ugr | Universidad de Granada

ESTUDIOS DE INGENIERÍA DE TELECOMUNICACIÓN

PROYECTO FIN DE CARRERA

Video Tester

CURSO: 2010/2011

Iñaki Úcar Marqués

El tribunal constituido para la evaluación del proyecto PFC titulado:

Video Tester

Realizado por el alumno: **Iñaki Úcar Marqués**
Y dirigido por los tutores: **Juan Manuel López Soler y Jorge Navarro Ortiz**

Ha resuelto asignarle la calificación de:

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

Con la nota: puntos.

El Presidente:

El Secretario:

El Vocal:

Granada, a de de 2011

ugr | Universidad de Granada

ESTUDIOS DE INGENIERÍA DE TELECOMUNICACIÓN

Video Tester

REALIZADO POR:
Iñaki Úcar Marqués

DIRIGIDO POR:
Juan Manuel López Soler y Jorge Navarro Ortiz

DEPARTAMENTO:
Teoría de la Señal, Telemática y Comunicaciones

Granada, junio de 2011

Video Tester

Iñaki Úcar Marqués

PALABRAS CLAVE:
calidad de servicio (QoS), calidad de la experiencia (QoE), transmisión de vídeo, redes IP, medidas de calidad, métricas de calidad, evaluación de la calidad de vídeo

RESUMEN:
Ante el crecimiento incesante durante los últimos años del tráfico de vídeo en Internet, se hace patente la necesidad de herramientas que automaticen la tarea de la evaluación de la calidad de vídeo en entornos de transmisión IP, tanto para el ámbito de la investigación como para la puesta en marcha y monitorización de servicios finales. Desafortunadamente, en la actualidad no existen apenas aplicaciones de este tipo. De entre las disponibles, destaca EvalVid por su versatilidad.

Este proyecto está dedicado al diseño e implementación de una aplicación extensible y reutilizable que suponga una solución completa al problema de la evaluación de la calidad de vídeo sobre redes IP: envío, recepción, codificación, descodificación, extracción de parámetros y características, y cálculo de medidas de calidad. Dicha extracción de parámetros se realiza en los tres niveles implicados en el procesado y transmisión de vídeo —nivel de paquete, nivel de flujo codificado y nivel de imagen— con el objetivo de reunir la mayor cantidad de información posible y, por tanto, ofrecer la posibilidad de implementar cualquier tipo de medida de calidad. Algunos de los parámetros y métricas implementados son los siguientes: ancho de banda, latencia, *jitter*, tasa de pérdidas, tamaño del GOP, PSNR, SSIM, MOS, etc.

Se ha escogido el lenguaje Python para su programación con el objetivo de favorecer la extensibilidad, y la asistencia del marco de trabajo *GStreamer* para el procesamiento de vídeo por su amplio soporte en este ámbito.

La aplicación resultante cubre las funcionalidades presentes en EvalVid y añade mejoras en términos de usabilidad, extensibilidad, soporte para codificadores, soporte para métodos de transmisión y fiabilidad en condiciones de pérdidas.

Video Tester

Iñaki Úcar Marqués

KEYWORDS:
quality of service (QoS), quality of experience (QoE), video transmission, IP networks, quality measures, quality metrics, video quality assessment

ABSTRACT:
The continual growth of Internet video traffic during the last years has made evident the need for tools that automate the task of video quality assessment in IP environments for both research and implementation and monitoring of end services. Unfortunately, there are yet hardly any applications of this type. Among those available, Evalvid stands out because of its versatility.

This M.Sc. Thesis is devoted to the design and implementation of an extensible and reusable application which provides a complete solution to the problem of video quality assessment over IP networks: sending, receiving, encoding, decoding, extracting parameters and characteristics, and calculating quality metrics. Parameter extraction is performed on the three levels involved in the video processing and transmission — packet level, bitstream level and picture level— in order to gather as much information as possible and, therefore, offer the possibility to implement any quality metric. Some of the implemented parameters and metrics are as follows: bandwidth, latency, jitter, packet loss rate, GOP size, PSNR, SSIM, MOS, etc.

Python has been chosen for programming with the aim of promoting extensibility, and the election of the GStreamer framework for video processing is due to its broad support in this area.

The resulting application covers EvalVid features and adds further improvements in terms of usability, extensibility, codec support, support of transmission methods and reliability in case of losses.

D. Juan Manuel López Soler y D. Jorge Navarro Ortiz, profesores del Departamento de Teoría de la Señal, Telemática y Comunicaciones de la Universidad de Granada, como directores del Proyecto Fin de Carrera de D. Iñaki Úcar Marqués

Informan:

que el presente trabajo, titulado:

Video Tester

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

Granada, a de de 2011

Fdo. Juan Manuel López Soler

Fdo. Jorge Navarro Ortiz

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

Granada, a de de 2011

Fdo. Juan Manuel López Soler

Fdo. Jorge Navarro Ortiz

Fdo. Iñaki Úcar Marqués

Agradecidimientos

Con esta gran coda, llega el momento que todo estudiante espera: el final de una etapa, la confirmación de unas espectativas, la realización de un sueño. Atrás quedan buenos años en los que no solo me he formado como ingeniero, sino también como adulto.

Alguien dijo una vez que las ingenierías no son carreras de conocimientos, sino de obstáculos. Y no le faltaba razón: ha sido un camino largo, lleno de pruebas y duro por momentos. Pero también ha sido un camino apasionante gracias a que me han acompañado, en mayor o menor medida, muchas personas de las que me gustaría acordarme:

A Almudena, mi novia, la armadura de esta sinfonía.

A mis padres, por su apoyo incondicional y su plena confianza en mí.

A mis compañeros, por los buenos momentos —que han sido muchos— y por hacer más llevaderas las interminables tardes de prácticas.

A todos aquellos profesores que han sabido inculcarme la pasión por esta profesión; y a los que no —¡qué diablos!—, también, porque me han enseñado a luchar contra las adversidades.

A Juanma y Jorge, mis tutores, por darme la oportunidad de acabar la carrera con este Do de pecho.

Gracias a todos.

Índice general

Índice de figuras

Índice de tablas

Índice de listados

Glosario

Símbolos

3G Tercera generación de tecnologías para comunicaciones móviles.

A

ANSI *American National Standards Institute*, organismo de estandarización de los Estados Unidos de América.

B

BGP *Border Gateway Protocol*, protocolo de encaminamiento entre sistemas autónomos.

F

FR *Full Reference* o referencia completa.

G

GOP *Group Of Pictures*, grupo de imágenes codificadas consistente en una imagen de referencia seguida de un conjunto de imágenes predictivas.

GPRS *General Packet Radio Service*, servicio de datos para comunicaciones móviles orientado a paquete.

GSM *Global System for Mobile Communications*, sistema global estándar para las comunicaciones móviles de segunda generación.

H

H.263 Estándar de la ITU para la codificación de vídeo con compresión.

H.264 Estándar de la ITU en colaboración con el Moving Picture Experts Group para la codificación de vídeo con alta compresión. Se corresponde con la parte 10 de MPEG-4.

HTML *HyperText Markup Language*, lenguaje de marcado para describir la estructura y el contenido de las páginas web.

HTTP *Hypertext Transfer Protocol*, protocolo de nivel de aplicación de la pila de protocolos TCP/IP.

I

I420 Formato YUV planar con submuestreo 4:2:0 y 12 bits por píxel.

IP *Internet Protocol*, protocolo de nivel de red de la pila de protocolos TCP/IP. También se llama así a la dirección de nivel de red que identifica a una máquina.

IPTV *Internet Protocol Television*, conjunto de tecnologías que permiten la distribución de servicios de televisión sobre redes IP.

ITU *International Telecommunications Union*, organismo especializado encargado de regular las telecomunicaciones a nivel internacional.

M

MOS *Mean Opinion Score*, medida numérica de la calidad percibida por un usuario de un servicio multimedia.

MP4 Parte 14 de MPEG-4 que define un formato contenedor multimedia.

MPEG-4 Grupo de estándares de codificación de audio y vídeo creado por el *Moving Picture Experts Group*. Habitualmente se denomina así a la parte 2 de este grupo de estándares, la cual define un codificador de vídeo con compresión.

MSE *Mean Squared Error* o Error Cuadrático Medio

N

NAT *Network Address Translation*, método de traducción de direcciones IP entre redes.

NR *No Reference* o sin referencia.

P

P2P *Peer to Peer*, arquitectura de red distribuida en la que los nodos que la componen se comunican entre sí de forma igualitaria sin asumir ningún rol concreto (cliente o servidor).

PSNR *Peak Signal to Noise Ratio*, relación entre el pico máximo de una señal y el ruido que afecta a su representación.

Q

QoE *Quality of Experience*, la aceptación general de una aplicación o servicio según la percepción subjetiva del usuario final.

QoS *Quality of Service*, la totalidad de las características de un servicio de telecomunicaciones que determinan su capacidad para satisfacer las necesidades explícitas e implícitas del usuario del servicio.

R

RR *Reduced Reference* o referencia reducida.

RTP *Real-time Transport Protocol*, protocolo de nivel de sesión de la pila de protocolos TCP/IP para la transmisión de información en tiempo real.

RTSP *Real-Time Streaming Protocol*, protocolo de nivel de sesión de la pila de protocolos TCP/IP para el control de flujos multimedia en tiempo real.

S

SDP *Session Description Protocol*, protocolo de descripción de los parámetros de inicialización de flujos multimedia.

SSIM *Structural Similarity*, método FR para medir la semejanza entre dos imágenes.

T

TCP *Transmission Control Protocol*, protocolo de nivel de transporte de la pila de protocolos TCP/IP orientado a conexión.

TV *Television*, medio de telecomunicación consistente en la transmisión y recepción de imágenes en movimiento.

U

UDP *User Datagram Protocol*, protocolo de nivel de transporte de la pila de protocolos TCP/IP orientado a datagrama.

UMTS *Universal Mobile Telecommunications System*, sistema de comunicaciones móviles de tercera generación sucesor de GSM.

URL *Uniform Resource Locator*, secuencia de caracteres usada para la localización e identificación de recursos en Internet.

V

VoD *Video on Demand*, aquellos sistemas multimedia que permiten a sus usuarios disfrutar de un servicio de vídeo bajo demanda.

VoIP *Voice over IP*, conjunto de tecnologías que permiten la distribución de servicios de voz sobre redes IP.

VQEG *Video Quality Experts Group*, grupo independiente de expertos dedicado al desarrollo de medidas de calidad de vídeo.

VQM *Video Quality Metric*, método para medir la semejanza entre dos imágenes.

X

XML *eXtensible Markup Language*, metalenguaje extensible de marcado creado por el World Wide Web Consortium (W3C).

XML-RPC Protocolo de llamada a procedimiento remoto que utiliza XML para la codificación de los datos y HTTP para la transmisión de mensajes.

Y

YUV Espacio de color que tiene en cuenta la percepción humana al descomponer la imagen en componentes de luminancia y cromitancia.

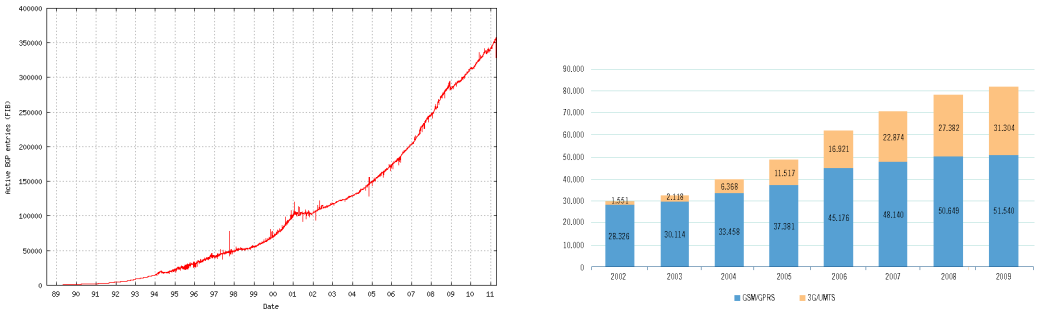
Capítulo 1

Introducción

1.1. Contexto

1.1.1. El auge del vídeo en Internet

En los últimos tiempos, las redes de datos, tanto fijas como móviles, han sufrido un crecimiento verdaderamente espectacular (véase la Figura 1.1 [1, 2]). La expansión y mejora de las mismas hacen que lleguen cada vez a más usuarios y propician la aparición de nuevos servicios y, al mismo tiempo, el aumento de usuarios y la proliferación de servicios demandan mayor capacidad en estas redes. Gran parte de la culpa de este crecimiento la tienen las aplicaciones multimedia puesto que, por su naturaleza, tienen unas peculiaridades que los distinguen del resto en cuanto a requerimientos para su transmisión (véase la Tabla 1.1 [3]).



(a) *Número de entradas BGP activas.*

(b) *Número de estaciones base GSM/GPRS y 3G/UMTS en España.*

Figura 1.1: Crecimiento de las redes de datos.

Uno de los contenidos multimedia que mayor auge está teniendo es el vídeo [4]. Existen muchos servicios y aplicaciones basados en él: *streaming* de vídeo, videoconferencia, televisión, vídeo bajo demanda (VoD), IPTV, etc. Las propiedades de este tipo de

Servicio	Parámetros de red			
	Sensibilidad a...			Ancho de banda necesario
	Latencia	*Jitter*	Pérdidas	
VoIP	alta	alta	media	bajo
Videoconferencia	alta	alta	media	alto
Streaming de video	media	media	media	medio-alto
Email	baja	baja	alta	bajo
Web	baja	baja	alta	bajo

Tabla 1.1: Requerimientos para algunos servicios típicos.

contenido son las siguientes [3]:

- Requiere, por lo general, un ancho de banda grande.
- Soporta cierto nivel de pérdidas, pero exige que la información llegue ordenada al destinatario.
- La fluctuación de la latencia (*jitter*) debe ser baja. Las aplicaciones interactivas requieren, a su vez, un bajo nivel de latencia, mientras que las no interactivas permiten relajar este aspecto gracias al *buffering* en el reproductor.

Según el informe anual del Cisco Visual Networking Index (VNI), en la actualidad el tráfico de vídeo supone aproximadamente un 40 % de todo el tráfico de usuario de Internet. Se estima que, en 2014, la suma de todas las formas de vídeo (TV, VoD, Internet y P2P) superará el 91 % del tráfico de usuario global [4].

1.1.2. QoS *vs.* QoE

Internet es un sistema global que se compone de la agregación de redes de ordenadores interconectados mediante la pila de protocolos TCP/IP. En su conjunto, puede verse como una inmensa red digital de comunicaciones *best-effort* basada en la conmutación de paquetes, donde cada uno de estos paquetes es tratado de la misma forma. Las funciones del protocolo IP permiten encaminar un paquete de un origen a un destino, pero no aseguran cuándo llegará; ni siquiera si llegará. La enorme ventaja que supone esta sencillez esconde, sin embargo, su mayor desventaja: no conoce prioridades, no contempla conexiones ni estado de la red y, por lo tanto, no asegura unas prestaciones para cada servicio.

Con este marco presente, se denomina *calidad de servicio* (QoS) a *la totalidad de las características de un servicio de telecomunicaciones que determinan su capacidad para satisfacer las necesidades explícitas e implícitas del usuario del servicio* [5]. Esto es, la capacidad de una red o un protocolo para asignar diferentes prioridades a diferentes

tipos de tráfico, aplicaciones o usuarios, de forma que se cumplan sus requerimientos de red y esto se traduzca en unas buenas prestaciones. Así pues, los mecanismos de QoS tienen por objetivo garantizar a cada flujo sus requerimientos críticos de tasa de bits, orden de llegada, retardo, *jitter* y pérdidas.

La calidad de servicio, por tanto, es un término que, aunque tiene un enfoque hacia capas superiores, se circunscribe principalmente —aunque no se limita— al ámbito de los parámetros medibles de transmisión. Por ello, no siempre existe una correspondencia entre QoS y el resultado final que obtiene el usuario a nivel de aplicación, de ahí que a menudo se distinga entre los siguientes términos [6]:

QoS planificada La calidad que el usuario espera recibir y el proveedor espera conseguir al establecer unas políticas y mecanismos de QoS determinados.

QoS conseguida La calidad medida por el proveedor a través de los parámetros de red conseguidos.

QoS percibida La calidad que percibe el usuario final del servicio.

QoS inferida La calidad que determina el proveedor mediante estudios de la opinión de los usuarios.

La *calidad de la experiencia* (QoE), en cambio, es un concepto mucho más amplio y puramente subjetivo que se utiliza para describir la calidad que experimenta un usuario ante un servicio multimedia. Formalmente, se define como *la aceptación general de una aplicación o servicio según la percepción subjetiva del usuario final* [7]. Es decir, viene a englobar lo que antes hemos definido como QoS percibida y QoS inferida, conceptos que ciertamente se escapan de las pretensiones de la definición inicial de QoS.

Con esta nueva perspectiva, podemos afirmar que la QoE es la medida subjetiva de la percepción del usuario, mientras que la QoS es la medida objetiva de los parámetros de transmisión que sirve como herramienta para conseguir una QoE determinada. De hecho, esta estrecha relación entre QoS y QoE es objeto de intensa investigación en la actualidad. En concreto, para el servicio de vídeo, en el cual se enfoca este proyecto, la evaluación de la calidad percibida es uno de los aspectos más importantes y más estudiados, lo que ha dado lugar a una nutrida cantidad estándares y recomendaciones, como se verá en la Sección 1.1.3.3.

1.1.3. Evaluación de la calidad de vídeo

Los sistemas de procesamiento y transmisión de vídeo pueden introducir distorsión en la señal proveniente de diversos componentes: *hardware* de proyección, convertidores, multiplexores, codificadores, enrutadores... Es por ello que la determinación de la calidad de vídeo es un problema complejo. No obstante, con el vídeo como pieza clave de la Internet del futuro más cercano, la evaluación de la calidad de vídeo se torna en pieza

fundamental para una gran variedad de propósitos: evaluación de codificadores, dimensionamiento de redes, evaluación de protocolos de QoS, test de equipos, monitorización de redes, verificación de la calidad en el usuario final, etc. De ahí que mucho esfuerzo investigador se centre en el desarrollo y estandarización de nuevas y mejores métricas de calidad, junto con toda la metodología asociada: requerimientos, procedimientos, buenas prácticas, planes de pruebas, condiciones experimentales, etc.

Estas medidas de calidad de vídeo pueden subdividirse principalmente en dos categorías: medidas subjetivas y medidas objetivas.

1.1.3.1. Medidas subjetivas

Se denominan medidas subjetivas aquellas métricas basadas en la percepción de un grupo de sujetos (típicamente de 15 a 30) en un entorno controlado, los cuales puntúan la calidad de unas secuencias de prueba según un procedimiento [8]. Dichos procedimientos pueden ser de comparación implícita o comparación explícita, de estímulo doble o simple. Estos son algunos de ellos [9]:

- *Double Stimulus Continuous Quality Scale* (DSCQS), por el que los sujetos comparan pares de secuencias cortas, una de prueba y su correspondiente referencia.
- *Double Stimulus Impairment Scale* (DSIS), por el que los sujetos puntúan la degradación de la secuencia de prueba con respecto a una de referencia.
- *Single Stimulus Continuous Quality Evaluation* (SSCQE), por el que los sujetos ven un programa de unos 20-30 minutos de duración y cada cierto tiempo puntúan la calidad instantánea percibida.
- *Absolute Category Rating* (ACR), por el que los sujetos puntúan cada secuencia individualmente, sin ninguna referencia.

La métrica más utilizada para expresar las puntuaciones es la *Mean Opinion Score* (MOS) [10], que varía en una escala de 1 a 5 (véase la Tabla 1.2).

MOS	Calidad	Degradación
5	Excelente	Imperceptible
4	Buena	Perceptible, pero no molesta
3	Regular	Ligeramente molesta
2	Pobre	Molesta
1	Mala	Muy molesta

Tabla 1.2: Escala MOS en relación con la calidad y la degradación percibida.

La percepción visual humana es compleja y difícil de modelar, por lo que las medidas subjetivas representan el método más acorde con nuestra percepción de obtener estimaciones de calidad. Por ello, sirven de base para evaluar la precisión de los modelos

de estimación objetiva a través de la correlación entre estos y aquellas. No obstante, debido a la naturaleza de las medidas subjetivas (son costosas en cuanto a tiempo y dinero, requieren gran cantidad de espectadores, las condiciones experimentales son muy estrictas, etc.), su aplicación directa en el mundo real no resulta factible: en su lugar, se utilizan medidas objetivas que, habitualmente, se mapean en escalas como la MOS.

1.1.3.2. Medidas objetivas

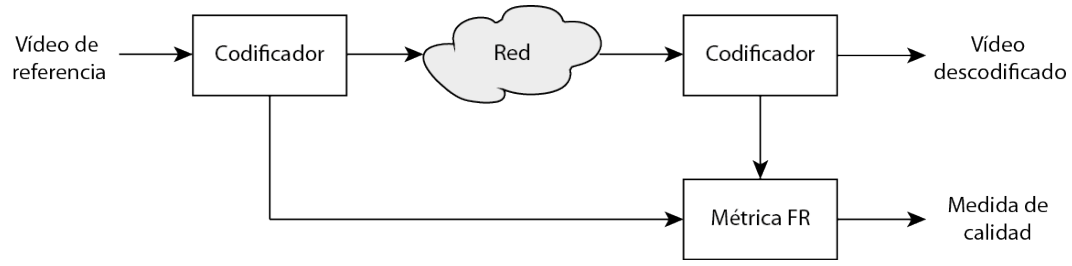
Se denominan medidas objetivas aquellas métricas que caracterizan la calidad basándose en parámetros medibles del sistema de procesamiento y transmisión de vídeo. Son algoritmos de aplicación inmediata y, por tanto, de gran utilidad práctica, a diferencia de las anteriores. Sin embargo, el grado de relevancia que otorgan a los diferentes tipos de defectos presentes en la señal no tiene por qué estar en consonancia con nuestro sistema de percepción, por lo que serán adecuadas en tanto en cuanto su correlación con las medidas subjetivas sea alta y puedan predecir con precisión la calidad experimentada. Las medidas subjetivas, por lo tanto, suponen un punto de referencia para el desarrollo de estas.

Las medidas objetivas se clasifican principalmente en tres categorías [11] dependiendo de la cantidad de información de referencia que requieren (véase la Figura 1.2):

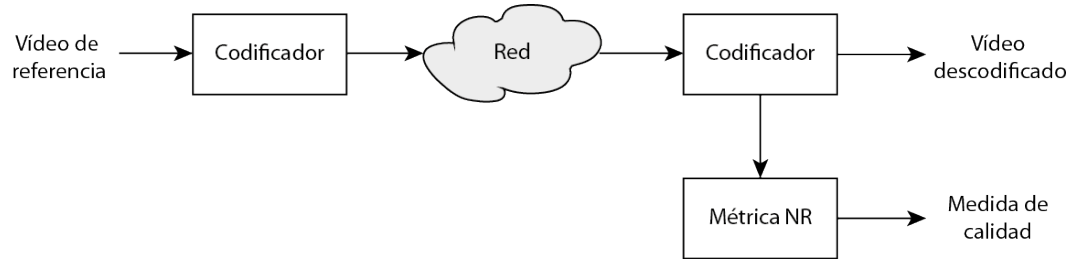
- Las medidas *Full Reference* (FR) requieren el vídeo original como referencia del vídeo recibido de manera que puedan compararse fotograma a fotograma. Por consiguiente, las secuencias de test deben estar almacenadas de forma local en el receptor o bien el emisor debe hacerlas llegar por otro canal.
- Las medidas *No Reference* (NR) analizan solo la secuencia recibida y no precisan de ninguna referencia, por lo que son capaces de funcionar en cualquier punto del sistema y en tiempo real.
- Las medidas *Reduced Reference* (RR) ofrecen un compromiso entre las medidas FR y NR. En este caso también se requiere información de referencia, pero no de la secuencia íntegra. En su lugar, se extraen una serie de características de ambos vídeos (el recibido y la referencia): una cantidad de información manejable con la cual se realiza la comparación.

Las medidas FR son más indicadas para pruebas de laboratorio o medidas *offline* por la gran cantidad de información que manejan y la carga computacional que acarrean, mientras que las RR o NR pueden utilizarse para monitorizar servicios reales en diferentes puntos de la red: las medidas RR son más indicadas para los extremos; en cambio, para puntos intermedios, las NR constituyen la mejor opción.

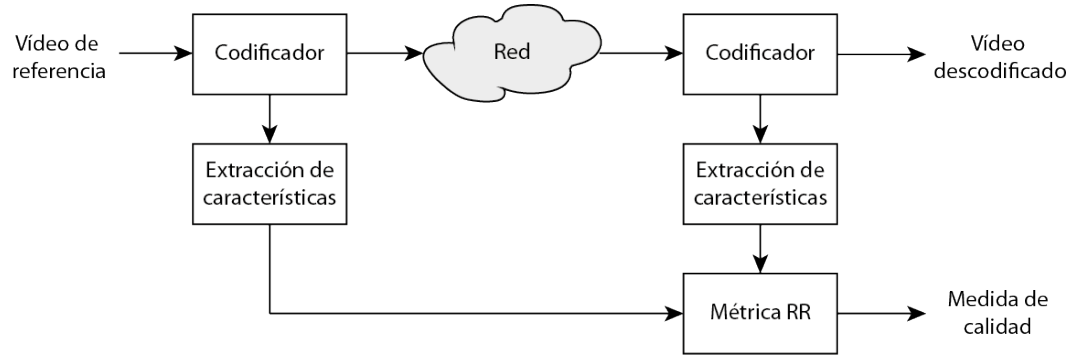
Paralelamente, puede establecerse otra clasificación en función del tipo de información que requiere cada métrica (véase la Figura 1.3) [8]:



(a) *Full Reference.*



(b) *No Reference.*



(c) *Reduced Reference.*

Figura 1.2: Clasificación de medidas de calidad de vídeo en función de la cantidad de información que manejan.

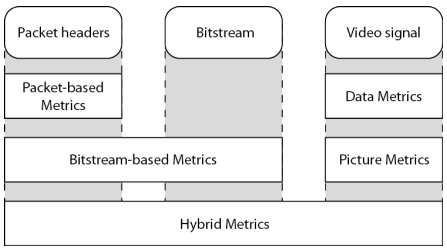
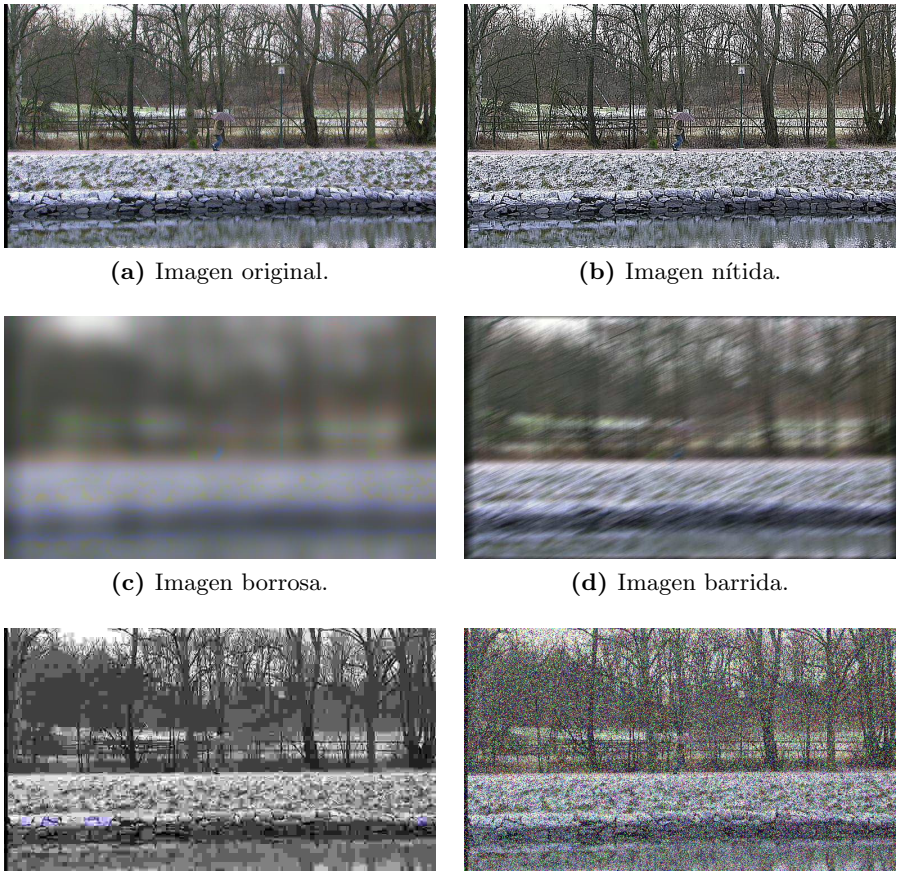


Figura 1.3: Clasificación de medidas de calidad de vídeo en función del tipo de información que manejan.

