Medidas de calidad para la transmisión de vídeo en entornos con restricciones de baja latencia

Jesús Molina Merchán

Máster en Ingeniería
de Telecomunicación

MÁSTERES
DE LA UAM
2017 - 2018

Escuela Politécnica Superior

UAM Universidad Autónoma de Madrid

UNIVERSIDAD AUTÓNOMA DE MADRID
ESCUELA POLITÉCNICA SUPERIOR

TRABAJO FIN DE MÁSTER

Medidas de calidad para la transmisión de vídeo en entornos con restricciones de baja latencia

Máster Universitario en Ingeniería de Telecomunicación

Autor: Molina Merchán, Jesús

Tutor: López de Vergara Méndez, Jorge E.
Departamento de Tecnología Electrónica y de las Comunicaciones

FECHA: Septiembre, 2018

MEDIDAS DE CALIDAD PARA LA TRANSMISIÓN DE VÍDEO EN ENTORNOS CON RESTRICCIONES DE BAJA LATENCIA

AUTOR: Molina Merchán, Jesús
DIRECTOR: López de Vergara Méndez, Jorge E.

High Performance Computing and Networking Research Group (HPCN)
Dpto. de Tecnología Electrónica y de las Comunicaciones
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Septiembre, 2018

Resumen

Resumen

El Trabajo de Fin de Máster realizado aborda la monitorización de la calidad de servicio de una red para asegurar la máxima calidad disponible a aplicaciones con restricciones de baja latencia. En particular, se centrará en la transmisión de vídeo en tiempo real para permitir la conducción remota de un coche teledirigido en el contexto del proyecto Racing Drones (MINECO/FEDER RTC-2016-4744-7). El objetivo de este proyecto es realizar carreras de drones terrestres, los cuales toman la imagen, la codifican con un códec LHE y la envían hasta un cliente sobre redes IP, por lo que requiere una latencia de red muy baja.

Por ello, este trabajo tiene que realizar una monitorización del estado de la red, y, por otro lado, se debe adecuar el tráfico generado por la aplicación a la capacidad de la red de forma que se ofrezca la máxima calidad de experiencia posible.

Con respecto a la monitorización, se ha implementado el protocolo Q4S. Este protocolo mide los parámetros de calidad de servicio de la red (ancho de banda, latencia, jitter y pérdidas) y determina si cumple con las restricciones impuestas por la aplicación, y en caso de que no cumpla, envíe una notificación al módulo actuador.

El módulo actuador es el encargado de, dadas las condiciones de la red, ajustar el tráfico de la aplicación, ofreciendo la máxima calidad de experiencia posible. Para ello, indicará las características del contenido multimedia generado por la aplicación, y tendrá también la capacidad de generar cambios sobre dicho contenido en el caso de que los parámetros de calidad de la red varíen.

Palabras Clave

Calidad de Servicio (QoS), Calidad de Experiencia (QoE), LHE, Q4S, medidas de red, monitorización, actuador, Zynqberry, Raspberry.

Abstract

This Master's Thesis explores the Quality of Service monitoring of a given network to ensure the maximum available quality for applications with low-latency restrictions. Particularly, the application in this project is related to real time video transmission to allow remote-control car driving in the context of Racing Drones project (MINECO/FEDER RTC-2016-4744-7). The target of this project is to allow races with terrestrial drones. Such drones capture the images to be coded with LHE codec and send them through an IP network, therefore requiring low latency.

This project monitors network status in order to adapt application's generated traffic to real network capacity to provide the best available quality to users.

Regarding the required monitoring, Q4S protocol has been implemented. This protocol measures network quality indicators (bandwidth, latency, jitter and loses) and qualifies their suitability to application's restrictions; if there is a deviation, it is automatically notified to the actuator module.

Actuator module is responsible for adapting the traffic to network conditions in order to ensure the best possible user experience; therefore, it must analyze generated multimedia content and it should be able to tune such content in case of deviation of the quality indicators.

Key words

Quality of Service (QoS), Quality of Experience (QoE), LHE, Q4S, network measures, monitoring, actuator, Zynqberry, Raspberry

Agradecimientos

En primer lugar me gustaría dar las gracias a mi tutor, Jorge E. López de Vergara, por guiarme y ayudarme durante la realización de este trabajo. También al grupo HPCN por haberme prestado la ayuda que necesitaba en tantas ocasiones, y por el buen trato recibido. Asimismo mencionar en estos agradecimientos al proyecto Racing Drones (MINECO/FEDER RTC-2016-4744-7), y agradecer a la Subdirección de Estudios de Posgrado y Formación Continua de la Escuela Politécnica Superior por la ayuda para material.

Agradecer también a mi familia, sobre todo a mi madre, por todo el apoyo recibido, y por trabajar duro para darme todas las facilidades.

A Beatriz, por compartir conmigo tantos y tantos momentos, y por apoyarme en los malos y darme las fuerzas cuando más las necesitaba, pero ante todo, por permanecer conmigo pase lo que pase.

Dar las gracias a mis amigos de la universidad, por todos los buenos ratos que hemos pasado, en especial a Belén, gracias por echarme una mano siempre que lo pedía (que han sido muchas) y, sobre todo, por tu compañía.

Índice general

Índice de figuras

Índice de tablas

1 Introducción

En este primer capítulo se va a realizar una breve introducción al trabajo realizado. Inicialmente se explicará la motivación de este trabajo, así como seguidamente se describirán los objetivos, la fase de realización y la estructura del documento. Para contextualizar este trabajo, cabe destacar que forma parte del proyecto Racing Drones, cofinanciado por el Ministerio de Economía y Competitividad (MINECO) y el Fondo Europeo de Desarrollo Regional (FEDER) con el número de referencia RTC-2016-4744-7. Este proyecto tiene como objetivo el manejo de un dron terrestre a control remoto mediante redes IP, por lo que el objetivo global es una transmisión de información de muy baja latencia entre la cámara del dron y el usuario final.

1.1. Motivación

Uno de los retos y de los objetivos más importantes para los proveedores de servicios multimedia y en los que más tiempo y recursos se destina es la disminución del retardo en la red. Gracias a las mejoras en la disminución del retardo de red que se están consiguiendo, cada vez hay mayor oferta de estos servicios y con mayor complejidad, como son los videojuegos online, la transmisión de vídeos sobre IP, etc.

Debido a la mayor exigencia en términos de requisitos de red, surge la necesidad de una monitorización continua de estas comunicaciones, que asegure que la conexión entre el proveedor de servicios y el cliente final cumpla con los requerimientos que dicho contenido necesita para transmitirse.

Sin embargo, la calidad de la red no es monótona a lo largo del tiempo. Existen distintos factores que influyen en la calidad de la red, como puede ser tráfico cruzado o cuellos de botella entre otros que hacen variar la calidad de la conexión a lo largo del tiempo entre el cliente y el servidor. Esta variación de la calidad de la red provoca que el servicio multimedia ofrecido por el proveedor pueda verse seriamente afectado o incluso interrumpido. Para evitarlo, se plantean

dos posibilidades, cambiar la red, lo que resulta muy costoso, o cambiar el contenido del servicio para que se adapte a los nuevos requisitos.

Por otro lado, atendiendo a la calidad del contenido recibido, desde hace unos años, los proveedores de servicios multimedia se centran en maximizar la calidad percibida por el usuario. No obstante, se trata de una medida subjetiva, por lo que la monitorización de esta calidad percibida, necesita una gran cantidad de recursos y resulta muy compleja. Para solucionarlo, se han desarrollado algoritmos capaces de pronosticar esta calidad de experiencia dada una entrada.

Este trabajo se centra en unificar ambos problemas, por ello se realizará un protocolo para monitorizar la red, y en función de la calidad de red medida, se realizarán cambios en la aplicación para tratar de maximizar la calidad percibida por el usuario.

1.2. Objetivos

El objetivo principal de este Trabajo de Fin de Máster es implementar el protocolo Q4S [1], y después desarrollar un actuador que ejecute cambios en la aplicación para ajustarla a los parámetros de red tomados por el Q4S. Posteriormente integrar ambos módulos y adaptarlos al proyecto Racing Drones. La estructura del trabajo realizado se corresponde a la figura 1.2. Para conseguir este resultado, los objetivos marcados han sido:

- Implementar el protocolo Q4S [1], que se encargará de medir en tiempo real diferentes parámetros de la red.
- Realizar un módulo actuador que, dadas unas características de red, envíe un mensaje HTTP a la aplicación.
- Estudiar el tráfico generado por la aplicación de Racing Drones y la QoE para las diferentes posibles salidas de la aplicación, de cara a maximizar la calidad de experiencia percibida por el usuario.
- Probar el funcionamiento del Q4S, realizando diferentes pruebas variando los parámetros QoS de la red con la herramienta netem en diferentes escenarios.

1.3. Fases de realización

Para la consecución de este proyecto, se han completado diferentes tareas, junto con sus respectivos entregables hasta llegar al objetivo global. Estas tareas siguen el diagrama de Gantt de la figura 1.1, que a continuación se exponen de manera más detallada.

- Realización del protocolo Q4S. El entregable de esta fase será un programa que ejecute dicho protocolo.
- Diseño de programas de automatización de pruebas estáticas y primeros resultados. Al final de esta tarea se entregará un documento en el que se detallen los resultados de las pruebas estáticas realizadas.

