir este mensaje es necesario enviar el audio desde el cliente al servidor. Para ello se ejecuta un archivo *batch*, con la siguiente información:

```
1 cd C:\Program Files\VideoLAN\VLC
2 start vlc.exe --play-and-exit %1
```

Como en los casos anteriores, primeramente es necesario ir al directorio donde se encuentra instalado, en este caso, el reproductor multimedia *VLC*. Para ello se usa el comando de la consola de *Windows* `cd [destino]`.

Finalmente se ejecuta el programa *VLC* con los siguientes parámetros:

  - `--play-and-exit`: con este parámetro se indica que al terminar de reproducir el audio el programa se cierre.
  - `.%1`: Ahora se introduce la ruta completa donde se localiza el audio elegido para el test. Esta dirección se introduce en forma de variable. Por tanto al ejecutar este *batch* desde *JAVA* se pasa la ruta del audio concatenada con la ruta donde se encuentra el *batch*.

- `INIT_REC`, la recepción de este mensaje en el servidor implica que este, debe ir modificando las características de la red según los datos de simulación introducidos por el cliente. Este proceso se especifica mas detalladamente en la sección 6.2.5,

- `FIN_CALL`, se envía este mensaje al cliente para notificar que la parte de la llamada de la simulación ha finalizado. Después del envío de este mensaje el servidor debe finalizar la llamada. Para ello se vuelve hacer uso de la herramienta *clisk*. El proceso es similar al de responder la llamada, el único cambio es que en el *batch* que se debe ejecutar el parámetro que se pasa a *Clisk*, en este caso es `ha`:

```
1 start clisk.py ha
```

- `ENVIO_RQ`, este mensaje implica tanto en el cliente como el servidor que se debe detener la captura, que se está realizando con *TShark*. Para ello es necesario ejecutar un *Batch* que contiene:

```
1 taskkill /IM dumpcap.exe /T
```

  Con este comando propio de la consola de *Windows*, se consigue finalizar una tarea o un proceso. En este caso los parámetros que se añaden, en primer lugar `/IM [nombre proceso]` indica el nombre del proceso que se desea finalizar. En este caso *dumpcap.exe* que corresponde con el proceso de *Tshark*. Y En segundo lugar `/T` indica que también se finalizan todos los procesos hijos del proceso principal.

- `TRANSMITIR`, este mensaje indica al cliente que debe prepararse para recepción de información.

- `RECIBIENDO_AUDIO`, Al recibir en el cliente este mensaje, el cliente ejecuta el método:

```
1 recibirFichero()
```

  Este método se ha especificado en la sección 6.2.4. Este método recoge la información enviada desde el servidor. Para el envio de esta información, se usa el método:

```
1 enviarFichero("C:\TestSkype\audios\servidor.wav")
```

  Este método también se ha especificado en 6.2.4. Como se puede observar el argumento que se le pasa, indica el archivo que se va enviar. En este caso se envía el audio recogido por el servidor.

- `RECIBIENDO_CAPTURA` Al recibir este mensaje en el cliente se repite el proceso realizado en el punto anterior. La única modificación en este caso es el argumento que se pasa al método `enviarFichero` ya que como en este caso se desea enviar la captura de tráfico. La ruta es:

```
1 C:\TestSkype\Capturas\servidor.pcap
```

- `END_ENV`, el servidor al enviar este mensaje da por finalizada la simulación. por tanto vuelve a ejecutar el programa desde el principio a la espera de nuevas conexiones de clientes.

  Al recibir este mensaje en el cliente comienza la ejecución de una serie de métodos, detallados en la sección 6.2.6. Con los cuales se pretende recoger los resultados de la simulación y procesar estos resultados. Se procesa con el objetivo de obtener información útil que más adelante se mostrará al usuario.

Finalmente el usuario a través de la interfaz gráfica elige las estadísticas que desea que se muestren. En la sección 6.2.7 se muestra detalladamente el proceso para obtener las gráficas.

6.2.1. Recogida de los datos de la simulación a través de la interfaz gráfica

Para que la aplicación creada sea capaz de realizar una simulación, es necesario que se dispongan de una serie de datos. La introducción de esta información es llevada a cabo por parte del usuario que actúa como cliente, por medio de la interfaz gráfica del programa.

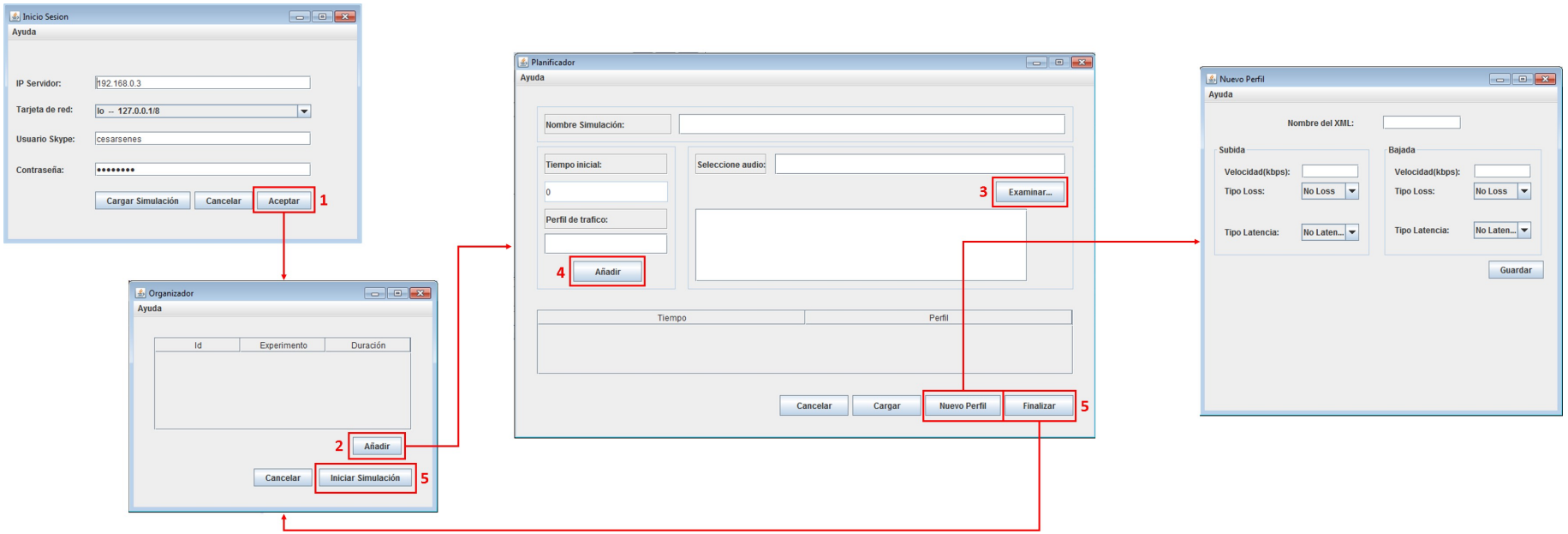


Figura 6.11: Esquema introducción de los datos de la simulación en el cliente.

Indicar que como se observa en la Figura 6.11, esta parte de la interfaz consta de cuatro ventanas. Estas cuatro ventanas en la implementación en *JAVA*, son cuatro clases diferentes y cada una de ellas extiende de la clase *javax.swing.JFrame*. Esta clase *JFrame* nos permite crear ventanas así como colocar y manejar diferentes elementos gráficos.

Con la ayuda del esquema mostrado en la figura 6.11, se va a explicar clase a clase como se realiza esta operación:

1. **Inicio sesión:** Al iniciar la simulación en el cliente se muestra esta ventana. En esta ventana se muestra al usuario los siguientes campos:

   - Dirección IP del servidor.
   - Tarjeta de red Cliente.
   - Usuario Skype.
   - Contraseña Skype.

   Estos campos son editables y todos ellos deben ser introducidos por el usuario. Una vez que se han rellenado todos los campos, para continuar con la simulación es necesario pulsar el botón *Aceptar*.

   Al pulsar este botón se ejecuta un método el cual realiza las siguientes operaciones:

   - En primer lugar es necesario que deje de mostrarse esta ventana, para ello se usa el método:

   ```
   setVisible(boolean b)
   ```

   Este método pertenece a la clase *Component* [33]. Todos las representaciones gráficas que se muestran en pantalla y el usuario puede interactuar con ellos, son objetos de la clase *Component*. Por tanto la ventana *InicioSesion* es un objeto de la clase *Component*. Por este motivo se puede usar este método para ocultar esta ventana, es necesario pasarle como argumento un *false*.

   - Ahora es necesario iniciar *Skype*. Para ello usando la información obtenida de la pagina de soporte de Skype [32]. Donde se nos indica como iniciar Skype desde linea de comandos introduciendo ciertos parámetros. Se crea un archivo *batch*, con el siguiente contenido:

   ```
   cd "C:\Program Files\Skype\Phone"
   start Skype.exe /minimized /nosplash /username:%1
       /password:%2
   ```

   Con la primera linea se traslada del directorio actual al directorio donde se encuentra instalado *Skype*. Con la siguiente linea se inicia Skype indicando que lo haga minimizado(*/minized*), que

no muestre la pantalla de presentación(*/nosplash*). Por ultimo se le pasa como variable del *batch* el usuario(*/username: %1*) y la contraseña(*/password: %2*). Por tanto para iniciar Skype es necesario ejecutar este archivo *batch*. Pasándole como argumento el usuario y la contraseña introducida por el usuario.

- Por último es necesario enviar esta información (usuario, dirección IP del cliente y del servidor) recogida a la siguiente clase (*Organizador*). Para ello es necesario crear un objeto de la clase *Organizador*. Para crear este objeto es necesario usar un constructor, al cual le pasamos la información que se va usar en las siguientes clases, de esta manera:

```
1 Organizador o = new Organizador(dirIPServ,dirIPCliente,user);
```

2. **Organizador:** En esta ventana se va ir mostrando el nombre y la duración de cada experimento realizado, como se muestra en la imagen 6.12. Una simulación esta compuesta por un número determinado por el usuario de experimentos. Cada experimento esta formado por un audio, una serie de instantes de tiempo y unos perfiles de tráfico.

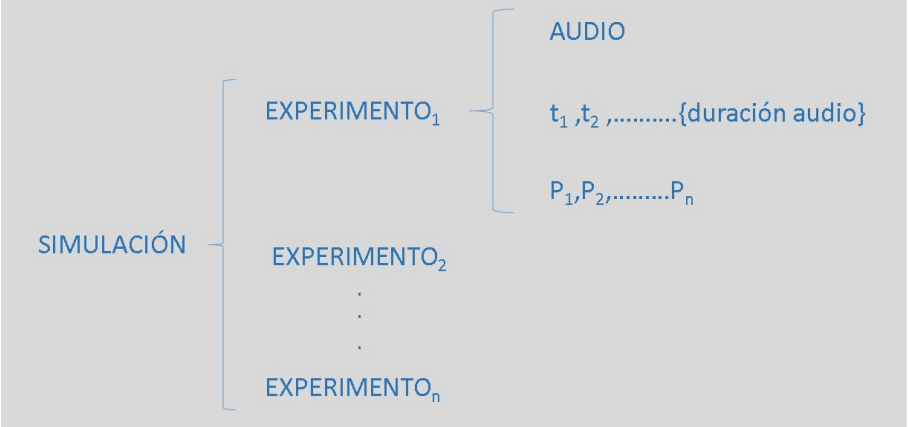


Figura 6.12: Esquema composición de una simulación.

Para continuar con la simulación es necesario pulsar el botón *Añadir*. Al pulsar este botón se crea la siguiente ventana. Y finalmente como hemos explicado anteriormente ocultamos esta ventana (setVisible(false)).

3. **Planificador:** Esta es la ventana donde se recoge toda la información de cada experimento. En primer lugar es necesario elegir un audio.

   Para ello se pulsa en el botón *Examinar...*. Al pulsar este botón se muestra un explorador de archivos del equipo cliente (por defecto se

abre en la carpeta C:/TestSkype/audios). A continuación se debe seleccionar un audio. Es necesario que el audio cumpla con las siguiente características:

- PCM_SIGNED
- 16000.0 Hz
- 16 bit
- mono
- 2 bytes/frame
- ittle-endian

En el caso de que audio no cumpla con estas especificaciones, se muestra un mensaje indicando que el audio no es permitido por el sistema. Por el contrario si el audio tiene un formato aceptado la ruta de este audio es añadida a un *arrayList* llamado *audios*.

Una vez seleccionado el audio se requiere que se especifique el perfil de trafico y el instante de tiempo en el cual comienza a ejecutarse. Para ello se rellenan los campos de *Tiempo inicial* y *Perfil de tráfico* posteriormente se pulsa el botón *Añadir*. Al pulsar este botón estos datos se van añadiendo a dos *arrayList*. Uno de ellos llamado *tiempo* y otro llamado *perfiles*. Esta operación se repite tantas veces como el usuario desee. Los tiempos introducidos son validados de manera que no se pueda introducir un tiempo mayor que la duración del audio. Así como tampoco esta permitido introducir un tiempo menor al introducido anteriormente.