Data Metrics

Son medidas que comparan las señales ideal y obtenida *byte* a *byte* sin reparar en qué representan. Se tratan los datos como algo genérico sin tener en cuenta ni el tipo de contenido ni el tipo de distorsión que presentan. Dos de las más conocidas y utilizadas en el campo del procesado de imágenes son el *Mean Squared Error* (MSE) y el *Peak Signal to Noise Ratio* (PSNR) —siendo el segundo una representación logarítmica del primero—. El PSNR es una medida que compara píxel a píxel y fotograma a fotograma la señal de vídeo. Es totalmente ignorante de las relaciones espaciales y temporales de los píxeles y, como resultado, distorsiones de muy distinto tipo que derivan claramente en percepciones de la calidad muy diferentes obtienen valores de PSNR similares (véase la Figura 1.4 [12]). No obstante, el PSNR es una medida bien conocida que se computa de manera sencilla y rápida, por lo que es muy útil como primera aproximación.



(a) Imagen original. **(b)** Imagen nítida.

(c) Imagen borrosa. **(d)** Imagen barrida.

(e) Imagen comprimida en JPEG. **(f)** Imagen con nieve.

Figura 1.4: Comparación de imágenes con diferentes tipos de distorsión, todas ellas con PSNR = 28 dB y una resolución de 768×432.

Picture Metrics

A diferencia de las anteriores, son medidas que tienen en cuenta el tipo de contenido que manejan y sus características, así como la manera en que degradan la calidad los diferentes tipos de distorsión. En esta subcategoría de medidas, podemos encontrar el enfoque del *modelado de la visión*, por el que se trata de modelar el sistema de percepción visual humano, y un enfoque más *ingenieril* que busca analizar y extraer ciertas características del vídeo. Ejemplos de esto último son el *Structural Similarity* (SSIM) *index* de Wang et al. [13] o el *Video Quality Metric* (VQM) de Pinson y Wolf [14].

Packet- and Bitstream-Based Metrics

Estas medidas extraen la información de las cabeceras de los paquetes que transportan la información multimedia (como las cabeceras RTP) y de las características de la técnica de compresión que utiliza el correspondiente codificador (frecuencia de los distintos tipos de *frames*, etc.). Así, la cantidad de información que manejan es muy inferior a las métricas anteriores, ya que no se requiere la descodificación del vídeo. Como desventaja, son medidas dependientes de los protocolos de transmisión y los codificadores utilizados, por lo que deben ser adaptadas a estos.

Hybrid Metrics

Por último, las medidas *híbridas* abarcan las tres áreas que cubren las categorías anteriores: cabeceras de paquetes, flujo de bits y señal de vídeo descodificada.

1.1.3.3. Actividades de estandarización

La actividad investigadora en este campo ha dado como resultado un buen número de recomendaciones de la Unión Internacional de Telecomunicaciones (ITU). Gran parte de estos logros se deben al trabajo del *Video Quality Experts Group* (VQEG) [15], un grupo de expertos (muchos de los cuales pertenecen a la ITU) establecido en 1997 y dedicado al desarrollo, validación y estandarización de nuevas medidas objetivas de evaluación de vídeo.

Algunas de las recomendaciones más importantes en este ámbito son las que se enumeran a continuación:

- Para medidas subjetivas:
  - ITU-R Rec. BT.500-12 [9]: Metodología para la evaluación subjetiva de la calidad de las imágenes de televisión.
  - ITU-T Rec. P.800 [10]: Métodos de determinación subjetiva de la calidad de transmisión.
  - ITU-T Rec. P.910 [16]: Métodos de evaluación subjetiva de la calidad de vídeo para aplicaciones multimedia.

  - ITU-T Rec. P.911 [17]: Métodos de evaluación subjetiva de la calidad audiovisual para aplicaciones multimedia.

- Para medidas objetivas:

  - ITU-T Rec. G.1070 [18]: Modelo de opinión para aplicaciones de vídeotelefonía.
  - ITU-T Rec. J.144 [19]: Técnicas de medición objetiva de la percepción de la calidad vídeo en televisión digital por cable en presencia de una referencia completa.
  - ITU-T Rec. J.246 [20]: Técnicas de medida de la calidad de vídeo percibida para servicios multimedia sobre redes de televisión digital por cable en presencia de ancho de banda de referencia reducido.
  - ITU-T Rec. J.247 [21]: Medida de la calidad de vídeo multimedia percibida en presencia de referencia completa.
  - ITU-T Rec. J.249 [22]: Técnicas de medida de la calidad de vídeo percibida para televisión digital por cable en presencia de referencia reducida.

1.2. Motivación

Como ya se ha apuntado en la Sección 1.1.3, la evaluación de la calidad de vídeo sirve a una gran cantidad de propósitos. En el ámbito de la investigación, es necesaria para la evaluación de codificadores, de protocolos de QoS, de métricas, test de equipos, etc. Adicionalmente, no menos interesantes resultan las aplicaciones empresariales en el ámbito de las telecomunicaciones: tanto *a priori*, para el dimensionamiento de redes e implantación de servicios de vídeo (p. ej.: IPTV), como *a posteriori*, para la monitorización de estos servicios, verificación de la calidad en el usuario final, etc.

No es complicado encontrar herramientas que ofrecen una solución parcial, ya que el sistema sobre el que se produce la evaluación de la calidad contempla el procesado del vídeo, pero no la transmisión. Son aplicaciones más enfocadas a la evaluación de codificadores, ya sean implementaciones de métricas concretas (como el caso de SSIM o PSNR) o plataformas más generalistas. En esta última categoría, destaca especialmente IVQUEST [23], una aplicación de MATLAB para la evaluación de métricas de calidad de imagen y vídeo que reúne una gran cantidad de medidas de calidad, además de filtros para generar archivos de test, herramientas de análisis y una interfaz para pruebas subjetivas.

Sin embargo, no son tantas las herramientas que contemplen la transmisión y que estén dirigidas a entornos IP, como se verá en la revisión del estado del arte, en el Capítulo 2. La razón principal de esta carencia puede ser la complejidad que conlleva: existen muchos codificadores y formatos disponibles, se han definido gran cantidad de medidas de muy diferente naturaleza y con distintos requerimientos, etc.

Por todo ello, se ve necesario el desarrollo de una aplicación flexible que abarque toda la problemática de la evaluación de la calidad de vídeo, de forma que sea susceptible de funcionar tanto en entornos de laboratorio como de producción, para propósitos de investigación o de dimensionamiento y monitorización de servicios de telecomunicaciones.

1.3. Objetivos y alcance del proyecto

El principal objetivo del presente proyecto es el diseño e implementación de una aplicación destinada a la estimación de la calidad de vídeo en un entorno de transmisión IP, tanto real como de laboratorio. Dicha aplicación aportará una solución completa al problema de la evaluación de la calidad: se encargará del envío, la recepción, la codificación, la descodificación, la extracción de parámetros y características, y el cálculo de medidas de calidad.

Dicha extracción de parámetros tendrá lugar en los tres niveles posibles anteriormente descritos, tanto para su análisis por separado como para su incorporación a las medidas de calidad de vídeo:

- A nivel de paquete, se extraerán parámetros de QoS tales como ancho de banda, latencia, *jitter*, tasa de pérdidas, etc.
- A nivel de *bitstream*, se extraerá información relativa al vídeo codificado: tamaño de los *frames*, tipo, tamaño del GOP, etc.
- A nivel de señal de vídeo, se extraerán los *frames* descodificados.

En resumen, se pretende realizar una plataforma o *framework* que ofrezca la mayor cantidad de información posible para cubrir los requerimientos de todos los tipos de métricas objetivas vistas en la Sección 1.1.3.2, y que permita la implementación e incorporación sencilla y rápida de nuevas medidas de calidad, tanto objetivas como mapeos de estas en métricas subjetivas como la MOS.

Capítulo 2

Estado del arte

En el Capítulo 1, se ha repasado el auge que está experimentando el tráfico de vídeo en Internet y se ha establecido el importante papel que juega en este escenario la evaluación de la calidad de vídeo sobre redes IP. A pesar de ello, no existen muchas aplicaciones que automaticen esta tarea.

En este capítulo se hará un repaso de algunas soluciones existentes, totales o parciales, sobre el ámbito del vídeo u otro relacionado.

2.1. VoIP Tester

Es una aplicación de código abierto (con licencia GNU GPLv3) para la evaluación de la calidad de voz sobre redes IP [24]. Se trata de un Proyecto Fin de Carrera realizado en la Escuela Superior de Ingenieros Informáticos y de Telecomunicación de la Universidad de Granada por el Ingeniero de Telecomunicación Antonio Sánchez Navarro, con la dirección de los profesores D. Juan Manuel López Soler y D. Jorge Navarro Ortiz, tutores asimismo del presente proyecto. VoIP Tester es el precursor de este o, más bien, Video Tester viene a continuar la línea de trabajo emprendida por aquel en otro ámbito de las comunicaciones multimedia.

2.1.1. Características

VoIP Tester está desarrollado en Java y su diseño gira en torno a una arquitectura de tipo cliente-servidor. Consta de dos aplicaciones: una para el cliente y otra para el servidor. El cliente es el agente activo que controla todo el proceso (mediante *Remote Method Invocation* de Java): escoge los parámetros de transmisión, establece el sentido de la comunicación (ascendente o descendente), la inicia y analiza y presenta los resultados.

La codificación de la voz se efectúa con la asistencia de *GStreamer* para Java y la transmisión, aunque utiliza UDP sobre IP como protocolo de transporte, viene deter-

minada por un formato de paquetes propio. Realiza medidas de calidad FR y basadas en parámetros de QoS.

2.1.1.1. Ventajas

- Es de código abierto, luego puede desarrollarse libremente.
- Es multiplataforma, ya que está escrito en Java.
- Desde el cliente se puede evaluar tanto el sentido descendente de la comunicación como el ascendente, ya que una comunicación VoIP es bidireccional.
- Una *paquetización* flexible permite que la transmisión sea completamente configurable: tamaño de los paquetes, tiempo entre paquetes, tamaño del *mensaje*, número de repeticiones del mensaje... son algunos de los parámetros que VoIP Tester es capaz de modificar.

2.1.1.2. Desventajas

- La herramienta consta de dos aplicaciones separadas. Esto, en un primer momento, no parece tan importante. Sin embargo, en secciones posteriores se verá cómo de lo que más adolecen este tipo de programas es precisamente de la complejidad en su uso, debido, en gran medida, a la fragmentación (muchas aplicaciones distintas en un mismo paquete de *software*).
- La invocación de métodos remotos de Java presenta problemas para funcionar a través de NAT, lo que obstaculiza la utilización de esta aplicación en entornos reales.
- La implementación no permite más de un cliente por servidor, por lo que pierde una funcionalidad importante de este tipo de arquitectura.
- *A priori*, no se dispone de un flujo VoIP real, así que las ventajas que supone un formato de paquetes propio son muy válidas para entornos de laboratorio, pero las mediciones de calidad no son directamente aplicables a transmisiones reales. No obstante, es cierto que pueden estudiarse los parámetros de una transmisión real y ajustarlos en VoIP Tester, pero esto requiere un esfuerzo adicional y solo es posible para transmisiones VoIP con tamaños de paquete fijos.

2.2. EvalVid

Se trata de un conjunto de herramientas de código abierto (publicadas sin licencia) para la evaluación de la calidad de vídeo transmitido a través de una red de comunicaciones real o simulada desarrollado por investigadores del Telecommunication Networks Group

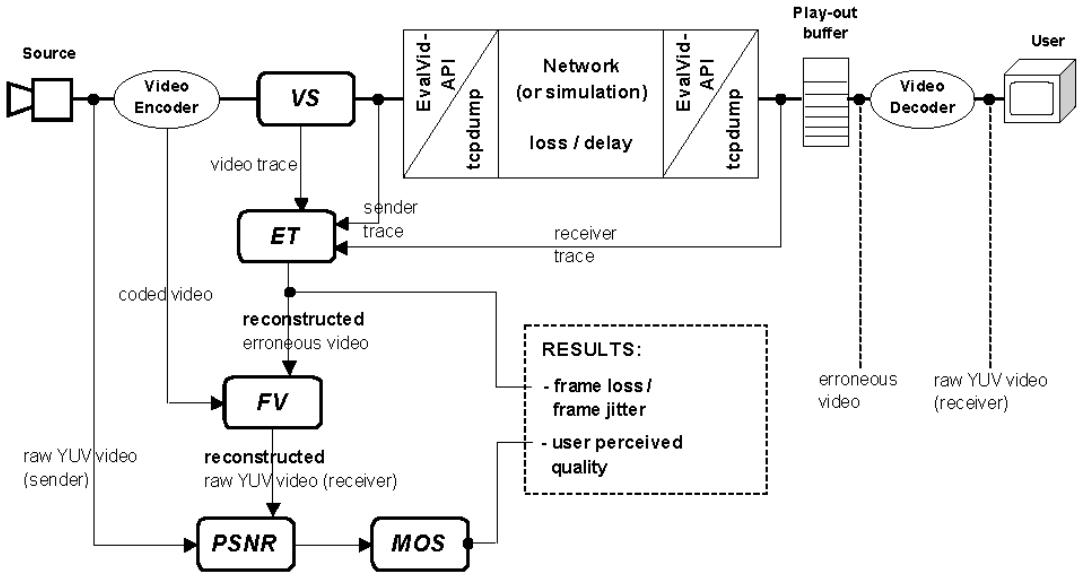


Figura 2.1: Esquema de funcionamiento de EvalVid.

(TKN) de la Technical University de Berlín [25]. De forma análoga al presente proyecto, supone una solución completa a la problemática que se plantea (véase la Figura 2.1).

2.2.1. Características

EvalVid está escrito en C. Desde su página web [26] puede descargarse tanto el código fuente como binarios compilados para Windows, Linux y Mac OS X. Como ya se ha dicho, no es una única aplicación, sino que es un conjunto de ellas. Concretamente, en la última versión (EvalVid 2.7) encontramos un total de ocho binarios que se describen a continuación:

`vsgen` Herramienta de manipulación de archivos YUV.

`mp4trace` Herramienta para el envío de un archivo MP4 hacia un *host* especificado a través de UDP/RTP.

`eg` Generador de errores. Dado un archivo de captura y una distribución de error, produce otro archivo de captura con pérdidas marcadas (marca los paquetes, no los elimina).

`psnr` Calcula el PSNR de cada fotograma descomprimido. También permite calcular la SSIM desde la versión 2.2.

`etmp4` Herramienta para la evaluación la traza de la transmisión de un archivo MP4. Básicamente lo que hace es reconstruir el vídeo en dos versiones: un archivo

probablemente corrupto (por las pérdidas) y otro que solo contiene los fotogramas sin daños.

`hist` Generador de histogramas a partir de datos de retardo o *jitter*.

`mos` Calcula la MOS de cada fotograma del vídeo recibido mediante un simple mapeo de la PSNR a la escala MOS.

`miv` Calcula el porcentaje de fotogramas con una MOS peor que la de referencia (la del vídeo codificado pero sin pasar por el sistema de transmisión) en un intervalo deslizante de 25 fotogramas. Se trata de una medida de calidad que únicamente se encarga de suavizar la respuesta escalonada que ofrece el mapeo de la herramienta `mos`.

Además, son necesarias otras herramientas externas para la codificación y la descodificación (p. ej.: `ffmpeg` [27]), la creación de contenedores MP4 (`MP4Box`, parte de GPAC [28]) o la captura de paquetes (`tcpdump`, `netcat`).

Ofrece valores de retardo, *jitter* y pérdidas, y realiza medidas de calidad de vídeo FR, concretamente PSNR y SSIM, además de la ya comentada MIV.

2.2.1.1. Ventajas

- Es de código abierto, luego puede desarrollarse libremente.
- Dispone de binarios para las tres plataformas más importantes: Windows, Linux y Mac OS X.
- Produce trazas de vídeo reales, ya que el vídeo se encapsula sobre RTP.
- Sirve tanto para redes reales como simuladas [29].

2.2.1.2. Desventajas

- La alta fragmentación de la aplicación hace que su uso sea más complejo.
- Se requieren herramientas externas para su uso.
- Su aplicación es limitada porque solo admite la transmisión de contenedores MP4. Eso significa que únicamente pueden evaluarse aquellos codificadores que están contemplados en la especificación de MP4. Por el momento, soporta los siguientes codificadores: MPEG-4, H.263 y H.264.
- Soporta pocas medidas tanto de QoS como de evaluación de la calidad de vídeo.
- Al estar desarrollado en C, tampoco facilita la ampliación de las herramientas para añadir mayor soporte a codificadores y medidas.

Por último, cabe destacar que, cuando en una transmisión se producen pérdidas, esto se traduce en que faltan fotogramas en el vídeo recibido; es decir, hay menos fotogramas que en el vídeo original y existen problemas de sincronización. Como consecuencia, los valores de PSNR pierden fiabilidad, ya que se comparan fotogramas desalineados, y las medidas de calidad se ven degradadas. Los autores de EvalVid conocen este problema y así lo expresan en la web, pero no han dado con una solución.

2.3. Video Quality Metric (VQM) Software

Investigadores del Institute for Telecommunication Sciences, del National Telecommunications and Information Administration y del Department of Commerce de Estados Unidos han desarrollado este set de herramientas para la evaluación de la calidad de vídeo y lo ponen a disposición pública de manera gratuita en su página web (tras previo registro en la misma) [30].

2.3.1. Características

Proporciona diferentes aplicaciones que implementan métodos estandarizados que se recogen tanto en la ANSI como en la ITU (concretamente en [19, 31, 32]). Se describen a continuación:

`VQM` Programa de Windows y Linux (sin soporte ya) para pruebas *offline*. Proporciona una interfaz gráfica de usuario para el procesado en lote y el análisis de múltiples vídeos y sistemas. Permite seleccionar varios algoritmos de calibración, así como modelos de medida de calidad.

`BVQM` Expansión del anterior (también se proporciona el código fuente en MATLAB).

`CVQM` Otra expansión del primero (también se proporciona el código fuente en MATLAB).

`FDF` Programa de Windows (también se proporciona el código fuente en MATLAB) para medidas NR y RR de las *Fraction Droped Frames* de un flujo de vídeo procesado.

`IVQM` Programa de Windows (también se proporciona el código fuente en MATLAB) para monitorización en tiempo real mediante una conexión RR.

`PSNR` Programa de Windows (también se proporciona el código fuente en MATLAB) para el cálculo de la PSNR de un flujo de vídeo procesado.

`VDF` Programa de Windows (también se proporciona el código fuente en MATLAB) para medidas FR de los *Variable Frame Delays* de un flujo de vídeo procesado.

2.3.1.1. Ventajas

- Dispone de herramientas para la monitorización tanto *offline* como en tiempo real.
- Los estándares implementados proporcionan métodos para la calibración y alineamiento de fotogramas, además de medidas FR, RR y NR.
- Los buenos resultados de la Fase II del VQEG lo avalan.

2.3.1.2. Desventajas

- Aunque se proporcionan los algoritmos en MATLAB, son herramientas cerradas que no se prestan a la ampliación con nuevas medidas.

2.4. MintMOS

Es un módulo del núcleo de Linux para la determinación de la calidad de la experiencia para flujos de vídeo en tránsito desarrollado por investigadores de la University of Central Florida [33].

2.4.1. Características

Se trata de una herramienta NR lista para actuar en puntos intermedios de la red como enrutadores. Sus autores presentan un modelo de reconocimiento de patrones en el que definen un espacio de características (*QoE space*) de k dimensiones y N particiones que es entrenado mediante vídeos con parámetros conocidos y a base de encuestas de opinión de los usuarios. De esta forma, se obtiene un clasificador que posteriormente puede actuar de manera autónoma sin ninguna referencia.

2.4.1.1. Ventajas

- Es una herramienta NR que puede incorporarse en puntos intermedios de la red para la monitorización de servicios sin tener que desplazarse hasta la ubicación del usuario final.
- Esta monitorización transparente para el usuario puede disparar alertas o desencadenar acciones que mejoren la QoE en caso de superar ciertos umbrales de degradación.

2.4.1.2. Desventajas

- Requiere un proceso de definición del modelo y del posterior entrenamiento muy cuidadoso.

- Como se ha comentado en la Sección 1.1.2, los parámetros de QoS no siempre encajan bien con la opinión del usuario y los resultados de las extensas investigaciones del VQEG no han dado resultados satisfactorios con medidas sin referencia (esto se refleja en que todavía no existe ninguna recomendación de la ITU para medidas NR). Por ello, no parece probable que esta herramienta, que trabaja únicamente con parámetros de nivel de red, arroje resultados muy fiables.

Capítulo 3

Especificación de requisitos

La especificación de requisitos queda dividida en los siguientes apartados: requisitos funcionales, no funcionales, de datos y de la interfaz.

3.1. Requisitos funcionales

Para la definición de los requisitos funcionales, se ha optado por un diagrama de flujo de datos en tres niveles: diagrama de contexto, diagrama de nivel superior y diagrama de expansión.

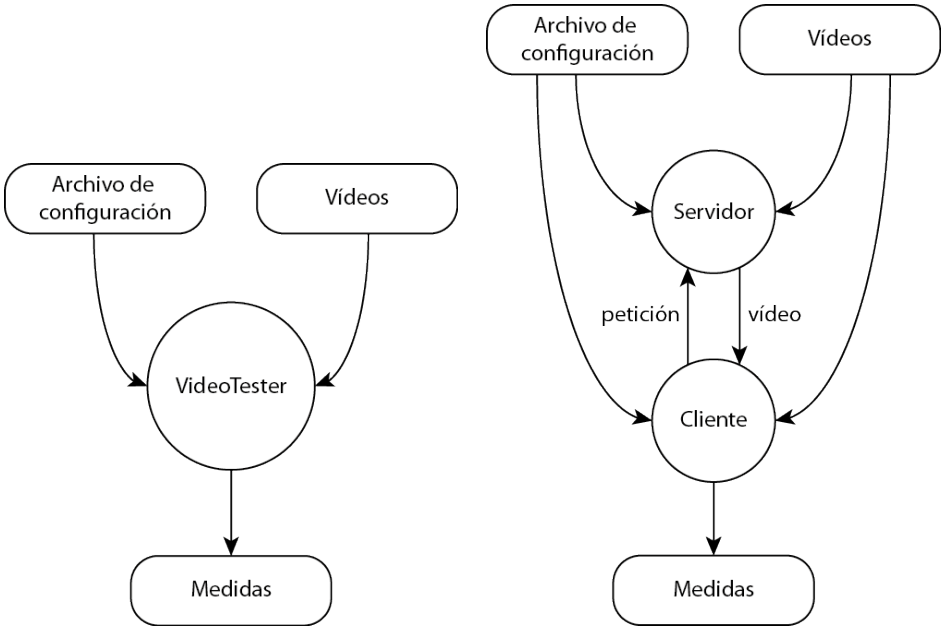
El diagrama de contexto define las relaciones del sistema con el exterior. Para el caso de Video Tester, esto se traduce en las entradas y salidas del mismo. Como se aprecia en la Figura 3.1a, Video Tester requiere dos entradas —un conjunto de vídeos de prueba y un archivo de configuración— y ofrece una salida: las medidas calculadas a nivel de red, de *bitstream* y de fotograma.

A un nivel de abstracción más bajo, la herramienta constará de una única aplicación que podrá lanzarse en modo cliente (el agente que recibe el vídeo) o servidor (el agente que envía el vídeo), como ilustra la Figura 3.1b. Típicamente, ambos procesos se ejecutarán en ordenadores distintos y separados por una red de comunicaciones IP con el objetivo de medir la calidad de vídeo en ese enlace. El servidor se lanzará en modo *standalone* a la espera de un cliente, el cual dirigirá todo el proceso. Además, el servidor será capaz de gestionar múltiples clientes de forma concurrente.

Por último, se describen los flujos de datos y los principales procesos involucrados presentes en el diagrama de expansión (véase la Figura 3.2):

`iniciar` Una vez lanzado el servidor, el cliente inicia el proceso realizando una petición que contiene los parámetros de vídeo configurados (*bitrate*, etc.).

`lanzar servidor RTSP` El programa servidor inicia un servidor RTSP con los parámetros de vídeo solicitados.



(a) *Diagrama de contexto* **(b)** *Diagrama de nivel superior*

Figura 3.1: Diagramas de flujo de datos de alto nivel.

`enviar vídeo` Se realiza el envío del vídeo solicitado a través de la red de comunicaciones.

`escuchar red` De nuevo del lado del cliente, se produce una escucha en la interfaz de red correspondiente para generar un archivo de captura de paquetes.

`recibir vídeo` Paralelamente al proceso anterior, se recibe el vídeo y se generan dos salidas: el vídeo recibido codificado (tras desempaquetar) y el vídeo recibido descodificado (tras desempaquetar y descodificar). Cuando se ha completado la recepción, se insta al programa servidor a parar el servidor RTSP.

`extraer parámetros` Este proceso puede dividirse en dos procesos diferentes. Por un lado, se toma el fichero de traza y se extraen los parámetros de QoS (definidos en la Sección 3.2.3); por otro lado, se toma el vídeo recibido codificado y se extraen los parámetros de *bitstream* (definidos en la Sección 3.2.4).

`calcular medidas` Este proceso puede dividirse en tres procesos diferentes. Uno de ellos calcula las medidas de QoS mediante los parámetros de QoS. Otro calcula las medidas de *bitstream* mediante los parámetros de *bitstream*. Un tercero recibe el vídeo de referencia, el vídeo recibido descodificado y la realimentación de las medidas de QoS y *bitstream* para el cálculo final de las medidas de calidad de vídeo correspondientes.

`parar servidor RTSP` Cuando el cliente solicita la parada del servidor RTSP con los parámetros indicados, se producirá la parada en caso de que no haya ningún cliente utilizándolo.

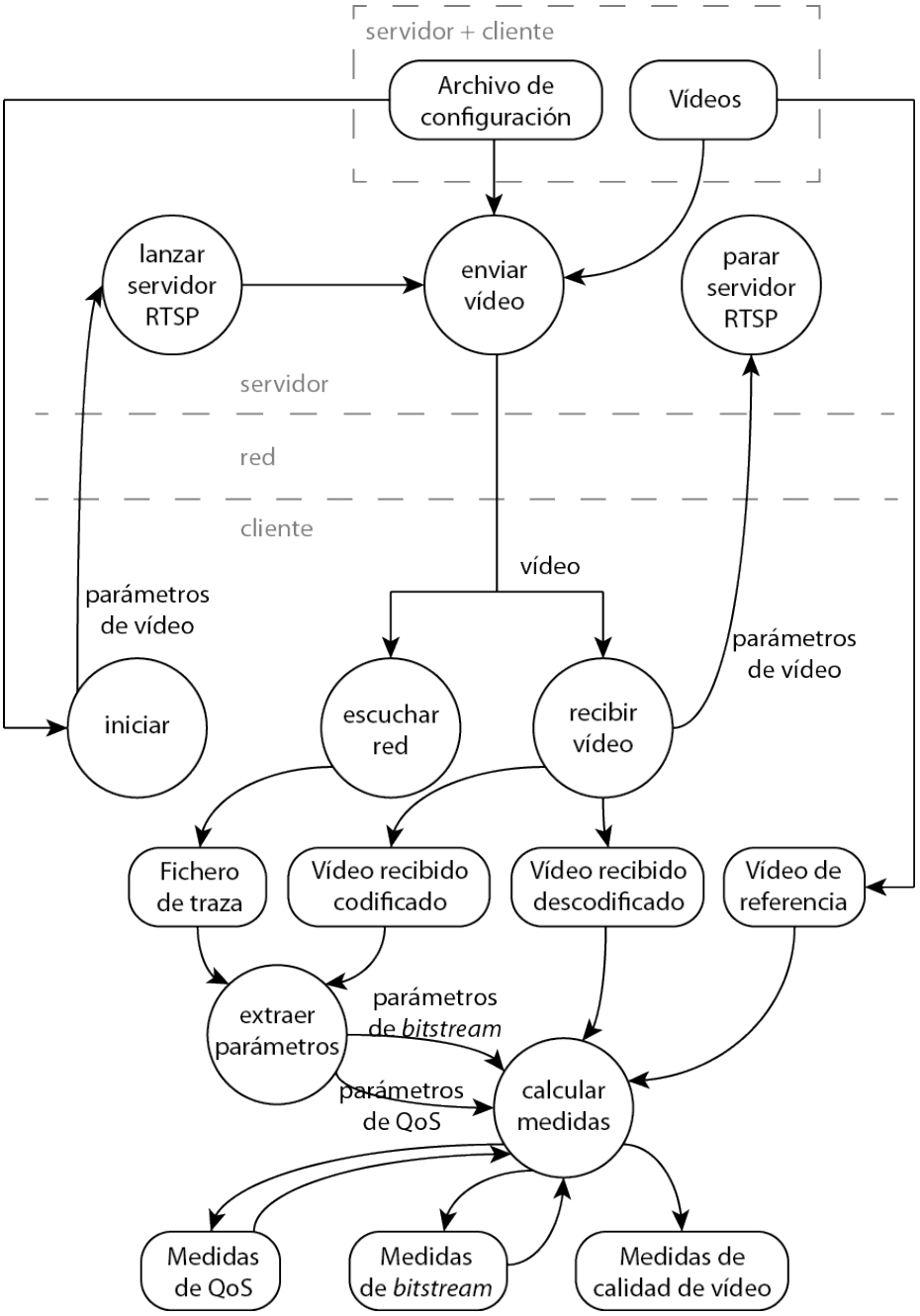


Figura 3.2: Diagrama de expansión.