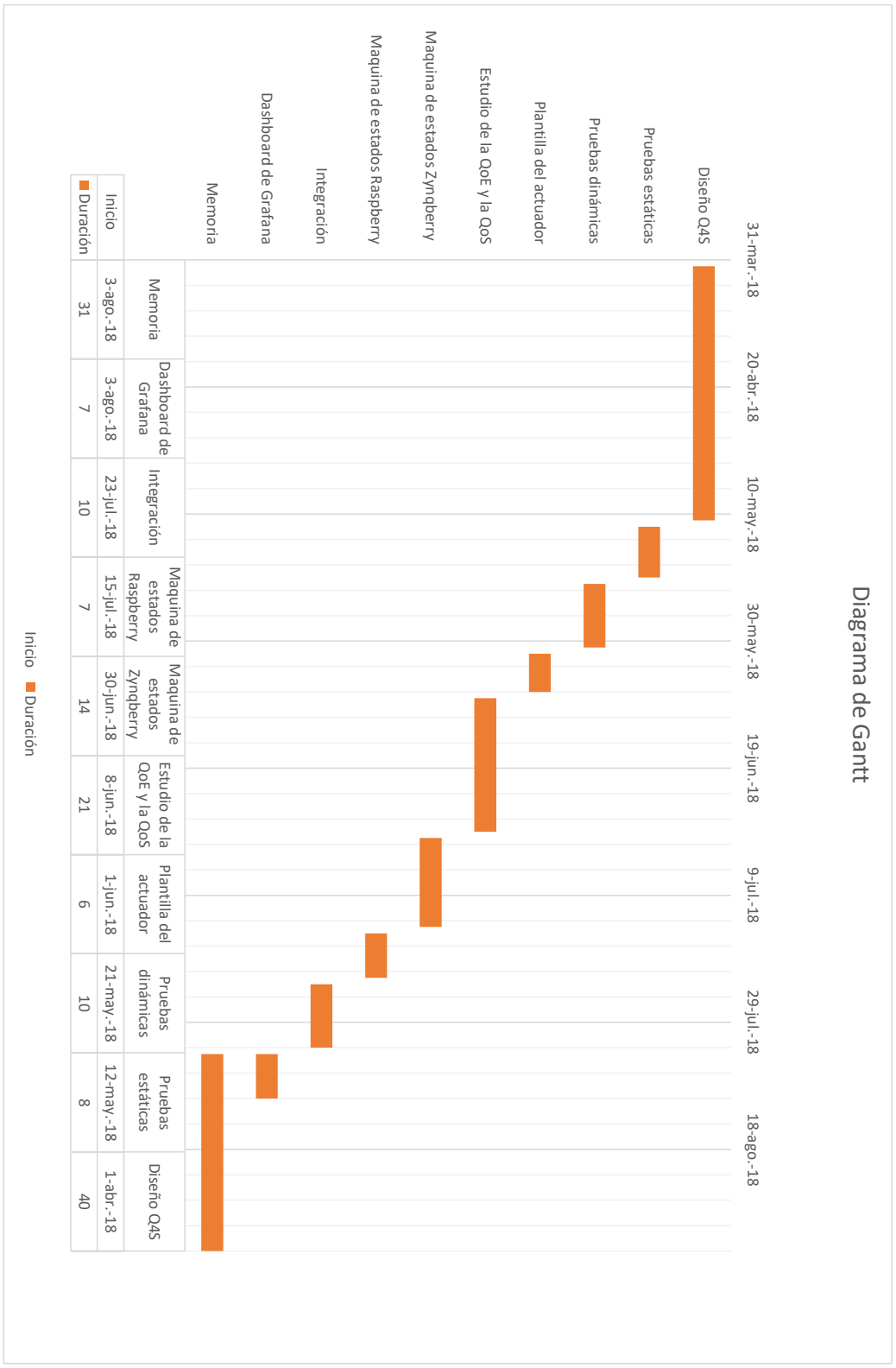

Figura 1.1: Diagrama de Gantt.

- Diseño de programas de automatización de pruebas dinámicas y primeros resultados. El entregable será un informe detallando los resultados de dichas pruebas.
- Realización de un módulo actuador genérico, el cual no efectúa ninguna función, sino que se buscará que esté al tanto de las restricciones medidas por el Q4S. El entregable de esta tarea será una esqueleto del actuador, el cual se conectará al programa del Q4S generado en la primera tarea.
- Estudio del tráfico generado por el códec LHE y del efecto de parámetros de la calidad de servicio sobre la calidad de experiencia. Se entregará un informe con los detalles del estudio, probado con diferentes configuraciones del LHE.
- Diseño de la máquina de estados del actuador para Zynqberry. Con los entregables de las tareas anteriores, se realizará un documento con la máquina de estados que tendrá lugar en la Zynqberry.
- Diseño de la máquina de estados e implementación de los actuadores para la Zynqberry y creación de un módulo para realizar los cambios en la aplicación. Los entregables serán dos actuadores funcionales en la Zynqberry.
- Diseño e implementación de la máquina de estados para Raspberry. Se entregará un módulo actuador compatible con la Raspberry.
- Integración de la aplicación, el protocolo y el actuador tanto en Zynqberry como en Raspberry. Los entregables serán un programa para cada dispositivo que ejecute todos los módulos interconectados entre sí.
- Desarrollo de un Dashboard de Grafana. En el final de esta tarea se entregará una interfaz donde se visualizará el comportamiento del programa implementado.
- Redacción de la memoria. El entregable final será la memoria del trabajo realizado.

1.4. Estructura del documento

En los siguientes capítulos, se explicará el proceso seguido para la consecución del trabajo, así como los resultados obtenidos, como se muestra en la figura 1.2. La memoria se estructura de la siguiente manera:

- El presente capítulo 1 incluye la exposición de los motivos por los que se ha realizado este proyecto, los objetivos que se han perseguido, sus fases de realización y una breve estructuración de la memoria.
- En el capítulo 2 se realiza un breve resumen de la documentación necesaria para la elaboración de este trabajo. Se explicarán los conceptos de QoS, QoE, y cómo se relacionan. Sobre el QoS, se comentan los parámetros a medir y cómo se realizan estas medidas, donde se hará hincapié en las utilizadas en este Q4S, y también qué factores afectan y cómo a la QoS. De la QoE se dará definición al concepto, detallando diferentes tipos de QoE, como son la QoE de vídeo sobre IP y QoE de jugabilidad.

- En el capítulo 3 se exponen algunas características del proyecto Racing Drones para poder dar un marco al trabajo. Para ello se comentan los fundamentos principales del codificador que se utiliza y la forma de empaquetarlos. También se describe la topología de la red utilizada.
- El capítulo 4, se habla del sistema implementado, desde el Q4S hasta el actuador, y muestra el estudio realizado del QoS y QoE para la aplicación en el proyecto Racing Drones. Se describe, también, la integración de ambos módulos en los distintos dispositivos (Raspberry y Zynqberry).
- En el capítulo 5 se detalla el procedimiento llevado a cabo en las pruebas realizadas y los resultados obtenidos en ellas.
- Por último, en el capítulo 6, se plantean las conclusiones sobre el trabajo realizado en función de las pruebas y los resultados expuestos en el capítulo anterior, además de la propuesta de un trabajo futuro.

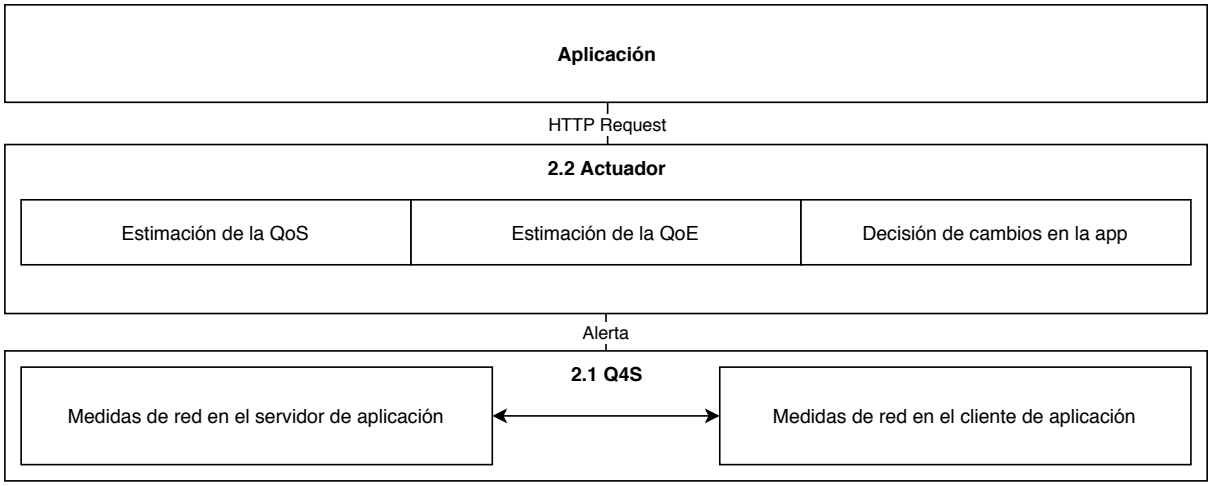

Figura 1.2: Estructura del trabajo.