Otra forma de rellenar todos estos datos es, cargar un experimento previamente guardado. Para ello es necesario pulsar el botón *cargar*. Al pulsar este botón se muestra un explorador de archivos y el usuarios debe elegir el archivo a cargar. El archivo mostrado en la Figura 6.13 es un ejemplo de un archivo en el cual se encuentra guardado un experimento, ya que contiene los datos de dicho experimento. En el archivo se diferencia cada tipo de dato, ya que en la linea anterior se escribe un texto diferente. Estos textos son:

- *AUDIO*: es único en cada experimento. Se localiza en la primera linea del archivo. Indica que la siguiente linea contiene la ruta donde se encuentra el audio del experimento.
- *TIME*: indica que en la siguiente linea se encuentra el instante de tiempo. Este instante de tiempo indica el momento justo en el cual unas condiciones de red deben ser aplicadas al enlace. Esas condiciones de red vienen determinadas por el perfil.

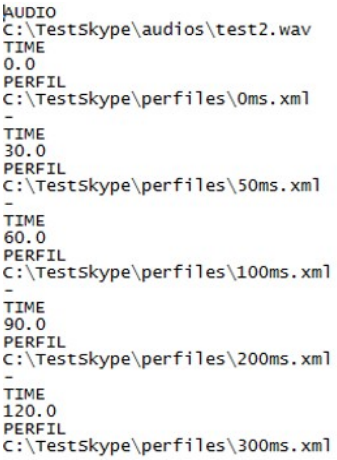


Figura 6.13: Estructura del archivo donde se guarda un experimento.

- *PERFILl*: indica que en la siguiente linea se muestra la ruta. En la cual se encuentra el archivo XML, que representa un determinado perfil de trafico.

Para diferenciar cada par de datos Time-perfil. Se inserta después de la dupla el carácter -. La aparición de este carácter implica que el indice del array perfil y time debe aumentar.

Por tanto conociendo la estructura de estos archivos, se recorre el archivo linea a linea comprobando que tipo de información se esta leyendo. Y por último se añade a uno de los tres *arrayList*. Para recorrer el archivo linea a linea es necesario el uso del método *readLine()* de la clase *BufferedReader*.

Una vez introducidos los datos para continuar con la simulación, se debe pulsar el botón finalizar. Al pulsar este botón el sistema nos pregunta si deseamos guardar este experimento, de indicar que se desea guardar se crea un archivo como el mostrado en la Figura 6.13.

Ahora se vuelve a mostrar la ventana del *Organizador*. En este punto tienes dos opciones o introduces un nuevo experimento, es decir se repite este proceso o se pulsa el botón de *Iniciar Simulación*. En caso de pulsar este botón se da por finalizada la fase de introducción de datos de la simulación.

4. **Nuevo perfil:** esta ventana es mostrada en el caso de que mientras se introducen los datos en el *Planificador*, sea necesario usar un perfil de tráfico no existente en tu equipo. Para añadir un nuevo perfil es necesario que el cliente pulse el botón *Nuevo perfil*. A continuación se muestra una ventana con múltiples opciones.

Una vez rellenada toda la información de esta ventana, se pulsa en el botón de *Aceptar*. Por ultimo estos datos son recogidos e introducidos en un archivo XML de manera tal que el emulador de red sea capaz de entenderlos.

6.2.2. Envío y recepción de mensajes desde un extremo a otro

Para realizar la operación de envío y recepción de mensajes entre ambos extremos de la simulación es necesario implementar estos dos métodos:

```
1 String envRecCliente(String msg, String condicion)
2 String envRecCServidor(String msg, String condicion)
```

El primero implementado en el cliente y el segundo en el servidor. Ambos métodos se implementan de la misma forma, salvo en la construcción de los *socket*. Estos métodos usan comunicación mediante UDP, por tanto el *socket* creado es del tipo *DatagramSocket*. Al crear este *socket* en cada uno de los extremos debemos introducir un puerto diferente de esta forma:

- `DatagramSocket socketC = new DatagramSocket(P_CLIENTE)`, siendo *P_CLIENTE* una constante, con valor 5050.
- `DatagramSocket socketS = new DatagramSocket(P_SERVIDOR)`, siendo *P_SERVIDOR* una constante, con valor 5051.

Con esta operación indicamos a cada uno de los *socket*, el puerto por el cual debe esperar la información del otro extremo.

Estos métodos están claramente diferenciados en dos partes. Al ejecutar el método dependiendo del parámetro introducido como argumento *condición* se ejecuta una parte o la otra. Diferenciamos entre:

- *ENVIAR*: En primer lugar es necesario crear un datagrama con la información que se desea enviar. Para este fin se usa la clase *DatagramPacket*. Para la creación de un objeto de esta clase es necesaria la siguiente información:
  - Información contenida en el datagrama. En este caso esta información, es el mensaje (*msg*) pasado como argumento al método. Es necesario transformar la cadena de caracteres, *msg*, en una secuencia de *bytes*. Para ello se usa el método *getBytes()* de la clase *String* sobre *msg*.
  - Tamaño de los datos contenidos en el paquete. En este caso la longitud de de la cadena *msg*. Para conseguir este dato usamos el método *length()* de la clase *String*sobre *msg*.
  - Dirección IP del destino.

- Puerto de destino.

A continuación se muestra como se ha creado el datagrama en el cliente con destino al servidor:

```
new DatagramPacket(msg.getBytes(),msg.length(),servidor,
    P_SERVIDOR)
```

Y ahora datagrama creado en el servidor con destino al cliente:

```
new DatagramPacket(msg.getBytes(),msg.length(),cliente,
    P_CLIENTE)
```

Una vez creado creado el datagrama, este se envia al otro extremo a través del *socket* (*SocketC* y *SocketS*). Para ello se usa el método *send(DatagramPacket p)* de la clase *DatagramSocket*. Finalmente se cierra el socket.

- *RECIBIR*: En primer lugar es necesario crear un datagrama, en el cual se almacena la información recibida del otro extremo. Para ello se usa la Clase *DatagramPAcket*, de esta forma:

```
byte[ ] buffer = new byte[50];
new DatagramPacket(buffer, buffer.length);
```

Únicamente es necesario un *buffer* donde se va almacenando los datos entrantes y el número de *bytes* a leer. En este caso se ha indicado que como máximo el número de *bytes* entrantes es 50.

Una vez creado el datagrama se usa el método *receive()* de la clase *DatagramSocket* sobre el *socket* creado anteriormente. Al ejecutar este método se bloquea la ejecución del programa hasta que se recibe un datagrama. Mientras tanto el *buffer* se llena con los datos recibidos. Finalmente es necesario pasar el contenido del *buffer* a una cadena de caracteres. Esta cadena es devuelta por el método, con esta información a lo largo de la ejecución del programa se va comprobando que la ejecución del programa va correctamente.

6.2.3. Sincronización del cliente y el servidor

Debido a que en este proyecto es importante el retardo extremo a extremo, ya que este influye de forma significativa en la calidad de la llamada de VoIP. Es necesario que los relojes de ambos equipos(cliente y servidor) estén sincronizados, es decir ambos tengan la misma referencia temporal.

El proceso que se sigue para la sincronización esta basado en el usado por SNTP, como se muestran Figura 6.14. Indicar que los paquetes UDP, usados para obtener las marcas de tiempo, tienen el mismo tamaño con el objetivo de que el tamaño no afecte al tiempo de propagación.

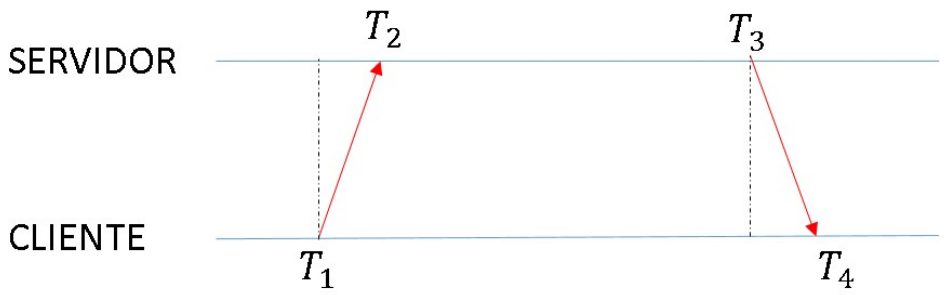


Figura 6.14: Mecanismo estimación offset mediante *time stamp*

El proceso seguido ha sido el siguiente:

1. El cliente marca el instante T_1', en el cual se envía el primer paquete al servidor.
2. El servidor marca el instante T_2, en el cual se recibe el primer paquete.
3. El servidor marca el instante T_3, en el cual se envía el siguiente paquete al servidor.
4. El cliente marca el instante T_4', en el cual se recibe el último paquete.

Indicar que los tiempos T_1' y T_4' incluyen el tiempo que el reloj del cliente esta adelantado o atrasado con respecto al servidor(*offset, O*). Por tanto:

$$T_1' = T_1 + O \tag{6.1}$$

$$T_4' = T_4 + O \tag{6.2}$$

Así para obtener el *offset* se calcula primeramente los retardos extremo a extremo, aplicando las ecuaciones6.1 y 6.2, definimos las ecuaciones 6.3 y 6.4:

$$R1 = T_2 - T_1' = T_2 - T_1 - 0 \tag{6.3}$$

$$R2 = T_4' - T_3 = T_4 - T_3 + 0 \tag{6.4}$$

Suponiendo que los tiempos de propagación en ambos sentidos son iguales ecuación 6.5, desarrollando 6.6 6.7 obtenemos que el resultado del tiempo que se debe sumar o restar(dependiendo del signo obtenido) al reloj del servidor viene definido por la ecuación 6.8.

$$R1 = R2 \tag{6.5}$$

$$T_2 - T1 - O = T_4 - T_3 + O \tag{6.6}$$

$$T_2 - T1 - T_4 + T_3 = 2O \tag{6.7}$$

$$O = \frac{T_2 - T1 - T_4 + T_3}{2} \tag{6.8}$$

Finalmente destacar que la exactitud de este método depende de las condiciones de red.

Implementación

La implementación del mecanismo de sincronización se realiza mediante cuatro métodos *JAVA*(dos en el servidor y dos en el cliente) definidos en el programa principal.

Para la comunicación de estos métodos se ha usado la clase *DatagramSocket* [34], la cual permite tanto el envío como recepción de datagramas.

A continuación se describe la funcionalidad de estos métodos de manera secuencial:

- `envSync()`: implementado en el cliente se encarga de tomar el instante de tiempo actual, el cual encapsula en un datagrama y lo envía al servidor. Destacar que este datagrama se rellena con el objetivo que tanto el datagrama enviado desde el cliente al servidor como el enviado del servidor tenga el mismo tamaño. Con esto se pretende que el tamaño del datagrama no influya en el tiempo de propagación y por tanto obtener unos resultados mas exactos, como se muestra en la Figura 6.15 a).

- `recSync()`implementado en el servidor recibe el datagrama enviado desde el cliente, de manera inmediata se toma el instante de tiempo y se encapsula en el datagrama.

- `envSync()` implementado en el servidor se vuelve a tomar el tiempo, se añade al datagrama, como se muestra en la Figura 6.15 b) y se envía al cliente.

- `recSync()` implementado en el cliente recibe el datagrama y toma el instante de tiempo. Ahora procesa el paquete recibido obteniendo los instantes de tiempo de cada fase. Por tanto ya se tienen todos los tiempos necesarios para aplicando la ecuación 6.8, se obtiene el tiempo a sumar o restar. Dependiendo de si un reloj está adelantado o retrasado con respecto al otro. Con esto se consigue sincronizar los relojes de ambas máquinas.

Por otro lado indicar que para los *timestamps*, es decir para la obtención de los instantes de tiempo se ha usado el método de la clase `System` [35] `currentTimeInMillis()`. Este método nos devuelve un dato de tipo `long` con la diferencia en milisegundos entre el instante de tiempo actual y las 00:00, del 1 de Enero de 1970 UTC(*Coordinated Universal Time*). UTC es el estándar principal para la regulación del tiempo.

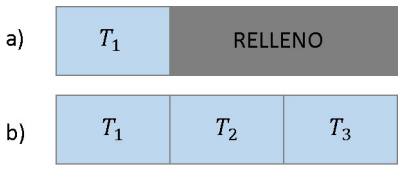


Figura 6.15: Contenido de los datagramas en las diferentes fases de la sincronización

6.2.4. Envío de datos de la simulación del cliente al servidor

Para la correcta realización de la simulación, es necesario enviar los datos introducidos por el cliente al servidor. Esto se debe a que en el lado del servidor es donde se simulan las perdidas- Por tanto necesita los archivos *xml* que necesita el emulador para definir un determinado perfil de tráfico. También es necesario que el servidor conozca el orden en el cual se aplica cada perfil, así como el instante de tiempo en el cual se ejecuta cada perfil. Con este objeto se implementan estos dos métodos:

- En el cliente `enviarDatos()`: Este método realiza dos operaciones diferentes. Por un lado el método envía los archivos xml con la información que debe cargar el emulador.

  Para esta tarea es necesario usar otro método *enviarFichero(String filename)*. Este método necesita dos flujos uno de entrada y otro de salida.

  Para crear el flujo de entrada se usa la clase *BufferedInputStream*, que almacena en un buffer el argumento de entrada. En este caso el argumento que se pasa son los bytes del fichero, que se introduce al método como argumento al método(*new File(filename)*),se implementa de este modo:

```
1 File file=new File(filename);
2 BufferedInputStream bis = new BufferedInputStream(new
      FileInputStream(file));
```

  Para crear el flujo de salida se usa la clase *BufferedOutputStream*. Es necesario indicar donde debe ir escribiendo los bytes almacenados en el buffer. En este caso queremos escribir estos bytes en el otro extremo (servidor). Por tanto es necesario usar un *socket* con destino al servidor, como se muestra a continuación:

```
1 socket_snd = new Socket(servidor,P_SERVIDOR);
2 BufferedOutputStream bos = new BufferedOutputStream(socket_snd.
      getOutputStream());
```

  Finalmente el método *enviarFichero(String filename)*, usando un bucle

lee byte a byte el flujo de entrada y lo va escribiendo en el de salida. Así se consigue enviar un fichero de un extremo a otro.