3.2. Requisitos de los datos

Como así se muestra en el diagrama de expansión (Figura 3.2), el sistema maneja cinco tipos de datos: vídeo, parámetros de vídeo, parámetros de QoS, parámetros de *bitstream* y archivos de configuración.

3.2.1. Vídeo

Es necesario que tanto el cliente como el servidor compartan un conjunto de vídeos de prueba. El archivo de vídeo, tras la codificación por parte del servidor, será enviado por un servidor RTSP. Dicho servidor permitirá la tranmisión sobre RTP/TCP y RTP/UDP, este último tanto en modo *unicast* como *multicast*.

En cuanto a los codificadores soportados, serán cuatro de los más comunes: H.263 [34], H.264 [35], MPEG-4 parte 2 [36] y Theora [37].

3.2.2. Parámetros de vídeo

Son los parámetros configurados en el lado del cliente necesarios para el envío del vídeo:

Vídeo El archivo seleccionado del conjunto que comparten cliente y servidor.

Codificador Uno entre los soportados.

Bitrate Tasa de bits por segundo a la que se codificará el vídeo.

Framerate Tasa de *frames* por segundo del vídeo.

3.2.3. Parámetros de QoS

Por cada paquete RTP, se extraerán los siguientes parámetros de QoS:

- Tamaño del paquete de red (con cabeceras) en *bytes*.
- Número de secuencia RTP (véase [38]).
- Marca de tiempo RTP (véase [38]).
- Tiempo de llegada.

Los valores de número de secuencia, marca de tiempo y tiempo de llegada estarán normalizados de forma que empiecen en cero para cada secuencia de vídeo analizada. Los paquetes RTP se reordenarán en función del número de secuencia y, por último, cada parámetro constituirá una lista:

`lengths` Lista de tamaños.

`sequences` Lista de números de secuencia.

`timestamps` Lista de marcas de tiempo.

`times` Lista de tiempos de llegada.

3.2.4. Parámetros de *bitstream*

Será necesario contar con un analizador específico para cada codificador soportado. De esta manera, podrán extraerse los siguientes parámetros a nivel de *bitstream* para cada uno de los *frames* codificados:

- Tipo de imagen: I (imagen de referencia que no necesita de otras para su descodificación), P (requiere imágenes anteriores para su descodificación), B (requiere imágenes anteriores y posteriores para su descodificación) u otras específicas del formato.
- Tamaño del *frame* en *bytes*.

Ambos parámetros constituirán una estructura de datos consistente en dos listas:

`lengths` Lista de tamaños.

`types` Lista de tipos.

3.2.5. Archivos de configuración

Existirá uno en común para el servidor y el cliente que definirá la ruta hasta los archivos de vídeo y los propios vídeos disponibles. El cliente dispondrá de un segundo archivo de configuración donde se podrán escoger las siguientes opciones:

- Parámetros de vídeo (véase la Sección 3.2.2), es decir: vídeo, codificador, *bitrate* y *framerate*.
- Opciones de red: interfaz, dirección IP del servidor, puerto del servidor y protocolo de transporte (UDP *unicast*, UDP *multicast* o TCP).
- Medidas de QoS a calcular.
- Medidas de *bitstream* a calcular.
- Medidas de calidad de vídeo a calcular.

3.3. Requisitos no funcionales

***Software* y compatibilidad**
: En la implementación se utilizará el lenguaje de programación Python para facilitar la incorporación de nuevas medidas. Aunque este es multiplataforma, en principio el desarrollo se restringirá a entornos Linux, dejando abierta la posibilidad futura de extensión a otros.

Dependencia de otras partes
: Se utilizarán las librerías de *GStreamer* [39] para la manipulación del vídeo por el amplio soporte que ofrece en este ámbito y porque puede manejarse a través de los *bindings* disponibles para Python.

Gestión de fallos
: Se incorporará un completo servicio de *logging* que registre mensajes de estado de la ejecución, información, advertencias, errores, etc.

Extensibilidad
: Se aislará convenientemente el módulo de medidas del resto de la aplicación con el objetivo de hacerla fácilmente extensible. Se proporcionará una interfaz estándar con unos requisitos bien definidos sobre la que implementar nuevas medidas sin alterar el núcleo del programa.

3.4. Requisitos de la interfaz

Para ambos modos de funcionamiento, cliente y servidor, se proporcionará una interfaz por línea de comandos: en un caso por su simplicidad (el servidor se lanza en *standalone* y no requiere más interacción por parte del usuario) y, en el otro, para facilitar el lanzamiento de baterías de pruebas. Adicionalmente, se proporcionará una interfaz gráfica para el modo cliente para demostraciones y presentación de resultados en gráficas sin la dependencia de herramientas externas.

3.4.1. Interfaz por línea de comandos

Argumentos opcionales:

`-h, --help` Muestra la ayuda.

Subcomandos:

`server` Inicia la aplicación en modo servidor.

`client` Inicia la aplicación en modo cliente.

3.4.1.1. Subcomando *server*

No requiere argumentos. Posee argumentos opcionales:

`-h, --help` Muestra la ayuda.

3.4.1.2. Subcomando *client*

No requiere argumentos. Posee argumentos opcionales:

`-h, --help` Muestra la ayuda.

`-c CONF, --conf CONF` Carga el archivo de configuración `CONF` en lugar del archivo por defecto.

`-g, --gui` Lanza la interfaz gráfica para el modo cliente.

3.4.2. Interfaz gráfica

La interfaz gráfica contará con una única ventana con cuatro pestañas y dos menús.

3.4.2.1. Pestañas

Configuration Sustituye al archivo de configuración, por lo que permite escoger los mismos parámetros (véase la Sección 3.2.5).

Log Muestra los mensajes del sistema.

Results Muestra los resultados en forma de gráfica de todas las medidas calculadas (inicialmente oculta).

Video Proporciona controles para reproducir los vídeos, degradados y sin degradar, con el objetivo de poder compararlos visualmente (inicialmente oculta).

3.4.2.2. Menús

Open Permite abrir archivos de medidas guardadas en ejecuciones anteriores para poder visualizarlos en la pestaña de resultados.

Run Inicia el proceso descrito en la Figura 3.2 con las opciones seleccionadas en la pestaña de configuración.

Capítulo 4

Planificación y estimación de costes

4.1. Recursos

4.1.1. Humanos

- D. Juan Manuel López Soler y D. Jorge Navarro Ortiz, profesores del Departamento de Teoría de la Señal, Telemática y Comunicaciones de la Universidad de Granada, en calidad de tutores del proyecto.
- Iñaki Úcar Marqués, alumno de la Escuela Técnica Superior de Ingenieros Industriales y de Telecomunicación de la Universidad Pública de Navarra, autor del proyecto.

4.1.2. *Hardware*

- Ordenador personal.
- Ordenador de la Escuela con IP pública para la fase de pruebas.

4.1.3. *Software*

- Sistema operativo GNU/Linux Fedora 14 y herramientas de virtualización basadas en el Kernel (KVM).
- Ofimática: *Kile*, *KBibTex*, *GIMP*, *Dia* y *GnuPlot*.
- Lenguaje de programación Python versión 2.7 y librerías estándar [40].
- Entorno de desarrollo *SPE* (*Stani's Python Editor*) [41] y diseñador de interfaces gráficas *wxGlade* [42].

- *GStreamer 0.10* [39].
- *GStreamer plugins 0.10*: *base*, *good*, *ugly*, *bad*.
- *GStreamer-FFmpeg plugins 0.10* [43].
- *GStreamer RTSP server 0.10.7* [44].
- Librerías no estándar para Python: *Scapy 2.0* [45], *Matplotlib 1.0.1* [46], *Numpy 1.4.1* [47], *GStreamer bindings 0.10*, *WxPython 2.8.11* [48], *WxPython backend* para *Matplotlib 1.0.1*, *OpenCV bindings 2.1* [49].

4.2. Fases de desarrollo

La distribución temporal de los seis paquetes de trabajo puede verse en el diagrama de Gantt de la Figura 4.1.

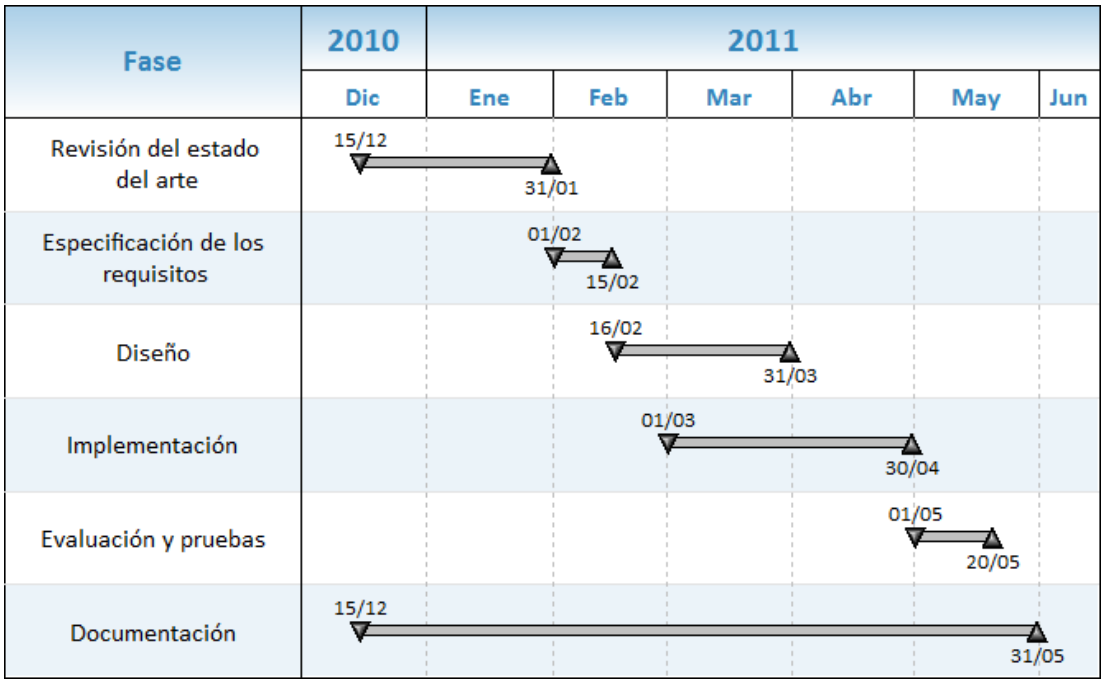


Figura 4.1: Planificación temporal del proyecto.

4.2.1. Revisión del estado del arte

Se hace un repaso de las soluciones existentes hasta el momento, totales o parciales, sobre el mismo ámbito de aplicación o relacionado, del problema a resolver con el objetivo de sentar una base que justifique la realización de este proyecto.

4.2.2. Especificación de los requisitos

Se analiza pormenorizadamente el problema en pos de una correcta especificación de requisitos, funcionalidades y restricciones que delimitan el posterior diseño de una solución.

4.2.3. Diseño

Se adopta justificadamente una solución previamente especificada y se procede a su diseño.

4.2.4. Implementación

Se implementa el diseño anterior en el lenguaje de programación escogido.

4.2.5. Evaluación y pruebas

Tras la comprobación y puesta a punto, se procede a la evaluación de la solución, contrastando su funcionamiento y resultados, si procede, con otras soluciones vistas durante la revisión del estado del arte.

4.2.6. Documentación

La redacción de la documentación es una tarea que se lleva a cabo de forma paralela al resto.

4.3. Estimación de costes

4.3.1. Recursos humanos

En la Tabla 4.1 puede verse el coste temporal de los diferentes paquetes de trabajo. El total se computa en días (jornadas de 8 horas a 20 €/hora), sin contar sábados, domingos y festivos. La documentación no aparece por ser una tarea que se realiza en paralelo. La Figura 4.2 refleja el coste monetario asociado a cada una de las fases del proyecto.

4.3.2. Herramientas

Se han empleado herramientas gratuitas siempre que ha sido posible para minimizar el coste del proyecto. En la Tabla 4.2 se recoge el coste monetario de los recursos *software* y *hardware*.

Fase	Duración (días)
Revisión del estado del arte	21
Especificación de los requisitos	11
Diseño	8
Diseño + Implementación	23
Implementación	15
Evaluación y pruebas	13
Total	91

Tabla 4.1: Coste temporal del proyecto.

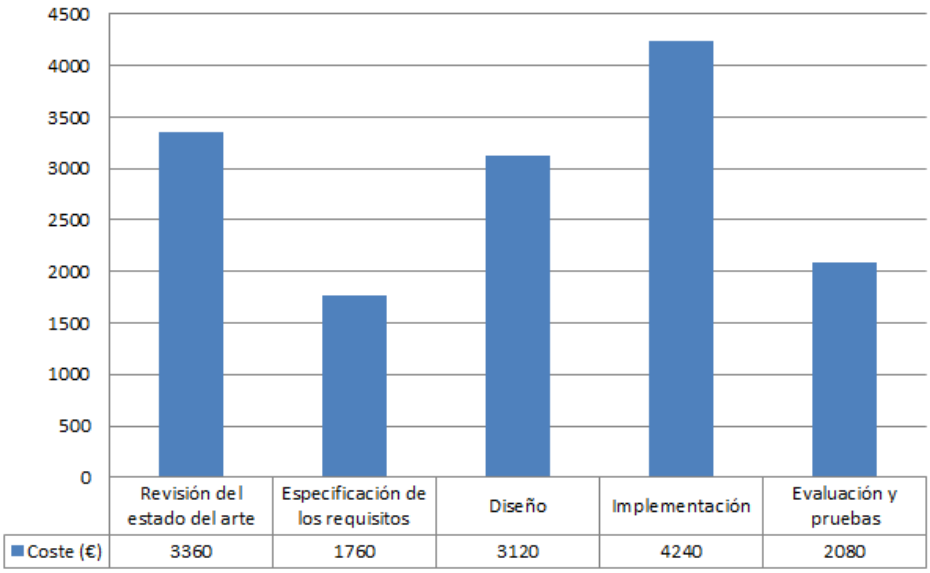


Figura 4.2: Coste monetario asociado a cada fase del proyecto.

Recurso	Vida media	Coste unitario
Ordenador personal	36 meses	1600 €
Sistema operativo	-	0 €
Entorno de desarrollo	-	0 €
Software complementario y librerías	-	0 €
Paquetes de ofimática	-	0 €

Tabla 4.2: Coste monetario del proyecto.

4.4. Presupuesto

Por último, la Tabla 4.3 registra el presupuesto del proyecto.

Concepto	Cantidad
Recursos humanos	
1 desarrollador × 91 días × 8 h/día × 20 €/h	14560 €
Herramientas	
1 ordenador personal × 1600 €/unidad × 5.5 meses / 36 meses	244.4 €
Sistema operativo	0 €
Entorno de desarrollo	0 €
Software complementario y librerías	0 €
Paquetes de ofimática	0 €
Total	14804.4 €

Tabla 4.3: Presupuesto.

Capítulo 5

Diseño

A continuación, se establecerá el diseño de la aplicación en dos pasos: en primer lugar y con un mayor nivel de abstracción, se procederá al diseño de la arquitectura de forma que quede definido el comportamiento global del sistema; en segundo lugar, se profundizará en su estructura a través de los diagramas de clases en el apartado de diseño de los datos.

5.1. Diseño de la arquitectura

Tal y como se ha especificado en el Capítulo 3, la arquitectura escogida es de tipo cliente-servidor. La herramienta constará de una única aplicación que podrá lanzarse indistintamente con uno u otro rol, como puede apreciarse en el diagrama de la Figura 5.1, que describe de forma esquemática las funciones de cada una de las partes.

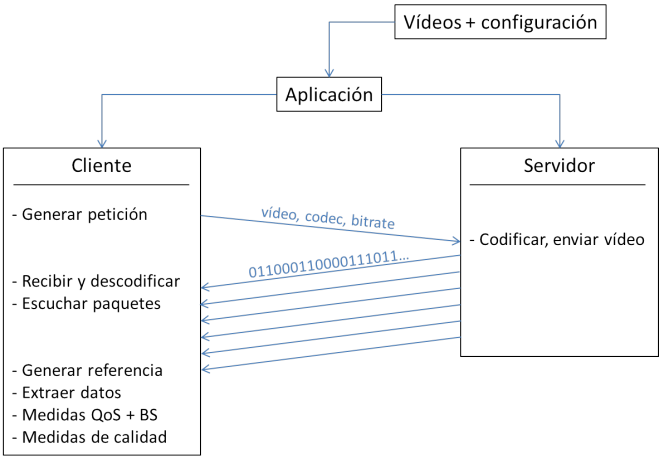


Figura 5.1: Arquitectura cliente-servidor.

5.1.1. Servidor

Como se ha visto en la figura anterior, el servidor tiene tan solo dos funciones básicas:

- Codificar las secuencias de vídeo de prueba según los parámetros que establece el cliente.
- Enviar el vídeo codificado a través de la red.

Así pues, se requiere un servidor RTSP flexible y máximamente configurable que, de alguna manera, permita escoger el codificador, el *bitrate* y el *framerate* de forma dinámica. Esto puede hacerse mediante una librería basada en *GStreamer* llamada *GStreamer RTSP server* [44]. Dicha librería permite crear servidores RTSP a medida: para cada medio mapeado en una determinada URL, puede establecerse el codificador utilizado, así como el *bitrate* y el *framerate*.

No obstante, esta solución todavía no aporta la flexibilidad necesaria: los parámetros *bitrate* y *framerate* no pueden ser modificados de forma dinámica. Por ello, se opta por un servidor XML-RPC que será lanzado en modo *standalone* a la espera de peticiones por parte de algún cliente.

La conexión se inicia con la solicitud por parte del cliente de un determinado par *bitrate-framerate* mediante una llamada a procedimiento remoto. Dicho procedimiento creará un nuevo proceso que ejecutará un servidor RTSP capaz de servir todas las secuencias de vídeo con los cuatro codificadores escogidos (véase la Sección 3.2.1) y al *bitrate-framerate* solicitado (si no existe uno ejecutándose en ese momento con esas características).

En la Figura 5.2 puede apreciarse un diagrama de secuencia del funcionamiento del servidor descrito. El servidor XML-RPC, que escucha en el puerto 8000, recibe una petición de un cliente. Al no haber un servidor RTSP lanzado con esas características,

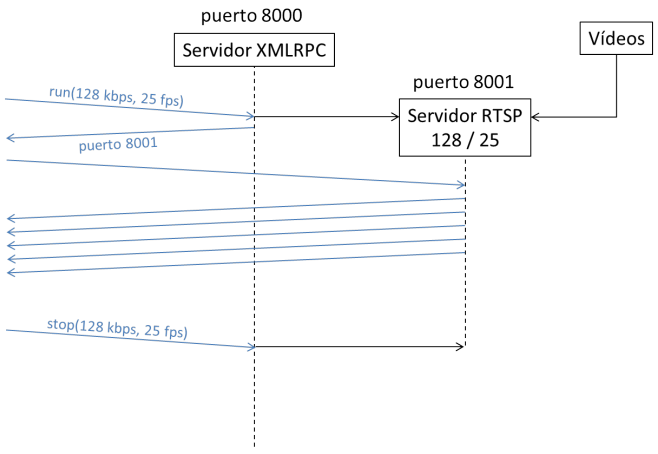


Figura 5.2: Diagrama de secuencia del servidor para un cliente.

ejecuta un nuevo proceso que permanece a la escucha en el puerto 8001, y devuelve al cliente la información de este puerto. En este punto, el cliente ya está en condiciones de realizar una conexión convencional con el servidor RTSP del puerto 8001 y, una vez recibido el vídeo, envía una petición de parada de dicho proceso. Como en este caso no existen más clientes conectados al mismo, el proceso principal (el servidor XML-RPC) mata al anterior.

En el ejemplo de la Figura 5.3, puede verse lo que ocurre cuando en la secuencia anterior un nuevo cliente realiza una petición con un par *bitrate-framerate* distinto: simplemente se lanza otro servidor RTSP con esas características en otro puerto y se informa a este nuevo cliente, sin interferir con la transmisión del primero.

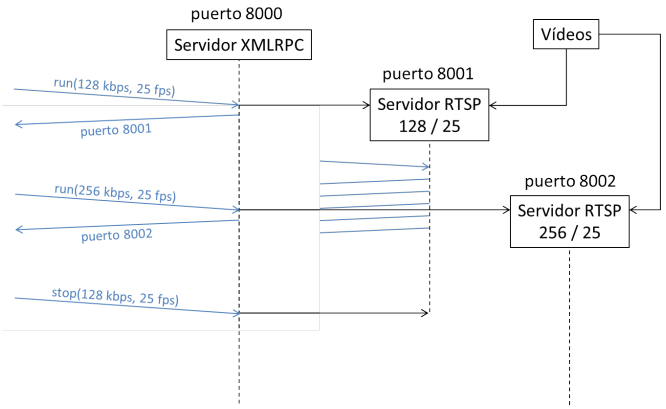


Figura 5.3: Diagrama de secuencia del servidor para dos clientes.

5.1.2. Cliente

El cliente, por su parte, consta de las siguientes funciones, tal y como describe la Figura 5.1:

- Solicitar al servidor un par *bitrate-framerate*.
- Recibir el vídeo y guardarlo en formato codificado y descodificado.
- Al mismo tiempo, escuchar en la interfaz de red correspondiente para guardar la traza de paquetes de la sesión.
- Generar la referencia de vídeo.
- Extraer datos de toda la información recogida.
- Calcular las medidas de QoS y *bitstream*.
- Calcular las medidas de calidad de vídeo.

En la Figura 5.4 se describe un diagrama de secuencia con el esbozo de estos pasos (nótese que el *sniffer* de red se lanza como un hilo paralelo al principal).

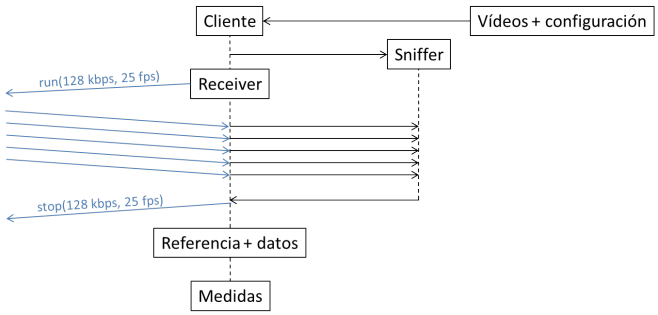


Figura 5.4: Diagrama de secuencia del cliente.

5.2. Diseño de los datos

Se definen los siguientes módulos, cuyas relaciones vienen representadas en la Figura 5.5:

`core` Núcleo de la aplicación. Reúne las clases que implementan las funcionalidades básicas tanto del cliente como del servidor.

`config` Módulo auxiliar con constantes y funciones comunes.

`sniffer` Agrupa las clases relacionadas con la escucha de paquetes y el procesamiento de los mismos para la extracción de informacion a nivel de red.

`gstreamer` Agrupa las funcionalidades de procesado de vídeo: todas aquellas clases que utilizan las librerías de *GStreamer*.

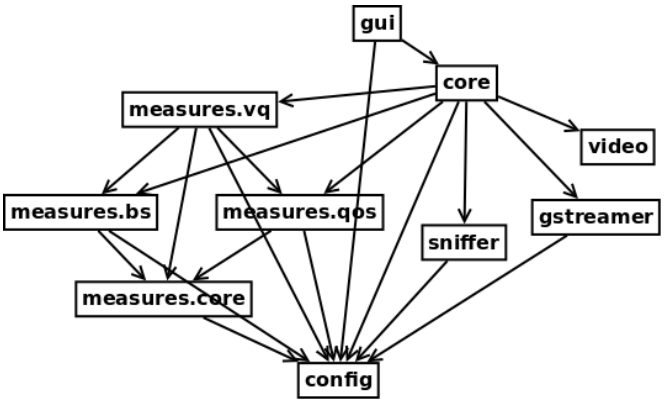


Figura 5.5: Relación entre módulos.

`video` Define objetos para contener vídeo codificado y descodificado.

`gui` Interfaz gráfica de usuario.

`measures.core` Núcleo del paquete de medidas (`measures`). Reúne las funcionalidades comunes a todo el subsistema de medidas.

`measures.qos` Agrupa las clases relacionadas con las medidas de QoS.

`measures.bs` Agrupa las clases relacionadas con las medidas a nivel de *bitstream*.

`measures.vq` Agrupa las clases relacionadas con las medidas de calidad de vídeo.

5.2.1. Módulo `core`

Contiene las clases `VT`, `Server` y `Client` (véase el diagrama de la Figura 5.6).

`VT` Superclase que agrupa funcionalidades compartidas por el cliente y el servidor.

Atributos

`videos` Lista de vídeos disponibles.

`path` Ruta hacia los vídeos.

Métodos

`__init__()` Método inicializador que procesa el archivo de configuración que contiene la ubicación de los vídeos.

`run()` Método abstracto que deberán implementar las subclases.

`parseConf()` Procesador de archivos de configuración que devuelve una lista de opciones.

`Server`
Clase principal del servidor.

Atributos

`servers` Conjunto de servidores RTSP activos (con algún cliente conectado).

`port` Siguiente puerto para un futuro servidor RTSP (va incrementandose de uno en uno).

`exports` Lista de métodos remotos exportados por el servidor XML-RPC: `run()` y `stop()`.

Métodos

`__init__()` Método inicializador.

`run()` Lanza un subproceso para un servidor RTSP con un par *bitrate-framerate* dado (si no está ya en ejecución) y devuelve el puerto de dicho servidor.

`stop()` Detiene el servidor RTSP identificado con el par *bitrate-framerate* (si no hay más clientes conectados).

`__freePort()` Cuando un nuevo servidor RTSP es invocado, comprueba si

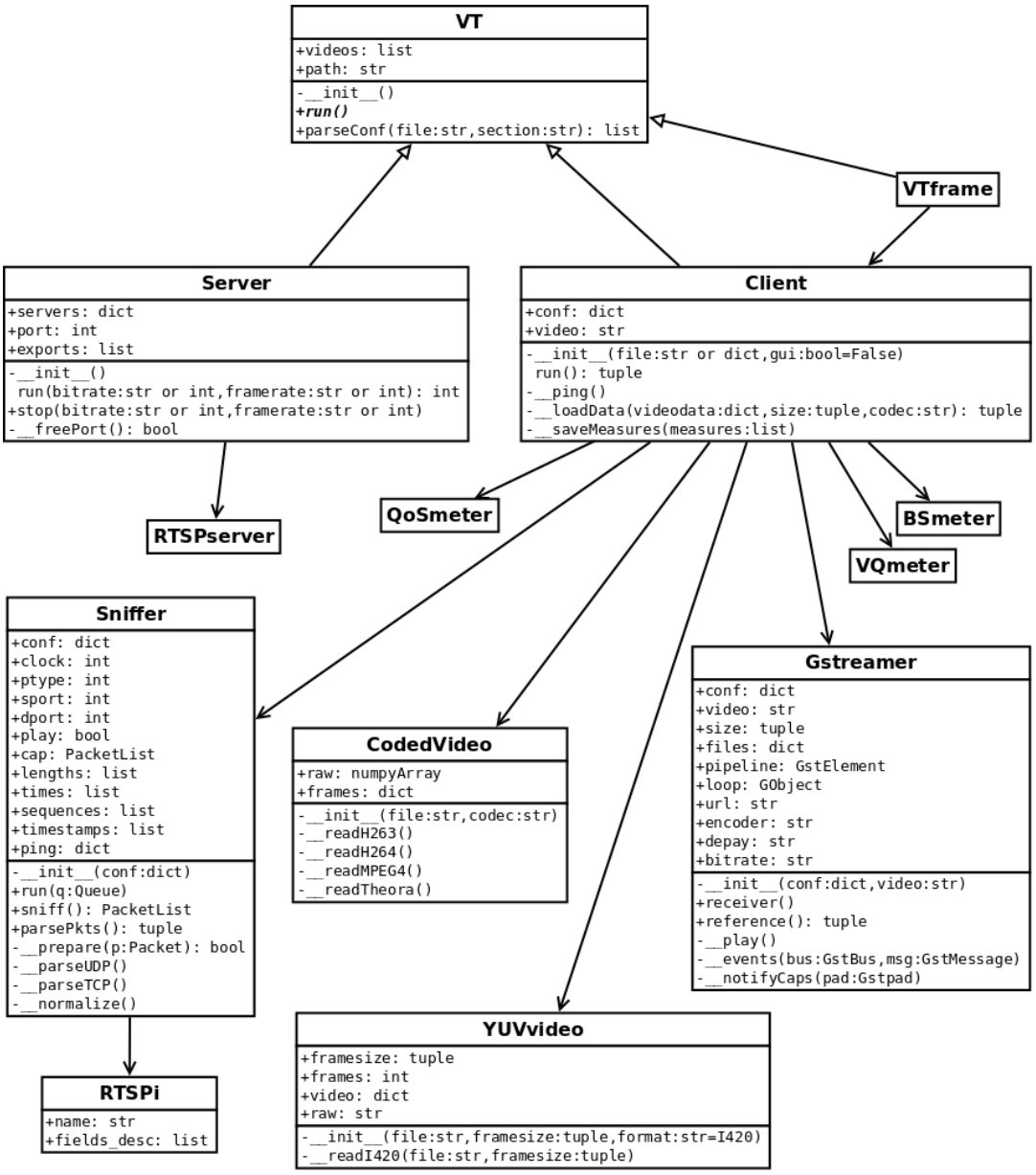


Figura 5.6: Diagrama de clases de la aplicación.

el puerto actual está libre (no existe ningún servicio escuchando en el mismo). En caso afirmativo, devuelve verdadero; falso en caso contrario.