2

Estado del arte

En este capítulo se expondrán los conceptos teóricos que fundamentan el trabajo realizado y son necesarios para la comprensión del mismo. En la figura 2.1 se visualiza cómo está estructurada esta sección. Inicialmente se explicará qué se entiende por calidad de servicio (*Quality of Service*,QoS), los parámetros que la definen, cómo se obtienen dichos parámetros y se describirá el protocolo Q4S, siendo éste el utilizado en el proyecto. Posteriormente se expondrá el concepto de calidad de experiencia (*Quality of Experience*,QoE), para después diferenciar entre la QoE para vídeos sobre IP y la QoE para la jugabilidad. Por último, se definirán los modelos para relacionar QoE y QoS.

2.1. Calidad de servicio (QoS)

Se trata de un concepto utilizado para describir la calidad de un servicio, entendido la calidad de servicio como la capacidad de intercambiar información a través de un medio de telecomunicaciones. Existen diferentes enfoques. Inicialmente se entendió la QoS como QoS

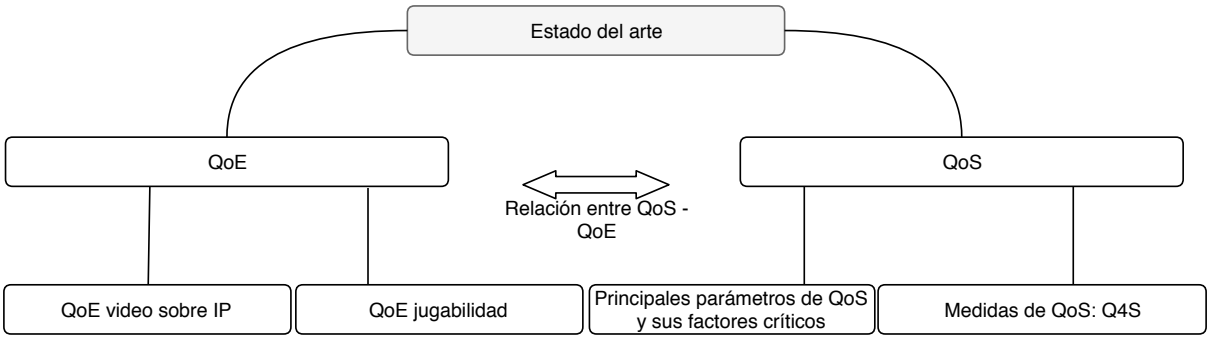

Figura 2.1: Estructura del estado del arte.

extremo a extremo, aunque, hoy en día la QoS orientada a la red está cobrando gran importancia, como en [2], donde definen la QoS de la red como el rendimiento del camino entre el servidor y el cliente.

Según se detalla en [3], en la cual el ITU-T asume el enfoque de QoS *end-to-end*, la QoS es "*la totalidad de las características de un servicio de telecomunicaciones que tienen que ver con su capacidad de satisfacer las necesidades tanto explícitas e implícitas del usuario del servicio*", mientras que el IETF entiende la QoS desde un enfoque del rendimiento de la red, como define en [4]: "*Un conjunto de requisitos a ser cumplidos por la red durante el transporte de un flujo*".

En este trabajo, el enfoque que se ha utilizado ha sido el de QoS orientado a red. Por tanto, a continuación se van a explicar diferentes parámetros que se deben monitorizar en una red para tener una medida de QoS fiable, y seguidamente se detallarán algunos métodos para realizar esta monitorización.

2.1.1. Principales parámetros de QoS y sus factores críticos

Existen multitud de parámetros medibles que caracterizan una red, aunque en esta subsección se van a detallar aquellos que serán empleados en la realización del TFM. Estos parámetros en concreto son: latencia, jitter, ancho de banda y pérdidas.

Latencia

La latencia es el retardo que añade una red. Hay diferentes interpretaciones de la latencia, entre las que conviene destacar el "*One Way Delay*"(OWD) y el "*Round Time Trip*"(RTT).

One Way Delay es el tiempo que pasa entre el envío del primer bit de un paquete y la recepción del último bit de este paquete en el destino. En otras palabras es el retardo medido en un solo sentido [5]. Para poder medir el OWD, es necesario tener sincronizados los relojes del emisor y el receptor. Estos entornos sólo suelen darse en laboratorios, ya que con máquinas reales, que suelen encontrarse alejadas, no es tan sencilla su sincronización. No obstante, existen protocolos que tratan de sincronizar los relojes de Internet, como es el caso de NTP [6], aunque no es suficientemente preciso para los requisitos que imponen los servicios interactivos.

Por otro lado, el *Round Time Trip* se define como el intervalo de tiempo entre el envío del primer bit de un paquete TCP y la recepción del último bit del ACK de respuesta del destinatario. También es extensible a otros protocolos bidireccionales. Además, si se asume simetría en el camino del emisor al receptor y viceversa (el tiempo de ida y vuelta es el mismo para ambos) se puede calcular el OWD como la mitad del RTT.

La latencia suele tomarse realizando medidas activas, ya que estimarlas de manera pasiva monitorizando conexiones TCP (u otro protocolo bidireccional) requiere conocer otros protocolos a nivel superior que se estén usando, para relacionar peticiones y respuestas.

Jitter

El jitter tiene diferentes significados dependiendo del contexto en que se utilice. En el contexto de QoS, el jitter es la variación de la latencia entre los distintos paquetes de un flujo de

paquetes. Se puede asumir que el jitter es la diferencia de OWD. Esta variación afecta mucho a la calidad de servicio ya que repercute directamente en el tiempo entre llegada de paquetes. Como en [7], la definición que se va a adoptar es la dada en [8], que a su vez es la utilizada en el Q4S implementado en este proyecto. En ésta se define que, dados $N+1$ paquetes unidireccionales,$\{l_i\}_{i=0}^{N}$ se pueden obtener N diferencias, $\{\delta_j\}_{j=1}^{N}$:

$$\delta_j = |l_j - l_{j-1}| \tag{2.1}$$

Por lo que se tomará el jitter como la media aritmética o la media de los $\{\delta_j\}$. También se puede estimar el jitter como la desviación estándar de $\{l_i\}$.

Ancho de banda

Se trata de la cantidad de información que se puede transmitir por un canal por unidad de tiempo. Existen diferentes definiciones, dependiendo de qué ancho de banda se quiere medir, en qué segmento, etc. En este estado del arte se va a diferenciar principalmente entre ancho de banda entendido como capacidad total del enlace y ancho de banda disponible, que es la capacidad de canal que se puede utilizar.

La capacidad total de una red se corresponde con el máximo ancho de banda que se puede ofrecer sin tener en cuenta el tráfico cruzado. El ancho de banda máximo de una red se corresponde con la capacidad del enlace con menor ancho de banda total entre todos los distintos enlaces de la red [9].

El ancho de banda disponible en la red también se refiere al enlace con menos capacidad disponible del camino, por lo que de ahora en adelante se hablará del ancho de banda disponible en un enlace. Es la capacidad de canal que no está siendo ocupada por otro tráfico en la red. Esta medida varía con el tiempo, ya que está relacionada con la utilización individual de cada enlace dando diferente cantidad de tráfico cruzado. El ancho de banda ocupado de un enlace $U(t, t+\delta)$ para un intervalo de tiempo $[t, t+\delta]$ puede expresarse como en [10]:

$$U(t, t+\delta) = \frac{1}{\delta}\int_{t}^{t+delta} U(x)dx. \tag{2.2}$$

Por lo que el ancho de banda disponible de ese enlace $B_\delta(t)$ es la media de la capacidad no utilizada en ese intervalo de tiempo:

$$B_\delta(t) = C(1 - U(t, t+\delta)). \tag{2.3}$$

Pérdida de paquetes

La pérdida de paquetes es una de las principales causas del descenso de la QoS. Se define como la cantidad de paquetes perdidos en un periodo de tiempo [11]. Se incluyen como perdidos los paquetes que sobrepasan un determinado tiempo sin llegar y aquellos que llegan defectuosos (más común en conexiones inalámbricas).

Existe una diferencia muy grande entre los servicios basados en TCP y los servicios basados en UDP, ya que los TCP pueden utilizar la retransmisión para recuperar dichos paquetes. No

obstante, es cierto que dicha retransmisión puede tardar un tiempo demasiado elevado y ser contabilizado como pérdida.

Las aplicaciones a tiempo real suelen estar basadas en UDP, por lo que se pueden ver fuertemente afectadas por este parámetro. Un ejemplo es el protocolo en tiempo real (RTP), el cual se usa para el envío de contenido multimedia sobre UDP.

Factores críticos

Estos parámetros adquieren un significado distinto dependiendo del tipo de enlace. Existen diferencias entre las características de una conexión inalambrica y las características de una conexión por cable. Como ejemplo, la latencia en una red cableada puede significar un camino lento entre los dos terminales, mientras que la misma latencia en una red wifi puede suponer que está habiendo pérdidas que están forzando el reenvío del mensaje.

Un factor crítico en ambos tipos de red es el tamaño de paquete. En [12] se realizó un estudio para observar la relación entre el tamaño de los paquetes y las pérdidas en un router. Se centra en las redes cableadas, asumiendo que los únicos puntos de la red en los que la conexión tiene posibilidades de perder paquetes es en los routers. Para realizar este estudio se enviaron dos tipos de tráfico, uno con un tamaño de paquete fijo y un segundo flujo con un ancho de banda igual la capacidad total del enlace de cara a saturarlo, ya que si el ancho de banda disponible es alto no debería haber pérdidas. Este segundo flujo incluía tanto paquetes grandes como paquetes pequeños y medianos para no influir en los resultados obtenidos. Además, para caracterizar mejor el funcionamiento real de los routers comerciales, se realizó el estudio con dos tipos diferentes de buffer. El primero de ellos mide el tamaño del buffer en bytes, mientras que el segundo lo mide en número de paquetes. Si no hay espacio para encolar los paquetes, éstos son descartados. Las ráfagas de paquetes mandados de tamaño fijo acababan aproximadamente cuando se habían enviado 25 000 paquetes. En los resultados se puede ver como para los buffer con tamaño medido en bytes, se observan más pérdidas cuanto más grande es el paquete. Sin embargo, para tamaños de buffer centrados en el número de paquetes, no se observa diferencia al variar el tamaño del paquete. Por tanto, en este trabajo se asume que cuanto más grande sea el paquete, más posibilidad de que se pierda.