Por tanto, para que *enviarDatos()* consiga enviar todos los perfiles de red de una determinada simulación, únicamente es necesario recorrer el *arrayList* perfiles y usando el método *enviarFichero(String filename)* enviarlos uno a uno. Destacar que entre archivo y archivo dentro del bucle se envía un mensaje al servidor. Con el objetivo de notificar a este que se envía otro perfil distinto. Para enviar este mensaje se usa el método *envRecCliente("NEXT", "ENVIAR")* 6.2.2.

Por otro lado el método *enviarDatos()* envía la información de la simulación. Esta información esta contenida en los vectores que contienen los instantes de tiempo en el cual se carga cada perfil de trafico, así como las rutas de los propios perfiles. Es necesario en este método la creación un flujo de salida usando la clase *DataOutputStream*, con el objeto de mandar la información al servidor. Por este motivo es necesario pasarle un *socket* con destino al otro extremo, como se muestra a continuación:

```
1 ServerSocket ssN = new ServerSocket(P_SERVIDOR);
2 Socket sN = ssN.accept();
3 DataOutputStream out = new DataOutputStream(new
      BufferedOutputStream(sN.getOutputStream()));
```

Una vez creado el flujo de salida únicamente es necesario recorrer los arrayList usando un bucle y enviar el String contenido en cada posición. Para este fin se usa el método *writeUTF(String str)* de la clase *DataOutputStream*.

- En el servidor `recibirDatos()`: Este método debe estar preparado para recibir la información desde el otro extremo. En primer lugar debe recibir los archivos del emulador.

  Para esta tarea es necesario implementar un método llamado *recibirFichero()*. Este método necesita como en el caso anterior dos flujos. Uno de entrada el cual guarda en un buffer la información recibida por un *socket*. Este *socket* conecta ambos extremos. Por otro el de salida almacena datos en un buffer para escribirlos en el flujo de salida. En este caso un nuevo archivo creado con la clase *File*.

  Por tanto en el método *recibirDatos()*, usando un bucle con el método *envRecServidor* 6.2.2 el mensaje enviado por el cliente. Dependiendo del mensaje recibido puede ocurrir:

  - *NEXT*: En el caso de recibir este mensaje, el servidor ejecuta el método *recibirDatos()*. Este método se mantiene en ejecución hasta recibir el fichero completo.

- *FIN*: En el caso de recibir este mensaje, se fuerza la salida del bucle y el método continua con la siguiente tarea.

Así se consigue la recepción de todos los ficheros enviados por el cliente. Por otro lado ahora es necesario recibir y almacenar el instante de tiempo y el orden de cada perfil de tráfico. Para esta tarea es necesario un flujo que reciba información del socket, que conecta ambos extremos. Una vez creado este flujo usando el método *readUTF()* de la clase *DataInputStream*, se van leyendo y almacenado en dos *ArrayList* (perfil y tiempo) la información recibida.

6.2.5. Modificación de los parámetros de la red según los datos iniciales de la simulación

Durante la llamada es necesario que las características de la red se modifiquen. Para conseguir este propósito es necesario realizar una serie de operaciones en el servidor.

Una vez obtenidos los *ArrayList* que contienen los instantes de inicio de cada perfil (tiempo) y el listado con los perfiles de tráfico (perfiles), realizado en la sección 6.2.4.

Es necesario calcular cuanto tiempo va estar en ejecución cada perfil. Para realizar esta operación es necesario implementar un bucle que recorra el *ArrayList* tiempo. Para cada iteracción del bucle es necesario obtener la posición actual de la lista así como la consecutiva. A continuación se restan el valor de ambos números y el valor obtenido se almacena en un *ArrayList*, llamado *auxt*. Son necesarios esta listas de datos ya que para ejecutar un determinado perfil se tiene que ejecutar un archivo batch, como el siguiente:

```
1 cd C:\Program Files\Network Emulator for Windows Toolkit\bin\
2 start newtman -t %1 %2
```

Este archivo batch para ser ejecutado es necesario pasarle dos variables. En primer lugar %1 la ruta del perfil que se va ejecutar y por otro lado %2 la duración.

Por tanto para realizar la simulación completa y que de forma automática se vaya cambiando el perfil aplicado, es necesario realizar un bucle que recorra el *ArrayList* perfiles. Dentro de este bucle en cada iteración se ejecuta el archivo batch, visto anteriormente, pasandole como argumentos el perfil actual, así como la posición actual de la lista *auxt* (duración). Destacar que después de ejecutar el archivo *batch*, para que la ejecución del programa no continué es necesario detenerla. Para pausar la ejecución de un programa en java es necesario usar el método *sleep(int milis)* de la clase *Thread*. A este método se le pasa como argumento el mismo valor de duración que se le a pasado al archivo *batch*.

6.2.6. Recogida y procesamiento de los resultados de la simulación

Una vez finalizada la llamada y enviada la trazas obtenidas mediante *Wireshark* en el servidor al cliente. Es el momento de transformar ambas capturas archivo con extensión *pcap* en un archivo de texto plano(*txt*), con el objetivo de poder tratar esta información en JAVA. Para realizar esta tarea se utilizan comandos de *TShark* 5.4. Estos comandos son recogidos en un archivo *batch* como el mostrado a continuación:

```
1 cd C:\Program Files\Wireshark
2
3 tshark -r C://TestSkype/Capturas/cliente.pcap -x  > C://TestSkype/
      Capturas/clDatos.txt
4
5 tshark -t e -r C://TestSkype/Capturas/cliente.pcap  > C://TestSkype/
      Capturas/clTime.txt
6
7 tshark -r C://TestSkype/Capturas/servidor.pcap -x  > C://TestSkype/
      Capturas/serDatos.txt
8
9 tshark -t e -r C://TestSkype/Capturas/servidor.pcap  > C://TestSkype/
      Capturas/serTime.txt
```

Cuando se ejecuta este archivo *batch*, en la carpeta en la que se guardan las capturas, aparecen cuatro archivos de texto. En dos de ellos (clDatos.txt y serDatos.txt) se muestran los datos de cada paquete. El contenido de estos archivos es como el mostrado en la Figura 6.16.

Mientras que los otros dos archivos(clTime.txt y serTime.txt) muestran el instante de tiempo en el que se ha capturado cada paquete. Así como otra información relevante, como es el tamaño de cada paquete y el protocolo usado. Un ejemplo de este tipo de archivos se muestra en la Figura 6.17.

Una vez realizada la transformación de las capturas a archivos de texto, en el programa se implementan los siguiente métodos:

- ***obDatos()***: El objetivo de este método es almacenar en dos *ArrayList* los datos de cada paquete, tanto del cliente como del servidor.
- ***obTime()***: Con este método se consigue almacenar en dos *arrayList* (cliente y servidor), el tiempo de llegada. Este tiempo se muestra en los archivos como los segundos transcurridos desde el uno de Enero de 1970 a las 00:00 hasta la llegada del paquete. De este archivo también se obtiene una lista ordenada con el tamaño de cada paquete. Antes de almacenar los datos de cada paquete se comprueba que este paquete es del tipo UDP. Los paquetes que no cumplen esta condición se desechan. También se elimina esta posición del arrayList de la lista de datos obtenida en el punto anterior.
- ***sincroTrazas()***: El objetivo de este método es el de conseguir sincronizar ambas trazas. Es decir, eliminar paquetes por el principio y por

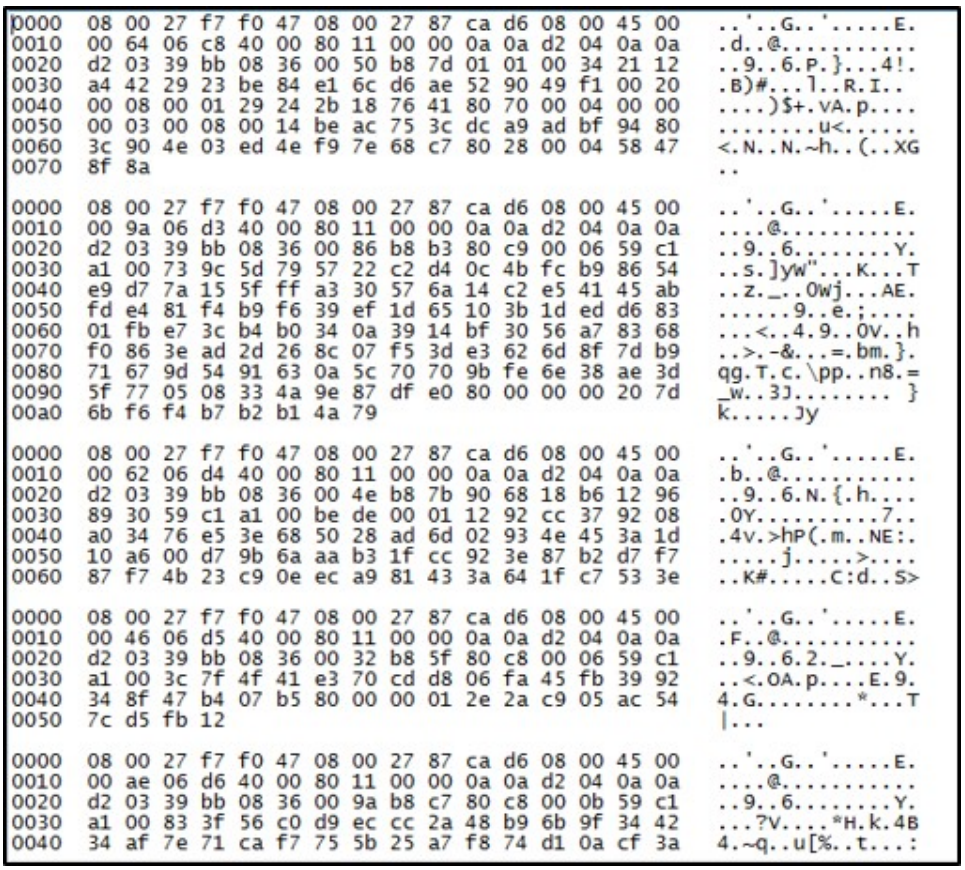
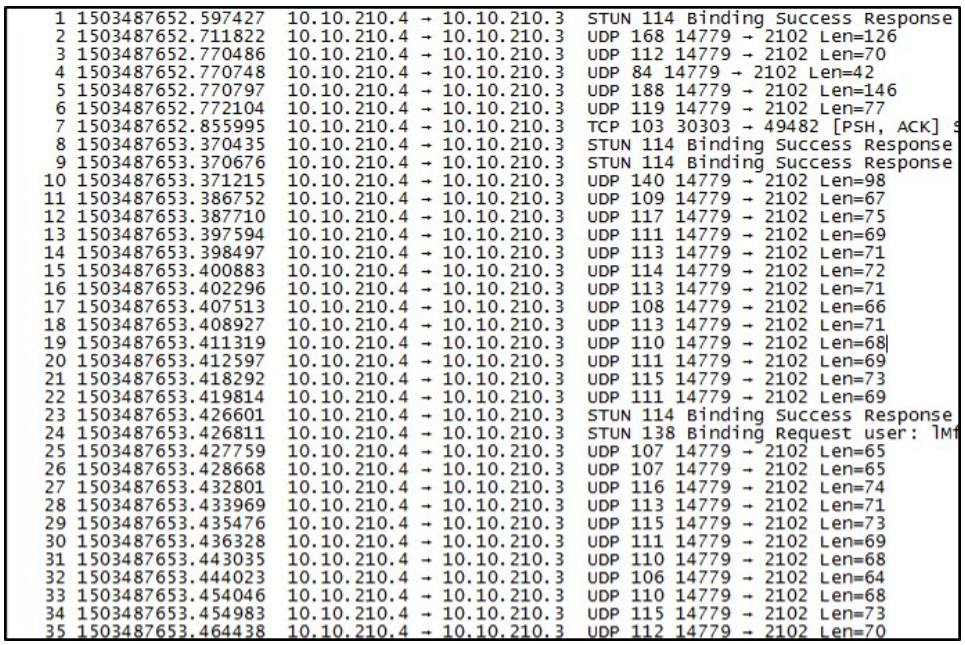


Figura 6.16: Archivo de texto plano obtenido tras la transformación con los datos de cada paquete.

el final para que ambas empiecen y acaben en el mismo paquete. Esto es necesario debido a que el proceso de capturar en ambos extremos no empieza exactamente en el mismo instante de tiempo.

- ***paquetesPerdidos()***: Este método almacena en una lista ordenada el instante de tiempo en el cual se produce una pérdida de un paquete. También elimina los paquetes perdidos del *ArrayList* que contiene los datos del cliente. Finalmente se crea una copia de la lista de tiempos del cliente, eliminado los tiempos que corresponden a paquetes perdidos.