`Client`
: Clase principal del cliente.

Atributos

`conf` Contenido procesado del archivo de configuración para el cliente.

`video` Vídeo seleccionado.

Métodos

`__init__()` Método inicializador que procesa el archivo de configuración del cliente.

`run()` Inicia el cliente y realiza todas las operaciones descritas en la Sección 5.1.2. Devuelve una tupla con la lista de medidas calculadas y la ruta hasta los archivos generados durante el proceso.

`__ping()` Realiza un *ping* al servidor para facilitar el cálculo posterior de la latencia.

`__loadData()` Carga la información de vídeo codificada y descodificada en objetos manejables (véase la Sección 5.2.4). Devuelve una tupla que contiene una lista de objetos de vídeo codificado y otra de objetos de vídeo descodificado.

`__saveMeasures()` Guarda una lista de medidas calculadas en disco para su posterior análisis.

5.2.2. Módulo `sniffer`

Contiene las clases `RTSPi` y `Sniffer` (véase el diagrama de la Figura 5.6).

`RTSPi`
: Clase derivada de la clase `Packet` de *Scapy* que define un descodificador de paquetes de tipo *RTSP interleaved*. Estos paquetes encapsulan los paquetes RTP cuando el flujo de vídeo va sobre TCP [50].

Atributos

`name` Nombre del descodificador.

`fields_desc` Lista de campos.

`Sniffer`
: *Sniffer* de red y procesador de paquetes.

Atributos

`conf` Opciones de configuración.

`clock` Frecuencia de reloj del servidor [38].

`ptype` *Payload type* o código del tipo de medio que transporta RTP [38].

`sport` Puerto de origen (servidor).

`dport` Puerto de destino (cliente).

`play` *Flag* que indica si durante el procesamiento de los paquetes se ha hallado uno de PLAY [50].

`cap` Captura de red.

`lengths` Lista de longitudes de los paquetes RTP hallados.

`times` Lista de tiempos de llegada de los paquetes RTP hallados.

`sequences` Lista de números de secuencia de los paquetes RTP hallados [38].

`timestamps` Lista de marcas de tiempo de los paquetes RTP hallados [38].

`ping` Información del *ping* realizado.

Métodos

`__init__()` Método inicializador.

`run()` Inicia la captura de red y la guarda en disco.

`sniff()` Método de escucha de paquetes. Termina cuando encuentra un paquete de TEARDOWN [50] y devuelve una lista de paquetes.

`parsePkts()` Procesador de los paquetes capturados. Extrae y devuelve los atributos `lengths`, `times`, `sequences`, `timestamps` y `ping`.

`__prepare()` Preprocesador de los paquetes capturados. Extrae del comienzo de la sesión RTSP los atributos `ping`, `ptype` y `clock`. Devuelve verdadero cuando se ha encontrado un paquete RTSP de PLAY (atributo `play` con valor verdadero), el puerto de origen (`sport`) y el puerto de destino (`dport`).

`__parseUDP()` Procesador de paquetes RTP sobre UDP.

`__parseTCP()` Procesador de paquetes RTP sobre TCP.

`__normalize()` Normaliza los atributos `times`, `sequences` y `timestamps` (resta el valor del primer elemento a todos).

5.2.3. Módulo `gstreamer`

Contiene las clases `RTSPserver`, y `Gstreamer` (véase el diagrama de la Figura 5.6).

`RTSPserver`

Clase para el servidor RTSP. En la versión utilizada en este proyecto de la librería *GStreamer RTSP server*, no se incluye soporte completo para el lenguaje Python. Queda reflejada en el diseño esta clase para una futura implementación y, mientras tanto, se salvará esta circunstancia de otra forma en el Capítulo 6.

`Gstreamer`
: Manejador de la librería *GStreamer* para el cliente para el procesado del vídeo.

Atributos

`conf` Opciones de configuración.

`video` Ruta completa al vídeo seleccionado.

`size` Resolución del vídeo: tupla ancho-alto.

`files` Rutas completas a los vídeos procesados: original codificado y descodificado; el producido en modo local, sin sufrir el proceso de transmisión) codificado y descodificado, y el recibido codificado y descodificado.

`pipeline` Conducto de *GStreamer*.

`loop` Bucle de *GObject*.

`url` URL del recurso remoto.

`encoder` Codificador seleccionado.

`depay` Desempaquetador RTP para el codificador seleccionado.

`bitrate` *Bitrate* seleccionado.

Métodos

`__init__()` Método inicializador.

`receiver()` Conecta con el servidor RTSP y recibe el vídeo seleccionado.

`reference()` Crea los vídeos de referencia y devuelve los atributos `files` y `size`.

`__play()` Adjunta el manejador de eventos e inicia la ejecución del bucle.

`__events()` Manejador de eventos.

`__notifyCaps()` Extrae la resolución del vídeo y la guarda en el atributo `size` y en disco.

5.2.4. Módulo `video`

Contiene las clases `YUVvideo` y `CodedVideo` (véase el diagrama de la Figura 5.6).

`YUVvideo`
: Lector de archivos YUV.

Atributos

`framesize` Resolución del vídeo.

`frames` Número de fotogramas del vídeo.

`video` Contiene las tres componentes por separado (Y, U, V) y, cada componente, contiene una lista de fotogramas.

`raw` El vídeo en formato binario.

Métodos

`__init__()` Método inicializador que lanza el método lector pertinente en función del formato.

`__readI420()` Lector de archivos I420.

`CodedVideo`

Lector de archivos codificados.

Atributos

`raw` El vídeo en formato binario.

`frames` Guarda una lista de tipos y una lista de tamaños de fotograma.

Métodos

`__init__()` Método inicializador que lanza el método lector pertinente en función del formato.

`__readH263()` Lector de archivos H.263.

`__readH264()` Lector de archivos H.264.

`__MPEG4()` Lector de archivos MPEG-4 parte 2.

`__Theora()` Lector de archivos Theora.

5.2.5. Módulo `gui`

La interfaz gráfica se ha diseñado haciendo uso del editor visual *wxGlade* [42], que genera el código asociado automáticamente, por lo que no se incluye ninguna descripción aquí.

5.2.6. Módulo `measures.core`

El subsistema de medidas ha sido diseñado para que quede lo más aislado posible del resto de la aplicación: el objetivo es facilitar la implementación de nuevas medidas y que este subsistema sea accedido siempre de la misma manera, de forma que la adición o substracción de una medida no provoque cambios en el resto del programa.

Con este propósito, se ha definido una clase `Measure` y otra `Meter`, de las que derivan los medidores y las medidas de cada tipo concreto (de QoS, *bitstream* y calidad de vídeo). La implementación de una nueva medida hereda de su tipo correspondiente y se registra en el medidor asociado (véase la Figura 5.7). De esta forma, el núcleo de la aplicación interactúa únicamente con los medidores, tal y como puede verse en el diagrama de la Figura 5.6, de forma totalmente transparente y sin importar qué o cuántas medidas hay registradas en ellos.

`Meter`

Medidor genérico. Contiene las funcionalidades comunes a sus subclases.

Atributos

`measures` Lista de medidas (instancias) registradas en el medidor.

Métodos

`__init__()` Método inicializador.

`run()` Por cada instancia registrada, lanza su método correspondiente de cálculo. Devuelve la lista de medidas calculadas.

`Measure`

Medida genérica. Contiene las funcionalidades comunes a sus subclases.

Atributos

`data` Diccionario de resultados.

Métodos

`__init__()` Método inicializador.

`calculate()` Método abstracto para el cálculo de la medida. Deberá ser implementado por cada medida específica.

`__max()` Devuelve el punto máximo de una función.

`__min()` Devuelve el punto mínimo de una función.

`__mean()` Devuelve el valor medio de una función.

`graph()` Para medidas multivalor, añade al atributo `data` los ejes X e Y para su posterior representación, el punto máximo, el punto mínimo y el valor medio.

5.2.7. Módulo `measures.qos`

Contiene las clases asociadas a las medidas de QoS: `QoSmeter`, `QoSmeasure`, `Latency`, `Delta`, `Jitter`, `Skew`, `Bandwidth`, `PacketLossRate` y `PacketLossDist` (véase el diagrama de la Figura 5.7).

`QoSmeter`

Medidor de QoS.

Métodos

`__init__()` Método inicializador. Registra las medidas de QoS seleccionadas en el archivo de configuración del cliente.

`QoSmeasure`

Superclase que agrupa funcionalidades compartidas por las medidas de QoS.

Atributos

`lengths` Lista de longitudes de los paquetes RTP hallados.

`times` Lista de tiempos de llegada de los paquetes RTP hallados.

`sequences` Lista de números de secuencia de los paquetes RTP hallados.

`timestamps` Lista de marcas de tiempo de los paquetes RTP hallados.

`ping` Información del *ping* realizado.

Métodos

`__init__()` Método inicializador.

`Latency`

Retardo de extremo a extremo.

Métodos

`__init__()` Método inicializador.

`calculate()` Método para el cálculo de esta medida. Devuelve el atributo `data` heredado de la clase `Measure` (véase la Sección 5.2.6).

`Delta`

Intervalo entre dos paquetes consecutivos.

Métodos

`__init__()` Método inicializador.

`calculate()` Método para el cálculo de esta medida. Devuelve el atributo `data` heredado de la clase `Measure` (véase la Sección 5.2.6).

`Jitter`

Desviación de cada paquete con respecto a la latencia.

Métodos

`__init__()` Método inicializador.

`calculate()` Método para el cálculo de esta medida. Devuelve el atributo `data` heredado de la clase `Measure` (véase la Sección 5.2.6).

`Skew`

Desviación temporal de cada paquete con respecto a su marca de tiempo RTP.

Métodos

`__init__()` Método inicializador.

`calculate()` Método para el cálculo de esta medida. Devuelve el atributo `data` heredado de la clase `Measure` (véase la Sección 5.2.6).

`Bandwidth`
Ancho de banda.

Métodos

`__init__()` Método inicializador.

`calculate()` Método para el cálculo de esta medida. Devuelve el atributo `data` heredado de la clase `Measure` (véase la Sección 5.2.6).

`PacketLossRate`
Tasa de pérdidas.

Métodos

`__init__()` Método inicializador.

`calculate()` Método para el cálculo de esta medida. Devuelve el atributo `data` heredado de la clase `Measure` (véase la Sección 5.2.6).

`PacketLossDist`
Distribución de pérdidas.

Métodos

`__init__()` Método inicializador.

`calculate()` Método para el cálculo de esta medida. Devuelve el atributo `data` heredado de la clase `Measure` (véase la Sección 5.2.6).

5.2.8. Módulo `measures.bs`

Contiene las clases asociadas a las medidas de *bitstream*: `BSmeter`, `BSmeasure`, `StreamEye`, `RefStreamEye`, `GOP` y `IFrameLossRate` (véase el diagrama de la Figura 5.7).

`BSmeter`
Medidor de *bitstream*.

Métodos

`__init__()` Método inicializador. Registra las medidas de *bitstream* seleccionadas en el archivo de configuración del cliente.

`BSmeasure`
Superclase que agrupa funcionalidades compartidas por las medidas de *bitstream*.

Atributos

`coded` Información del vídeo recibido sin descodificar (longitud de los fotogramas y tipo).

`codedref` Información del vídeo codificado generado localmente —que no ha sufrido las consecuencias de la transmisión por la red— (longitud de los fotogramas y tipo).

Métodos

`__init__()` Método inicializador.

`StreamEye`

Visualización del flujo codificado. Muestra el tamaño de los fotogramas y su tipo.

Atributos

`v` Selecciona el vídeo a mostrar: el recibido o el generado localmente (por defecto, el primero).

Métodos

`__init__()` Método inicializador.

`calculate()` Método para el cálculo de esta medida. Devuelve el atributo `data` heredado de la clase `Measure` (véase la Sección 5.2.6).

`RefStreamEye`

Hereda de la clase anterior. Modifica su atributo `v` para seleccionar el vídeo generado localmente.

Métodos

`__init__()` Método inicializador.

`GOP`

Estimación del tamaño del GOP del vídeo recibido.

Métodos

`__init__()` Método inicializador.

`calculate()` Método para el cálculo de esta medida. Devuelve el atributo `data` heredado de la clase `Measure` (véase la Sección 5.2.6).

`IFrameLossRate`

Tasa de pérdidas de fotogramas I.

Métodos

`__init__()` Método inicializador.

`calculate()` Método para el cálculo de esta medida. Devuelve el atributo `data` heredado de la clase `Measure` (véase la Sección 5.2.6).

5.2.9. Módulo `measures.vq`

Contiene las clases asociadas a las medidas de calidad de vídeo: `VQmeter`, `VQmeasure`, `PSNR`, `SSIM`, `G1070`, `PSNRtoMOS` y `MIV` (véase el diagrama de la Figura 5.7).

`VQmeter`

Medidor de calidad de vídeo.

Métodos

`__init__()` Método inicializador. Registra las medidas de calidad de vídeo seleccionadas en el archivo de configuración del cliente.

`VQmeasure`

Superclase que agrupa funcionalidades compartidas por las medidas de calidad de vídeo.

Atributos

`conf` Parámetros del vídeo: codificador, *bitrate*, *framerate* y resolución.

`rawdata` Archivos de vídeo descodificado (véase el método `__loadData()` en la Sección 5.2.6).

`yuv` Vídeo recibido descodificado.

`yuvref` Vídeo original descodificado.

`codecdata` Información a nivel de *bitstream*.

`packetdata` Información a nivel de paquete.

Métodos

`__init__()` Método inicializador.

`getQoSm()` Si una medida de calidad de vídeo requiere para su cálculo medidas de QoS, puede invocar este método para obtenerlas.

`getBSm()` Si una medida de calidad de vídeo requiere para su cálculo medidas de *bitstream*, puede invocar este método para obtenerlas.

`PSNR`

Cálculo de la PSNR para la componente Y (luminancia).

Métodos

`__init__()` Método inicializador.

`calculate()` Método para el cálculo de esta medida. Devuelve el atributo `data` heredado de la clase `Measure` (véase la Sección 5.2.6).

`SSIM`

Cálculo del índice SSIM para la componente Y.

Métodos

`__init__()` Método inicializador.

`calculate()` Método para el cálculo de esta medida. Devuelve el atributo `data` heredado de la clase `Measure` (véase la Sección 5.2.6).

`__array2cv()` Convierte entre estructuras de datos de los fotogramas.

`__SSIM()` Calcula el índice SSIM de dos imágenes y devuelve su valor.

`G1070`

Estimación de la calidad de vídeo según la recomendación ITU-T G.1070 [18].

Métodos

`__init__()` Método inicializador.

`calculate()` Método para el cálculo de esta medida. Devuelve el atributo `data` heredado de la clase `Measure` (véase la Sección 5.2.6).

`PSNRtoMOS`

Mapeo de la PSNR a la escala MOS (véase la Tabla 1.2).

Métodos

`__init__()` Método inicializador.

`calculate()` Método para el cálculo de esta medida. Devuelve el atributo `data` heredado de la clase `Measure` (véase la Sección 5.2.6).

`MIV`

Medida del porcentaje de fotogramas con un MOS peor que el de referencia en un intervalo deslizante.

Atributos

`interval` Número de fotogramas del intervalo deslizante sobre el que se hacen los cálculos.

Métodos

`__init__()` Método inicializador.

`calculate()` Método para el cálculo de esta medida. Devuelve el atributo `data` heredado de la clase `Measure` (véase la Sección 5.2.6).

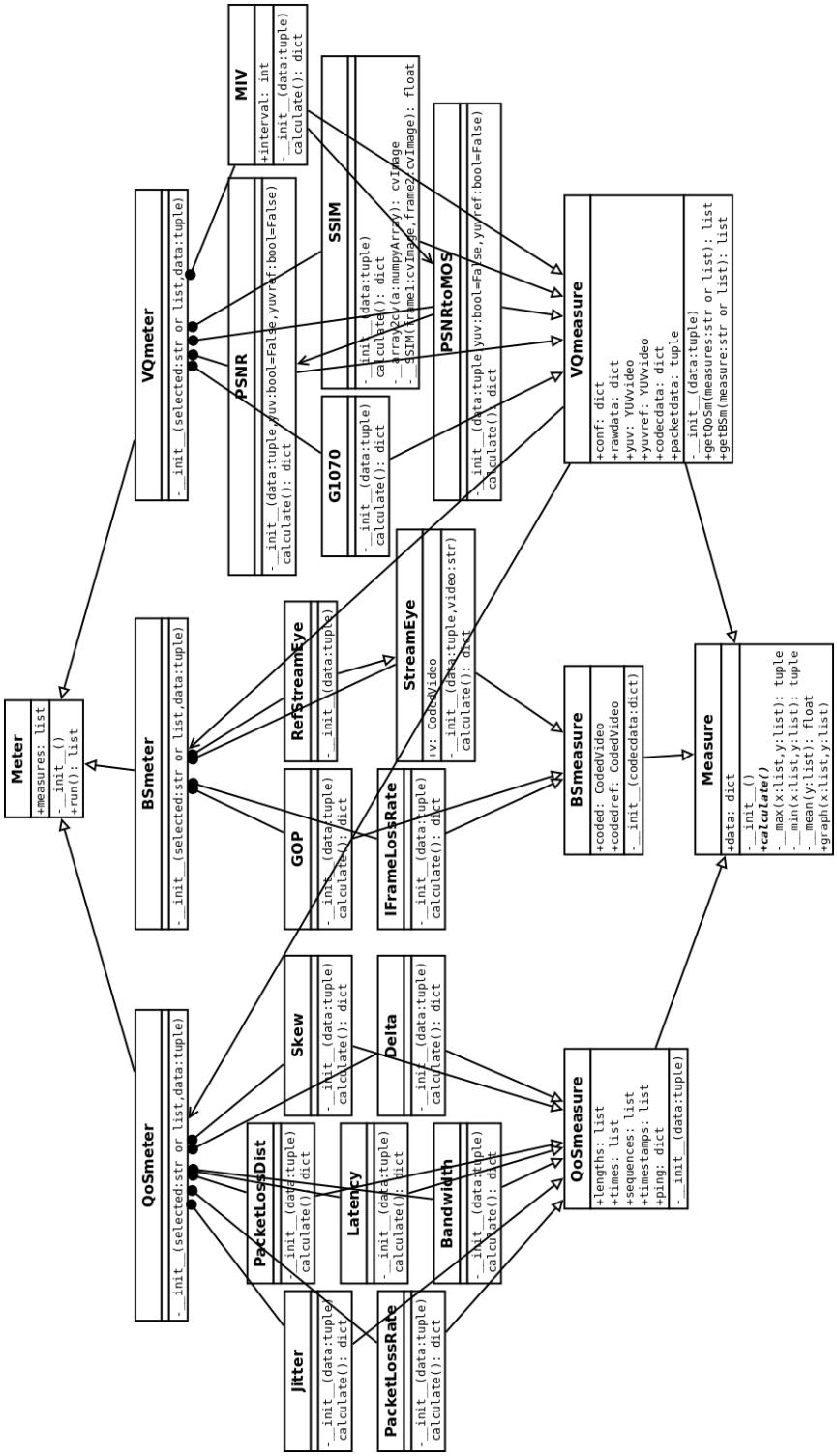


Figura 5.7: Detalle del diagrama de clases del subsistema de medidas.

Capítulo 6

Implementación

El presente capítulo tiene por objetivo detallar las soluciones adoptadas a la hora de acometer la implementación del sistema planteado en el capítulo anterior. Los puntos a tratar son el servidor, el cliente, el *sniffer* de red, el procesamiento de vídeo mediante las librerías de *GStreamer* (para la recepción y generación de referencias), los lectores de archivos de vídeo (codificado y descodificado), el subsistema de medidas y, por último, la implementación del *script* principal de la aplicación.

6.1. Servidor

Tal como se ha descrito en la fase de diseño, el servidor debe soportar llamadas XML-RPC para lanzar servidores RTSP con configuraciones concretas bajo demanda. Aprovechando la herencia múltiple de Python, existe una solución simple gracias a la clase `SimpleXMLRPCServer.SimpleXMLRPCServer` perteneciente a la librería estándar del lenguaje: proporciona un servidor XML-RPC con funcionalidades básicas. Por tanto, añadiendo esta herencia, la clase `Server` (véase la Sección 5.2.1) automáticamente es capaz de ofrecer todos sus métodos de manera remota. No obstante, por motivos de seguridad, es conveniente rehacer un método interno de la clase `SimpleXMLRPCServer` llamado `_dispatch()` cuyo cometido es cursar las solicitudes remotas recibidas. En el Listado 6.1 se muestra la implementación de dicha función, según la cual el cliente solo puede emplear los métodos registrados en el atributo `exports`.

Listado 6.1: Método `_dispatch()` del servidor.

```
1  def _dispatch(self, method, params):
2      if method in self.exports:
3          func = getattr(self, method)
4          return func(*params)
5      else:
6          raise Exception('method "%s" is not supported' % method)
```

Para llevar la contabilidad de los servidores RTSP activos y los clientes conectados, se hace uso del atributo `servers`, el cual tiene la siguiente forma:

```
self.servers = { '128 kbps - 25 fps':{}, '256 kbps - 25 fps':{} ... }
```

Es decir, cada llave de ese diccionario se corresponde con un identificador de un servidor RTSP activo distinto, y su contenido también es un diccionario como el que sigue:

```
self.servers['128 kbps - 25 fps'] = { 'clients':5, 'server':0 }
```

Como se desprende del ejemplo anterior, dentro de cada identificador se halla una llave que guarda la cuenta del número de clientes conectados a ese servidor RTSP y otra llave con el identificador del subproceso en cuestión. Los procedimientos a seguir por los métodos remotos `run()` y `stop()` que se ofrecen al cliente (véase la Sección 5.2.1) se ilustran en la Figura 6.1.

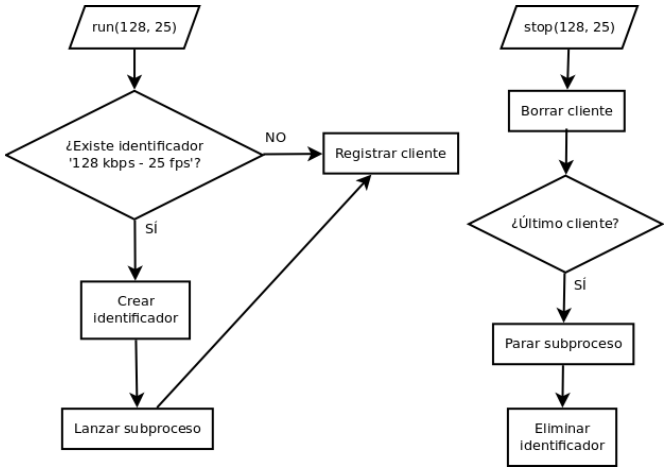


Figura 6.1: Métodos remotos `run()` y `stop()`.

6.1.1. Servidor RTSP

La implementación del servidor RTSP utilizando la librería *GStreamer RTSP server* no resulta trivial dado que esta librería no incluye soporte completo para el lenguaje Python, tal y como se ha comentado anteriormente (véase la Sección 5.2.3). La solución adoptada pasa por programar y compilar una aplicación en C e incluirla dentro del paquete `VideoTester`. Este binario se ejecuta como el subproceso descrito anteriormente y es el encargado de mediar con dichas librerías. Recibe como parámetros (en este orden) el puerto donde se lanzará (opción `-p`), el *framerate* (opción `-f`), el *bitrate* (opción `-b`) y una lista de vídeos (opción `-v`).

El código fuente creado se incluye en el Listado 6.2. En él puede verse cómo, para cada vídeo y para cada codificador soportado, se construye una tubería de *GStreamer* con esta forma:

```
filesrc location=<path_to_video> ! decodebin2 ! videorate ! video/x-raw-yuv,
    framerate=<framerate>/1 ! <codec> bitrate=<bitrate> ! rtppay name=pay0
```

Es decir, se impone tanto el *bitrate* como el *framerate*, y el vídeo resultante codificado es encapsulado en paquetes RTP. Los vídeos quedan accesibles en el servidor RTSP bajo las siguientes URL:

- `rtsp://<RTSPserver>:<port>/video0.h263`
- `rtsp://<RTSPserver>:<port>/video1.h263`
- ...
- `rtsp://<RTSPserver>:<port>/video0.theora`
- `rtsp://<RTSPserver>:<port>/video1.theora`
- etc.

La compilación del servidor se realiza mediante la siguiente orden:

```
$ gcc server.c -o server $(pkg-config --libs --cflags gstreamer-0.10 gst-rtsp-server
    -0.10)
```

Listado 6.2: Implementación del servidor RTSP.

```
#include <gst/gst.h>
#include <gst/rtsp-server/rtsp-server.h>
#include <string.h>

int main (int argc, char *argv[])
{
    GMainLoop *loop;
    GstRTSPServer *server;
    GstRTSPMediaMapping *mapping;
    GstRTSPMediaFactory *factory;
    gchar *pipe, *url;
    int i, j, cont;

    gst_init (&argc, &argv);
    if (argc < 7 || strcmp(argv[1], "-p") != 0 || strcmp(argv[3], "-f") != 0 ||
        strcmp(argv[5], "-b") != 0 || strcmp(argv[7], "-v") != 0) {
        g_print ("usage: %s -p <port> -f <framerate> -b <bitrate(kbps)> -v <video1> <
            video2> ...\n", argv[0]);
        return -1;
    }
    loop = g_main_loop_new (NULL, FALSE);
```

```
20      /* create a server instance */
21      server = gst_rtsp_server_new ();
22      gst_rtsp_server_set_service (server, argv[2]);
23      /* map media */
24      cont = 0;
25      for (i = 8; i < argc; i++) {
26          for (j = 0; j <= 3; j++) {
27              switch (j){
28                  case 0: /* h263 */
29                      pipe = g_strdup_printf("( filesrc location=%s ! decodebin2 !
                            videorate ! video/x-raw-yuv,framerate=%s/1 ! ffenc_h263
                            bitrate=%s000 ! rtph263pay name=pay0 )", argv[i], argv[4],
                            argv[6]);
30                      url = g_strdup_printf("/video%d.h263", cont);
31                      break;
32                  case 1: /* h264 */
33                      pipe = g_strdup_printf("( filesrc location=%s ! decodebin2 !
                            videorate ! video/x-raw-yuv,framerate=%s/1 ! x264enc bitrate=%
                            s ! rtph264pay name=pay0 )", argv[i], argv[4], argv[6]);
34                      url = g_strdup_printf("/video%d.h264", cont);
35                      break;
36                  case 2: /* mpeg4 */
37                      pipe = g_strdup_printf("( filesrc location=%s ! decodebin2 !
                            videorate ! video/x-raw-yuv,framerate=%s/1 ! ffenc_mpeg4
                            bitrate=%s000 ! rtpmp4vpay name=pay0 )", argv[i], argv[4],
                            argv[6]);
38                      url = g_strdup_printf("/video%d.mpeg4", cont);
39                      break;
40                  case 3: /* theora */
41                      pipe = g_strdup_printf("( filesrc location=%s ! decodebin2 !
                            videorate ! video/x-raw-yuv,framerate=%s/1 ! theoraenc bitrate
                            =%s ! rtptheorapay name=pay0 )", argv[i], argv[4], argv[6]);
42                      url = g_strdup_printf("/video%d.theora", cont);
43                      break;
44              }
45              /* get the mapping for this server */
46              mapping = gst_rtsp_server_get_media_mapping (server);
47              /* make a media factory */
48              factory = gst_rtsp_media_factory_new ();
49              gst_rtsp_media_factory_set_launch (factory, pipe);
50              gst_rtsp_media_factory_set_shared (factory, TRUE);
51              gst_rtsp_media_factory_set_eos_shutdown (factory, TRUE);
52              /* attach the test factory to the /test url */
53              gst_rtsp_media_mapping_add_factory (mapping, url, factory);
54              /* don't need the ref to the mapper anymore */
55              g_object_unref (mapping);
56              g_free (url);
57              g_free (pipe);
58          }
59          cont = cont + 1;
60      }
61      /* attach the server to the default maincontext */
62      if (gst_rtsp_server_attach (server, NULL) > 0)
63          /* start serving */
64          g_main_loop_run (loop);
65      return 0;
66  }
```

6.2. Cliente

La secuencia de acciones que implementa el cliente (método local `run()`) es la siguiente:

1. Invocar el método remoto `run()` del servidor.
2. Iniciar el *sniffer* de red como un proceso aparte.
3. Realizar un *ping* al servidor con cuatro peticiones de eco para promediar los tiempos en el cálculo de la latencia.
4. Iniciar el receptor de vídeo.
5. Descargar el vídeo desde el servidor RTSP.
6. Invocar el método remoto `stop()` del servidor.
7. Crear los vídeos de referencia. Aparte del vídeo original (sin ninguna degradación), se debe procesar el vídeo que sufre la degradación del codificador, pero no la del proceso de transmisión. De esta forma, es posible medir el efecto aislado de la red sobre la calidad percibida.
8. Procesar el archivo de captura para obtener los parámetros de red.
9. Procesar los archivos de vídeo para obtener los parámetros de *bitstream* y la información YUV.
10. Ejecutar los medidores.
11. Guardar las medidas obtenidas.