Con respecto a las conexiones inalámbricas, el tamaño del paquete también resulta bastante importante. Tal y como se expone en [13], un tamaño de paquete más grande, puede provocar que al menos uno de sus bits sea erróneo. Saca esta conclusión mediante un estudio teórico sobre la posibilidad de que un bit esté corrupto. Además, dado que cuando llega un paquete corrupto al router dicho paquete se vuelve a enviar, el estudio también modela la latencia que habrá en función del tamaño.

2.1.2. Medidas de QoS: Q4S

Para llevar un control sobre los niveles de la QoS, es necesario realizar una monitorización. Existen diferentes formas de realizar estas medidas, en función de la participación en el tráfico presente en la red. A pesar de que hay otras medidas, este trabajo, basándose en [14], se va a centrar en las medidas pasivas y las medidas activas.

Medidas pasivas

Las medidas pasivas de red son las medidas en las que se observa el tráfico llegado a un dispositivo y, a raíz de ese tráfico, se extraen las características de red. Estas medidas se basan en la idea de recolectar datos y después procesarlos de cara a estimar los parámetros de red y analizar el rendimiento de la misma. Por otro lado, las medidas de red suelen obtenerse en dispositivos dedicados o es desarrollado en routers y switches. Esta información puede ser recolectada desde cualquiera de las siguientes categorías:

1. Desde el tráfico capturado: como las trazas PCAP o los paquetes capturados por algún otro tipo de hardware.

2. Información pre-procesada y estadísticas recogidas por los dispositivos.

Dependiendo del tipo de información que se recoja se obtendrá un resultado diferente de monitorización. Unas características de bajo nivel permiten obtener estimaciones de algunos parámetros de QoS, como pérdida de paquetes, número de flujos diferentes, etc.

La principal ventaja de las medidas pasivas es la no intrusión en la red. Generalmente, se puede capturar el tráfico sin apenas interferir en la producción de tráfico y, aunque a veces se inserte cierto tráfico extra, estas medidas son mucho menos intrusivas que las activas. Sin embargo, calcular los parámetros QoS de una manera precisa resulta muy difícil debido a la variación de las condiciones de red y el intervalo de adquisición de información. Por tanto, si se tiene un intervalo de recogida de información muy amplio, cambios muy abruptos en un corto periodo de tiempo no van a ser detectados; mientras que si se tiene un intervalo de tiempo muy pequeño, el coste computacional y el tiempo dedicado se incrementa considerablemente. Existen dos técnicas predominantes para realizar medidas pasivas: medidas a nivel de flujo y medidas a nivel de paquete. Éstas medidas a nivel de flujo, según [15], mejoran en rendimiento aunque empeoran en precisión frente a las medidas basadas en paquete.

Medidas activas

Las medidas activas son técnicas que se basan en introducir tráfico a la red con el fin de caracterizar la misma. Cada aproximación puede aplicarse a diferentes escalas de red, ya sea extremo a extremo o por enlace para obtener estadísticas fiables y determinar la calidad observada en un segmento de red. Esta forma permite medir directamente lo que los operadores quieren observar sin la necesidad de esperar a un evento. Por eso es necesario determinar dos parámetros que definan estas medidas, como son el tamaño de la información que se inyecta en la red y la frecuencia con la que se harán las pruebas.

Algunos parámetros de QoS, como puede ser *One Way Delay*, ancho de banda disponible, capacidad de enlace o pérdida de paquetes, por norma general son medidos con este método. Para el cálculo del OWD , como se ha explicado anteriormente, se necesitaría una sincronización entre los relojes de los dos terminales que se quieren medir, proceso que resulta bastante complejo y no del todo preciso. Además, tanto el ancho de banda disponible como la capacidad del enlace sólo pueden conocerse utilizando medidas activas, ya que rara vez el tráfico alcanza el 100 % de la capacidad de canal.

Las medidas activas tienen el inconveniente de que resultan invasivas, puesto que inyectan tráfico a la red, pudiendo influenciar en las características de la red. Por ello, para monitorizar el comportamiento normal de la red, es necesario minimizar el impacto sobre el tráfico normal y el comportamiento de las operaciones de red. Sin embargo, si lo que se desea es caracterizar el mal uso o disponibilidad de la red cuando ésta está saturada, se deberá estresar la red mediante dicha inyección de tráfico.

A continuación se explicarán tres de las técnicas más generalizadas que componen el estado del arte de las medidas activas.

1. Descarga de fichero: se trata de una técnica definida en la *European Standards Institute* (ETSI)[16]. Consiste en paralelamente subir y descargar un archivo entre una web y el usuario final. Estima la QoS utilizando una descarga HTTP. El fichero descargado debe ser 8 veces el ancho de banda del enlace, y haber sido generado aleatoriamente para evitar que haya una optimización del rendimiento en el servidor web. La principal ventaja de este método es la sencilla implementación que conlleva. Las características de esta técnica y del fichero a descargar suponen también que el tiempo requerido para estas medidas sea de mínimo 8 segundos, lo que conduce a la conclusión de que el tráfico cruzado tiene una influencia muy grande.

2. Par de paquetes: Esta técnica trata de estimar la capacidad de un camino mediante la dispersión de dos paquetes del mismo tamaño enviados *back-to-back* [17]. Se envían múltiples pares de paquetes desde el origen al destino para calcular los parámetros QoS. En el lado del receptor cada paquete es analizado para estimar la capacidad de enlace. Sin embargo, esta técnica conlleva una sobreestimación o subestimación de la capacidad real debido al tráfico interferente [18].

3. Tren de paquetes: dadas las limitaciones en la varianza de la capacidad de la técnica anterior, se plantea como solución la técnica de tren de paquetes [19]. En esta se envía un grupo de N paquetes *back-to-back* desde el emisor al receptor. La dispersión media de esos N elementos se utiliza para calcular la capacidad extremo a extremo del camino o enlace [18]. Además esta técnica permite estimar también latencia, pérdidas y jitter si se incluyen las marcas de tiempo de los paquetes.

Este trabajo se centra en las medidas de red realizadas por el protocolo Q4S, puesto que es el que se va a implementar. El Q4S realiza medidas activas de red poco intrusivas. Estas medidas de red son en función del parámetro a medir, las siguientes:

1. Latencia: para medir la latencia, Q4S utiliza el método PING para intercambiar paquetes entre el cliente y el servidor. Basándose en el intercambio de PING y la respuesta entre cliente y servidor, se calcula el RTT. Para ello, se necesita un intercambio mínimo de 255 muestras. La latencia se calcula como la media del envío de cada PING y la respuesta del mismo. Por ello, se envía el PING con el timestamp (marca temporal) en la cabecera del mensaje, y cuando se recibe la respuesta se guarda el timestamp de la recepción. Esto se hace con la finalidad de que no influya la desincronización entre los relojes de los distintos equipos.

2. Jitter: las medidas de jitter se realizan con todos los intervalos de tiempo entre la llegada de los PINGs. No se tiene en cuenta el tiempo de respuesta. Para esto, tanto cliente como servidor deben enviar estos mensaje PING con una periodicidad fija.

3. Ancho de banda: en este protocolo no se mide el ancho de banda disponible ni la capacidad, se mide si la comunicación entre el cliente y el servidor alcanza un ancho de banda determinado. En otras palabras, busca comprobar si el ancho de banda disponible por la red es igual o mayor que el ancho de banda demandado. Para ello, tanto cliente como servidor se intercambian mensajes UDP simultáneamente.

   Los mensajes UDP intercambiados deben tener un tamaño de 1 KByte o superior. Además, deben incluir un número de secuencia. El método empleado para realizar las medidas de ancho de banda es el siguiente: dado un ancho de banda B que se quiere alcanzar, se enviarán 1000 trenes de N número de paquetes con una longitud L fijada (un tren por milisegundo), por lo que $N = \frac{B}{L*1000}$. En el cálculo se contará el número de paquetes recibidos, y se dividirá entre el intervalo de tiempo entre el primer y último paquete UDP de la medida de ancho de banda recibido.

4. Pérdida de paquetes: la medida de pérdida de paquetes se realiza simultáneamente con la medida de ancho de banda. La pérdida de paquetes se calcula gracias al número de secuencia que se incluye en la cabecera de estos mensajes UDP. El cálculo del porcentaje de paquetes perdidos es:

$$PL\,\% = \frac{NUM\_SEQ_{MAX} - \#PAQUETES\_RECIBIDOS}{NUM\_SEQ_{MAX}} 100 \tag{2.4}$$

2.2. Calidad de Experiencia (QoE)

La Quality of Experience (QoE) está definida por la ITU como [20] *la aceptabilidad general de una aplicación o servicio, tal como lo percibe subjetivamente el usuario final.* Afecta al sistema end-to-end, incluyendo el cliente, la terminal, la red, etc.