- ***delay()***: Para este método se van a usar dos *arrayList*. Estas listas contienen los instantes de tiempo de los paquetes en el cliente y en el servidor. Destacar que los instantes de tiempo en el cliente son sin los paquetes perdidos. Por tanto ambas listas tiene el mismo tamaño. Finalmente es necesario un bucle que recorra posición a posición una de estas listas. En cada iterción se guarda en otra lista la diferencia



```
    1 1503487652.597427  10.10.210.4 → 10.10.210.3  STUN 114 Binding Success Response
    2 1503487652.711822  10.10.210.4 → 10.10.210.3  UDP 168 14779 → 2102 Len=126
    3 1503487652.770486  10.10.210.4 → 10.10.210.3  UDP 112 14779 → 2102 Len=70
    4 1503487652.770748  10.10.210.4 → 10.10.210.3  UDP 84 14779 → 2102 Len=42
    5 1503487652.770797  10.10.210.4 → 10.10.210.3  UDP 188 14779 → 2102 Len=146
    6 1503487652.772104  10.10.210.4 → 10.10.210.3  UDP 119 14779 → 2102 Len=77
    7 1503487652.855995  10.10.210.4 → 10.10.210.3  TCP 103 30303 → 49482 [PSH, ACK] S
    8 1503487653.370435  10.10.210.4 → 10.10.210.3  STUN 114 Binding Success Response
    9 1503487653.370676  10.10.210.4 → 10.10.210.3  STUN 114 Binding Success Response
   10 1503487653.371215  10.10.210.4 → 10.10.210.3  UDP 140 14779 → 2102 Len=98
   11 1503487653.386752  10.10.210.4 → 10.10.210.3  UDP 109 14779 → 2102 Len=67
   12 1503487653.387710  10.10.210.4 → 10.10.210.3  UDP 117 14779 → 2102 Len=75
   13 1503487653.397594  10.10.210.4 → 10.10.210.3  UDP 111 14779 → 2102 Len=69
   14 1503487653.398497  10.10.210.4 → 10.10.210.3  UDP 113 14779 → 2102 Len=71
   15 1503487653.400883  10.10.210.4 → 10.10.210.3  UDP 114 14779 → 2102 Len=72
   16 1503487653.402296  10.10.210.4 → 10.10.210.3  UDP 113 14779 → 2102 Len=71
   17 1503487653.407513  10.10.210.4 → 10.10.210.3  UDP 108 14779 → 2102 Len=66
   18 1503487653.408927  10.10.210.4 → 10.10.210.3  UDP 113 14779 → 2102 Len=71
   19 1503487653.411319  10.10.210.4 → 10.10.210.3  UDP 110 14779 → 2102 Len=68
   20 1503487653.412597  10.10.210.4 → 10.10.210.3  UDP 111 14779 → 2102 Len=69
   21 1503487653.418292  10.10.210.4 → 10.10.210.3  UDP 115 14779 → 2102 Len=73
   22 1503487653.419814  10.10.210.4 → 10.10.210.3  UDP 111 14779 → 2102 Len=69
   23 1503487653.426601  10.10.210.4 → 10.10.210.3  STUN 114 Binding Success Response
   24 1503487653.426811  10.10.210.4 → 10.10.210.3  STUN 138 Binding Request user: lMf
   25 1503487653.427759  10.10.210.4 → 10.10.210.3  UDP 107 14779 → 2102 Len=65
   26 1503487653.428668  10.10.210.4 → 10.10.210.3  UDP 107 14779 → 2102 Len=65
   27 1503487653.432801  10.10.210.4 → 10.10.210.3  UDP 116 14779 → 2102 Len=74
   28 1503487653.433969  10.10.210.4 → 10.10.210.3  UDP 113 14779 → 2102 Len=71
   29 1503487653.435476  10.10.210.4 → 10.10.210.3  UDP 115 14779 → 2102 Len=73
   30 1503487653.436328  10.10.210.4 → 10.10.210.3  UDP 111 14779 → 2102 Len=69
   31 1503487653.443035  10.10.210.4 → 10.10.210.3  UDP 110 14779 → 2102 Len=68
   32 1503487653.444023  10.10.210.4 → 10.10.210.3  UDP 106 14779 → 2102 Len=64
   33 1503487653.454046  10.10.210.4 → 10.10.210.3  UDP 110 14779 → 2102 Len=68
   34 1503487653.454983  10.10.210.4 → 10.10.210.3  UDP 115 14779 → 2102 Len=73
   35 1503487653.464438  10.10.210.4 → 10.10.210.3  UDP 112 14779 → 2102 Len=70
```

Figura 6.17: Archivo de texto plano obtenido tras la transformación con el tiempo de llegada de cada paquete.

entre el tiempo del servidor y el del cliente. El tiempo del servidor debe incluir el tiempo de sincronización de ambos extremos 6.2.3.

- ***jitter()***: Este método recorre posición a posición la lista que contiene los tiempos del servidor. En cada iteración del bucle se almacena en un nuevo *ArrayList*, la diferencia entre el valor actual de la lista de tiempos en el servidor y el anterior de la misma lista.

Con la información recogida por estos método es posible construir las gráficas y mostrar los resultados de la simulación.

6.2.7. Construcción de las gráficas de resultados.

Todas las gráficas mostradas son editables, es decir, el usuario puede indicar un valor deslizando un control en cada una de las gráfica. Este valor corresponde con un intervalo de tiempo, por tanto al calcular los resultados se calculan en cada intervalo. Las gráficas se dividen en dos grupos:

- **Gráficas objetivas**
  - Paquetes perdidos: Es necesario usar la lista ordenada que contiene los instantes de tiempo en los que se produce una pérdida. Por tanto se realiza un bucle que recorra todos los elementos de

la lista anterior. Dentro del bucle se comprueba si el instante de la perdida es menor que el intervalo de tiempo definido por el usuario. de ser así aumentamos en una unidad el contador. En caso contrario añadimos el contador a un *arrayList* e incrementamos el intervalo de tiempo. Se repite esta operación y dentro de la lista se dispone del numero de paquetes perdidos en cada intervalo.

Por otro lado para obtener el tanto por ciento de paquetes perdidos, es necesario almacenar el numero de paquetes totales que se tienen en el mismo intervalo. Para esta tarea solo es necesario recorrer el *ArrayList* que contiene los instantes de tiempo de todos los paquetes en el cliente. Cada posición se compara con el incremento. Si es mayor que el incremento se añade una unidad a un contador. En caso contrario se añade este contador a una lista. Obteniendo así los paquetes totales en un intervalo. Con esta información es sencillo obtener el tanto por ciento de los paquetes perdidos. Basta con dividir posición a posición, los paquetes perdidos entre los totales y dividir por cien.

- *Throughput*: Para obtener esta gráfica es necesario volver a realizar lo explicado en el punto anterior, obtener todos los paquetes de un intervalo.

  Por otro lado para obtener *Throughput* en bits por segundo. Se usa el método explicado anteriormente de obtener todos los paquetes de un intervalo. Con la salvedad que el contador no se incrementa en uno sino se incrementa el tamaño del paquete. Como se dispone de una lista con el tamaño de todos los paquetes únicamente se va accediendo de uno a uno en el *arrayList*.

- Retardo: Para obtener esta gráfica son necesarios dos *arraysList* obtenidos en la sección 6.2.6. Uno contiene el tiempo que tarda cada paquete del cliente al servidor. Mientras que el otro contiene todos los instantes de llegada de los paquetes en el cliente (Sin paquetes perdidos). Por tanto basta con recorrer ambas listas. Dependiendo si el instante de llegada es mayor o menor que el incremento introducido. Aumentamos un contador en el tiempo de retardo. O añadimos ese contador a un *arrayList* con los retardos por incremento.

- **Gráficas subjetivas**

  - *Perceptual Evaluation of Speech Quality*, PESQ [11]: Para calcular el PESQ en un determinado incremento de tiempo, es necesario disponer de los dos audios, tanto el original como el grabado en el servidor. Después se dividen ambos audios usando la aplicación *Sox* en trozos del tamaño del incremento. Una vez divididos

los dos audios, usando un bucle ejecutamos tantas veces como sea necesario la implementación de la ITU del PESQ. En cada iteracción del bucle guardamos en un arrayList los resultados.

- E-Model [12]: Es necesario calcular, como se ha indicado antes, los paquetes perdidos en tanto por ciento y el retardo. Una vez que tenemos estos datos, solo es necesario aplicar el procedimiento indicado en [12].

Capítulo 7

Realización de simulaciones y resultados obtenidos

En este capítulo se muestran los resultados de varias simulaciones. Las simulaciones elegidas son representativas ya que estas muestran claramente como funciona el codec de *bitrate* variable de *Skype*. Para cada simulación, se muestran tanto estadísticas objetivas como subjetivas con el fin de realizar un análisis lo mas detallado posible.

7.1. Variando el *Throughput*

Para esta simulación se usa un audio de unos 600 seg (10 min). Este audio es el recomendado por la *UIT* (Unión Internacional de Telecomunicaciones) en la recomendación P.501 [36] para la medición de calidad de transmisión telefónica. Destacar que el emulador durante la simulación no va a introducir otro tipo de perturbaciones, como son la perdida de paquetes o el retardo, sino que sólo se varía el ancho de banda del enlace.

La planificación de la simulación realizada se muestra en la tabla 7.1. Esta figura muestra, para cada instante de tiempo de la simulación, el perfil usado. En este caso, el perfil únicamente modifica la velocidad del enlace. Indicar que durante los primeros cinco segundos no actúa el emulador de red con el objetivo de que la llamada se establezca correctamente. A partir de ese instante cada 30 segundos va cambiando de manera progresiva el perfil.

En primer lugar, se muestran las gráficas en la Figura 7.1 que muestran la evolución del *throughput*, tanto en paquetes por segundo como en kilobits por segundo. Para estas imágenes se ha elegido un intervalo de tiempo de diez segundos para el cálculo del *throughput*.

En dicha figura se puede observar cómo, a pesar de que la velocidad del enlace es mayor a la mostrada, la tasa máxima de generación de paquetes para una llamada de Skype está entorno a 55 kbps en este caso. Como es de esperar, al bajar la velocidad del enlace, la generación de paquetes de *Skype*

Instante inicial	Perfil
0	sincronización
5	200kbps
35	150kbps
65	100kbs
95	75kbps
125	50kbps
155	25kbps
185	10kbps
215	25kbps
245	50kbps
275	75kbps
305	100kbps
335	150kbps
365	200kbps
395	150kbps
425	100kbps
455	75kbps
485	50kbps
515	25kbps
581	-

Cuadro 7.1: Simulación *Throughput* variable.

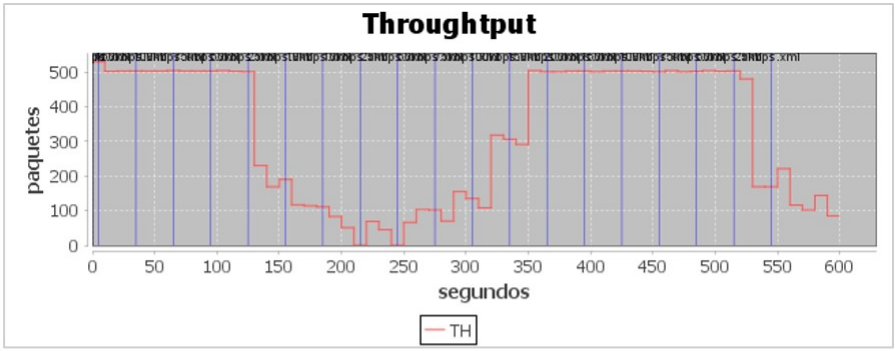
también disminuye.

El estudio realizado para esta simulación se divide en dos partes dependiendo de la objetividad de los resultados obtenidos, como se muestra a continuación:

- **Gráficas Objetivas:**

  Aquí se muestran, en la Figura 7.2, las gráficas con las estadísticas objetivas de la simulación. Para estas gráficas también se ha escogido un intervalo de tiempo de diez segundos para realizar los cálculos.

a)



b)

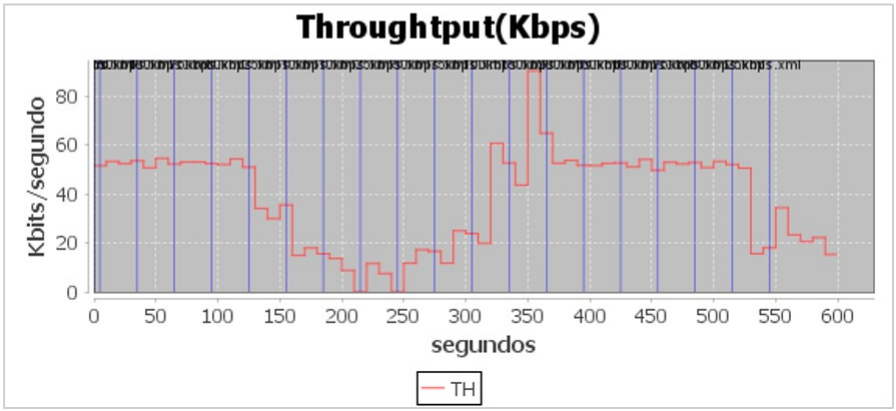


Figura 7.1: Gráfica evolución del *Throughput* durante la simulación.