6.3. *Sniffer* de red

La clase `Sniffer` (véase la Sección 5.2.2) consta de dos partes diferenciadas: por un lado, la escucha de paquetes en red y, por otro, el procesado del archivo de captura.

6.3.1. Generación del archivo de captura

El método `run()` está diseñado para ser ejecutado como un proceso paralelo de forma que recoja los paquetes de red al mismo tiempo que el proceso principal recibe el vídeo gracias a las librerías de *GStreamer*. Desde `run()`, se lanza el método `sniff()`, que se ha tomado de la librería *Scapy*. Concretamente, se ha adaptado el código de la función `scapy.sendrecv.sniff()` mediante las siguientes modificaciones:

- Se eliminan los parámetros innecesarios.

- La implementación original tenía una precisión de un segundo para los tiempos de llegada. Esto se ha solucionado generando una nueva marca de tiempo con la función `time.time()` de Python.

- Se ha adaptado para que el bucle infinito de escucha se detenga cuando se reciba un paquete con el mensaje TEARDOWN, que marca el final de un flujo RTSP.

6.3.2. Procesado de la captura

Del procesado de la captura se encarga el método `parsePkts()`, que a su vez invoca un método diferente en función de la capa de transporte sobre la que viajan los paquetes RTP: `__parseUDP()` para el caso de UDP y `__parseTCP()` para TCP.

Ambos casos tienen en común el procedimiento de preprocesado del flujo RTSP, que tiene por objetivo extraer la información que se intercambian los interlocutores antes del comienzo del envío del vídeo. Dicho procedimiento es llevado a cabo por el método `__prepare()`, el cual realiza las siguientes acciones:

1. Recolecta y almacena las peticiones de eco del comienzo de la traza en el atributo `ping`.

2. Busca la información SDP, en concreto,

   a) guarda el tipo de carga RTP (*payload type*) en el atributo `ptype`.

   b) guarda la frecuencia de reloj en el atributo `clock`. El emisor utiliza esta frecuencia para generar los *timestamps* de los paquetes RTP —cuyas unidades son "muestras de reloj"—, por lo que resulta imprescindible para convertirlos posteriormente a segundos.

3. Busca un paquete RTSP donde figuren el puerto del emisor (servidor) y del receptor (cliente), y los almacena en los atributos `sport` y `dport` respectivamente.

4. Busca el paquete con el mensaje PLAY, indicador de que el emisor va a comenzar a enviar el vídeo.

6.3.2.1. UDP

Cuando el protocolo de transporte es UDP, el procesado es sencillo ya que cada paquete UDP contiene un solo paquete RTP. Por tanto, el procedimiento se limita a decodificar la cabecera RTP y extraer la información relevante cada vez que se encuentra un paquete que cumple las condiciones de pertenencia al flujo de vídeo: el protocolo de transporte es UDP, procede de la IP y el puerto del servidor (`sport`), va destinado al puerto del cliente (`dport`) y contiene el tipo de carga descrita por la sesión RTSP (`ptype`).

6.3.2.2. TCP

El caso de TCP es distinto: cada paquete TCP puede llevar más de un paquete RTP encapsulado bajo una cabecera RTSP *interleaved* [50], lo que complica el procesado. Además, los paquetes del flujo de vídeo ya no se distinguen de los paquetes RTSP de señalización a nivel de transporte: proceden del mismo puerto y se dirigen al mismo puerto.

En la solución implementada, se ha optado por concatenar todos los paquetes del flujo de vídeo tras eliminar las cabeceras hasta el nivel de transporte tal y como se ilustra en la Figura 6.2. Para preservar el tiempo de llegada del paquete IP y la longitud del mismo, se recurre a insertar dicha información entre cadenas de control de final de paquete ("ENDOFPACKET"). A pesar de que se trata de TCP y que, en principio, este protocolo asegura la entrega de todos los paquetes, se tienen en cuenta posibles pérdidas que podrían darse en casos extremos a pesar de las retransmisiones. Así, cuando se detecta una pérdida debido a que el número de secuencia TCP no concuerda con los datos recibidos (existen huecos), se inserta la cadena de control "PACKETLOSS".

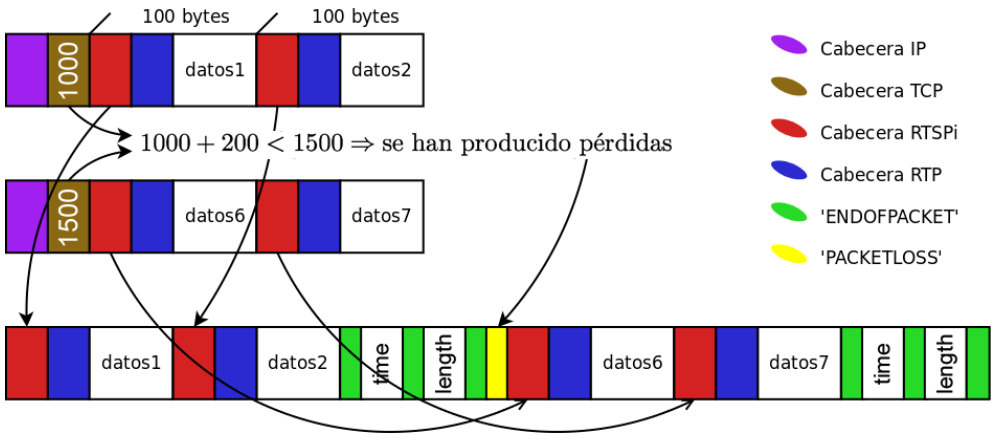


Figura 6.2: Concatenación de paquetes en el procesado TCP.

Una vez obtenido el *stream* de paquetes RTSP *interleaved*, se sigue un procedimiento recursivo para la extracción de parámetros.

6.4. Procesamiento de vídeo con *GStreamer*

El procesamiento de vídeo se concentra en la clase `Gstreamer` (véase la Sección 5.2.3), particularmente en sus dos métodos públicos: `receiver()`, encargado de la recepción del vídeo desde el servidor RTSP, y `reference()`, encargado de generar los vídeos de referencia.

6.4.1. Receptor de vídeo

El método receptor se encarga de conectar con la fuente RTSP, solicitar el vídeo y guardarlo tanto en formato codificado (archivo `00.h263` del ejemplo de la Sección C.3.4) como descodificado (archivo `00.yuv` del ejemplo de la Sección C.3.4). Implementa la tubería que se muestra a continuación y que, a su vez, se corresponde con el diagrama de la Figura 6.3. Como puede apreciarse, el flujo de vídeo recibido se divide en dos: por un lado se archiva en formato codificado y por el otro en formato YUV.

```
1 self.pipeline = parse_launch('rtspsrc name=source ! tee name=t ! queue ! ' + self.
      depay + self.\_\_add + ' ! filesink name=sink1 t. ! queue ! decodebin !
      videorate skip-to-first=True ! video/x-raw-yuv,framerate=' + self.conf['
      framerate'] + '/1 ! filesink name=sink2')
```

El elemento `rtspsrc` permite escoger el protocolo de trasporte (UDP *unicast*/*multicast* o TCP) mediante la propiedad *protocols*. Además, en modo UDP y de forma predeterminada, envía *dummy packets* antes de la transmisión del vídeo para abrir un agujero en el sistema NAT si lo hubiere.

Cabe destacar el elemento `videorate`, que impone un *framerate* determinado a la salida (el configurado con anterioridad). Funciona de la siguiente manera:

- Si el flujo entrante tiene el *framerate* exigido, no hace nada.
- Si el flujo entrante tiene un *framerate* superior al exigido, elimina fotogramas para adecuarlo.
- Si el flujo entrante tiene un *framerate* inferior al exigido, duplica fotogramas para adecuarlo.

Esto garantiza que, a la salida, el *framerate* sea constante e igual al configurado, se produzcan o no pérdidas durante la transmisión. Así, se mitigarán los problemas de alineamiento de fotogramas de los que adolece EvalVid, tal y como se comentaba en la Sección 2.2.1.2.

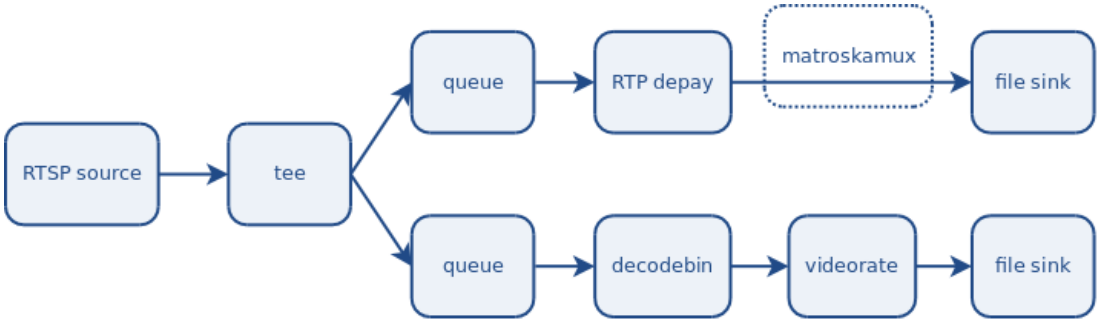


Figura 6.3: Tubería *GStreamer* del receptor de vídeo.

6.4.2. Generador de referencias

El método generador de referencias se encarga de producir tres archivos: *a*) el vídeo original descodificado —es decir, sin degradar— (archivo `00_ref_original.yuv` del ejemplo de la Sección C.3.4), *b*) el vídeo codificado —degradado por el codificador pero que no ha pasado por el proceso de transmisión— (archivo `00_ref.h263` del ejemplo de la Sección C.3.4) y *c*) el vídeo descodificado degradado previamente por el codificador (archivo `00_ref.yuv` del ejemplo de la Sección C.3.4).

Implementa la dos tuberías que se muestran a continuación y que están representadas en el diagrama de la Figura 6.4. La tubería simple produce el vídeo *a*), mientras que la doble produce *b*) y *c*).

```
1  self.pipeline = parse_launch('filesrc name=source ! decodebin ! videorate ! video/x-
       raw-yuv,framerate=' + self.conf['framerate'] + '/1 ! filesink name=sink1')
2
3  self.pipeline = parse_launch('filesrc name=source ! decodebin ! videorate ! video/x-
       raw-yuv,framerate=' + self.conf['framerate'] + '/1 ! ' + self.encoder + '
       bitrate=' + self.bitrate + ' ! tee name=t ! queue' + self.\_\_add + ' ! filesink
        name=sink2 t. ! queue ! decodebin ! filesink name=sink3')
```

En las Figuras 6.3 y 6.4, aparece un elemento difuminado llamado `matroskamux` cuyo origen proviene de la variable `self.__add` presente en el código de las tuberías. Su presencia se debe a que, solamente cuando el codificador es Theora, el vídeo se encapsula en un contenedor Matroska para facilitar el análisis posterior (más información al respecto en el Anexo B).

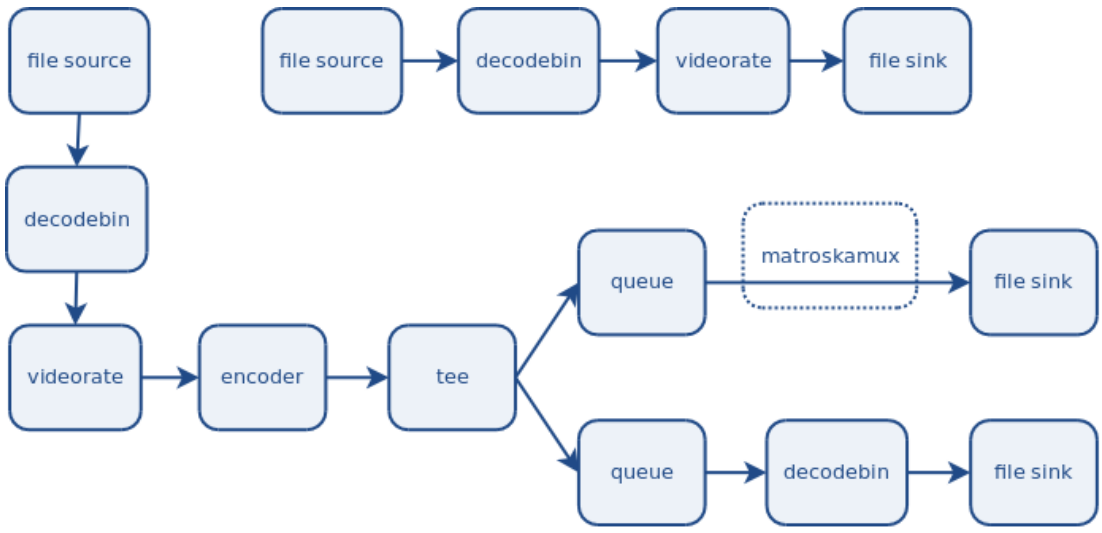


Figura 6.4: Tuberías *GStreamer* del generador de vídeos de referencia.

6.5. Lectores de archivos de vídeo

Para la extracción de información de los archivos de vídeo generados por *GStreamer* (parámetros de *bitstream* e información YUV), se han implementado dos lectores: `YUVvideo` para el vídeo en formato descodificado YUV I420 y `CodedVideo` para formatos los formatos codificados H.263, H.264, MPEG-4 parte 2 y Theora (véase la Sección 5.2.4).

- El lector de archivos YUV extrae las tres componentes (Y, U, V) y las almacena por separado para cada fotograma del vídeo (véase el Anexo A para más detalles).
- El lector de archivos codificados extrae la longitud del fotograma y su tipo (véase el Anexo B para más detalles).

6.6. Subsistema de medidas

En esta sección se recogen las fórmulas y supuestos utilizados en la implementación de las medidas presentes en Video Tester. Para una descripción detallada de las directrices a seguir para la extensión de la herramienta con nuevas medidas, véase el manual de referencia, Sección C.4.

6.6.1. Medidas de QoS

Latencia (clase `Latency`)

El *round-trip time* (RTT) se define como el tiempo transcurrido desde que se envía un paquete hasta que se recibe su confirmación. Por tanto, la latencia es igual, aproximadamente, a la mitad de ese tiempo. Como la aplicación cliente realiza un proceso de *ping* al servidor al comienzo de la comunicación, este retardo extremo a extremo L puede calcularse promediando los valores extraídos de N peticiones de eco:

$$L = \frac{1}{N} \sum_{n=1}^{N} \frac{RTT(n)}{2} \tag{6.1}$$

Tiempo entre paquetes (clase `Delta`)

Sea R_i el tiempo de llegada del paquete i-ésimo, el tiempo entre paquetes (Δ) se calcula de la siguiente forma:

$$\Delta(i) = R_i - R_{i-1} \tag{6.2}$$

Jitter (clase `Jitter`)

Para su cálculo, se emplea el método descrito en [38]. Sea S_i el *timestamp* RTP

del paquete i-ésimo y R_i el tiempo de llegada del mismo (ambos valores en las mismas unidades), se define D para dos paquetes i, j:

$$D(i,j) = (R_j - R_i) - (S_j - S_i) = (R_j - S_j) - (R_i - S_i) \tag{6.3}$$

Entonces, el *jitter* J para un paquete i se calcula como sigue:

$$J(i) = J(i-1) + \frac{|D(i-1,i)| - J(i-1)}{16} \tag{6.4}$$

Desviación (clase `Skew`)

Sea S_i el *timestamp* RTP del paquete i-ésimo y R_i el tiempo de llegada del mismo (ambos valores en las mismas unidades), la desviación σ sigue la siguiente expresión:

$$\sigma(i) = S_i - R_i \tag{6.5}$$

Ancho de banda (clase `Bandwidth`)

Se implementa el ancho de banda instantáneo, es decir, la cantidad recibida de *bytes* en el último segundo. Así, sea T_n el tamaño del paquete enésimo y N el número de paquetes recibidos en el último segundo, el ancho de banda B se calcula de la siguiente manera:

$$B(i) = \sum_{n=i-N}^{i} T_n \tag{6.6}$$

Pérdidas (clase `PacketLossRate`)

Recibidos N paquetes, los números de secuencia I_n son consecutivos, a no ser que existan pérdidas. Por tanto, las pérdidas ($Loss$) pueden calcularse así:

$$Loss = \frac{1}{N} \sum_{n=2}^{N} I_n - (I_{n-1} + 1) \tag{6.7}$$

Distribución de pérdidas (clase `PacketLossDist`)

Conocido el tiempo transcurrido entre la llegada del primer paquete y el último, puede subdividirse en K intervalos de un segundo. Aplicando la ecuación (6.7) a cada intervalo k, se llega a la siguiente expresión:

$$LossDist(k) = \frac{1}{N_k} \sum_{n=2}^{N_k} I_{n,k} - (I_{n-1,k} + 1) \tag{6.8}$$

6.6.2. Medidas de *bitstream*

Vista del flujo (clase `StreamEye`)

Se trata de conseguir una visión general del flujo codificado recibido. Para ello, se crean tres vectores (uno para imágenes I, otro para P y el último para B) cuya longitud es igual al número de fotogramas. Por cada fotograma recibido, su tamaño se inserta en el vector de su tipo correspondiente. Posteriormente, al pintar cada vector de un color distinto, esta "medida" —realmente no es una medida en el sentido estricto del término— permite reconocer de forma visual el tamaño del GOP o si se han producido pérdidas (véase el ejemplo de la Figura 6.5).

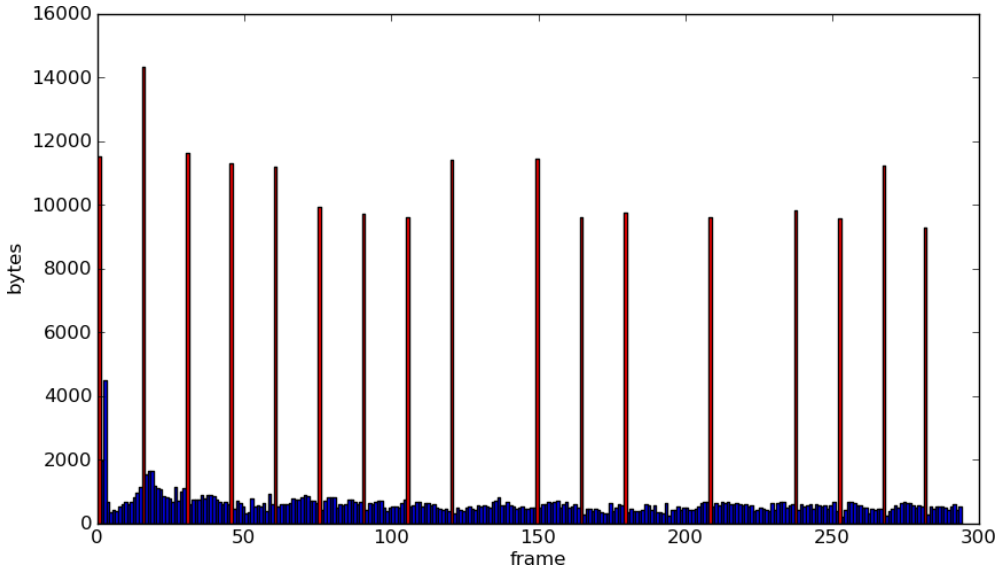


Figura 6.5: Ejemplo de salida de la medida `StreamEye`: en rojo, imágenes I; en azul, imágenes P.

Vista del flujo de referencia (clase `RefStreamEye`)

Misma concepción que la medida anterior, pero aplicada al vídeo de referencia.

Tamaño del GOP (clase `GOP`)

Podría calcularse de forma sencilla sobre el flujo de referencia como el número de fotogramas entre dos imágenes I. No obstante, se ha preferido utilizar el flujo recibido, donde puede haber pérdidas. Por ello, se obtiene una lista de tamaños para todos los GOP identificados en el vídeo. En aras de ajustar al valor real, se calcula la media μ y la desviación típica σ de los valores medidos y se desechan todos aquellos que quedan fuera del intervalo $[\mu - \sigma/2, \mu + \sigma/2]$. Sobre la nueva lista, se vuelve a aplicar la media y este es el valor final para el tamaño del GOP.

Pérdidas de imágenes I (clase `IFrameLossRate`)

La pérdida de una imagen I provoca un tamaño de GOP inusualmente grande. Por ello, en esta medida se calcula una lista de tamaños de GOP de la misma manera que en la anterior, incluyendo la media y la desviación típica. Se considera una pérdida cuando un valor de esa lista supera $\mu + \sigma$, y la tasa de pérdidas es igual al número de pérdidas dividido por el número de imágenes I (número de imágenes I presentes en el vídeo recibido más el número de pérdidas obtenidas).

6.6.3. Medidas de calidad de vídeo

PSNR (clase `PSNR`)

Sean I, K dos imágenes monocromas (para imágenes en color, se aplica a la luminancia) de tamaño $M \times N$ píxeles y sean $I(i,j)$, $J(i,j)$ el valor de los píxeles i, j de dichas imágenes, se define el error cuadrático medio MSE de la siguiente forma:

$$MSE = \frac{1}{MN} \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} ||I(i,j) - J(i,j)||^2 \tag{6.9}$$

Sea B el número de bits por píxel de las imágenes, la $PSNR$ queda definida así:

$$PSNR = 20 \log_{10} \left(\frac{2^B - 1}{MSE} \right) \tag{6.10}$$

SSIM (clase `SSIM`)

El índice SSIM se calcula sobre una ventana cuadrada que barre toda la imagen, obteniendo un conjunto de valores que después se promedia. Sean x e y dos ventanas cuadradas sobre las imágenes X e Y respectivamente, el índice SSIM se calcula como sigue:

$$SSIM(x,y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)} \tag{6.11}$$

donde:

- μ_x media de x;
- μ_y media de y;
- σ_x^2 varianza de x;
- σ_y^2 varianza de y;
- σ_{xy} covarianza de x e y;
- $c_1 = (k_1 L)^2$, $c_2 = (k_2 L)^2$
- $L = 2^B - 1$, con B el número de bits por píxel;
- $k_1 = 0{,}01$ y $k_2 = 0{,}03$ por defecto.

Existen numerosas implementaciones de esta medida [51]. Por ello, se ha optado por tomar una escrita en C++ que hace uso de la librería OpenCV [52] y traducirla a Python.

Recomendación ITU-T G.1070 (clase `G1070`)

Esta medida no aporta mucha información a la estimación de la calidad de vídeo ya que el algoritmo que se describe en dicha recomendación forma parte de uno más amplio para la estimación de la calidad multimedia. No obstante, se ha decidido incluir para aportar un ejemplo de una medida que hace uso en sus cálculos de parámetros como el *bitrate*, el *framerate* o las pérdidas de paquetes.

La recomendación G.1070 utiliza tablas de coeficientes dependientes del codificador, la resolución y otros parámetros que deberían calcularse siguiendo un método descrito en el propio documento. Dado que se trata de un método complejo que no entra en las pretensiones de este proyecto, se ha escogido un conjunto de coeficientes a modo de ejemplo (extraídos del Apéndice I de [18]).

Mapeo de PSNR a MOS (clase `PSNRtoMOS`)

Se implementa un mapeo simple, el mismo que se utiliza en EvalVid y se propone en [53]:

PSNR (dB)	MOS
> 37	5
31 - 37	4
25 - 31	3
20 - 25	2
< 20	1

Tabla 6.1: Mapeo de PSNR a MOS.

Degradación por intervalos (clase `MIV`)

Esta medida (propuesta también en [53] y utilizada en EvalVid) compara la MOS de cada fotograma del vídeo recibido con su homólogo en el vídeo de referencia. Se define como *Distortion in Interval* (DIV) el porcentaje de fotogramas con una MOS peor que en el vídeo de referencia dado un intervalo deslizante (25 fotogramas en este caso).

6.7. *Script* principal

En el Listado 6.3 se muestra el *script* principal que permite ejecutar la aplicación. Sus funciones son procesar los argumentos pasados por línea de comandos, iniciar el *logger* e instanciar la clase del cliente o el servidor en función del rol escogido.

Obsérvese que, para el cliente, el *script* se asegura de que ha sido lanzado con privilegios de administrador: estos son requeridos por el *sniffer* de red.

Listado 6.3: *Script* principal.

```
1  from VideoTester.config import parseArgs, initLogger, VTLOG
2  from sys import exit
3
4  if __name__ == '__main__':
5      # Parse arguments
6      args = parseArgs()
7      # Initialize VT logger (VTLOG)
8      initLogger(args)
9      # Server mode
10     if args.mode == "server":
11         from VideoTester.config import SERVERIP, SERVERPORT
12         from VideoTester.core import Server
13         server = Server()
14         try:
15             VTLOG.info('XMLRPC Server running at ' + SERVERIP + ':' + str(SERVERPORT))
16             VTLOG.info('Use Control-C to exit')
17             server.serve_forever()
18         except KeyboardInterrupt:
19             pass
20         VTLOG.info("Exiting...")
21     # Client mode
22     else:
23         from os import getuid
24         if getuid() != 0:
25             VTLOG.error("You need administrator privileges to run this program as
                   client")
26             exit()
27         # With GUI
28         if args.gui:
29             from VideoTester.gui import ClientGUI
30             client = ClientGUI(0)
31             client.MainLoop()
32         # Without GUI
33         else:
34             from VideoTester.config import CONF
35             from VideoTester.core import Client
36             if args.conf:
37                 CONF = args.conf[0]
38             client = Client(CONF)
39             client.run()
40         VTLOG.info("Exiting...")
```

Capítulo 7

Evaluación

En este capítulo se procede a la evaluación del proyecto llevado a cabo. Para ello, en primer lugar se define el escenario empleado para la realización de las pruebas con el objetivo de que los resultados mostrados posteriormente sean reproducibles.

7.1. Escenario

Se propone un escenario virtualizado con para la evaluación de la aplicación con los siguientes componentes:

Máquina *host*
De aquí en adelante, HOST. Ordenador personal con las siguientes características:

- Procesador Intel®Core™i7 Q820 con cuatro núcleos a 1.73 GHz.
- 4 GB de memoria RAM.
- Sistema operativo GNU/Linux Fedora 14 (64 bits).
- Herramientas de virtualización basadas en el Kernel (KVM).

Máquina virtual 1
De aquí en adelante, MV1. Máquina virtual que se ejecuta sobre el HOST con las siguientes características:

- Procesadores asignados: 1.
- Memoria RAM asignada: 1024 MB.
- Sistema operativo GNU/Linux Fedora 14 (64 bits).
- Video Tester instalado junto con todas sus dependencias.

Máquina virtual 2
De aquí en adelante, MV2. Máquina virtual que se ejecuta sobre el HOST con las siguientes características:

- Procesadores asignados: 1.
- Memoria RAM asignada: 1024 MB.
- Sistema operativo GNU/Linux Fedora 14 (64 bits).
- Video Tester instalado junto con todas sus dependencias.

Máquina virtual NAT

De aquí en adelante, MVNAT. Máquina virtual que se ejecuta sobre el HOST con las siguientes características:

- Procesadores asignados: 1.
- Memoria RAM asignada: 512 MB.
- Sistema operativo GNU/Linux Fedora 14 (64 bits).
- Dos interfaces de red disponibles: `eth1` y `eth2`.

7.1.1. Topologías utilizadas

7.1.1.1. Topología 1

Se establece una subred para las máquinas virtuales en el rango 192.168.0.0/24. La MV1 ejecuta el cliente con la IP 192.168.0.1, mientras que la MV2 ejecuta el servidor con la IP 192.168.0.2 (véase la Figura 7.1).