Como se matiza en [21], esta medida depende de la percepción visual o auditiva humana, y del grado de satisfacción que el usuario final siente usando un determinado servicio. Esta dependencia nos lleva a la conclusión de que se trata de un parámetro totalmente subjetivo. Además de esta subjetividad, dicha calidad percibida puede variar por diferentes factores independientes del servicio o aplicación, como son el contexto en el que se mide la QoE, el ambiente y otros factores. Una correcta medida de la QoE supone realizar varios test al usuario, lo que conlleva un consumo de tiempo elevado, así como procesos muy costosos.

A pesar de que los parámetros que definen la QoS dependen de la aplicación, en esta sección tomaremos la clasificación dada por [22], donde son agrupados en tres clases:

1. La calidad del servicio/aplicación en la fuente.
2. La calidad de la red.
3. La percepción humana, donde se incluyen factores externos que afectan a ésta.

Mientras que los dos primeros son objetivamente mensurables, la tercera resulta un poco más difícil de medir. Esta última suele expresarse como la puntuación de la opinión media (MOS, (*Mean Opinion Score*). El MOS es un sistema de medida en el que la ITU-T [23] estandarizó una escala del 1 al 5, en el que 5 es “excelente”, 4 es ”buena”, 3 es razonable”, 2 es “pobre” y 1 es ”mala”.

2.2.1. QoE para vídeo sobre IP

Los servicios de vídeo deben de cumplir unas expectativas del usuario diferentes de otros servicios. La percepción del usuario va a ir muy relacionada a eventos como un tiempo de carga muy elevado, interrupciones o un descarte de paquetes que afecte a la percepción del vídeo. Para medir este grado de satisfacción, se realizan métricas subjetivas sobre la calidad de experiencia percibida teniendo en cuenta estos eventos. Además, como se explica en la sección 2.3, existen también medidas objetivas que tratan de predecir la QoE en función de los parámetros de QoS.

Las medidas subjetivas están basadas en la recolección de la información dada por los usuarios en vista de su experiencia con el servicio. Los resultados de estos test subjetivos se utilizan normalmente como *ground truth* para la validación del rendimiento de los modelos de medida de calidad objetivos. El procedimiento normal de este tipo de medidas es el siguiente:

1. Preparación del test. El test debe ser probado en el laboratorio con condiciones controladas y en un ambiente de casa, el cual, es más difícil de medir. Estas condiciones, tanto las controladas en el laboratorio, como las de un ambiente de hogar, vienen recomendadas por la ITU-R BT. 500-11. Con respecto a la selección del vídeo fuente, se deben tener diferentes factores en cuenta, como el color, la luminancia y otras características espaciales y de movimiento. Otro punto determinante es la elección de evaluadores. Se requieren al menos 15 evaluadores no expertos, de los cuales se debe tener en cuenta sus características y cierta información personal (edad, sexo, ocupación, etc.).

2. Ejecución del test. La ejecución del test abarca desde la conducción de la prueba hasta la recolección de los datos. En la ejecución, el orden de la presentación de los vídeos debería ser aleatorio, de cara a cubrir el máximo de posibilidades bajo estudio. Existen diversos métodos que pueden ser utilizados. En el método de escala de doble estímulo, inicialmente se muestra el vídeo original y posteriormente el procesado. En el método de escala de un único estímulo, solo se muestra el vídeo procesado. Por último, en el método de comparación de estímulo, se comparan dos vídeos procesados.

3. Procesado de datos. El procesado incluye verificar la integridad de los datos y el descarte de valores atípicos y evaluadores poco convincentes. La puntuación dada por los evaluadores puede darse en distintas métricas. El MOS, por ejemplo, es para test con un sólo estímulo. Por otro lado, el DMOS, que es la diferencia entre MOS, es para test de doble estímulo.

2.2.2. QoE: jugabilidad

En esta subsección se van a analizar los juegos online y la jugabilidad de éstos según los parámetros QoS. Además, se pondrá el foco especialmente en los llamados juegos de tirador en primera persona (*First Person Shooter*, FPS), ya que se asume que requieren unas características parecidas, o incluso más restrictivas que las demandadas en la aplicación del proyecto Racing Drones.

Existen diferentes estudios, como [24], donde unos usuarios jugaban a dos juegos en un entorno controlado. En este entorno, se iban introduciendo pérdidas y latencia a las distintas partidas. Finalmente se preguntaba a los jugadores por la calidad percibida. Las principales conclusiones extraídas fueron las siguientes:

1. Diferentes niveles de QoS provocan un desequilibrio en los juegos donde no hay mecanismos para mitigar estas diferencias de QoS.

2. La percepción de las pérdidas depende de los mecanismos que tenga el juego para paliarlas, ya que en este caso se utilizaban dos juegos distintos y uno de ellos dejaba de funcionar con el 4 % de pérdidas, mientras que el otro juego funcionaba hasta con un 35 % de pérdidas.

3. La latencia tiene un efecto más severo sobre la QoE que las pérdidas.

También merece la pena incluir otro estudio que incluía el efecto del jitter y de la latencia [25]. En éste, mientras un conjunto de jugadores juegan, van variando la latencia y el jitter de la red. La pruebas se llevaron a cabo en diferentes juegos: un FPS, uno de deportes y un juego de carreras. Además, aparte de preguntar a los usuarios acerca de la calidad percibida para obtener el MOS, miden la puntuación media de un jugador (*Game Outcome Score*, GOS).

$$GOS_i = \frac{score_i}{\sum_{j=1}^{n} score_j} 100 \tag{2.5}$$

De este estudio se dedujo que la latencia tiene un efecto muy significativo en el MOS, mientras que en el GOS no se aprecia dicho empeoramiento. En cuanto al jitter, los jugadores si empiezan a percibirlo cuando este es muy alto, sin embargo no influye en gran medida en la calidad percibida. Con respecto al GOS, el jitter resulta completamente irrelevante.

Dados los resultados de los estudios anteriores, se puede decir que el parámetro de QoS que más influye es la latencia. Debido a esto, en [26], se concretan ciertas recomendaciones sobre los retardos límites para distintas aplicaciones. Para juegos online en tiempo real, basándose en [27], asume tres categorías distintas, según el punto de vista del jugador:

1. Omnipresente. Este tipo de juegos es el que tiene una mayor resistencia a la latencia. Tiene un umbral de latencia aceptable (latencia en la cual el 75 % del rendimiento sigue intacto) de 1000 ms. El género más representativo son los juegos de estrategia.

2. Avatar en tercera persona. En esta perspectiva se controla un avatar, aunque con una visión, por lo general, de vista de pájaro sobre el mismo. Tiene un umbral de latencia aceptable de 500 ms.

3. Avatar en primera persona. Son los más restrictivos. Tienden a exigir una mayor precisión que los de tercera persona, ya que la perspectiva en primera persona exige una respuesta inmediata. Es por ello que la máxima latencia aceptable es 100 ms.

Con todo ello, y debido a las características del proyecto Racing Drones, el cual podemos clasificar como un juego en primera persona, podemos concluir que para mantener la jugabilidad deberemos prestar especial atención a la latencia. Además, al ser un juego en el mundo real, no se podrán implementar mecanismos con los que sí se cuenta cuando el entorno es virtual, como la predicción de movimiento.

2.3. Relación entre QoS y QoE

La relación entre QoS y QoE es difícil de determinar ya que la calidad percibida del vídeo es subjetiva y puede cambiar según el ambiente. Realizar una correlación entre la QoS y la QoE resulta complejo, ya que, dependiendo del tipo de vídeo, un parámetro de QoS puede afectar más que otro [28]. Sin embargo, sí que existen distintos estudios que tratan de estimar la QoE dados unos parámetros de QoS. Para ello, los modelos de calidad objetivos calculan las métricas en función de parámetros QoS y otros factores externos. Existen diferentes maneras de obtener las métricas de la QoE, que en [29] se clasifican en tres grupos:

1. Referencia completa (*Full Reference*, FR): se compara el vídeo original, sin distorsiones, con el vídeo recibido.
2. Sin Referencia (*No Reference*, NR): solo está disponible para el usuario el vídeo recibido, haciendo la estimación de la calidad sin la referencia.
3. Referencia reducida (*Reduced reference*, RR): los parámetros vienen dados tanto del vídeo original como del vídeo recibido.

Los modelos de referencia total y referencia reducida necesitan la comparación con los vídeos originales, lo que provoca que estos modelos sean poco sostenibles para la estimación de la QoE online. Ambos modelos son intrusivos, ya que agregan carga adicional e innecesaria a la red o servicio. El modelo sin referencia no añade ninguna carga, por lo que es un modelo no intrusivo, y más favorable para realizar estimaciones online. Por ello, la velocidad y el rendimiento de estas estimaciones son factores críticos a favor de este modelo.

Como clasificación según la entrada, la ITU propone la agrupación de los distintos métodos en cinco clases [30]:

1. Modelos de capa de medios: utilizan la señal de vídeo para predecir la QoE. No requieren información a priori del sistema bajo prueba, como el tipo de codificador o la tasa de pérdida de paquetes. Se puede usar para evaluar sistemas desconocidos (por ejemplo, comparación u optimización de códecs).
2. Modelos de la capa de parámetros del paquete: predicen la QoE gracias a la información de la cabecera del paquete, lo que posibilita una medida muy ligera. La cabecera no contiene la señal de medios por ella misma. Por tanto, conlleva una difícil evaluación de la QoE dependiente del contenido.
3. Modelos de planificación paramétrica: se toman los parámetros de calidad para redes y terminales como su entrada. Requiere información a priori acerca del sistema bajo pruebas.
4. Modelos de la capa de flujo de bits: se encuentran entre los modelos de capa de medios y los modelos de la capa de parámetros del paquete. Utilizan información del flujo de bits codificado, además de la información de la capa de paquete, de modo que puede tener en cuenta las características de evaluación de calidad dependientes del contenido, con una carga de cálculo relativamente ligera.
5. Modelos híbridos: son una combinación de los modelos previos. Son efectivos en términos de explotación de la máximo cantidad de información posible para predecir la QoE.