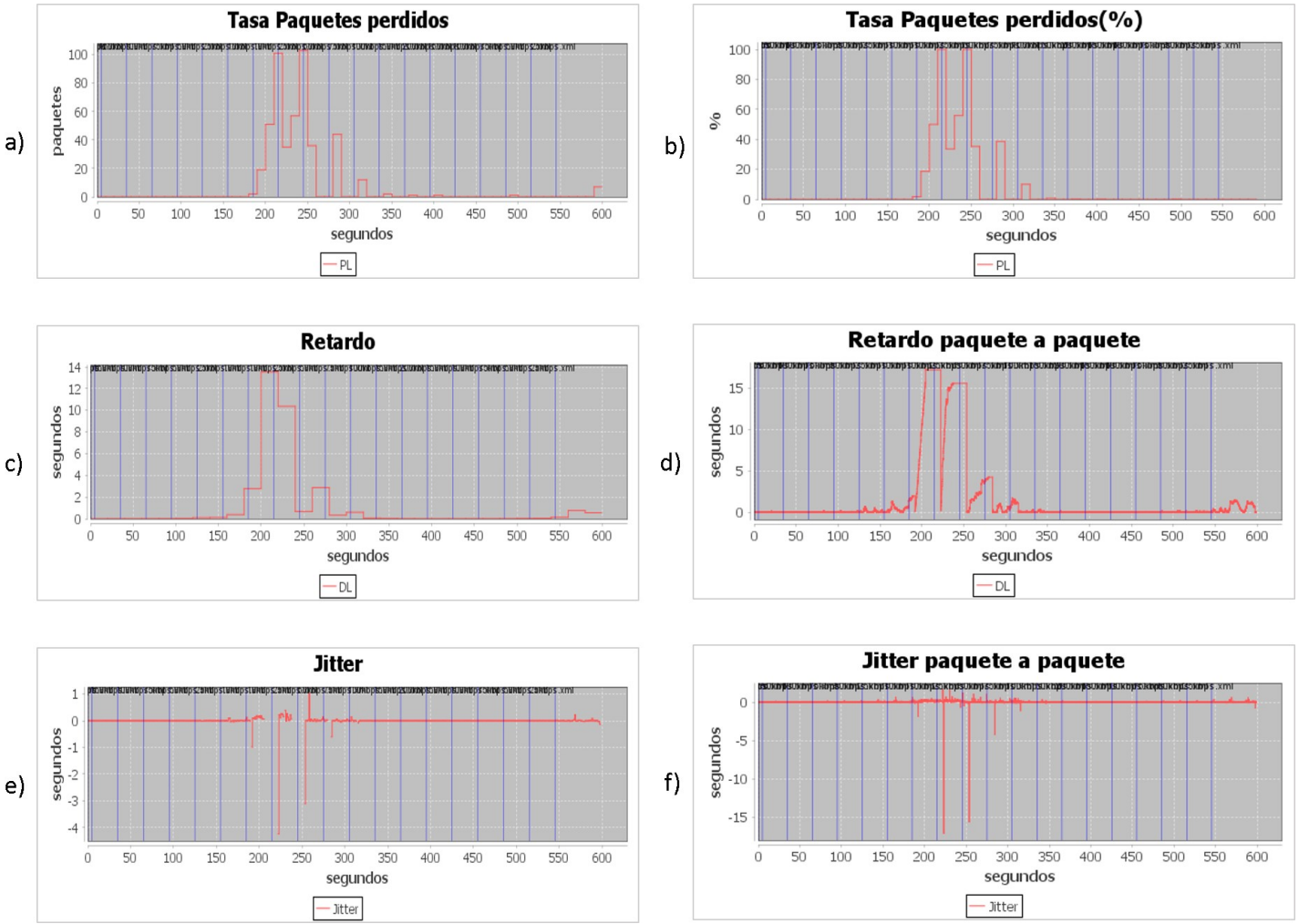


Figura 7.2: Gráficas objetivas para la simulación con *throughput* variable.

En primer lugar se observa la tasa de paquetes perdidos, tanto en número de paquetes perdidos 7.2 a) como en porcentaje 7.2 b). Se observa cómo, cuando la velocidad de *Skype* es igual a la máxima (55 kbps), la pérdida de paquetes es nula. Pero cuando la velocidad disminuye por debajo de este valor, aparecen paquetes perdidos. Este efecto se debe al tiempo que tarda *Skype* en adaptar su *bitrate*. Es decir, en los cambios de perfil de red (en el servidor), el cliente continúa emitiendo a la velocidad anterior hasta que descubre que ahora las condiciones de red son otras. Hasta ese instante de tiempo, el cliente emite más información de la que puede recibir el servidor (debido a las restricciones impuestas por el emulador de red). Por este motivo se produce la perdida de paquetes.

A continuación, se muestran las gráficas relacionadas con el retardo de la simulación. Se observa cómo, al disminuir la velocidad del enlace, aparece retardo en los paquetes. Esto, como en el caso anterior, se debe a que el cliente emite a mayor velocidad que lo que se puede transmitir por la red hacia el servidor (cuando se produce un cambio en las condiciones de la red). Los paquetes que no pueden ser recibidos en el servidor se almacenan en una cola del emulador de red, a la espera de que puedan ser enviados, ocasionando así un retardo mayor.

Finalmente, se muestra las gráficas relacionas con el *jitter*. En estas gráficas también aparecen efectos de *jitter* cuando el cliente modifica su tasa de generación de paquetes, ocasionando que el tiempo entre paquetes no sea siempre constante y aumentando así el *jitter*.

- **Gráficas Subjetivas**: Se muestra en la 7.3 las gráficas con las estadísticas subjetivas de la simulación.

  En primer lugar, se muestra la gráfica con la información de *Perceptual Evaluation of Speech Quality*, PESQ. En esta gráfica, al comparar ambas señales de voz al bajar la velocidad del enlace, empeora la calidad de la llamada.

  Finalmente se muestra la información relacionada con el E-Model.

Como conclusión general, en este caso se ve claramente cómo, al bajar la velocidad del enlace de la red por debajo de 30 kbps, la calidad de la llamada realizada se ve directamente afectada. Este resultado era de esperar, ya que la función E-Model depende de las estadísticas objetivas de retardo y de tasa de paquetes perdidos.

7.2. Variando la tasa de paquetes perdidos

Esta simulación usa un audio con las mismas características que el utilizado en la sección 7.1 pero, en esta ocasión la duración es de 500 segundos.

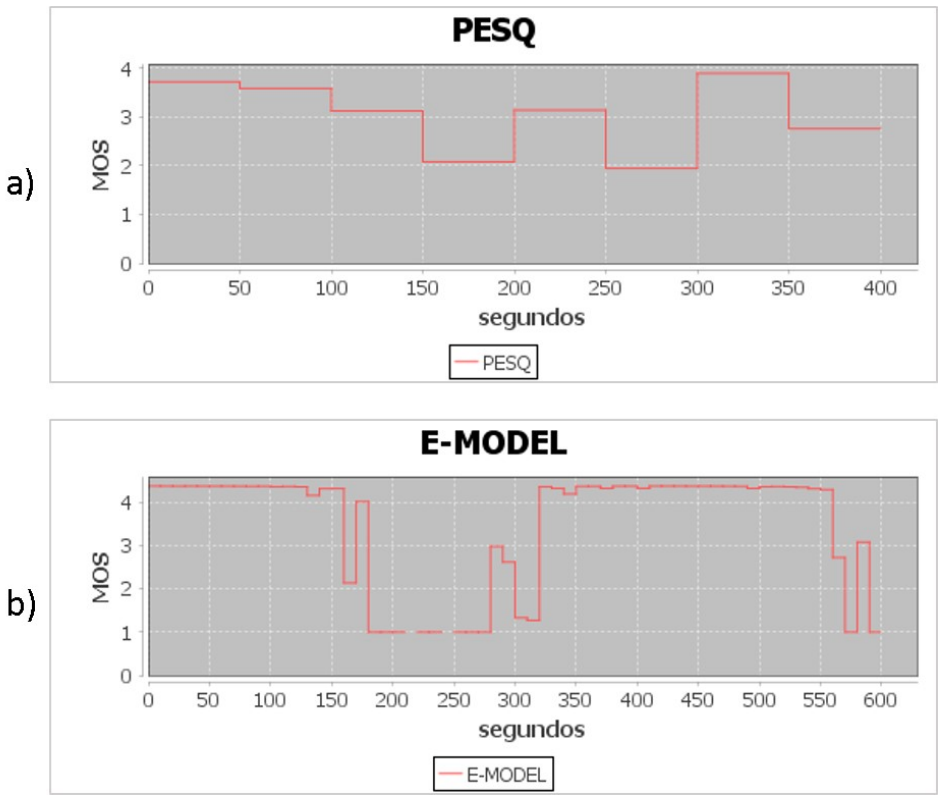


Figura 7.3: Gráficas subjetivas para la simulación con *throughput* variable.

En este caso, la velocidad del enlace definida por el emulador de red es lo suficientemente alta para que no introduzca ninguna perturbación. De este modo, durante la simulación solo se introducen perdidas por parte del emulador.

En el cuadro 7.2 se muestra cómo va ir evolucionando la simulación a lo largo del tiempo. Los perfiles de trafico mostrados en la tabla indican el porcentaje de paquetes perdidos debido al emulador de red.

En primer lugar, se muestra la Figura 7.4 que incluye las gráficas relacionadas con la pérdida de paquetes.

Destacar que la Figura 7.4 a) muestra el porcentaje de paquetes perdidos. Dado que el emulador no introduce otro tipo de perdidas, este porcentaje coincide en cada instante de tiempo con el mostrado en el Cuadro 7.2.

Por otro lado, la gráfica mostrada 7.4 b) muestra los paquetes perdidos. Se puede observar cómo, al aumentar el porcentaje de pérdidas, disminuyen los paquetes perdidos. Esto se debe a que el cliente detecta que la red tiene perdidas por lo que usa un codec con el que se mandan menos paquetes, lo que implica que se pierdan también menos paquetes.

El estudio realizado para esta simulación se divide en dos partes depen-

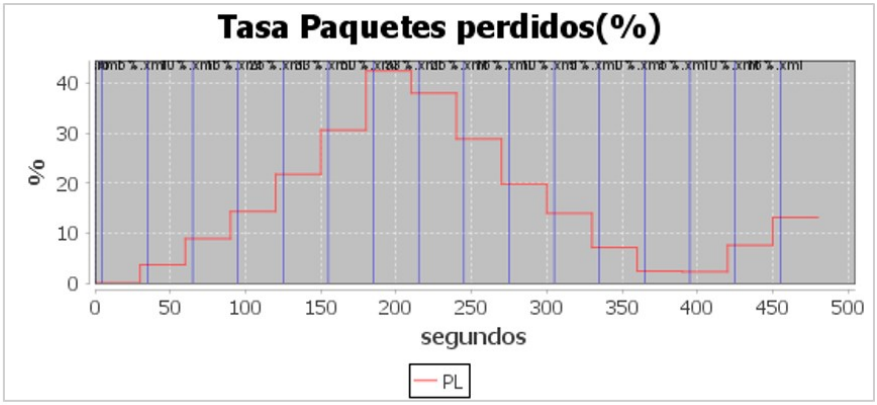
Instante inicial	Perfil
0	sincronización
5	0 %
35	5 %
65	10 %
95	16 %
125	25 %
155	33.3 %
185	50 %
215	33.3 %
245	25 %
275	16 %
305	10 %
335	5 %
365	0 %
395	5 %
425	10 %
455	16 %
500	-

Cuadro 7.2: Simulación tasa de paquetes perdidos variable.

diendo de la objetividad de los resultados obtenidos, como se muestra a continuación:

- Gráficas Objetivas: Ahora se muestran en la Figura 7.5 las gráficas con las estadísticas objetivas de la simulación.

a)



b)

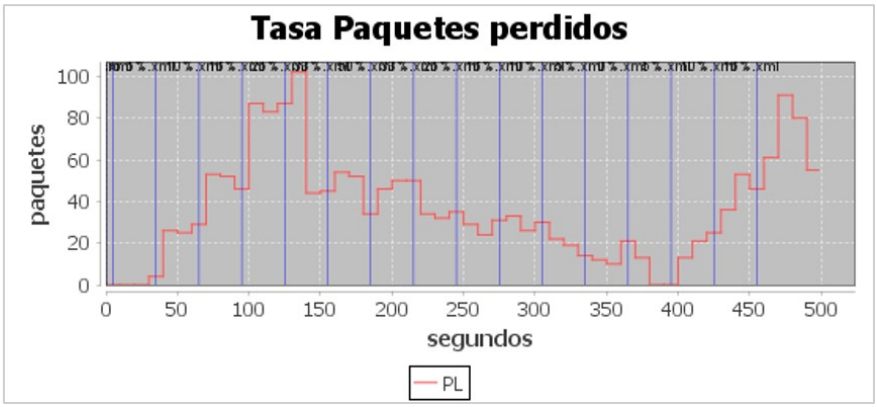


Figura 7.4: Gráfica evolución de la tasa de paquetes perdidos durante la simulación.