En la MV2 se ejecuta el siguiente comando (con permisos de administrador):

```
1 $ tc qdisc add dev eth0 root netem delay 100ms loss 1%
```

De esta forma, se establecen pérdidas del 1 % y un retardo (de ida y vuelta) para los paquetes de salida de 100 ms. Como el retardo para los paquetes que viajan de la MV1 a la MV2 es prácticamente nulo, desde la MV1 se ve un retardo extremo a extremo (solo de ida) de 50 ms.

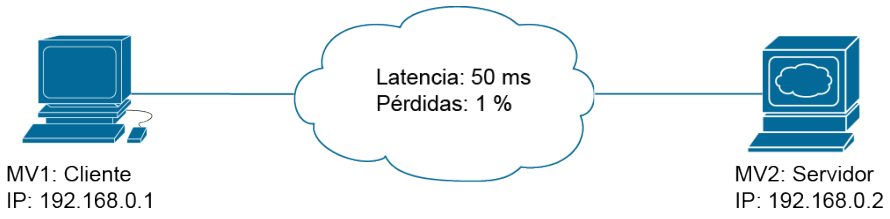


Figura 7.1: Topología 1.

7.1.1.2. Topología 2

Se establecen las subredes 192.168.0.0/24 y 10.0.0.0/24: la primera actúa como red local privada situada tras un dispositivo NAT, mientras que la segunda lo hace como red pública. La máquina MV1 ejecuta el cliente con la IP 192.168.0.2 y la MV2 ejecuta el servidor con la IP 10.0.0.2. Entre ellas, se sitúa la MVNAT a modo de enrutador con NAT de dos interfaces con las IP 192.168.0.1 y 10.0.0.1 respectivamente (véase la Figura 7.2).

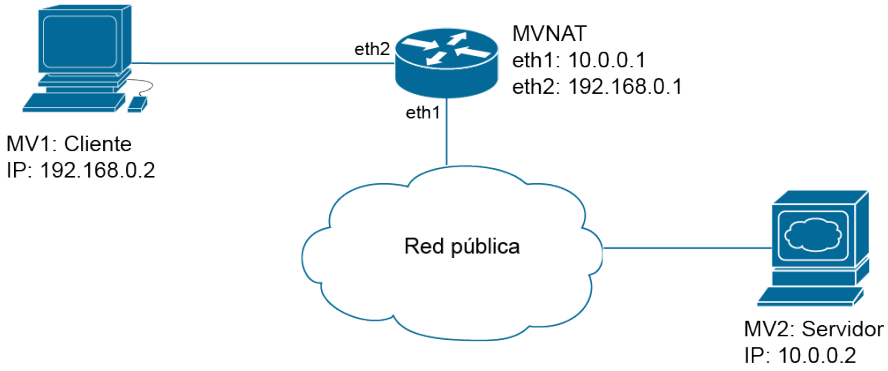


Figura 7.2: Topología 2.

Considerando que `eth1` es la interfaz de red conectada a la red pública y `eth2` es la interfaz conectada a la red privada, los comandos para habilitar MVNAT como enrutador NAT son los siguientes:

```
$ iptables -t nat -A POSTROUTING -o eth2 -j MASQUERADE
$ iptables -A FORWARD -i eth1 -j ACCEPT
$ echo 1 > /proc/sys/net/ipv4/ip_forward
```

Por último, la MV1 necesita una ruta por defecto hacia la red pública:

```
$ route add default gw 192.168.0.1
```

7.1.1.3. Topología 3

Se establece la misma disposición y configuración de la topología anterior, solo que se intercambian los papeles: esta vez el cliente se ejecuta en la red pública (MV2) y el servidor en la red privada (MV1), tal y como se aprecia en la Figura 7.3.

Dado que desde la red pública no existe acceso a la privada, la MVNAT debe implementar redirección de puertos hacia la IP de la MV1 en un rango suficiente (p. ej.: desde el 8000, que es el puerto por defecto del servidor, hasta el 8005, para permitir hasta 5 servidores RTSP). Para ello, se ejecutan los siguientes comandos en MVNAT:

```
$ iptables -t nat -A PREROUTING -i eth2 -p tcp --dport 8000:8005 -j DNAT --to
    192.168.0.2
$ iptables -t nat -A PREROUTING -i eth2 -p udp --dport 8001:8005 -j DNAT --to
    192.168.0.2
```

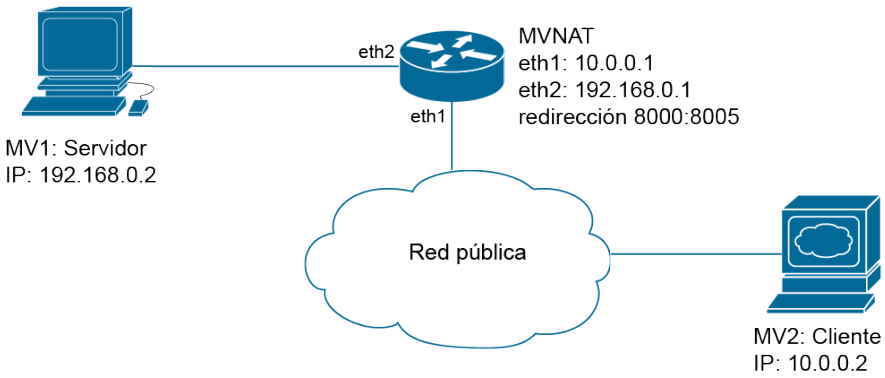


Figura 7.3: Topología 3.

7.1.2. Configuración

- El archivo de vídeo utilizado para las pruebas es `akiyo_cif.h264` (resolución de 352×288 píxeles, 11.96 segundos de duración), que puede descargarse desde la web de EvalVid [26].
- Las opciones de configuración fijadas son las siguientes (véase el Listado 7.1): codificador H.263, 128 kbps, 25 fps y todas las medidas seleccionadas para su cálculo.

Listado 7.1: Archivo `VT.conf` utilizado para las pruebas.

```
[video]

path=video/
video0=akiyo_cif.264

[client]

video=video0
codec=h263
bitrate=128
framerate=25
iface=eth0
ip=192.168.0.2
port=8000
protocols=udp-unicast
```

```
16  qos=latency, delta, jitter, skew, bandwidth, plr, pld
17  bs=streameye, refstreameye, gop, iflr
18  vq=psnr, ssim, g1070, psnrtomos, miv
```

7.1.3. Pruebas

Se ha considerado que el uso de un protocolo u otro varía sustancialmente el funcionamiento del sistema —sobre todo en lo que a la parte de procesamiento de la captura se refiere— además de las medidas obtenidas —ya que TCP recupera las pérdidas y UDP no—, no así la elección de un codificador u otro, que produce resultados similares de cara a la evaluación de la aplicación. En cuanto a UDP *multicast*, tampoco presenta diferencia alguna con respecto a *unicast* —exceptuando el hecho de que la IP de destino es una dirección *multicast*.

En la Sección 6.4.1, se defendía la inclusión del elemento `videorate` de *GStreamer* como una posible solución al desalineamiento de fotogramas que sufre EvalVid cuando se producen pérdidas en la transmisión. Se debe, por consiguiente, valorar el impacto de dicho elemento.

Por último, el elemento `rtspsrc` (descrito también en la Sección 6.4.1) teóricamente proporciona acceso a clientes situados detrás de un dispositivo NAT por defecto. No obstante, conviene realizar la comprobación pertinente. En la configuración contraria —es decir, con el servidor situado tras un NAT—, sin embargo, hay que implementar una redirección de puertos en el dispositivo NAT.

Por todo ello, con el objetivo de evaluar Video Tester, se han establecido cuatro pruebas mediante el escenario descrito en la sección anterior:

1. Funcionamiento con UDP *unicast* como protocolo de transporte mediante la topología número 1 (véase la Sección 7.1.1.1).
2. Funcionamiento con TCP como protocolo de transporte (cambiando la opción `protocols` del Listado 7.1 de *udp-unicast* a *tcp*) mediante la topología número 1 (véase la Sección 7.1.1.1).
3. En un escenario con pérdidas (es decir, con UDP como protocolo de transporte), se comprobará el impacto del elemento `videorate` mediante la topología número 1 (véase la Sección 7.1.1.1).
4. Funcionamiento tras un dispositivo enrutador con NAT.
   a) Con el cliente en la red privada (cambiando la opción `ip` del Listado 7.1 de *192.168.0.2* a *10.0.0.2*) mediante la topología número 2 (véase la Sección 7.1.1.2).
   b) Con el servidor en la red privada (cambiando la opción `ip` del Listado 7.1 de *192.168.0.2* a *10.0.0.1*) mediante la topología número 3 (véase la Sección 7.1.1.3).

7.2. Resultados

Las medidas de QoS y de *bitstream* serán evaluadas teniendo en cuenta los parámetros de red del escenario propuesto y la interrelación entre ellas. En cuanto a las medidas de calidad de vídeo, para la recomendación ITU-T G.1070 no se dispone de una referencia, por lo que el valor obtenido no será significativo (no obstante, recuérdese que el objetivo de su implementación es mostrar las posibilidades de Video Tester para la incorporación de la información de red y *bitstream* en las medidas de calidad de vídeo). Para las demás —PSNR, SSIM, MOS y MIV—, los resultados se compararán con los obtenidos por EvalVid para demostrar que se cubren las funcionalidades de este.

Los comandos para generar los resultados con EvalVid son los siguientes:

```
$ psnr 352 288 420 00_ref_original.yuv 00_ref.yuv > ref_psnr.txt
$ psnr 352 288 420 00_ref_original.yuv 00.yuv > work/psnr.txt
$ psnr 352 288 420 00_ref_original.yuv 00.yuv ssim > work/ssim.txt
$ mos work/ ref_psnr.txt > mos.txt
```

Dichos comandos generan los archivos que se describen a continuación, los cuales pueden ser representados con la herramienta *Gnuplot*:

- `work/psnr.txt` almacena la PSNR de cada fotograma.
- `work/ssim.txt` almacena el índice SSIM de cada fotograma.
- `work/qual.txt` almacena el mapeo de PSNR a MOS para cada fotograma.
- `work/miv.txt` almacena el MIV de cada intervalo de fotogramas.

7.2.1. Prueba con UDP

Una vez puesto en marcha el servidor en MV2 (mediante el comando `VT server`), la prueba se lanza con el siguiente comando:

```
$ python -m cProfile -o udp.log VT client
```

Al finalizar la ejecución, el archivo `udp.log` recoge las estadísticas de la misma. A modo de ejemplo, se muestran las diez primeras ordenadas por tiempo de ejecución acumulado:

```
>>> import pstats
>>> p = pstats.Stats('udp.log')
>>> p.sort_stats('cumulative').print_stats(10)

         3718543 function calls (3629163 primitive calls) in 51.620 CPU seconds

```

```
7     Ordered by: cumulative time
8     List reduced from 2415 to 10 due to restriction <10>
9
10    ncalls tottime percall cumtime percall filename:lineno(function)
11         1    0.000   0.000  51.622  51.622 {execfile}
12         1    0.021   0.021  51.622  51.622 VT:8(<module>)
13         1    0.195   0.195  51.536  51.536 core.py:216(run)
14         3    0.000   0.000  18.514   6.171 gstreamer.py:99(__play)
15         1    0.036   0.036  17.396  17.396 core.py:285(__loadData)
16         3   14.723   4.908  16.441   5.480 {method 'run' of 'glib.MainLoop' objects}
17         1    0.007   0.007  14.866  14.866 gstreamer.py:109(receiver)
18         3    0.000   0.000   9.144   3.048 video.py:13(__init__)
19         2    0.000   0.000   8.168   4.084 video.py:61(__init__)
20         2    4.818   2.409   8.166   4.083 video.py:84(__readH263)
```

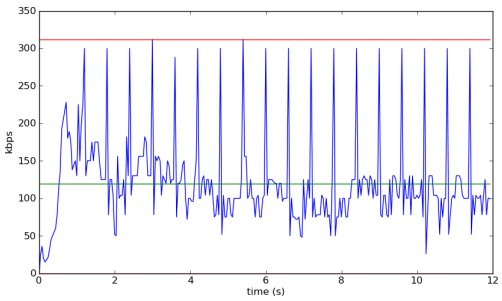
Se recogen 3718543 llamadas a funciones y un tiempo total de ejecución de 51,62 segundos. Teniendo en cuenta que la transmisión del vídeo dura unos 12 segundos —ya que se transmite en tiempo real—, el tiempo de procesamiento del cliente es de $51{,}62 - 12 = 39{,}62$ segundos.

Aparte de las tres primeras líneas, que se corresponden con el núcleo del programa, y de la cuarta, que se corresponde con el procesamiento de *GStreamer* y es inevitable, se observa que la función que más tiempo acumula es la de carga de los archivos de vídeo (`__loadData()`) con un total de 17,4 segundos.

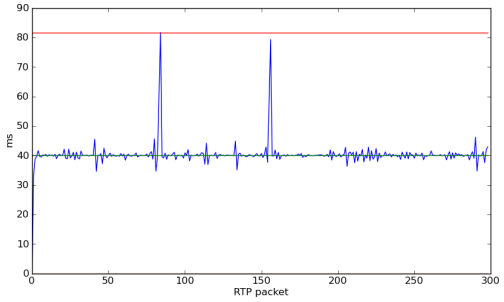
7.2.1.1. Medidas de QoS

Se mide una latencia de 50,46 ms y unas pérdidas del 0,67 %, lo cual concuerda con el escenario propuesto. El resto de medidas se representan en la Figura 7.4.

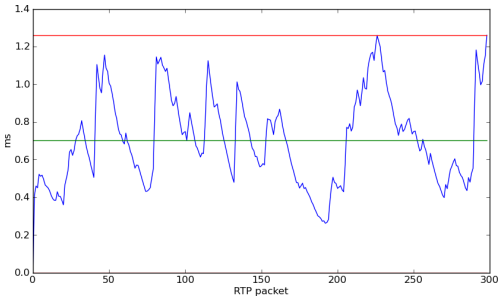
- La media del ancho de banda (Figura 7.4a) se sitúa en 119,8 kbps, cercana a los 128 kbps teóricos de la configuración. Los picos a intervalos regulares se deben a la llegada de paquetes con imágenes I, las cuales son sustancialmente más grandes, como se aprecia en la Figura 7.5.
- El tiempo entre paquetes (Figura 7.4b) es muy constante e igual a 40 ms, salvo en dos puntos: se observa que los picos presentes en dicha gráfica se corresponden con los intervalos donde se producen pérdidas, tal y como refleja la distribución de pérdidas (Figura 7.4d).
- Debido a la regularidad del tiempo entre paquetes, el *jitter* (Figura 7.4c) apenas varía en un rango menor de 1,3 ms.
- La desviación (Figura 7.4e) también es pequeña ya que se mueve en un rango de unos 9 ms.



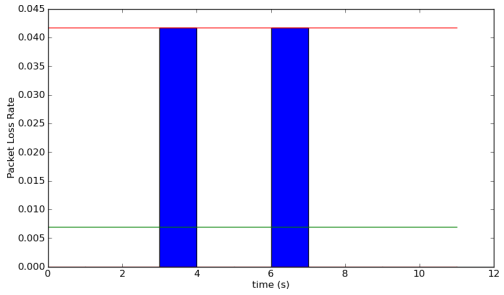
(a) Ancho de banda.



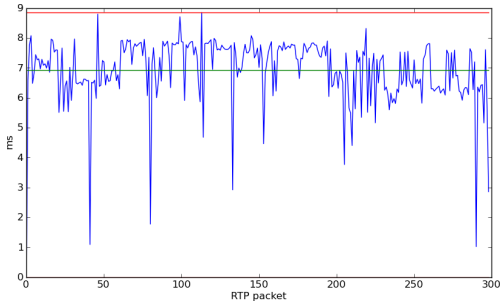
(b) Tiempo entre paquetes.



(c) *Jitter*.

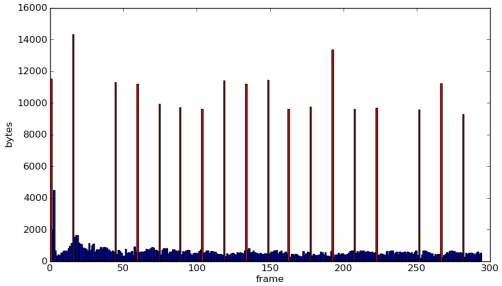


(d) Distribución de pérdidas.

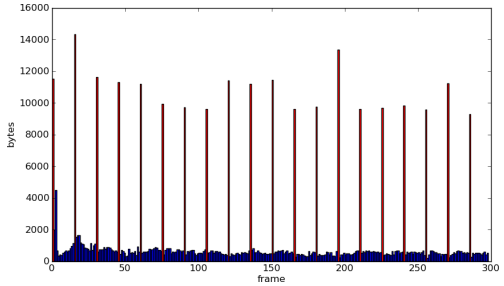


(e) Desviación.

Figura 7.4: Prueba con UDP: medidas de QoS.



(a) Vista del flujo.



(b) Vista del flujo de referencia.

Figura 7.5: Prueba con UDP: medidas de *bitstream*.

7.2.1.2. Medidas de *bitstream*

Se mide un tamaño de GOP de 15 fotogramas y unas pérdidas de imágenes I del 10 %. En la Figura 7.5 se representan las vistas del flujo codificado para el vídeo recibido y de referencia. En la vista del primero, se observa que hay menos fotogramas. Una rápida inspección de las gráficas permite comprobar que se han perdido 2 de los 20 fotogramas I (en rojo, frente a los P en azul), lo cual concuerda con el dato del 10 % mencionado anteriormente.

7.2.1.3. Medidas de calidad de vídeo

El valor medido para la recomendación ITU-T G.1070 es de 1,66. En cuanto al resto de medidas (véase la Figura 7.6), se aprecia una correlación clara en los picos de degradación (degradación implica caída en los valores de PSNR, SSIM y MOS, y aumento del porcentaje en el MIV). Asimismo, las gráficas de VideoTester coinciden exactamente con las obtenidas con EvalVid.

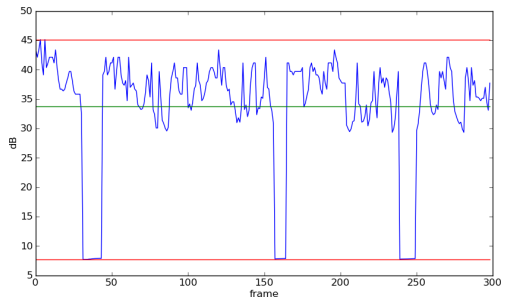
7.2.2. Prueba con TCP

Una vez puesto en marcha el servidor en MV2, la prueba se lanza con el siguiente comando:

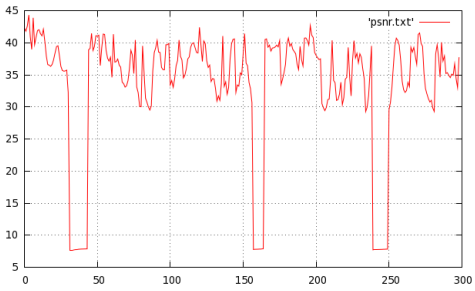
```
$ python -m cProfile -o tcp.log VT client
```

Al finalizar la ejecución, el archivo `tcp.log` recoge las estadísticas de la misma. A modo de ejemplo, se muestran las diez primeras ordenadas por tiempo de ejecución acumulado:

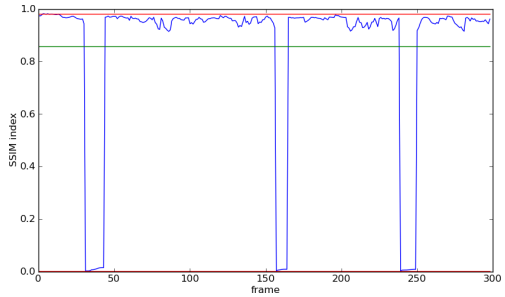
```
>>> import pstats
>>> p = pstats.Stats('tcp.log')
>>> p.sort_stats('cumulative').print_stats(10)

         11277866 function calls (10782713 primitive calls) in 57.029 CPU seconds

   Ordered by: cumulative time
   List reduced from 2430 to 10 due to restriction <10>

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    0.000    0.000   57.031   57.031 {execfile}
        1    0.020    0.020   57.030   57.030 VT:8(<module>)
        1    0.002    0.002   56.930   56.930 VideoTester/core.py:216(run)
        3    0.000    0.000   17.135    5.712 VideoTester/gstreamer.py:99(__play)
        3   14.565    4.855   16.425    5.475 {method 'run' of 'glib.MainLoop' objects}
        1    0.000    0.000   15.854   15.854 VideoTester/sniffer.py:114(parsePkts)
```



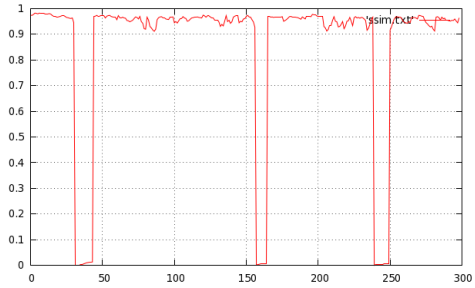
(a) PSNR.



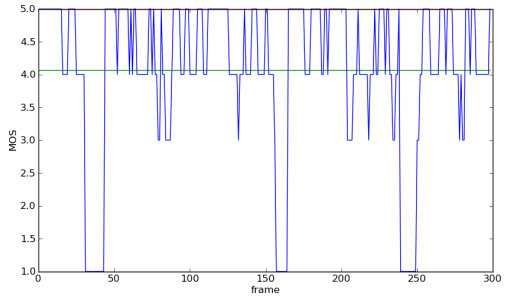
(b) PSNR con EvalVid.



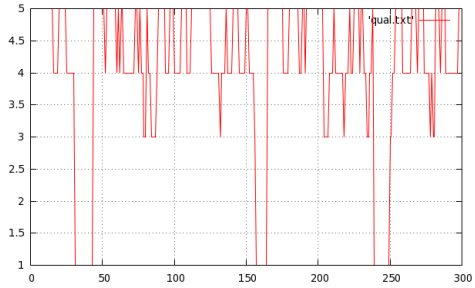
(c) SSIM.



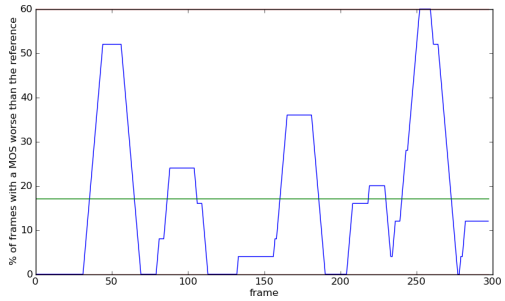
(d) SSIM con Evalvid.



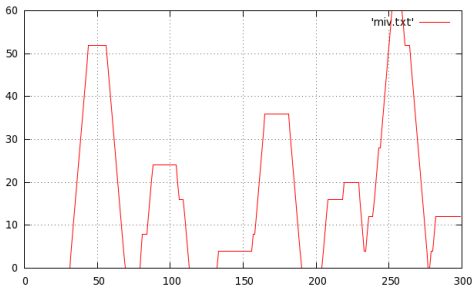
(e) MOS.



(f) MOS con Evalvid.



(g) MIV.



(h) MIV con Evalvid.

Figura 7.6: Prueba con UDP: medidas de calidad de vídeo.

```
17          1    0.019    0.019   15.851   15.851 VideoTester/sniffer.py:219(__parseTCP)
18          1    0.005    0.005   14.472   14.472 VideoTester/core.py:285(__loadData)
19      297/1    0.631    0.002   14.202   14.202 VideoTester/sniffer.py:223(extract)
20          1    0.000    0.000   13.462   13.462 VideoTester/gstreamer.py:109(receiver)
```

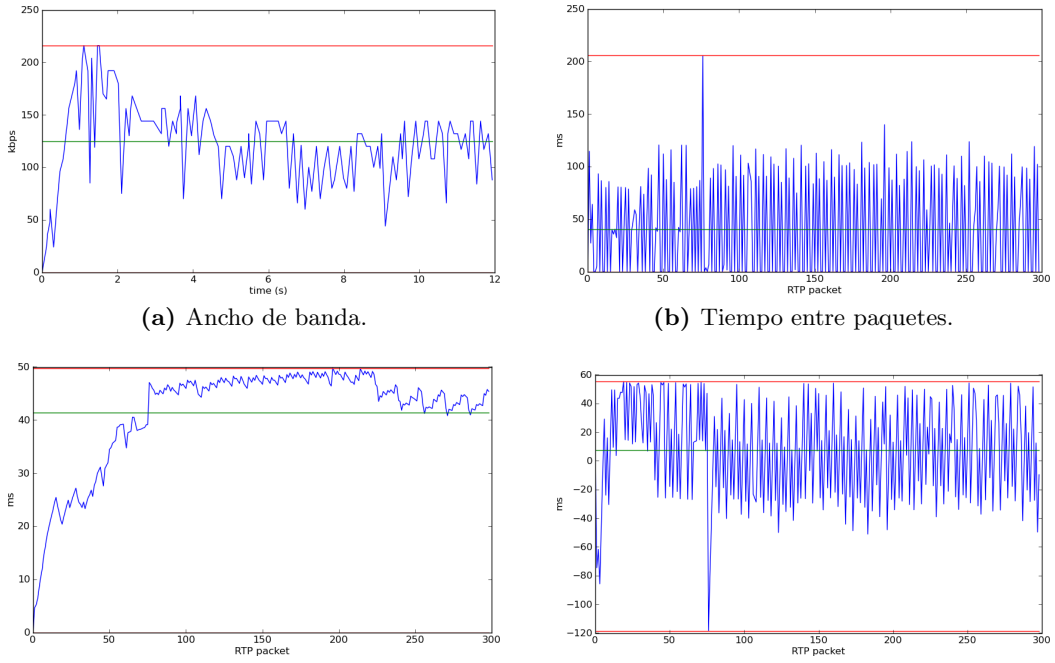
Se recogen 11277866 llamadas a funciones (más del triple que en la prueba con UDP) y un tiempo total de ejecución de 57,03 segundos (5,41 segundos más que en la prueba con UDP). Teniendo en cuenta que la transmisión del vídeo dura unos 12 segundos —ya que se transmite en tiempo real—, el tiempo de procesamiento del cliente es de $57{,}03 - 12 = 45{,}03$ segundos.

Como en la prueba con UDP, aquí también destaca la carga de los archivos de vídeo (`__loadData()`) con un tiempo acumulado de 14,47 segundos. No obstante, se aprecia una diferencia sustancial con aquella, y es que aquí aparece la función de procesamiento de paquetes TCP (`__parseTCP()`) con un gran consumo de tiempo: 15,85 segundos.

7.2.2.1. Medidas de QoS

Se mide una latencia de 50,51 ms y unas pérdidas del 0 %, lo cual concuerda con el escenario propuesto. El resto de medidas se representan en la Figura 7.7.

- La media del ancho de banda (Figura 7.7a) se sitúa en 124,6 kbps, cercana a los



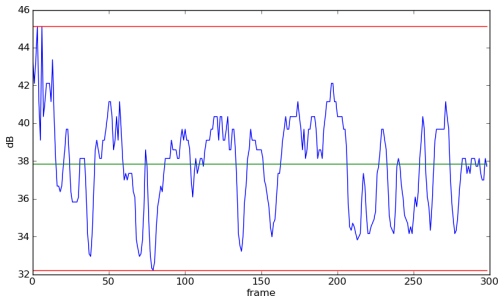
(a) Ancho de banda. **(b)** Tiempo entre paquetes.

(c) *Jitter*. **(d)** Desviación.

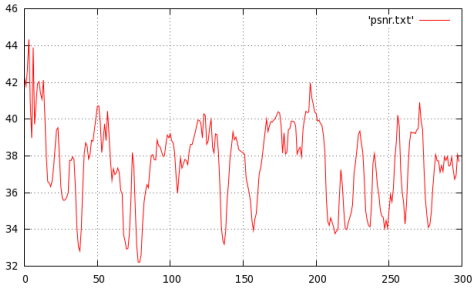
Figura 7.7: Prueba con TCP: medidas de QoS.

128 kbps teóricos de la configuración. A diferencia del caso de UDP, esta vez no se aprecian picos a intervalos regulares debido a que en TCP se encapsulan a menudo varios paquetes RTP en uno TCP, por lo que se produce menos variación del ancho de banda.

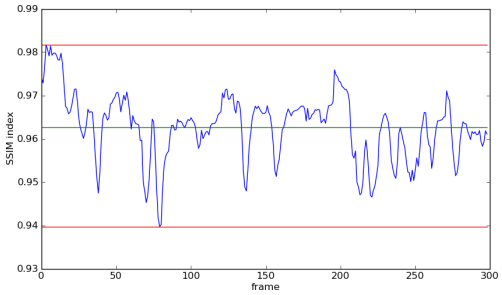
- En el tiempo entre paquetes (Figura 7.7b), la media coincide aproximadamente con el caso UDP —como es lógico, ya que la tasa de envío no ha variado—. No obstante, debido de nuevo al tipo de encapsulamiento, se producen grandes variaciones: se pasa constantemente de 0, para paquetes RTP que llegan encapsulados juntos, a unos 100 ms para el primer paquete RTP del siguiente segmento TCP.