Un ejemplo de estos modelos objetivos es VMAF (*Video Multimethod Assessment Fusion*)[31]. Se trata de una herramienta que predice la calidad subjetiva combinando múltiples métricas de calidad elementales. El principio básico es que cada métrica elemental puede tener sus propias fortalezas y debilidades con respecto a las características del contenido fuente, el grado de distorsión, etc. Gracias al fusionado de métricas elementales, VMAF obtiene una métrica final usando machine learning, concretamente Support Vector Machine, el cual asigna pesos a las distintas métricas, proporcionando un valor final más preciso. Además el modelo es entrenado y probado usando la puntuación de opinión media obtenida a través de un experimento subjetivo. Las métricas elementales empleadas por VMAF son:

- Fidelidad de la información visual (VIF). VIF es una medida de calidad de imagen bien adoptada basada en la premisa de que la calidad es complementaria a la medida de la pérdida de fidelidad de la información. En su forma original, la puntuación VIF es medida como la pérdida de la fidelidad combinando cuatro escalas. Sin embargo, en VMAF se adopta una versión modificada de VIF, en la cual la pérdida de fidelidad en cada escala se asume como métrica elemental.

- Métrica de pérdida de detalle (DLM). Es una métrica de calidad de imagen basada en el fundamento de medir de forma separada las pérdidas de detalle que afectan la visibilidad del contenido y el deterioro redundante que distrae la atención del espectador. Las métricas originales combinan la DLM con otras medidas que miden el deterioro para alcanzar el resultado final. Esta herramienta tan solo usa el DLM como métrica elemental. Se prestó especial atención a casos especiales, como cuadros negros, donde los cálculos numéricos para la formulación original se descomponen.

  Tanto VIF como DLM son métricas de calidad de imagen. Adicionalmente, en VMAF también se tiene en cuenta las características temporales del vídeo:

- Movimiento. Es una simple medida de la diferencia temporal entre frames adyacentes. Se consigue mediante el cálculo de la media absoluta de las diferencias entre el componente de luminancia de cada píxel.

Estas métricas y características elementales fueron seleccionadas entre otros candidatos a través de iteraciones de prueba y validación.

2.4. Conclusiones

En este estado del arte se han estudiado los diferentes parámetros, técnicas y factores que afectan a la QoS y la QoE. En concreto, los conceptos más importantes que conviene extraer son los siguientes:

- La QoS puede dividirse en dos enfoques principales, la QoS extremo a extremo y la QoS de la red. Dadas las características de este trabajo, nos centraremos en la QoS de la red. Los parámetros que vamos a medir de la red serán la latencia, el jitter, el ancho de banda y las pérdidas. También conviene distinguir entre ancho de banda disponible y capacidad total, siendo el ancho de banda disponible el más interesante para este TFM.

- Existen dos técnicas predominantes para realizar las medidas: activas y pasivas. La medidas activas son más precisas y más sencillas pero, sin embargo, son más intrusivas, pudiendo influir en los parámetros de la QoS de la red. El protocolo Q4S realiza medidas activas, sin embargo, trata de ser lo menos intrusivo posible.
- La QoE es la experiencia percibida por el usuario. Esta QoE se puede dividir entre la QoE del vídeo y la QoE aplicada a la jugabilidad.
- Con respecto a la jugabilidad, existen diferentes factores que son relevantes, poniendo el foco principalmente en la latencia.
- Dado que la percepción del usuario es muy costosa de estimar y requiere participación de usuarios, se han desarrollado diferentes técnicas que tratan de estimar esta QoE objetivamente, en especial, en este TFM, se utilizará la herramienta VMAF.

A partir de estos conceptos se ha desarrollado el trabajo expuesto en esta memoria.

3

Marco de trabajo

Este capítulo describe el entorno de trabajo. Como se explicaba en el capítulo 1, este trabajo está enmarcado en el proyecto Racing Drones. Racing Drones es un proyecto basado en un dron terrestre manejado vía IP.

Para este proyecto se cuenta con un códec LHE a nivel de hardware, que codifica las imágenes tomadas por la cámara y las envía al paquetizador. Este paquetizador inicia una sesión RTSP (Protocolo de transmisión en tiempo real) entre el servidor y el cliente, donde se inicia una sesión RTP [32] sobre UDP, en la cual el servidor envía al cliente el producto de la codificación del LHE.

3.1. Protocolos y codificador

El codificador utilizado por el proyecto Racing Drones es el códec LHE [33] (*Logarithmical Hop Encoding*). Se trata de un codificador que se caracteriza por funcionar en el dominio del espacio, tener una baja complejidad computacional y una latencia muy baja. Sin embargo, existen dos codec LHE diferentes sobre los que se va a trabajar.

El primero de ellos es el códec realizado por el grupo HPCN, el cual esta diseñado para funcionar a nivel de hardware. Para realizar la codificación, este códec divide el frame en bloques LHE de tamaño fijo $H \times W$, los cuales se codifican de manera independientemente unos de otros. Tanto la altura como el ancho de los bloques pueden ser modificados durante la ejecución de la aplicación de manera manual en el lado del servidor. Adicionalmente, cuenta con la posibilidad de codificar con distintos perfiles de color (YUV 4:0:0, YUV 4:2:0, YUV 4:2:2 o YUV 4:4:4) y con diversos *framerates*. Como se ha comentado antes, al funcionar en el dominio del espacio, codifican más o menos en función de la redundancia espacial. Por consiguiente, los bloques codificados van a tener un tamaño variable. A dichos bloques se les añade una cabecera específica, como es la mostrada en la figura 3.1, cuyo tamaño máximo es 8192 bytes. Como se puede ver en esta cabecera, se incluye el tipo de bloque y el tamaño de éste.

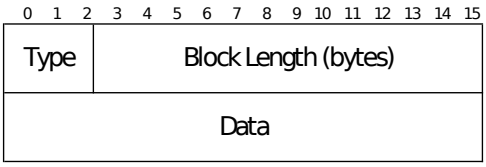

Figura 3.1: Cabecera del bloque LHE.

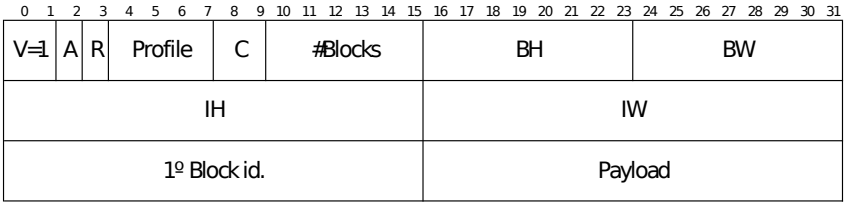

Figura 3.2: Cabecera del paquete LHE.

El resultado se encapsula en un paquete específico para LHE [34]. Este paquete tiene la cabecera de la figura 3.2, en la que los campos que más nos interesa son los siguientes:

1. Profile: indica el perfil de color utilizado.
2. C(codec): indica la versión de LHE utilizada.
3. # Blocks: número de bloques en el paquete.
4. BH: altura del bloque.
5. BW: ancho del bloque.
6. IH: altura en bloques de la imagen.
7. IW: ancho en bloques de la imagen.
8. 1st block id: identificador del primer bloque que aparece en el paquete. En caso de aparecer varios bloques solo se transmite el ID del primero.

Estos paquetes tienen un tamaño siempre menor de 1500 bytes, para así evitar la fragmentación en redes Ethernet, ya que se asume que la *Maximum Transmission Unit*(MTU) generalizada de Internet es de 1500 bytes.

Posteriormente, ese paquete se envía bajo el protocolo RTP. El formato de la cabecera RTP obedece a la figura 3.3; donde M es el marker bit, el cual indica el final de un cuadro. PT o payload type, indica el contenido de payload. P es una bandera que indica si hay padding, X una bandera que indica si hay extensión de cabecera y por último CC indica por cuántas fuentes se transmite el vídeo. Para el caso de paquetes LHE, deben ir a 0. Por último, sequence number es el número de secuencia asignado.

Por otro lado,en la Raspberry, el códec LHE con el que contaba estaba diseñado para software. Este códec codificaba la imagen línea a línea, no por bloques. Además, este códec cuenta con dos parámetros modificables en caliente que son el nivel de cantidad de líneas que se tiran y el número de frames que se descartan. La salida de este codificador se empaqueta en un paquete H264.

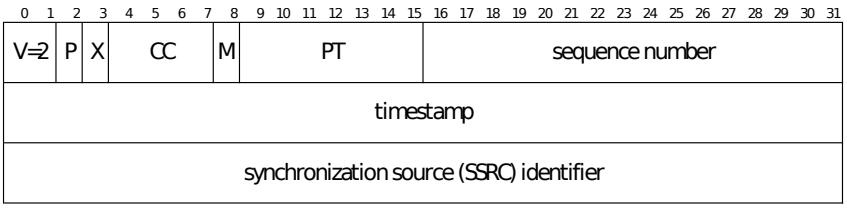

Figura 3.3: Cabecera de RTP.