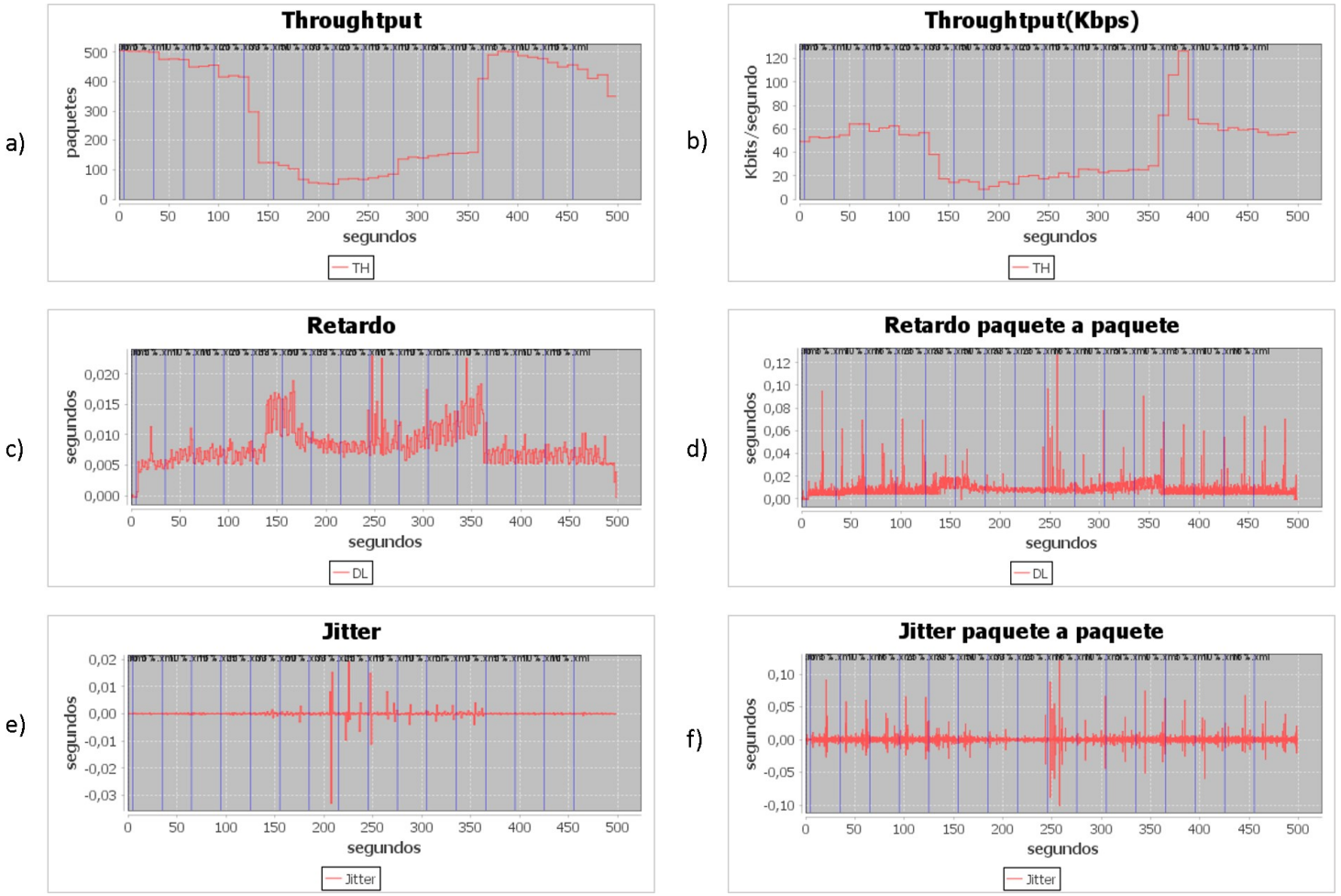


Figura 7.5: Gráficas objetivas para la simulación con tasa de paquetes perdidos variable.

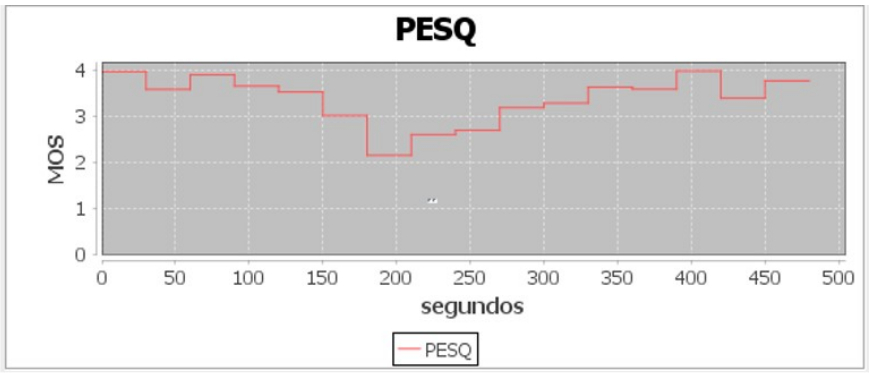
En primer lugar, observando 7.5 a) y b), al detectar *Skype* que en el enlace hay perdidas comienza a enviar a una velocidad menor, seguramente con el objetivo de que se congestione menos la red y se pierdan el menor número posible de paquetes.

Con respecto al retardo y al *jitter*, se observan pequeñas fluctuaciones poco significativas. Estas fluctuaciones es probable que aparezcan debido al cambio de bitrate de la llamada.

- **Gráficas Subjetivas**:

  A continuación, se muestra en la Figura 7.3 las gráficas con las estadísticas subjetivas de la simulación.

a)



b)

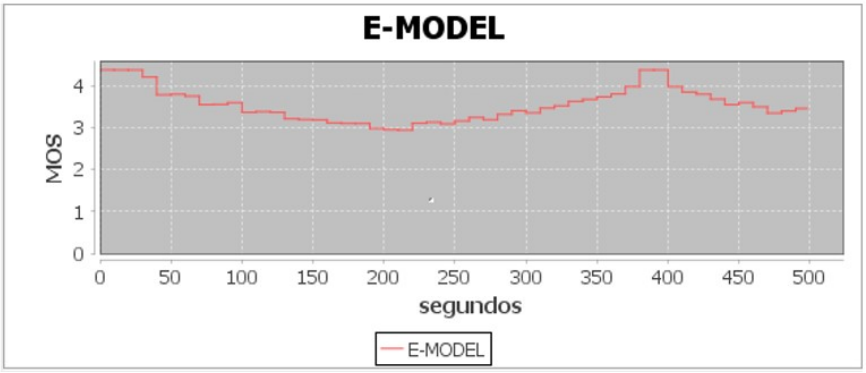


Figura 7.6: Gráficas subjetivas para la simulación con tasa de paquetes perdidos variable.

En este caso se puede observar cómo ambas gráficas mantienen una forma simular. Estos gráficos indican que, al comenzar la simulación, la comunicación era muy buena pero, conforme han ido aumentando el porcentaje de paquetes perdidos, la calidad de la llamada se ha deteriorado. Se comprueba así que las pérdidas afectan de forma similar tanto al PESQ como al E-Model.

7.3. Variando el retardo

Esta simulación usa un audio con las mismas características que el utilizado en la sección 7.1 y 7.2, con una duración de 500 segundos. En este caso, la velocidad del enlace definida por el emulador de red es lo suficientemente alta para que no introduzca ninguna perturbación. De este modo, durante la simulación solo se producen perturbaciones y retrasos debido al retardo introducido por el emulador.

En el cuadro 7.3 se muestra cómo va ir evolucionando la simulación a lo largo del tiempo. Los perfiles de trafico mostrados en la tabla indican los milisegundos que va a tardar un paquete en llegar de un extremo a otro.

Instante inicial	Perfil
0	sincronización
5	0 ms
35	50 ms
65	100 ms
95	200 ms
125	300 ms
155	500 ms
185	750 ms
215	1000 ms
245	750 ms
275	500 ms
305	300 ms
335	200 ms
365	100 ms
395	50 ms
425	0 ms
500	-

Cuadro 7.3: Simulación retardo variable.

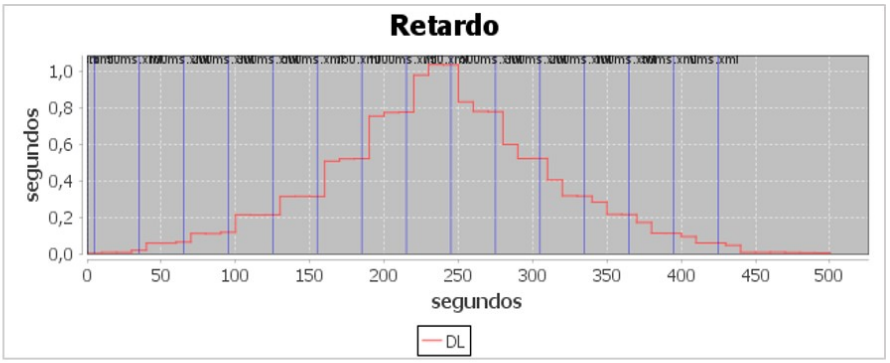
La Figura 7.7 muestra la evolución del retardo durante la simulación.

El estudio realizado para esta simulación se divide en dos partes dependiendo de la objetividad de los resultados obtenidos, como se muestra a continuación:

- Gráficas Objetivas:

  Ahora se muestran en la 7.8 las gráficas con las estadísticas objetivas de la simulación.

a)



b)

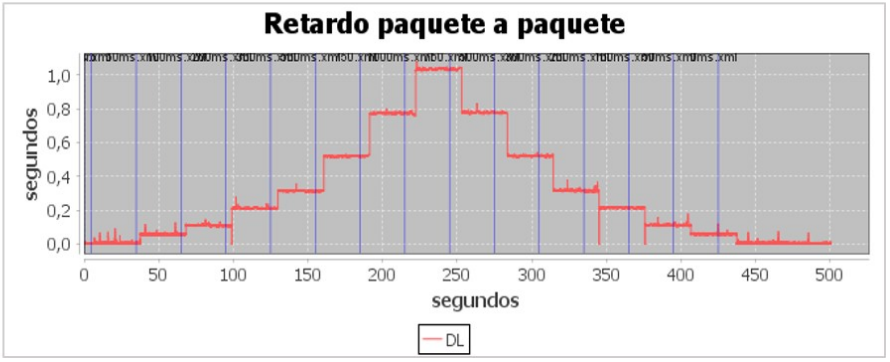


Figura 7.7: Gráfica evolución del retardo durante la simulación

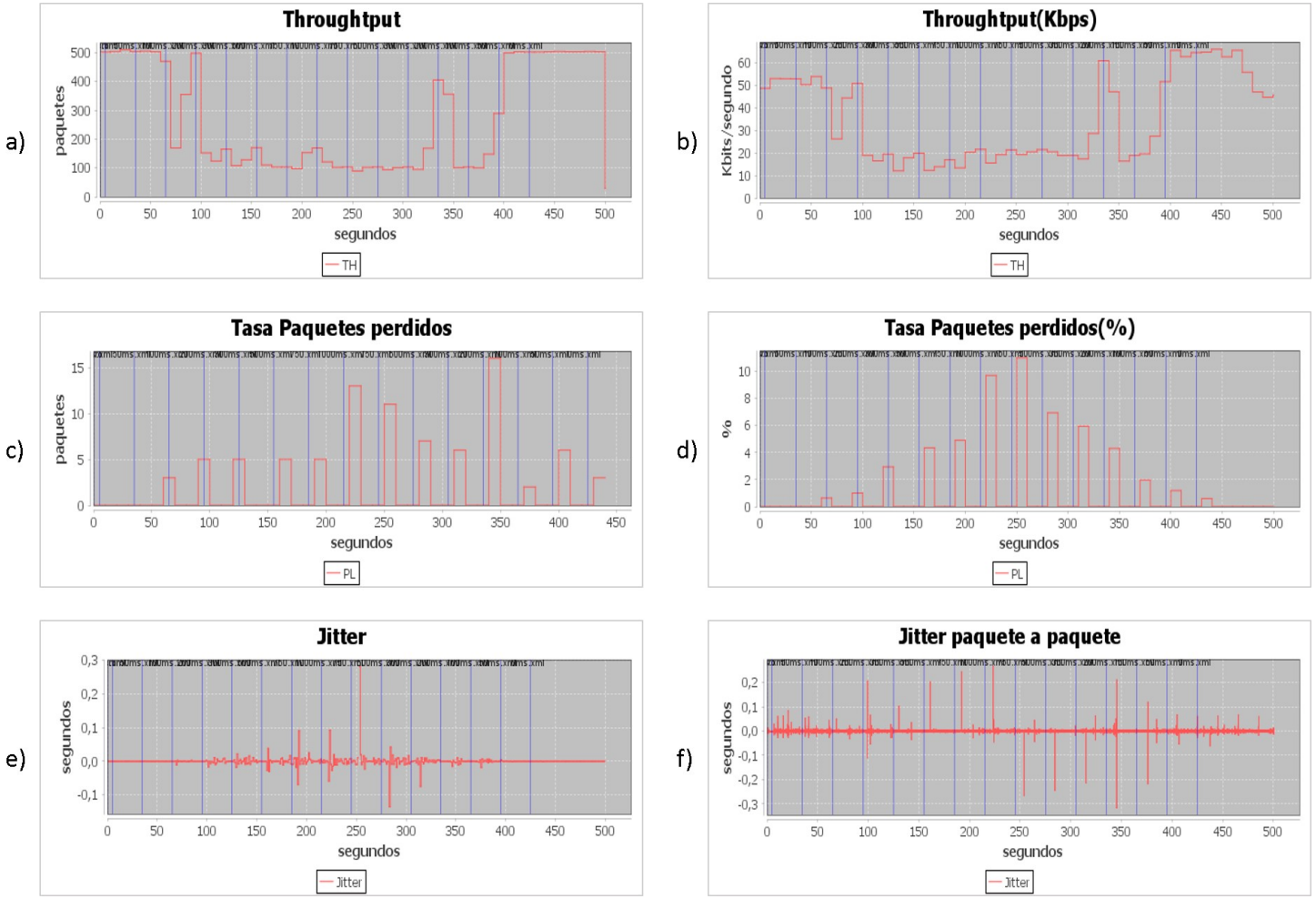


Figura 7.8: Gráficas objetivas para la simulación con retardo variable.