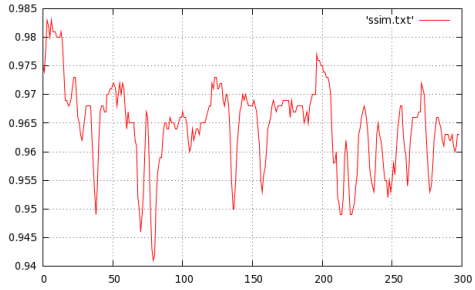
(a) PSNR.



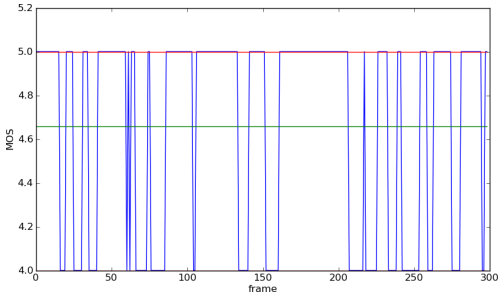
(b) PSNR con EvalVid.



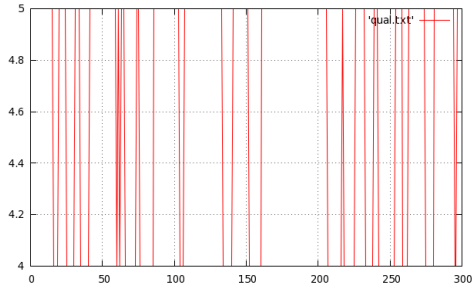
(c) SSIM.



(d) SSIM con Evalvid.



(e) MOS.



(f) MOS con Evalvid.

Figura 7.8: Prueba con TCP: medidas de calidad de vídeo.

- El *jitter* (Figura 7.7c) se acumula hasta alcanzar una media de unos 40 ms, que es precisamente la variación que sufre la media del tiempo entre paquetes.

- La desviación (Figura 7.7d) sufre, de igual forma, grandes variaciones. Pasa con rapidez de valores positivos —que se producen cuando han llegado más paquetes RTP juntos de la cuenta (el *timestamp* "adelanta" al tiempo de llegada)— a negativos —el siguiente segmento TCP se demora para mantener el *bitrate* y esto hace que el tiempo de llegada vuelva a "adelantar" al *timestamp*.

- El pico anómalo que figura tanto en la gráfica del tiempo entre paquetes como en la desviación (de 200 y −120 ms respectivamente) se debe a una recuperación de TCP ante la pérdida de un paquete.

7.2.2.2. Medidas de *bitstream*

Se mide un tamaño de GOP de 15 fotogramas y unas pérdidas de imágenes I del 0 %, lógico ya que TCP es un protocolo fiable y recupera las pérdidas. Las vistas del flujo codificado coinciden exactamente y son iguales a la gráfica de la Figura 7.5b.

7.2.2.3. Medidas de calidad de vídeo

El valor medido para la recomendación ITU-T G.1070 es de 1,63, levemente inferior al caso UDP, y el MIV es constante e igual a 0 porque no se han producido pérdidas de transmisión. En cuanto al resto de medidas (véase la Figura 7.8), de nuevo se aprecia que las gráficas de VideoTester coinciden exactamente con las obtenidas con EvalVid.

7.2.3. Prueba de alineamiento de fotogramas

Para esta prueba, se ha empleado el mismo escenario y la misma configuración que en la prueba con UDP; tan solo se ha modificado un detalle: se han incrementado las pérdidas de la MV2 al 2 % para que los desalineamientos, si se producen, sean más severos. Con estas condiciones, se han realizado dos ejecuciones: una con el elemento `videorate` presente y otra sin él para comprobar su impacto.
Una vez obtenidos los dos vídeos YUV recibidos, se ha procedido a escoger un fotograma de la parte final del vídeo; concretamente, el número 261 de los 297 que posee el vídeo de referencia. A continuación, se busca dicho fotograma en los vídeos recibidos. En condiciones de alineamiento perfecto, el fotograma escogido debería ser el 261 en las tres secuencias.

Con el elemento `videorate` presente, el fotograma de referencia se sitúa en la posición 265 (4 de diferencia). En cambio, sin él, el fotograma de referencia acaba en la posición 236 (25 de diferencia o, lo que es lo mismo, 1 segundo). En la Figura 7.9 se reproducen los fotogramas de dichas posiciones y se puede comprobar que coinciden.

(a) Fotograma de referencia en la posición 261.

(b) Con `videorate`: posición 265.

(c) Sin `videorate`: posición 236.

Figura 7.9: Prueba de alineamiento de fotogramas.

7.2.4. Prueba de funcionamiento con NAT

7.2.4.1. Cliente situado en la red privada

Las ejecuciones realizadas con la topología correspondiente verifican que el elemento `rtspsrc` garantiza el funcionamiento de la aplicación cuando el cliente está situado tras un enrutador con NAT, tanto en el caso TCP como UDP *unicast*, sin requerir acciones adicionales.

En el caso de UDP *multicast*, no se puede asegurar el funcionamiento ya que se hace necesario que todos los enrutadores intermedios soporten *multicast*.

7.2.4.2. Servidor situado en la red privada

En este caso, dado que desde la red pública no existen rutas hacia redes privadas, es indispensable configurar una redirección de puertos en el enrutador con NAT tal y como se describe en la Sección 7.1.1.3. Las ejecuciones realizadas con dicha topología verifican el correcto funcionamiento de la aplicación cuando el servidor está situado en una red privada bajo las condiciones de redirección citadas.

Capítulo 8

Conclusiones

8.1. Resultados

En este proyecto se ha diseñado e implementado una herramienta que aporta una solución completa a la problemática de la evaluación de la calidad de vídeo sobre redes IP en entornos reales en explotación. Las contribuciones más destacables son las siguientes:

- Se han analizado las diversas aproximaciones existentes al problema, sus fortalezas y sus debilidades; en especial, el conjunto de herramientas EvalVid, que sirve de referencia por ser el más completo hasta el momento.
- Se ha propuesto un diseño flexible con capacidad para extraer información de los tres niveles implicados en el procesado y transmisión de vídeo —nivel de paquete, nivel de *bitstream* y nivel de imagen— y, por tanto, con potencial para implementar cualquier tipo de medida de calidad.
- Se ha escogido una arquitectura de tipo cliente-servidor concurrente. El servidor se limita a enviar el vídeo desde una IP pública (o bien una IP privada con redirección de puertos configurada en el dispositivo NAT correspondiente), mientras que el cliente controla todo el proceso pudiendo ser ubicado detrás de un NAT sin necesidad de configuración adicional.
- El diseño define un subsistema de medidas reutilizable y extensible: proporciona aislamiento mediante una interfaz estándar para la interacción con el mismo a través de los medidores y facilidad para la implementación e incorporación de nuevas medidas —a lo cual contribuye el lenguaje de programación escogido.
- Se proporciona soporte para cuatro de los codificadores más comunes —H.263, H.264, MPEG-4 parte 2 y Theora— a través del marco de trabajo *GStreamer*. Esto supone una mejora con respecto a EvalVid, el cual está limitado por el contenedor MP4 del que hace uso.

- La transmisión se realiza en forma de *streaming* real gracias a un servidor RTSP con capacidad para encapsulado TCP, UDP con envío *unicast* o UDP con envío *multicast*.
- Los parámetros de vídeo seleccionables, aparte del propio vídeo y el codificador, son el *bitrate* y el *framerate*.
- La herramienta consta de una única aplicación y su uso básico se realiza a través de línea de comandos, aunque también se proporciona una interfaz gráfica que permite la representación de los resultados.
- Se ha aportado una solución al desalineamiento de fotogramas en transmisiones con pérdidas que sufre EvalVid. Las pruebas realizadas demuestran que dicha solución mitiga en gran medida el problema, lo que proporciona medidas de calidad más exactas.
- Se han implementado todas las medidas de calidad que soporta EvalVid y se ha demostrado que el cálculo coincide en ambas herramientas.

En definitiva, Video Tester cubre las funcionalidades de EvalVid y añade mejoras en cuanto a usabilidad, extensibilidad, soporte para codificadores, soporte para métodos de transmisión y fiabilidad en condiciones de pérdidas.

8.2. Líneas de trabajo futuras

Quedan abiertas las siguientes líneas de trabajo:

- La implementación del servidor RTSP como un binario aparte produce inestabilidades a menudo, lo que hace necesario el reinicio del mismo. No obstante, la librería *GStreamer RTSP server* de la que se hace uso se encuentra en un estadio temprano de desarrollo. En próximas versiones, se prevé que se completen los *bindings* para Python, lo que permitirá portar todo el código en C de dicho servidor RTSP a la aplicación principal.
- A medida que el marco de trabajo *GStreamer* proporcione soporte para más codificadores, estos podrán incorporarse a Video Tester con un esfuerzo mínimo.
- Resulta posible rehacer ciertas partes del procesado del fichero de captura para que los paquetes que lleguen demasiado tarde en función de un *buffer* de reproducción seleccionable sean considerados como pérdidas.
- Aunque se desechó la idea en las primeras etapas de desarrollo por las inestabilidades que producía, podría resultar interesante añadir la posibilidad —al igual que se ha hecho con el *bitrate* y el *framerate*— de seleccionar la resolución de

los vídeos. Actualmente, si se desea evaluar un mismo vídeo con distintas resoluciones, estos vídeos se deben producir de antemano con alguna herramienta externa.

- El vídeo es un tipo de contenido pesado en cuanto a cantidad de información. Por otro lado, Python es un lenguaje de *scripting* que, si bien proporciona la inmediatez necesaria para el requisito de extensibilidad, tiene la contrapartida de la lentitud en la ejecución frente a lenguajes compilados como C. Estos dos factores unidos, hacen que el tiempo de procesamiento necesario sea elevado. Para aumentar el rendimiento, se proponen dos posibilidades no excluyentes:
  - Rehacer ciertas partes del código para que sean más eficientes. La evaluación de la aplicación (véase el Capítulo 7) muestra que hay procesos que consumen demasiado tiempo, tales como la carga de los archivos de vídeo y el procesamiento de paquetes en modo TCP.
  - Adaptar el código para producir un binario compilado (como si se tratase de un programa en C) mediante herramientas existentes destinadas a tal efecto.
- Se ha considerado que la problemática a resolver en este proyecto era lo suficientemente compleja como para centrar el desarrollo en un único sistema operativo, en concreto Linux. No obstante, Python es un lenguaje multiplataforma y, por tanto, la aplicación implementada es susceptible de ser extendida a plataformas Windows y Mac OS X.
- Ante el inminente cambio a la versión 6 del protocolo IP que va a sufrir Internet, Video Tester necesitará ser revisado y adaptado.

Apéndice A

Formatos YUV

En este anexo se procederá a retratar las generalidades de los formatos YUV, además de describir de forma precisa el formato concreto que se maneja en el presente proyecto.

A.1. Descripción general

Los formatos YUV no están definidos de forma precisa en la literatura científica y técnica. Conforman una familia heterogénea de espacios de color que tienen en común la divisón de la señal en tres componentes: una de luminancia o brillo Y, y dos de crominancia o color U-V (véanse las Figuras A.1 y A.2). La luminancia es percibida con mayor precisión que la crominancia por el ojo humano, por lo que puede ahorrarse ancho de banda en la codificación de esta última. Además, el espacio YUV es retrocompatible con los antiguos televisores en blanco y negro que solo soportan la señal de luminancia.

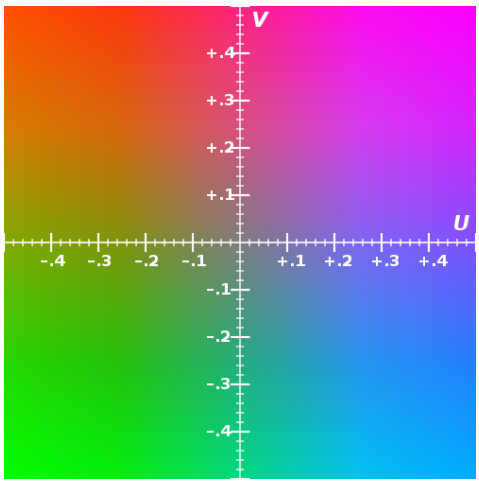
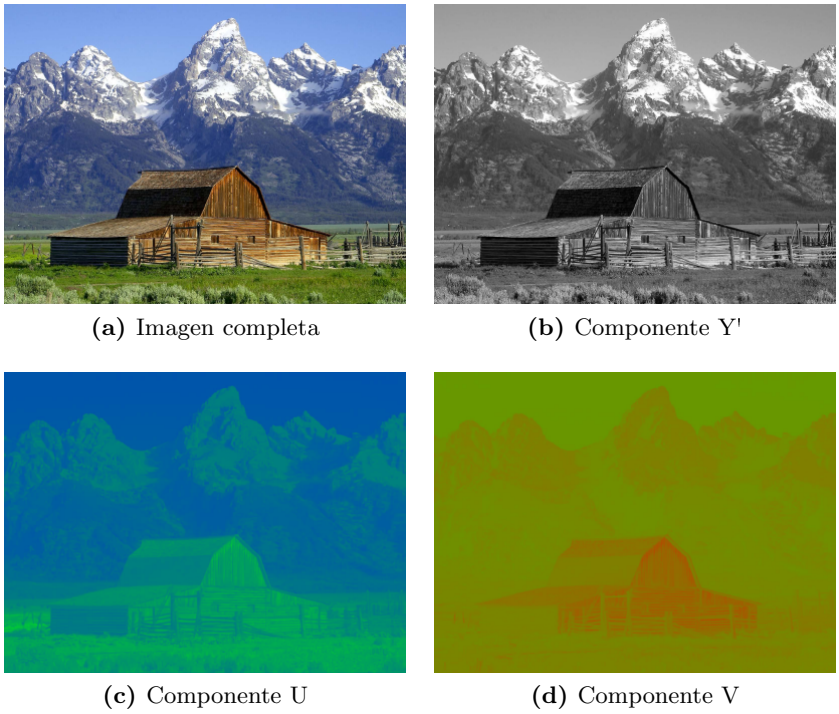


Figura A.1: Plano de color U-V.



(a) Imagen completa **(b)** Componente Y'

(c) Componente U **(d)** Componente V

Figura A.2: Imagen y su descomposición en componentes de luminancia y crominancia.

Se distinguen dos grandes grupos de formatos: planares (*planar*) y empaquetados (*packed*) [54].

Packed Cada punto de la imagen está formado por un macropíxel que contiene las tres componentes empaquetadas en una misma matriz.

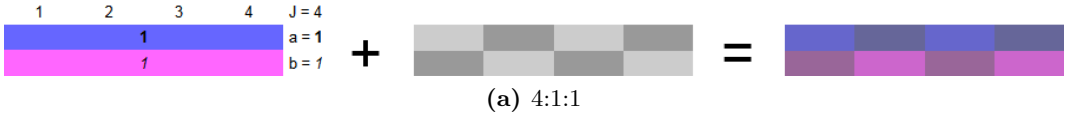
Planar Las componentes se almacenan como tres planos separados.

Dentro de estos grupos, la cantidad de bits por píxel que utiliza un formato YUV para su representación depende del muestreo aplicado a las componentes y de si incorpora información sobre la opacidad (canal alfa). Lo habitual es que se utilicen menos bits para representar la crominancia, ya que el ojo responde peor al color; esta situación se denomina *submuestreo de la crominancia*. El tipo de muestreo se representa como un ratio de tres partes $J : a : b$ (p. ej.: 4:2:0) que describe el número de muestras de luminancia y crominancia que se toman en una región de $J \times 2$ píxeles (véase la Figura A.3):

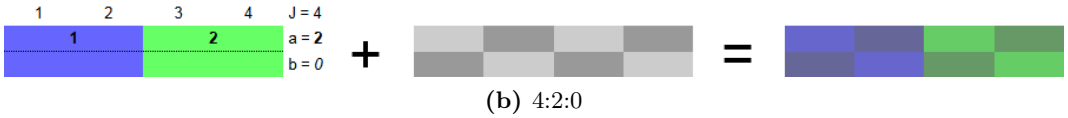
J Referencia horizontal (habitualmente 4).

a Número de muestras de crominancia en la primera fila de J píxeles.

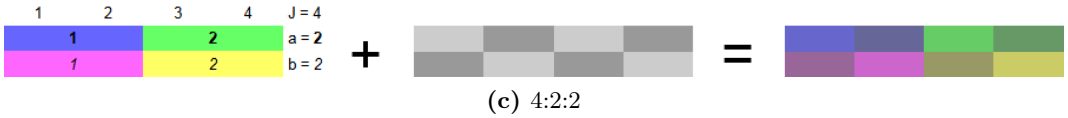
b Número adicional de muestras de crominancia en la segunda fila de J píxeles.



(a) 4:1:1



(b) 4:2:0



(c) 4:2:2

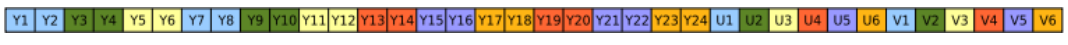
Figura A.3: $UV + Y = YUV$ para diferentes tipos de muestreo.

Es decir, no existe submuestreo de la crominancia si el ratio es 4:4:4. Cuando existe información sobre la opacidad, se añade un cuarto número al ratio que debe ser igual a J (p. ej.: 4:2:2:4).

A.2. Formato I420

En este proyecto se utiliza el formato I420 (también llamado IYUV) para almacenar y analizar el vídeo descomprimido porque es el que utiliza por defecto *GStreamer*. Se trata de un formato planar con un ratio 4:2:0 y 12 bits por píxel, de los cuales, 8 se correponden con la componente Y, 2 con la componente U y otros 2 con la V. Cada fotograma se almacena de tal forma que el plano Y completo va en primer lugar seguido por el U y el V (véase la Figura A.4).

(a) Almacenamiento del fotograma



(b) Posición en el flujo de bits

Figura A.4: Orden de las componentes. Cada cuadradito se corresponde con una componente de un píxel.

Apéndice B

Codificadores de vídeo

En el desarrollo de Video Tester, tal y como se especifica en el Capítulo 3, se ha añadido soporte para los siguientes codificadores de vídeo: H.263, H.264, MPEG-4 parte 2 y Theora. Así pues, el lector de vídeo codificado diseñado en la Sección 5.2.4 requiere un método diferente para cada uno de los cuatro codificadores soportados. Este anexo recoge la información recopilada y las decisiones tomadas durante el desarrollo de dichos métodos.

B.1. H.263

El flujo se compone de una serie de fotogramas que sigue la estructura que se describe a continuación [34]:

SC Código de inicio de 22 bits. Siempre es el mismo:

```
0000 0000 0000 0000 1000 00
```

TR Referencia temporal, 8 bits.

PTYPE Tipo de información, longitud variable.

Bit 1 Siempre a 1 para evitar la emulación del código de inicio.

Bit 2 Siempre a 0 para diferenciarse con la recomendación H.261.

Bit 3 Indicador de pantalla dividida.

Bit 4 Indicador de cámara de documento.

Bit 5 Liberación de congelación de toda la imagen.

Bit 6-8 Formato de fuente.

Si los bits 6-8 no son iguales a `111`, que indica PTYPE extendido, los siguientes bits también se incluyen en el PTYPE.

Bit 9 Tipo de codificación: `0` INTRA (imagen I) o `1` INTER (imagen P).

Bit 10 Modo vector de movimiento sin restricción.

Bit 11 Modo codificación aritmética basada en sintaxis.

Bit 12 Modo predicción avanzada.

Bit 13 Modo trama PB.

Existen otros campos que no se describen aquí porque no interesan para los propósitos de este proyecto. La especificación completa de este formato puede encontrarse en la recomendación ITU-T H.263 [34].

De lo anterior se extrae que, para separar los fotogramas de un flujo H.263, basta con identificar los códigos de inicio de cada uno de ellos: el tamaño del fotograma será la longitud en *bytes* desde un código de inicio hasta el siguiente. Por otro lado, como por inspección se ha determinado que *GStreamer* no utiliza un PTYPE extendido, el tipo de fotograma queda reflejado en el bit 9 del campo PTYPE.

B.2. H.264

Los flujos H.264 se dividen en unidades NAL (*Network Abstraction Layer*) precedidas por un código de inicio de 32 bits [35]:

```
0000 0000 0000 0000 0000 0000 0000 0001
```

Desafortunadamente, los archivos codificados generados de manera local (los que no han sufrido el proceso de transmisión) no implementan este código de inicio. Por ello, Video Tester es capaz de extraer la información relativa al vídeo recibido en H.264, pero no lo es para el de referencia.

La información relativa al tipo de unidad NAL se encuentra en los primeros campos de su estructura [35]:

`forbidden_zero_bit` Un bit siempre a 0.

`nal_ref_idc` 2 bits.

`nal_unit_type` Tipo de unidad NAL, 5 bits.

Cuando el `nal_unit_type` es igual a 1 o 5, el contenido de la unidad NAL es un fotograma (todos los valores posibles de este campo y su significado se encuentran en la Tabla B.1). Una vez determinado que se trata de un fotograma, la información sobre el tipo de imagen puede hallarse en los primeros campos de la siguiente cabecera (`slice_header`):

`first_mb_in_slice` Dirección del primer macrobloque, número de bits variable.

`slice_type` Tipo de fotograma, número de bits variable.

nal_unit_type	Contenido de la unidad NAL
0	Unspecified
1	Coded slice of a non-IDR picture
2	Coded slice data partition A
3	Coded slice data partition B
4	Coded slice data partition C
5	Coded slice of an IDR picture
6	Supplemental enhancement information (SEI)
7	Sequence parameter set
8	Picture parameter set
9	Access unit delimiter
10	End of sequence
11	End of stream
12	Filler data
13	Sequence parameter set extension
14	Prefix NAL unit
15	Subset sequence parameter set
16..18	Reserved
19	Coded slice of an auxiliary coded picture without partitioning
20	Coded slice extension
21..23	Reserved
24..31	Unspecified

Tabla B.1: Códigos de tipos de unidad NAL.

Código	Valor
1	0
010	1
011	2
00100	3
00101	4
00110	5
00111	6
0001000	7
0001001	8
0001010	9
...	...

Tabla B.2: Códigos *Exponential-Golomb*.

Estos dos campos contienen códigos *Exponential-Golomb*. La correspondencia entre estos códigos y su valor puede verse en la Tabla B.2. En las pruebas hechas con *GStreamer*, se comprobó que el primer bit de la `slice_header` siempre es igual a 1. Esto significa que el código para el primer campo es siempre de un bit de longitud según la tabla mencionada anteriormente.

Para el segundo campo, la asociación entre el valor del código encontrado y el tipo de fotograma se encuentra en la Tabla B.3.

`slice_type`	Tipo de fotograma
0	P
1	B
2	I
3	SP
4	SI
5	P
6	B
7	I
8	SP
9	SI

Tabla B.3: Códigos de tipos de fotograma para H.264.

B.3. MPEG-4 parte 2

Cada fotograma está contenido en una estructura VOP (*Video Object Plane*) cuyos dos primeros campos son los siguientes [36]:

`vop_start_code` Código de inicio de 32 bits. Siempre es el mismo:

```
0000 0000 0000 0000 0000 0001 1011 0110
```

`vop_coding_type` Tipo de fotograma, 2 bits (véase la Tabla B.4).

`vop_coding_type`	Tipo de fotograma
00	I
01	P
10	B
11	S

Tabla B.4: Códigos de tipos de fotograma para MPEG-4 parte 2.

B.4. Theora

Este formato resulta más complejo que los anteriores a la hora de determinar dónde empieza un fotograma y dónde acaba: estos no están separados por ningún código de inicio y contienen muchos campos opcionales [37]. Por ello, en este proyecto se ha optado por encapsular el vídeo codificado con Theora en un contenedor Matroska [55] que sí separa los fotogramas al introducirlos en sus propias estructuras.

Matroska embebe los fotogramas en bloques que, a su vez, se agrupan en *clusters* con el siguiente código de inicio de 32 bits:

```
0001 1111 0100 0011 1011 0110 0111 0101
```

En las pruebas realizadas, se ha comprobado que *GStreamer* utiliza los llamados "bloques simples" que tienen un código de inicio de 8 bits característico:

```
1010 0011
```

Una vez identificado el bloque simple, el siguiente bit (FTYPE [37]) ya es parte del flujo Theora e informa del tipo de fotograma: si es igual a 1, se trata de una imagen I; si es igual a 0, se trata de una P.

Apéndice C

Manual de referencia

C.1. Instalación

Video Tester es una aplicación para Linux y tiene las siguientes dependencias (las versiones indicadas son aquellas con las que se ha desarrollado el proyecto: con versiones anteriores, Video Tester probablemente no funcionará; con versiones posteriores, es probable que sí, pero no se garantiza):

- Python versión 2.7 y librerías estándar [40].
- *GStreamer 0.10* [39].
- *GStreamer plugins 0.10*: *base*, *good*, *ugly*, *bad*.
- *GStreamer-FFmpeg plugins 0.10* [43].
- *GStreamer RTSP server 0.10.7* [44].
- Librerías no estándar para Python: *Scapy 2.0* [45], *Matplotlib 1.0.1* [46], *Numpy 1.4.1* [47], *GStreamer bindings 0.10*, *WxPython 2.8.11* [48], *WxPython backend* para *Matplotlib 1.0.1*, *OpenCV bindings 2.1* [49].

Para la instalación de las mismas en Fedora (versión 14 al menos), basta con ejecutar el siguiente comando (con permisos de administrador):

```
1 $ yum install gstreamer gstreamer-plugins-base gstreamer-plugins-good gstreamer-
      plugins-ugly gstreamer-plugins-bad-free gstreamer-plugins-bad-nonfree gstreamer-
      python gstreamer-ffmpeg gstreamer-rtsp scapy numpy python-matplotlib python-
      matplotlib-wx wxpython opencv-python
```

La aplicación se distribuye mediante un archivo comprimido y autoinstalable llamado `VideoTester-0.1.tar.gz`. Los pasos para su instalación son los siguientes (el último comando, con permisos de administrador):

```
$ tar -xvf VideoTester-0.1.tar.gz
$ cd VideoTester-0.1
$ python setup.py install
```

Durante la instalación se piden dos parámetros (en este orden):

- La interfaz de red de salida a Internet del servidor (por omisión, `eth0`).
- El puerto de escucha del servidor (por omisión, el 8000).

Al finalizar la instalación, la aplicación queda instalada en el directorio de paquetes por defecto de Python (normalmente, `/usr/lib/python2.7/site-packages/VideoTester`) y estará disponible un nuevo comando: `VT`.

C.2. Documentación

Tras la instalación, puede encontrarse la documentación de Video Tester en el directorio `/usr/share/doc/VideoTester-0.1`:

`/usr/share/doc/VideoTester-0.1/html`
: Directorio con la documentación de Video Tester en formato HTML.

`/usr/share/doc/VideoTester-0.1/test`
: Directorio con archivos de test.

`/usr/share/doc/VideoTester-0.1/README.txt`
: Fichero "léeme" con este manual de referencia.

`/usr/share/doc/VideoTester-0.1/VT.conf`
: Fichero de configuración de ejemplo.

C.2.1. Archivos de test

En el directorio de test pueden encontrarse tres *scripts* que implementan algunas funcionalidades complementarias:

`test-generate-conf.py` Generador de archivos de configuración interactivo. Permite generar gran cantidad de archivos de configuración destinados a baterías de pruebas de forma rápida.

`test-pkl2csv.py` Conversor de formatos. Video Tester guarda todas las medidas generadas en cada ejecución en archivos independientes; para ello, utiliza una librería estándar de serialización llamada Pickle. Este conversor permite convertir recursivamente los archivos Pickle (extensión `*.pkl`) a un formato de valores separados por comas (CSV).

`test-series.py` Ejemplo de batería de pruebas en serie con diferentes archivos de configuración.

C.2.2. Formato de los archivos de configuración

Los archivos de configuración se componen de secciones que determinan el contexto al que se aplican las diferentes variables de configuración. Una sección comienza con un nombre encerrado entre corchetes. A continuación y hasta el siguiente nombre de sección, se considera el contenido de la misma. Este contenido se compone por pares "llave=valor". Si una línea comienza por una almohadilla, será considerada un comentario.

Tanto cliente como servidor requieren obligatoriamente una sección llamada `[video]` (véase el Listado C.1 para un ejemplo). Video Tester busca esta sección en un archivo llamado `VT.conf` que deberá estar presente obligatoriamente en el directorio actual de trabajo (aquel desde el que se ejecuta la aplicación).

Listado C.1: Ejemplo de la sección de vídeo.

```
[video]

# Ruta hasta el emplazamiento de los archivos
path=video/

# Cada archivo se asigna a una llave de la siguiente forma...
video0=akiyo_cif.264
video1=bridge-close_cif.264
video2=bridge-far_cif.264
video3=bus_cif.264
... = ...
```

El cliente, adicionalmente, buscará en el archivo `VT.conf` otra sección denominada `[client]` (véase el Listado C.2 para un ejemplo). No obstante, puede indicarse otro archivo de configuración alternativo tal y como se explica en la Sección C.3.