3.2. Topología

La topología diseñada para Racing Drones es la correspondiente con la figura 3.4. Se trata de un dron terrestre que transmite el contenido a un servidor, el cual lo envía a través de la red del proveedor de servicios de internet (*Internet Service Provider*, ISP) hasta llegar al usuario final. Este usuario envía una orden de vuelta al coche.

Como se puede observar, existen dos enlaces principales en esta red. El primero de ellos, es el enlace entre el coche y el servidor, mientras que el segundo es el enlace entre el servidor y el usuario final. Como ambos enlaces son independientes entre si, se tomó la decisión de monitorizarlos de manera independiente. Para la realización de pruebas se caracterizará el enlace entre el coche y el servidor como un enlace inalámbrico mientras que el enlace entre el servidor y el usuario final se caracterizará como un enlace cableado. En este proyecto, esta monitorización se hace mediante el protocolo Q4S, el cual se explicará en el capítulo siguiente. Este protocolo cuenta con un cliente y un servidor. En el enlace inalámbrico se tomará el coche como servidor Q4S y el servidor como cliente Q4S, y para el enlace cableado el servidor será el servidor Q4S y el usuario final será el cliente. Ambos servidores Q4S cuentan con un actuador independiente. Cada actuador realizará los cambios que considere oportuno sobre la aplicación, mediante peticiones HTTP POST.

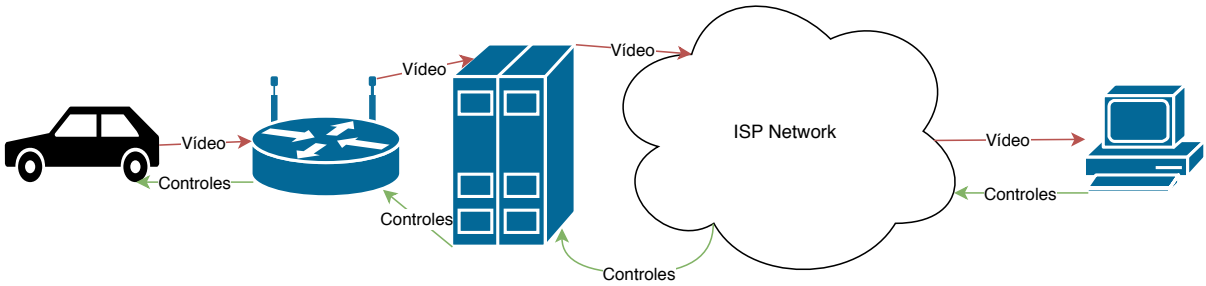

Figura 3.4: Topología de Racing Drones

3.3. Conclusiones

En este capítulo se ha expuesto el entorno en el que se va a trabajar. En concreto se ha descrito la aplicación sobre la que se diseñará el actuador, explicando la codificación y el empaquetado para los diferentes dispositivos. Asimismo, se ha explicado la topología de la red

de Racing Drones, en la cual se deberá llevar a cabo la monitorización con el protocolo Q4S. Además, la topología también se va a tener en cuenta para el diseño del actuador, que dependerá del escenario para el que actúe.

4

Sistema, diseño y desarrollo

En este capítulo se explica el sistema completo realizado. Dicho sistema consta de la implementación de un protocolo, en concreto el protocolo Q4S, encargado de medir la calidad de la red entre dos terminales. Por otro lado, en este sistema se ha desarrollado un módulo actuador encargado de ajustar la aplicación para que el contenido multimedia que transmite se ajuste a la calidad de red disponible, así como encargado de realizar cambios en caliente sobre dicha aplicación para tratar de ofrecer la máxima calidad de experiencia. Por último, se combinarán ambos módulos de forma que las medidas efectuadas por el protocolo Q4S, sean aprovechadas por el módulo actuador, así como que el protocolo mida los umbrales que demanda la aplicación sobre la que se centra este trabajo.

4.1. Quality for Service (Q4S)

El protocolo Q4S [1] fue desarrollado con la motivación de poder adaptar una aplicación en tiempo real (como juegos online, aplicaciones de vídeo interactivo, etc.) a las cualidades de la red, y reaccionar, lo más rápido posible, a cambios en la misma. Por ello, el Q4S debe tener la capacidad de trabajar conjuntamente con cualquier tipo de aplicación. Cabe destacar que el Q4S no establece ningún tipo de sesión multimedia, ni es aprovechada para el transporte de ninguna información de la aplicación, sino que se trata de un protocolo de comunicación orientado a mensajes. De esta manera, se posibilita que este protocolo pueda ser ejecutado en segundo plano y que su funcionalidad quede totalmente restringida a la monitorización de los requisitos de red entre un servidor y un cliente. Sin embargo, estas medidas realizadas sí deben de estar disponibles para que otra aplicación pueda hacer uso de ellas.

El protocolo de Q4S consta de cuatro etapas:

1. Saludo o *Handshake*: en esta etapa se establece la conexión entre cliente y servidor.

2. Negociación o *Negotiation*: donde se negocian las condiciones de red iniciales entre ambos.
3. Continuación o *Continuity*: etapa en la cual se realizarán de manera continua medidas para asegurar que las condiciones marcadas en la etapa anterior se están cumpliendo y, en caso contrario, notificarlo.
4. Terminación o *Termination*: en esta etapa se da por finalizada la conexión entre el cliente y el servidor.

Este protocolo se arrancará justo antes de iniciar una aplicación cliente-servidor que necesite cumplir requisitos de red relativos a latencia, jitter, ancho de banda disponible o pérdida de paquetes. Una vez que se hayan acordado las condiciones de red en la etapa *Negotiation*, se lanzará la aplicación y el protocolo Q4S continuará midiendo la red y alertando en caso de ser necesario.

Como se ha recalcado, este protocolo está diseñado principalmente para trabajar en segundo plano. Esto conlleva que el consumo de CPU del Q4S debe de ser muy bajo y, por otro lado, que el tráfico que genere no resulte demasiado intrusivo, en yuxtaposición a lo explicado en el Estado del arte (capítulo 2), en el que se exponía cómo las medidas activas eran intrusivas por definición.

4.1.1. Funcionamiento

A continuación se explicará el funcionamiento del Q4S y posteriormente se detallará la implementación que se ha realizado. La figura 4.1 describe gráficamente dicho funcionamiento.

Como se ha explicado antes, este protocolo consta de 4 etapas: *Handshake*, en la que se establece la conexión; *Negotiation*, en la que se negocian los requisitos de red; *Continuity*, en la que se comprueba el cumplimiento de los requisitos previamente negociados; y *Termination*, etapa en la que tan solo se cierra sesión.

La sesión comienza una vez que el cliente envía una petición, consistente en un mensaje BEGIN. El servidor devuelve el mensaje con un 200 OK, en el cual en nuestro caso se incluyen en un cuerpo SDP los requisitos de calidad impuestos por la aplicación (umbrales de latencia, jitter, ancho de banda y pérdidas en ambas direcciones), el procedimiento de medida que se va a utilizar (en nuestro caso el utilizado por defecto), el tiempo entre PINGs consecutivos, el tiempo mínimo entre alertas y el número de PINGs que se envían en una misma ráfaga y el modo de alerta. Existen dos modos de alerta posibles, según el tipo de red:

1. *q4s-aware-network*: en la cual la red conoce el protocolo y se puede adaptar a los requisitos de éste. Estos requisitos son leídos por el cliente y se da paso al siguiente estado.
2. Política del servidor o reactivo: la red no conoce el protocolo y no interviene en el cumplimiento de los requisitos, por lo que las políticas empleadas corren de parte de un nodo actuador al cual el Q4S se limitará a enviarle mensajes UDP del tipo ALERT cada vez que se infrinjan los umbrales de red.

Una vez que ambas partes se han saludado, comienza la etapa *Negotiation*. La negociación se realiza en dos etapas. En la primera (fase 0) se mide la latencia y el jitter y en la segunda (fase