Observando la Figura 7.8 a) y b), como ha ocurrido en las otras simulaciones, al detectar el cliente que algo va mal, la cantidad de información enviada del cliente al servidor disminuye drasticamente.

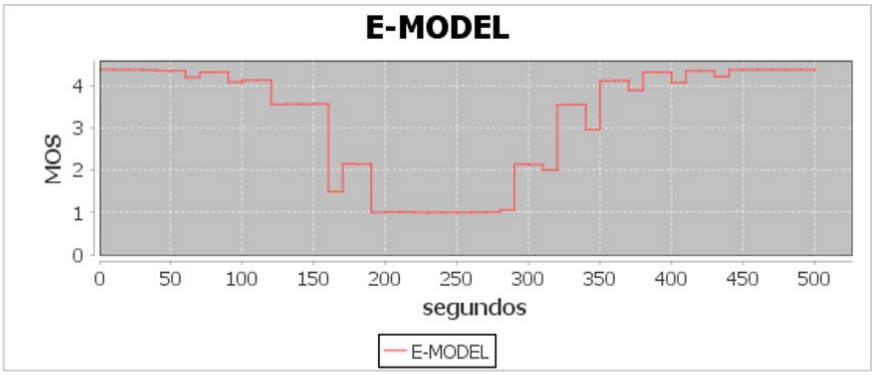
Por otro lado observando la Figura 7.8 c) y d), se detecta que se pierden algunos paquetes. A mayor retardo, mayor número de paquetes perdidos, si bien las pérdidas no son demasiado elevadas.

Finalmente, se observa que el *jitter* permanece principalmente constante, a excepción de algunas pequeñas fluctuaciones.

- **Gráficas Subjetivas**:

  A continuación se muestra en la Figura 7.9 las gráficas con las estadísticas subjetivas de la simulación.

a)



b)

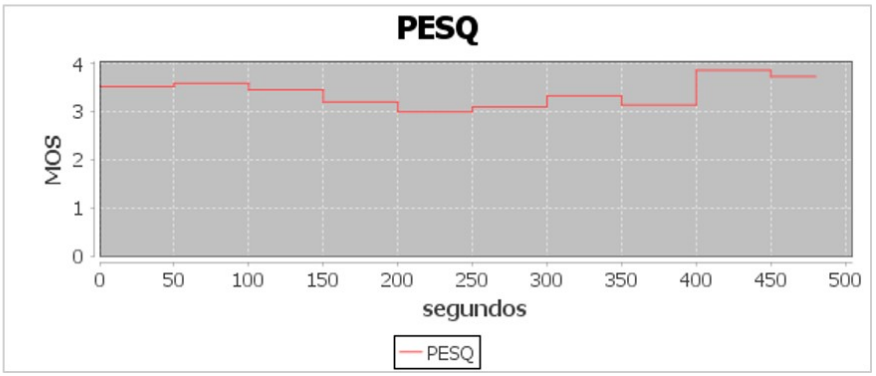


Figura 7.9: Gráficas subjetivas para la simulación con retardo variable.

En este caso entre las dos gráficas de E-Model y PESQ son muy diferentes. Esto se debe a que E-Model se calcula usando las estadísticas objetivas, por lo que al aumentar el retardo disminuye el MOS según el E-model.

Sin embargo, para calcular el MOS según PESQ se utilizan los audios capturados en ambos extremos. PESQ realiza operaciones de sincro-

nización entre ambos audios para poder comparar ambas señales, la original y la recibida tras atravesar la red. Esta sincronización hace que el retardo de los paquetes no afecte a los valores calculados por PESQ, que permanece prácticamente constante. Las pequeñas variaciones seguramente se deben a los paquetes perdidos al introducir retardos elevados.

Capítulo 8

Conclusiones y líneas futuras

En este capítulo se especifican las contribuciones generales que aporta este proyecto. Para este propósito se va a realizar un análisis objetivo sobre el resultado conseguido e indicando posibles líneas de trabajo futuras que complementen o mejoren el proyecto realizado. Finalmente se va realizar una valoración personal del trabajo realizado.

8.1. Conclusiones

Como conclusión principal, la solución realizada consigue cumplir con todos los requisitos iniciales para este trabajo. Ya que se ha conseguido realizar una herramienta capaz de analizar el tráfico generado por *Skype*. Tal como se comprueba en el Capítulo 2 del estado del arte, no se ha encontrado ninguna aplicación que tenga esta funcionalidad.

Las principal contribución que aporta este proyecto es la disposición de una herramienta altamente configurable con la que poder evaluar un gran número de escenarios y observar cómo se comporta *Skype* en estas situaciones.

8.2. Líneas futuras

Pese a que cumple con todas las especificaciones iniciales, como trabajos futuros se presentan una serie de posibles mejoras:

- Mejorar la interfaz con el usuario, haciéndola más atractiva y mas funcional para el usuario.
- Instalar las aplicaciones en dos ordenadores diferentes, ya que para este trabajo se han usado dos maquinas virtuales. Sería interesante hacerlas funcionar en dos ordenadores dentro o fuera de una misma red realizando NAT (suponiendo que usen direcciones IP privadas) y realizar test en redes reales.

- Adaptar la herramienta desarrollada para que sea capaz de funcionar en otros sistemas operativos.
- Realizar más tests, ya que sería interesante mezclar distintos tipos de pérdidas y retardos para comprender pormenorizadamente el funcionamiento de *Skype*.
- Añadir mas gráficas subjetivas. En este proyecto se han incluido dos medidas subjetivas, pero sería interesante implementar alguna más.
- Para que el estudio subjetivo sea más completo, seria interesante realizar tests con usuarios reales y recoger la experiencia del usuario.

8.3. Valoración personal

En primer lugar, cabe destacar que la realización de este proyecto ha sido un gran reto debido a los numerosos obstáculos encontrados. Sin embargo, con perseverancia y aplicando todo lo aprendido durante mis años de formación, se han conseguido alcanzar todos los objetivos propuestos.

Por otro lado, reseñar que durante la realización de este proyecto se han adquirido una gran cantidad de conocimientos. Ya sea en las primeras fases, leyendo gran cantidad de documentación o durante la fase de la implentación. Estos conocimientos serán muy útiles en futuros proyectos y en mi vida laboral.

En resumen, realizar un proyecto de este tipo ha sido muy productivo y ha hecho que el autor se sienta realizado y con la satisfacción de conseguir llevar a cabo un proyecto de este tipo en su totalidad.

Apéndice A

Manual de uso

En este apartado se explica como utilizar la aplicación desarrollada en el entorno diseñado. Por lo tanto las pruebas se van a realizar en un único ordenador.

En primer lugar es necesario iniciar las maquinas virtuales, que actúan como cliente y servidor, para este fin se usa *Virtualbox*. Una vez inicializados los equipos, en el escritorio de cada uno de ellos se encuentra un ejecutable denominado *TestSkype_Cliente.exe* y *TestSkype_Servidor.exe* respectivamente. Es necesario hacer doble click en estos ejecutable.

Al ejecutar *TestSkyp_Servidor.exe* aparece la ventana mostrada en la Figura A.1.

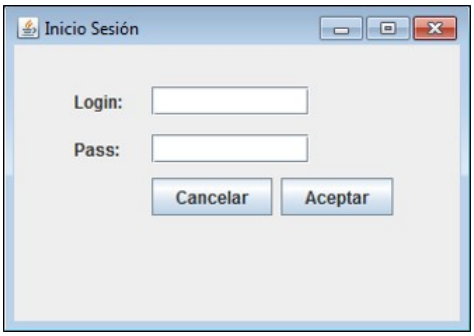


Figura A.1: Ventana mostrada en el servidor al iniciar el programa.

Es necesario introducir en el campo *Login*, un usuario dado de alta previamente en *Skype*. También se debe rellenar el campo *Pass* con la contraseña asociada al usuario introducido. Al rellenar esta información se pulsa el botón *Aceptar*, quedando el servidor a la espera de que un cliente inicie una simulación. Por parte del servidor no se requiere más atención.

Por otro lado al ejecutar el programa del cliente se muestra una ventana como la de la Figura A.2 .

En esta ventana se pueden realizar dos operaciones:

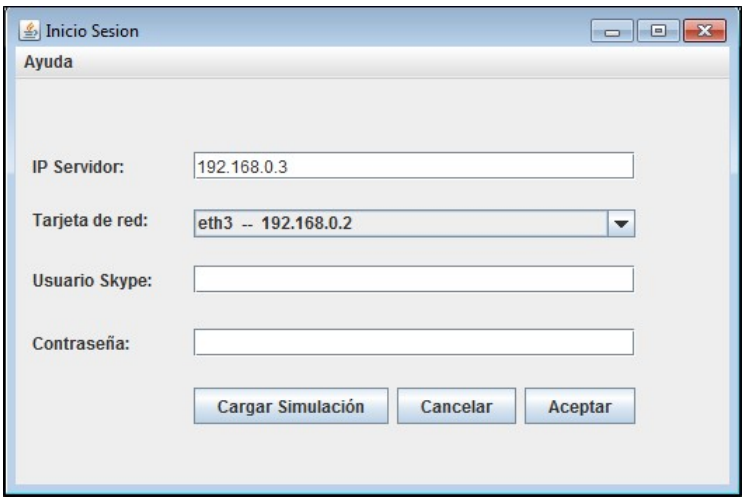


Figura A.2: Ventana mostrada en el cliente al iniciar el programa.

- **Testear un nuevo escenario**: para realizar esta operación es necesario rellenar todos los campos de esta ventana. Los dos primeros denominados *IP Servidor* y *Tarjeta de red* (dirección IP del cliente) vienen determinados por el diseño de ambas maquinas virtuales. Por tanto sus valores son los mostrados en la Figura A.2. También como en el servidor es necesario introducir un usuario y una contraseña válida previamente registrados en *Skype*. Una vez introducida esta información se pulsa el botón Aceptar, al realizar esta acción se muestra la ventana mostrada en la Figura A.3

- **Cargar los resultados de un escenario anterior**: Al final de testear un escenario los resultados se guardan automáticamente. Estos resultados se pueden volver a ver. Para ello únicamente es necesario pulsar el botón *Cargar Simulación* y seleccionar una de las carpetas mostradas en un explorador de archivos. Destacar que las carpetas que contienen los resultados están nombradas con la fecha en la que se realizó el test.

En esta Figura A.3, se muestra un listado con la información introducida que compone el escenario que se desea evaluar.

Es necesario introducir información por lo tanto es necesario pulsar el botón *Añadir*, al realizar esta operación se muestra la ventana mostrada en la Figura A.4.

En esta ventana es necesario introducir un audio disponible en tu equipo. Así como un perfil de trafico y el instante de tiempo en el cual comienza a ejecutarse. Esta ultima operación se puede repetir tantas veces como el usuario desee. En el caso de no existir el perfil deseado en tu equipo se puede crear pulsando el botón *Nuevo Perfil*, Figura A.5.

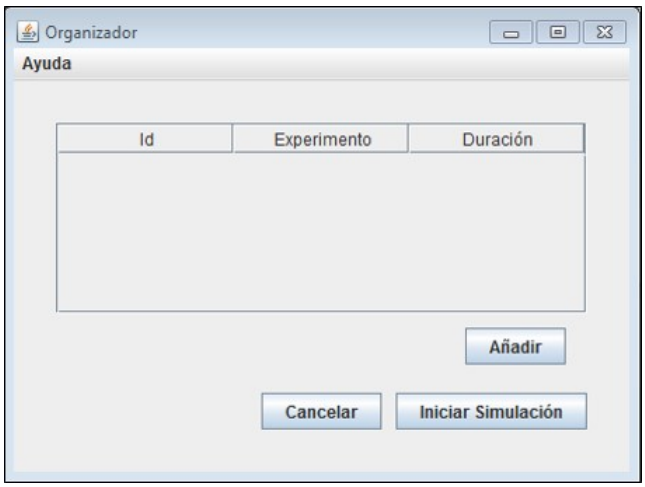


Figura A.3: Listado con la información que compone un escenario

En el caso de no existir el perfil deseado en tu equipo se puede crear pulsando el botón *Nuevo Perfil*, Figura A.5. En esta ventana rellenando la información se genera un archivo con la información introducida.

Volviendo a la Figura A.4, una vez introducida la información se pulsa el botón *Finalizar*. Al pulsar este botón volvemos a la ventana A.3. Llegados a este punto tenemos dos opciones añadir mas información al escenario (pulsar *Añadir*) o realizar la simulación (pulsar *Iniciar Simulación*).

Al iniciar la simulación se muestra la ventana, Figura A.6. En esta ventana se va mostrando el progreso de la simulación.

Al finalizar es posible pulsar pulsar los botones:

- *Gráficas Objetivas*: Al pulsar este botón se muestra la ventana mostrada en la Figura A.7. En esta ventana se seleccionan las gráficas que se desean visualizar. Al pulsar *Aceptar* en esta ventana se muestran finalmente la representación de las estadísticas objetivas.

- *Gráficas Subjetivas*: Al pulsar este botón se muestran gráficamente las estadísticas subjetivas del escenario introducido.