Listado C.2: Ejemplo de la sección del cliente.

```
[client]

# Archivo a recibir (se asigna una llave de definidas en [video])
video=video2

# Codificador
# Opciones: h263, h264, mpeg4, theora
codec=h264
```

```
9
10  # Bitrate en kbps
11  bitrate=128
12
13  # Framerate en fps
14  framerate=25
15
16  # Interfaz por la que se accede al servidor
17  iface=eth0
18
19  # IP del servidor
20  ip=192.168.229.131
21
22  # Puerto donde escucha el servidor
23  port=8000
24
25  # Protocolo de transporte
26  # Opciones: udp-unicast, tcp, udp-multicast
27  protocols=udp-unicast
28
29  # Medidas de QoS a calcular
30  # Opciones: latency, delta, jitter, skew, bandwidth, plr, pld
31  qos=latency, jitter, bandwidth, plr
32
33  # Medidas de bitstream a calcular
34  # Opciones: streameye, refstreameye, gop, iflr
35  bs=streameye, iflr
36
37  # Medidas de calidad a calcular
38  # Opciones: psnr, ssim, g1070, psnrtomos, miv
39  vq=psnr, ssim
```

C.3. Uso

Resulta indispensable que el cliente y el servidor dispongan de un repositorio de vídeo local con los mismos archivos, así como el mismo archivo de configuración de vídeo. Las secuencias utilizadas deberán estar codificadas en un formato sin pérdidas. Un buen ejemplo son los vídeos recopilados en la sección *YUV 4:2:0 video sequences* del proyecto EvalVid [26].

Tanto el cliente como el servidor deben ejecutarse en el directorio de trabajo donde se halla el archivo `VT.conf`. El primero, además, requiere permisos de administrador.

Siempre se recoge un *log* de la ejecución en el archivo `temp/VT.log` (ruta relativa al directorio de trabajo). Este *log* también se muestra en la salida estándar en el caso del servidor y del cliente sin interfaz gráfica, y aparece en una de las pestañas del cliente con interfaz gráfica.

C.3.1. Servidor

El comando y el resultado se muestra a continuación:

```
$ VT server
[<fecha-y-hora> VTServer] INFO : XMLRPC Server running at <ip>:<puerto>
[<fecha-y-hora> VTServer] INFO : Use Control-C to exit
```

El servidor queda a la espera de clientes y no requiere más operación. Para pararlo, basta con presionar las teclas `Control + C`.

C.3.2. Cliente (por línea de comandos)

El cliente sin interfaz gráfica se ejecuta como sigue:

```
$ VT client
```

De esta manera, la aplicación busca la sección de configuración del cliente en el mismo archivo `VT.conf`. Si se ha ubicado dicha sección en un archivo diferente, puede indicarse así:

```
$ VT client -c otroarchivo.conf
```

C.3.3. Cliente (con interfaz gráfica)

El cliente con interfaz gráfica se ejecuta como sigue:

```
$ VT client -g
```

No se requiere un archivo de configuración para el cliente ya que todos esos parámetros serán configurados a través de la propia interfaz gráfica, en la pestaña de configuración (véase la Figura C.1).

Una vez escogidas las opciones convenientes, se procede a la ejecución pulsando en el menú `Run`. Durante la ejecución se muestra la pestaña de *log* (véase la Figura C.2).

Cuando acaba la ejecución, aparecen las pestañas de resultados y de vídeo (véase la Figura C.3). En de resultados, se muestran las gráficas de todas las medidas marcadas en la pestaña de configuración (véase la Figura C.4; en la de vídeo, hay un botón que permite reproducir el vídeo recibido y los de referencia (original sin pérdidas y con pérdidas debidas al codificador).

Adicionalmente, la interfaz gráfica puede utilizarse para representar medidas calculadas en otras ejecuciones. Para ello, hay que abrir los archivos Pickle (extensión `*.pkl`) deseados mediante el menú `File`, y aparecen inmediatamente en la pestaña de resultados.

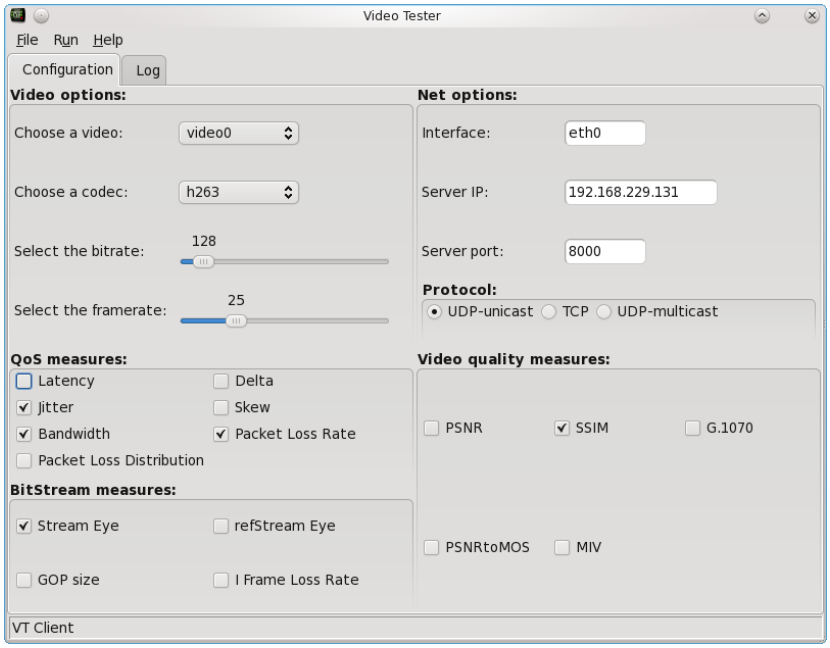


Figura C.1: Pestaña de configuración.

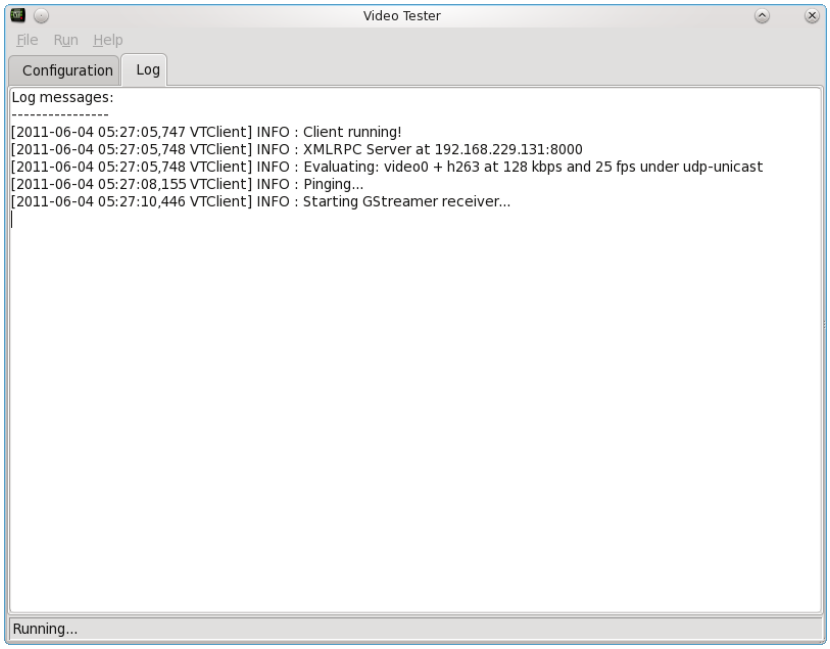


Figura C.2: Pestaña de *log* al comienzo de la ejecución.

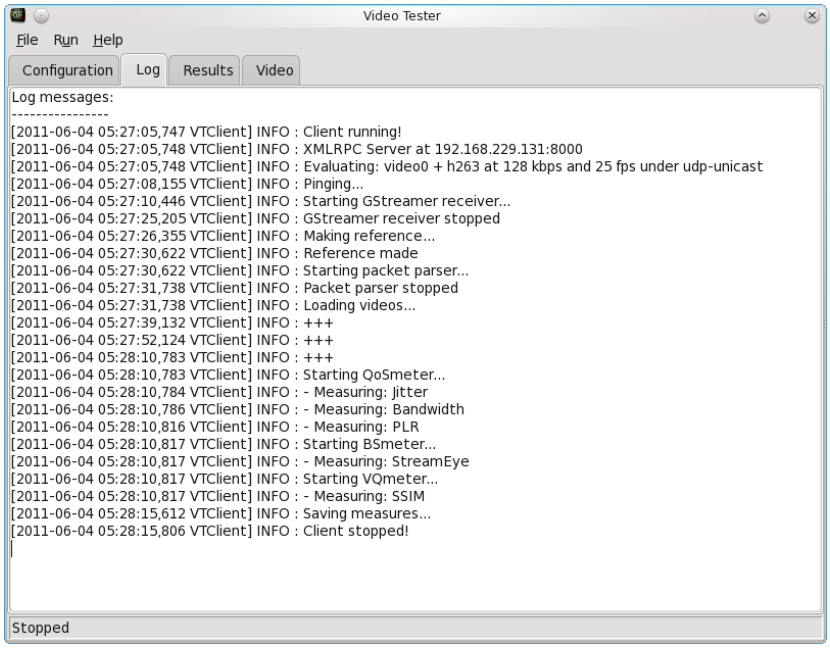


Figura C.3: Pestaña de *log* al final de la ejecución.

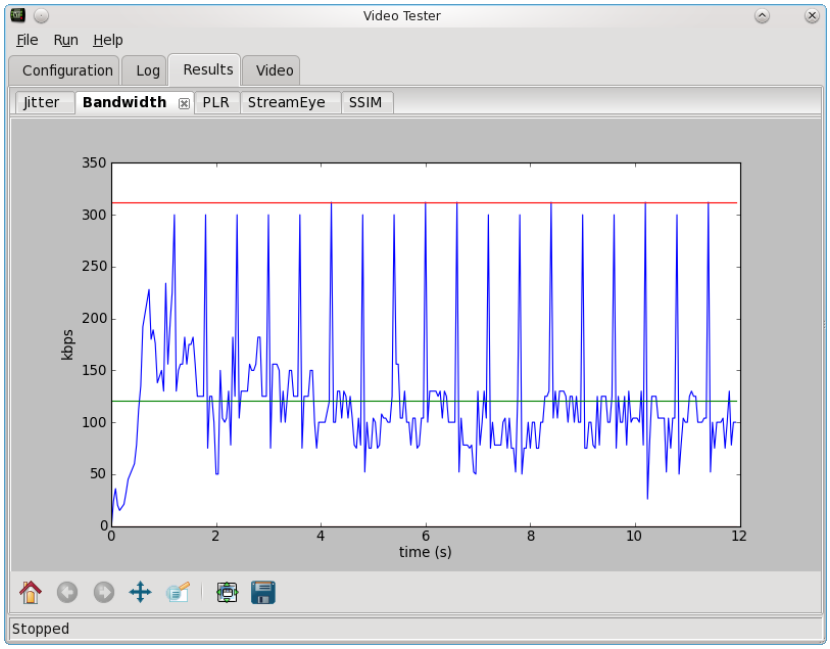


Figura C.4: Pestaña de resultados.

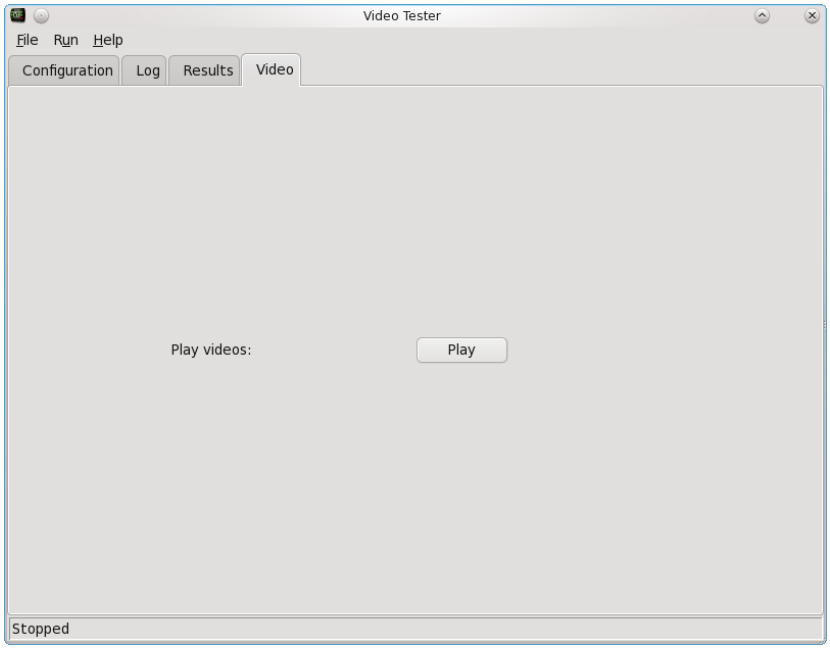


Figura C.5: Pestaña de vídeo.

C.3.4. Archivos generados tras la ejecución

Video Tester genera gran cantidad de información que se almacena en archivos contenidos en un directorio temporal que tendrá un nombre como el que sigue:

`<llave_del_vídeo>_<codificador>_<bitrate>_<framerate>_<protocolo>`

Supóngase, por ejemplo, que el cliente se ha configurado para pedir el video número 2 por UDP *unicast*, codificado con H.263 a 128 kbps y 25 fps. Tras la ejecución y bajo estas condiciones, se crea un directorio con el siguiente nombre (ruta relativa al directorio de trabajo):

`temp/video2_h263_128_25_udp-unicast`

En este directorio se almacenarán toda la información. Por ejemplo:

`00.cap` Archivo de captura de red.

`00.h263` Vídeo recibido sin descodificar.

`00.yuv` Vídeo recibido descodificado.

`00_ref.h263` Vídeo de referencia codificado.

`00_ref.yuv` Vídeo de referencia degradado por el codificador y descodificado.

`00_ref_original.yuv` Vídeo de referencia original.

`00_caps.txt` Datos sobre el vídeo YUV: *framerate*, resolución, formato, etc.

`00_<medida>.pkl` Medida calculada en formato Pickle.

Como se aprecia, los archivos generados en cada ejecución comparten un mismo prefijo de dos cifras; en el ejemplo, "00". Las subsiguientes ejecuciones, por tanto, producirán archivos con prefijos "01", "02", "03" y así sucesivamente.

C.4. Medidas

C.4.1. Formato de los resultados

Los resultados de las medidas se presentan en forma de diccionario de Python con las siguientes llaves:

name (Obligatorio) Nombre de la medida en formato cadena.

type (Obligatorio) Tipo de salida en formato cadena. Opciones:

- **"plot"** Para medidas multivalor adecuadas para ser representadas en forma de gráfica.
- **"bar"** Para medidas multivalor adecuadas para ser representadas en gráficas de barras (p. ej.: distribuciones de pérdidas, etc.).
- **"value"** Para medidas que tienen como salida un único valor.

units (Obligatorio) Unidades: una sola cadena en el caso de medidas de tipo "value" y una tupla con dos cadenas (una por cada eje) para medidas de tipo "plot" o "bar".

axes (Opcional: solo para medidas de tipo "plot" o "bar") Tupla con los ejes X e Y (en ese orden). Cada eje es una lista de valores.

min (Opcional: solo para medidas de tipo "plot" o "bar") Tupla con dos valores que hacen referencia al punto de la gráfica donde se encuentra el valor mínimo.

max (Opcional: solo para medidas de tipo "plot" o "bar") Tupla con dos valores que hacen referencia al punto de la gráfica donde se encuentra el valor máximo.

mean (Opcional: solo para medidas de tipo "plot" o "bar") Valor medio de la gráfica.

value (Opcional: solo para medidas de tipo "value") Valor de la medida.

width (Opcional: solo para medidas de tipo "bar") Intervalo de representación (ancho de la barra).

A continuación, se presentan tres ejemplos, uno para cada uno de los tipos:

```
{ 'name':'Bandwidth', 'type':'plot', 'units':('time (s)', 'kbps'), 'axes':([...],
    [...]), 'min':(3, 7), 'max':(5, 0), 'mean':5 }

{ 'name':'PLD', 'type':'bar', 'units':('time (s)', 'Packet Loss Rate'), 'axes'
    :([...], [...]), 'min':(3, 7), 'max':(5, 0), 'mean':5, 'width':1 }

{ 'name':'Latency', 'type':'value', 'units':'ms', 'value':50 }
```

C.4.2. Implementación de nuevas medidas

Video Tester dispone de tres categorías de medidas:

- Las medidas de QoS se encuentran en el módulo `measures.qos`.
- Las medidas de *bitstream* se encuentran en el módulo `measures.bs`.
- Las medidas de calidad de vídeo se encuentran en el módulo `measures.vq`.

Para implementar una medida nueva, hay que acudir al módulo correspondiente y añadir una clase cumpliendo una serie de directrices:

- Debe heredar de su categoría de medida: `QoSmeasure`, `BSmeasure` o `VQmeasure`.
- Debe contener dos métodos: `__init__()` y `calculate()`. El primero es el método inicializador y se encarga de definir el nombre, el tipo y las unidades en el atributo de resultados de la clase. El segundo realiza los cálculos y devuelve el atributo de resultados.

Estrictamente, el método inicializador no es necesario: podría incluirse su código en el método `calculate()`. No obstante, es preferible implementarlo por una cuestión de orden.

En el Listado C.3 se incluyen tres prototipos de medida a modo de ejemplo. Tras su creación, solo queda registrarlas en el medidor correspondiente tal y como queda reflejado en el Listado C.4. Las cadenas que se evalúan en los condicionales del medidor serán las que posteriormente deberán ser incluidas en el archivo de configuración para obtener su cálculo (véase el Listado C.5).

Listado C.3: Ejemplo de nuevas medidas.

```
class MedidaPlot(QoSmeasure):
    def __init__(self, data):
        QoSmeasure.__init__(self, data)
        self.data['name'] = 'Medida "plot"'
        self.data['type'] = 'plot'
        self.data['units'] = ('unidades eje x', 'unidades eje y')

    def calculate(self):
        # Operaciones...
        x = ... # Lista de valores (eje x)
        y = ... # Lista de valores (eje y)
        self.graph(x, y)
        return self.data

class MedidaValue(QoSmeasure):
    def __init__(self, data):
        QoSmeasure.__init__(self, data)
        self.data['name'] = 'Medida "value"'
        self.data['type'] = 'value'
        self.data['units'] = 'unidades'

    def calculate(self):
        # Operaciones...
        self.data['value'] = ... # Valor de salida
        return self.data

class MedidaBar(QoSmeasure):
    def __init__(self, data):
        QoSmeasure.__init__(self, data)
        self.data['name'] = 'Medida "bar"'
        self.data['type'] = 'bar'
        self.data['units'] = ('unidades eje x', 'unidades eje y')
        self.data['width'] = ... # Ancho de la barra

    def calculate(self):
        # Operaciones...
        x = ... # Lista de valores (eje x)
        y = ... # Lista de valores (eje y)
        self.graph(x, y)
        return self.data
```

Listado C.4: Ejemplo de registro en un medidor.

```
class QoSmeter(Meter):
    def __init__(self, selected, data):
        Meter.__init__(self)
        if 'latency' in selected:
            self.measures.append(Latency(data))
        # Otras medidas...
        if 'medidaplot' in selected:
            self.measures.append(MedidaPlot(data))
        if 'medidavalue' in selected:
            self.measures.append(MedidaValue(data))
        if 'medidabar' in selected:
            self.measures.append(MedidaBar(data))
```

Listado C.5: Ejemplo de cómo se invocan las nuevas medidas.

```
[client]

# ...

qos=latency, ..., medidaplot, medidavalue, medidabar

# ...
```

Referencias

[1] G. Huston. **BGP Routing Table Analysis Reports** [online]. Disponible en: `http://bgp.potaroo.net/`. 1

[2] Comisión del Mercado de las Telecomunicaciones. **Informe anual 2009**. Julio 2010. Disponible en: `http://www.cmt.es/es/publicaciones/anexos/20100705_IA09_CMT_INFORME_ANUAL_2009_SENCER_BAIXA.pdf`. 1

[3] T. C. Kwok. **Residential broadband Internet services and applications requirements**. *IEEE Communications Magazine*, **35**(6):76–83, Junio 1997. 1, 2

[4] Cisco. **Hyperconnectivity and the Approaching Zettabyte Era** [online]. Junio 2010. Disponible en: `http://www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/ns827/VNI_Hyperconnectivity_WP.pdf`. 1, 2

[5] ITU-T Recommendation E.800. **Definitions of terms related to quality of service**. *International Telecommunication Union*, Septiembre 2008. 2

[6] F. Fluckiger. *Understanding Networked Multimedia*. Prentice Hall, 1995. 3

[7] ITU-T Recommendation P.10/G.100 Amendment 2. **Vocabulary for performance and quality of service**. *International Telecommunication Union*, Julio 2008. 3

[8] S. Winkler and P. Mohandas. **The Evolution of Video Quality Measurement: From PSNR to Hybrid Metrics**. *IEEE Trans. Broadcasting*, **54**(3):660–668, Septiembre 2008. 4, 5

[9] ITU-R Recommendation BT.500-12. **Methodology for the subjective assessment of the quality of television pictures**. *International Telecommunication Union*, Septiembre 2009. 4, 8

[10] ITU-T Recommendation P.800. **Methods for subjective determination of transmission quality**. *International Telecommunication Union*, Agosto 1996. 4, 8

[11] S. Winkler. **Video quality measurement standards — Current status and trends**. pages 1–5, Diciembre 2009. 5

[12] S. Abeijón. *Performance evaluation of quality metrics for video quality assessment*. Master's thesis, Fakultät IV - Elektrotechnik und Informatik, Septiembre 2010. 7

[13] Z. Wang, L. Lu, and A. C. Bovik. **Video quality assessment based on structural distortion measurement**. *Signal Processing: Image Communication*, **19**(2):121–132, Febrero 2004. 8

[14] M. H. Pinson and S. Wolf. **A new standardized method for objectively measuring video quality**. *IEEE Trans. Broadcasting*, **50**(3):312–322, Septiembre 2004. 8

[15] VQEG. **Video Quality Experts Groups web page** [online]. Disponible en: `http://www.its.bldrdoc.gov/vqeg/`. 8

[16] ITU-T Recommendation P.910. **Subjective video quality assessment methods for multimedia applications**. *International Telecommunication Union*, Abril 2008. 8

[17] ITU-T Recommendation P.911. **Subjective audiovisual quality assessment methods for multimedia applications**. *International Telecommunication Union*, Diciembre 1998. 9

[18] ITU-T Recommendation G.1070. **Opinion model for video-telephony applications**. *International Telecommunication Union*, Abril 2007. 9, 48, 64

[19] ITU-T Recommendation J.144. **Objective perceptual video quality measurement techniques for digital cable television in the presence of a full reference**. *International Telecommunication Union*, Marzo 2004. 9, 15

[20] ITU-T Recommendation J.246. **Perceptual visual quality measurement techniques for multimedia services over digital cable television networks in the presence of a reduced bandwidth**. *International Telecommunication Union*, Agosto 2008. 9

[21] ITU-T Recommendation J.247. **Objective perceptual multimedia video quality measurement in the presence of a full reference**. *International Telecommunication Union*, Agosto 2008. 9

[22] ITU-T Recommendation J.249. **Perceptual video quality measurement techniques for digital cable television in the presence of a reduced reference**. *International Telecommunication Union*, Enero 2010. 9

[23] A. V. Murthy and L. J. Karam. **IVQUEST- Image and Video QUality Evaluation SofTware** [online]. Disponible en: `http://ivulab.asu.edu/Quality/IVQUEST`. 9

[24] A. Sánchez. *VoIP Tester*. Master's thesis, ETSIIT de la Universidad de Granada, 2010. Disponible en: `http://code.google.com/p/voiptester/`. 11

[25] J. Klaue, B. Rathke, and A. Wolisz. **EvalVid - A Framework for Video Transmission and Quality Evaluation**. pages 255–272, Septiembre 2003. Disponible en: `http://www.tkn.tu-berlin.de/publications/papers/evalvid.pdf`. 13

[26] J. Klaue. **EvalVid - A Video Quality Evaluation Tool-set** [online]. Disponible en: `http://www.tkn.tu-berlin.de/research/evalvid/`. 13, 70, 98

[27] **FFmpeg** [online]. Disponible en: `http://www.ffmpeg.org/`. 14

[28] **GPAC, an Open Source multimedia framework for research and academic purposes** [online]. Disponible en: `http://gpac.wp.institut-telecom.fr/`. 14

[29] **How to evaluate MPEG video transmission using the NS2 simulator?** [online]. Disponible en: `http://hpds.ee.ncku.edu.tw/~smallko/ns2/Evalvid_in_NS2.htm`. 14

[30] **Video Quality Metric (VQM) Software** [online]. Disponible en: `http://www.its.bldrdoc.gov/vqm/`. 15

[31] ANSI T1.801.03-2003. **Digital Transport of One-Way Video Signals - Parameters for Objective Performance Assessment**. *American National Standards Institute*, 2003. 15

[32] ITU-R Recommendation BT.1683. **Objective perceptual video quality measurement techniques for standard definition digital broadcast television in the presence of a full reference**. *International Telecommunication Union*, Junio 2004. 15

[33] M. Venkataraman and M. Chatterjee. **Inferring video QoE in real time**. *IEEE Network*, **25**(1):4–13, Enero 2011. 16

[34] ITU-T Recommendation H.263. **Video coding for low bit rate communication**. *International Telecommunication Union*, Enero 2005. 22, 89, 90

[35] ITU-T Recommendation H.264. **Advanced video coding for generic audiovisual services**. *International Telecommunication Union*, Marzo 2010. 22, 90

[36] ISO/IEC 14496-2. **Information technology - Coding of audio-visual objects**. *International Standard Organization*, Diciembre 2001. 22, 92

[37] Xiph.Org Foundation. **Theora Specification**. Marzo 2011. 22, 93

[38] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. **RTP: A Transport Protocol for Real-Time Applications**. *IETF RFC 3550*, Julio 2003. Disponible en: `http://www.ietf.org/rfc/rfc3550.txt`. 22, 40, 60

[39] **GStreamer - open source multimedia framework** [online]. Disponible en: `http://gstreamer.freedesktop.org/`. 24, 28, 95

[40] **The Python standard library** [online]. Disponible en: `http://docs.python.org/library/index.html`. 27, 95

[41] S. Michiels. **Stani's Python Editor (SPE)** [online]. Disponible en: `http://sourceforge.net/projects/spe/`. 27

[42] **wxGlade - a GUI builder for wxWidgets** [online]. Disponible en: `http://wxglade.sourceforge.net/`. 27, 42

[43] **GStreamer FFmpeg plug-in** [online]. Disponible en: `http://gstreamer.freedesktop.org/modules/gst-ffmpeg.html`. 28, 95

[44] **GStreamer RTSP server** [online]. Disponible en: `http://gstreamer.freedesktop.org/modules/gst-rtsp-server.html`. 28, 34, 95

[45] **Scapy - interactive packet manipulation program** [online]. Disponible en: `http://www.secdev.org/projects/scapy/`. 28, 95

[46] **Matplotlib - 2D plotting library** [online]. Disponible en: `http://matplotlib.sourceforge.net/`. 28, 95

[47] **Numpy - scientific computing with Python** [online]. Disponible en: `http://numpy.scipy.org/`. 28, 95

[48] **wxPython - a blending of the wxWidgets C++ class library** [online]. Disponible en: `http://www.wxpython.org/`. 28, 95

[49] **OpenCV - library of programming functions for real time computer vision** [online]. Disponible en: `http://opencv.willowgarage.com/wiki/`. 28, 95

[50] H. Schulzrinne, A. Rao, and R. Lanphier. **Real Time Streaming Protocol (RTSP)**. *IETF RFC 2326*, Abril 1998. Disponible en: `http://www.ietf.org/rfc/rfc2326.txt`. 39, 40, 57

[51] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli. **The SSIM Index for Image Quality Assessment** [online]. Disponible en: `https://ece.uwaterloo.ca/~z70wang/research/ssim/`. 64

[52] M. Rabah. **A C++ implementation of SSIM** [online]. Disponible en: `http://mehdi.rabah.free.fr/SSIM/`. 64

[53] J. Gross, J. Klaue, H. Karl, and A. Wolisz. **Cross-layer optimization of OFDM transmission systems for MPEG-4 video streaming**. *Computer Communications*, **27**, Febrero 2004. Disponible en: `http://www.tkn.tu-berlin.de/publications/papers/elsevier.pdf`. 64

[54] **YUV pixel formats - FOURCC.org** [online]. Disponible en: `http://www.fourcc.org/yuv.php`. 86

[55] **Matroska specifications** [online]. Disponible en: `http://matroska.org/technical/specs/index.html`. 93