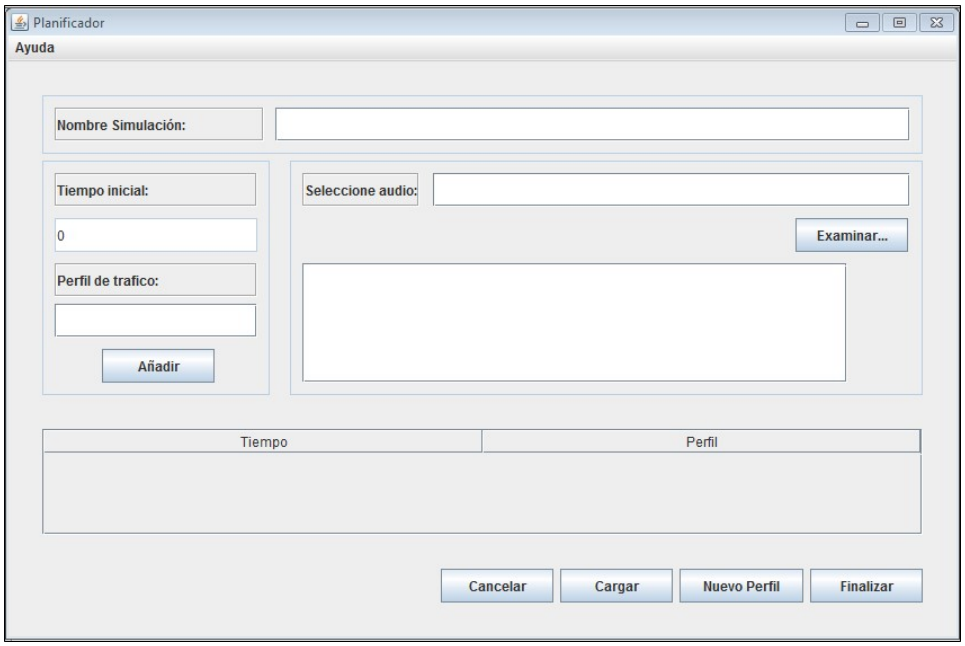


Figura A.4: Introducción de información

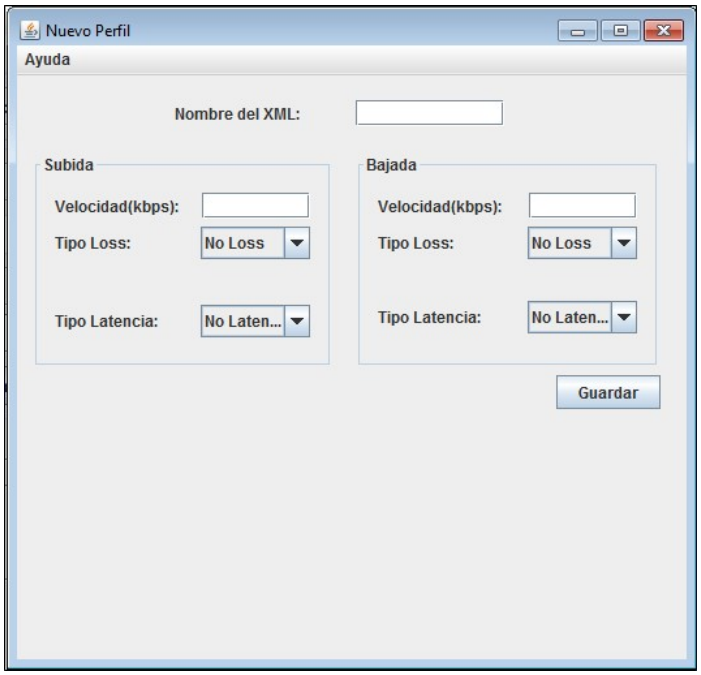


Figura A.5: Nuevo perfil de tráfico.

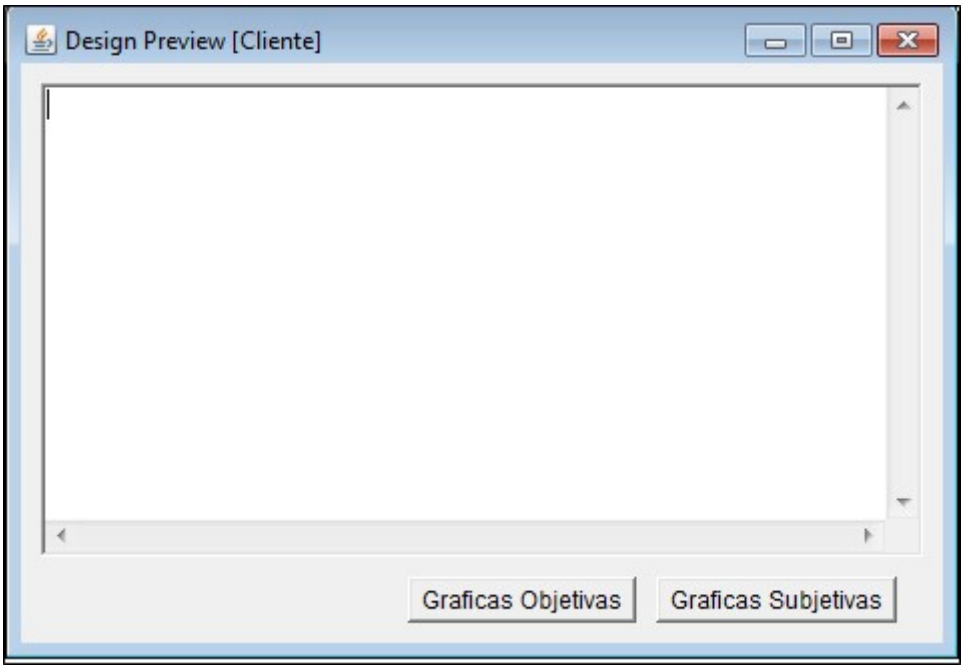


Figura A.6: Estado de la simulación

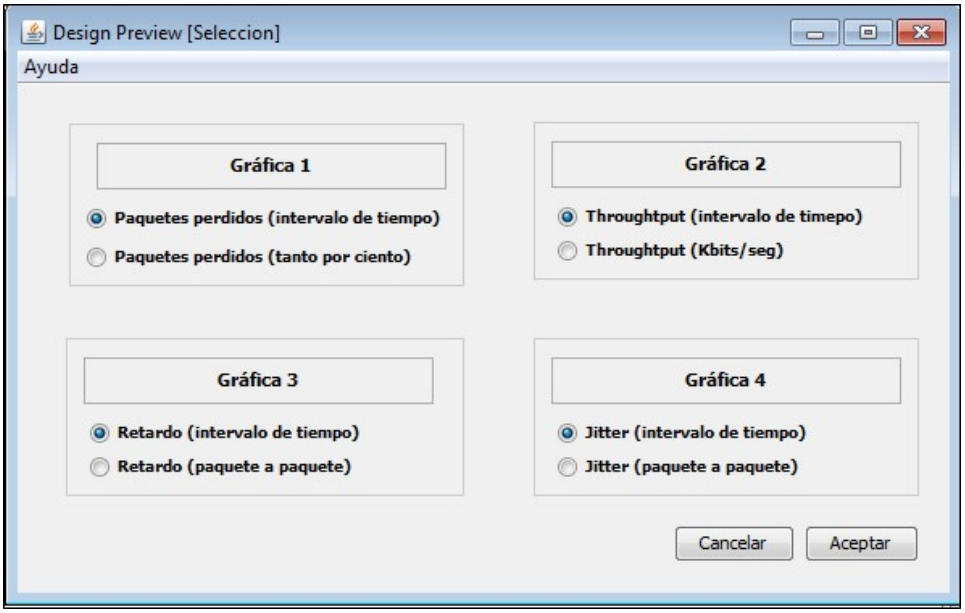


Figura A.7: Elección de las gráficas objetivas que se desean visualizar.

Bibliografía

[1] B. Goode, "Voice over internet protocol (VoIP)," *Proceedings of the IEEE*, vol. 90, no. 9, pp. 1495–1517, Sep 2002.

[2] Microsoft, "Web skype (accedido el 2017-09-12)." [Online]. Available: https://www.skype.com/es/

[3] C. C. Satya Nadella, "Envision 2016 (accedido el 12/09/2017)." [Online]. Available: https://news.microsoft.com/speeches/satya-nadella-and-chris-capossela-envision-2016/

[4] D. York, "Understanding Today's Skype Outage: Explaining Supernodes (accedido 12/09/2017)." [Online]. Available: http://www.disruptivetelephony.com/2010/12/understanding-todays-skype-outage-explaining-supernodes.html

[5] S. Jensen, K. Vos, and K. Soerensen, "SILK Speech Codec," Tech. Rep. [Online]. Available: https://tools.ietf.org/html/draft-vos-silk-01

[6] Skype, "Silkdatasheet," Nov. 2011. [Online]. Available: https://web.archive.org/web/20111123141335/http://developer.skype.com/resources/SILKDataSheet.pdf

[7] M. W. <mark@digitalfountain.com>, "Forward Error Correction (FEC) Framework." [Online]. Available: https://tools.ietf.org/html/rfc6363

[8] J. Postel, "User Datagram Protocol." [Online]. Available: https://tools.ietf.org/html/rfc768

[9] ITU, "G.114:Tiempo de transmisión en un sentido," May 2003. [Online]. Available: https://www.itu.int/rec/T-REC-G.114-200305-I/es

[10] "P.800.1:Mean opinion score (MOS) terminology," Tech. Rep. [Online]. Available: https://www.itu.int/rec/T-REC-P.800.1-201607-I/en

[11] ITU, "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." [Online]. Available: https://www.itu.int/rec/T-REC-P.862-200102-I/es

[12] ITU, "G.107:The E-model: a computational model for use in transmission planning," Tech. Rep. [Online]. Available: http://www.itu.int/rec/T-REC-G.107-201506-I/en

[13] A. Sanchez, "VoIPTester," Tech. Rep., 2010. [Online]. Available: http://dtstc.ugr.es/it/pfc/proyectos_realizados/downloads/Memoria2010_AntonioSanchez.pdf

[14] "PRTG Manual: Monitoring Quality of Service and VoIP." [Online]. Available: https://www.paessler.com/manuals/prtg/quality_of_service_monitoring

[15] "Monitoreo de VoIP y herramientas de calidad de servicio | SolarWinds." [Online]. Available: http://www.solarwinds.com/es/voip-network-quality-manager

[16] Cisco, "Cisco IOS IP Service Level Agreements (SLAs) - Cisco." [Online]. Available: http://www.cisco.com/c/en/us/products/ios-nx-os-software/ios-ip-service-level-agreements-slas/index.html

[17] P. Wuttidittachotti and T. Daengsi, "Subjective mos model and simplified e-model enhancement for skype associated with packet loss effects: a case using conversation-like tests with thai users," *Multimedia Tools and Applications*, pp. 1–25, 2016. [Online]. Available: http://dx.doi.org/10.1007/s11042-016-3901-5

[18] O. C. Ramiro, G. B. Rubén, and G. B. Rubén, "Estudio sobre calidad de servicio (QoS) para comunicaciones multimedia usando Skype," vol. REVISTA DE INGENIERÍA ELÉCTRICA, ELECTRÓNICA Y COMPUTACIÓN, VOL 1 NO., p. 6, Aug. 2007.

[19] "Wireshark · Go Deep." [Online]. Available: https://www.wireshark.org/

[20] "Oracle VM VirtualBox." [Online]. Available: https://www.virtualbox.org/

[21] "NetBeans IDE - Overview." [Online]. Available: https://netbeans.org/features/index.html

[22] "Maven Repository: org.jfree » jfreechart » 1.0.19." [Online]. Available: https://mvnrepository.com/artifact/org.jfree/jfreechart/1.0.19

[23] "Welcome to Python.org." [Online]. Available: https://www.python.org/download/releases/2.7/

[24] "Pamela for Skype." [Online]. Available: http://www.pamela.biz/

[25] "Clisk 2.5.1 User's Guide." [Online]. Available: http://www.dlee.org/skype/clisk/man.php

[26] itu, "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," Tech. Rep. [Online]. Available: https://www.itu.int/rec/T-REC-P.862/es

[27] "Tshark - The Wireshark Network Analyzer 2.4.1 (Accedido 12/09/2017)." [Online]. Available: https://www.wireshark.org/docs/man-pages/tshark.html

[28] "Official download of VLC media player, the best Open Source player - VideoLAN (Accedido 12/09/2017)."

[29] "VB-Audio VoiceMeeter (Accedido 12/09/2017)." [Online]. Available: http://www.vb-audio.com/Voicemeeter/index.htm

[30] "SoX - Sound eXchange | HomePage (Accedido 12/09/207)." [Online]. Available: http://sox.sourceforge.net/

[31] JAVA, "Runtime (Java Platform SE 7)." [Online]. Available: https://docs.oracle.com/javase/7/docs/api/java/lang/Runtime.html#exec(java.lang.String)

[32] Skype, "¿Puedo ejecutar Skype para el escritorio de Windows desde la línea de comandos?" [Online]. Available: https://support.skype.com/es/faq/FA171/puedo-ejecutar-skype-para-el-escritorio-de-windows-desde-la-linea-de-\comandos

[33] JAVA, "SetVisible (Java Platform SE 7)." [Online]. Available: https://docs.oracle.com/javase/7/docs/api/java/awt/Component.html

[34] "DatagramSocket (Java Platform SE 7)." [Online]. Available: https://docs.oracle.com/javase/7/docs/api/java/net/DatagramSocket.html

[35] "System (Java Platform SE 7)." [Online]. Available: https://docs.oracle.com/javase/7/docs/api/java/lang/System.html

[36] ITU, "P.501:Señales de prueba para utilización en telefonometría." [Online]. Available: https://www.itu.int/rec/T-REC-P.501/es