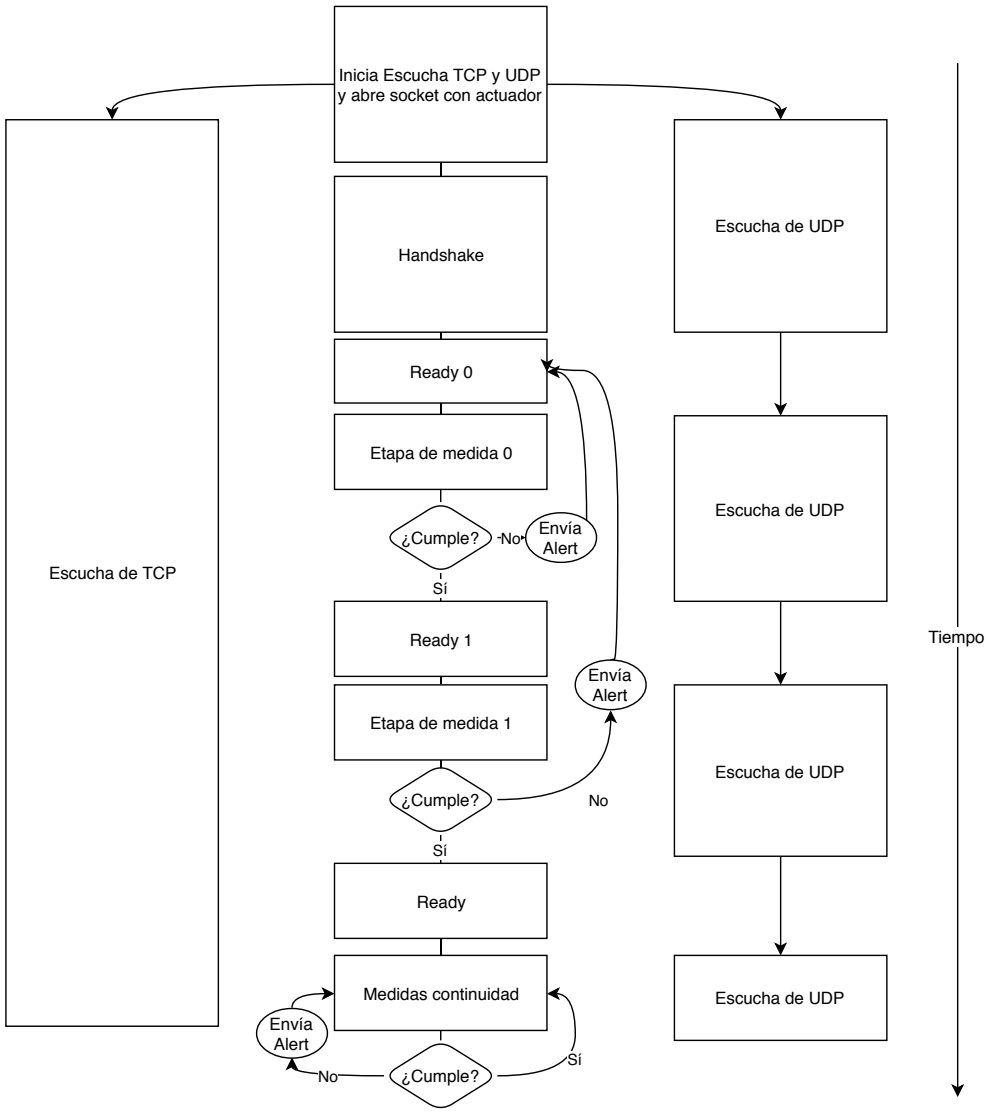

Figura 4.1: Funcionamiento del Q4S.

1) se mide ancho de banda y pérdidas. Además, la negociación consta de 11 niveles, que van del nivel 0, en el que las restricciones son mayores, al nivel 10, en el que los umbrales son mucho más laxos. Después del nivel 10 se asume que la red no cumple con los requisitos mínimos de la aplicación, de forma que el cliente envía un mensaje CANCEL al servidor, dando por terminada la sesión.

Para iniciar la fase 0, el cliente envía un mensaje READY 0. Posteriormente, el servidor responde con un mensaje 200 OK que, como en la fase *Handshake*, incluye de nuevo en el cuerpo SDP las especificaciones de red. La inclusión de estos requisitos en el protocolo es una de las mejoras realizadas, ya que se ha considerado más conveniente actualizar esta información recursivamente, para asegurarse de que ambos terminales comparten los mismos datos. El cliente, después de recibir la respuesta del servidor, comienza a enviar PINGs bajo UDP de manera equiespaciada. Tanto el número de PINGs que se envían como el tiempo que transcurre entre dos emisiones consecutivas es impuesto por el servidor. El servidor, en cuanto recibe el primer PING, empieza también el envío de mensajes PING. Los PINGs son respondidos por ambas partes según se reciben, tratando de minimizar el tiempo de respuesta del receptor. Estos mensajes PING tienen incluidos el timestamp del sistema y el número de secuencia que le corresponda, mientras que la respuesta es un 200 OK al que se le incluye el número de secuencia y el timestamp incluidos en el PING recibido. Además, cuando el 200 OK es recibido se marca con el timestamp de la llegada. Posteriormente, con el timestamp de llegada del 200 OK y el timestamp de emisión del PING, se calcula la latencia y jitter tal y como se explica en la subsección 2.1.2.

Al final de esta fase 0, el cliente y el servidor se intercambian las medidas tomadas, incluidas en un mensaje TCP tipo PING. En caso de que se cumplan los requisitos en ambas direcciones se continua a la fase 1, y en caso contrario se vuelve al inicio de la etapa *Negotiation*, donde se envían los nuevos requisitos de red impuestos por la aplicación correspondientes con el siguiente nivel de QoS. Adicionalmente, en el caso de los escenarios reactivos, el servidor envía un mensaje ALERT al actuador con las medidas que no superan el umbral acordado.

Una vez superada la fase 0, se alcanza la fase 1. Esta fase sigue el proceso enumerado a continuación:

1. Al inicio el cliente envía un mensaje READY 1, siendo respondido por el servidor con un mensaje 200 OK al que se adjuntan nuevamente los requisitos de la red.

2. Tras ello comienza el intercambio entre servidor y cliente de trenes de paquetes PING. Según [1], los paquetes PING deberán tener un payload de 1000 bytes, aunque por motivos que se explicarán al final de esta subsección, en esta implementación el tamaño será variable. Se enviarán cada milisegundo N paquetes, siendo N el número de PINGs a enviar para garantizar el ancho de banda requerido, tal y como se explica en la subsección 2.1.2.

   Para realizar las medidas, tanto en el servidor como en el cliente, se inicia un hilo en paralelo que se encargará de realizar el envío de los PINGs así como el cálculo de N, mientras que en el proceso principal se contará con un semáforo que controlará el hilo en paralelo. Esto se ha ideado debido a que si se enviasen los mensajes en el hilo principal y se esperase 1 milisegundo para proseguir el envío, se asumiría que el tiempo que se tarda en enviar los mensajes es despreciable, lo que se ha comprobado experimentalmente que para

umbrales de ancho de banda altos, el tiempo consumido se acerca bastante al milisegundo. Con respecto al cálculo de N, dado que el número de paquetes a enviar cada milisegundo no puede ser decimal, se ha implementado un bucle en el que se va aproximando al decimal, de forma que en determinados trenes de paquetes incluiría un paquete extra.

3. Finalmente se vuelven a intercambiar las medidas obtenidas con un mensaje TCP de tipo PING igual que el de la fase 0. Si ambos sentidos cumplen con las restricciones de la aplicación, se da por terminada la etapa *Negotiation*, comenzando así con la etapa *Continuity*.

Un cambio que se ha realizado con respecto al borrador del protocolo Q4S es el envío de un mensaje tipo ALERT Continuity al Actuador, de forma que posibilita a éste para arrancar la aplicación. En el borrador, se señalaba que era el protocolo Q4S el que enviaba un mensaje HTTP a la aplicación, pero con la intención de estandarizar el protocolo y hacerlo interoperable, se ha propuesto dicha solución. Si por el contrario, no se cumplen las restricciones de red, de nuevo se vuelve a empezar la etapa *Negotiation*, aumentando el nivel de QoS. En caso de que el nivel QoS sea máximo y no pueda aumentarse se pasaría a la etapa *Termination*.

En la etapa de *Continuity*, los únicos parámetros de red que se miden son latencia, jitter y pérdida de paquetes. Al igual que en la etapa anterior, el cliente envía un mensaje READY al servidor y éste responde con un 200 OK con los requisitos de la aplicación en el cuerpo del mensaje. Después, cliente y servidor empiezan a intercambiar mensajes PING con sus respectivas respuestas 200 OK, de la misma manera que se hacía en la fase 0 de *Negotiation*, con la salvedad de que además de medirse la latencia y el jitter, también se miden las pérdidas de paquetes. Al final de cada ráfaga se calculan los parámetros de QoS de la red y se intercambian las medidas con un mensaje PING sobre TCP. El siguiente paso se dará en función de los resultados:

- Si no se cumplen los límites y el nivel QoS no es el máximo, se envía un ALERT al actuador, aumenta en uno su nivel QoS y vuelve a empezar la etapa *Continuity*, esperando, además, un tiempo preacordado sin enviar un ALERT.
- Si no se cumplen los límites y el nivel QoS es el máximo, se pasa a la etapa *Termination*.
- Si se cumplen los límites y el nivel QoS es el negociado en la etapa anterior, se vuelve al inicio de la etapa para seguir con la monitorización.
- Si se cumplen los límites y el nivel QoS no es el negociado, se manda un RECOVERY al actuador, se reduce en uno el nivel de QoS, y se vuelve a empezar la etapa, dejando un tiempo negociado hasta enviar el siguiente RECOVERY.

Por último, en la etapa *Termination*, se da por finalizada una comunicación. El cliente enviará un mensaje TCP tipo CANCEL al servidor y cerrará el socket sin recibir respuesta, y el servidor a su vez enviará un mensaje UDP tipo ALERT Terminity al actuador. Posteriormente se cierra el cliente mientras que el emisor se queda a la escucha a la espera de nuevas conexiones de clientes.

Un ejemplo de esta comunicación es la figura 4.2, en la que un cliente inicia la conversación con el servidor y al terminar la fase 0 de la etapa *Negotiation* la red no cumple con los requerimientos impuestos. Entonces el servidor envía un ALERT al módulo actuador y vuelve

Figura 4.2: Ejemplo de conversación Q4S.

a empezar la etapa *Negotiation*. En el segundo intento sí que cumple con las restricciones que la aplicación impone y pasan a *Continuity*, enviando el servidor un ALERT Continuity al actuador. Por último, el cliente cuando da por finalizada la sesión envía un mensaje CANCEL al servidor, y éste a su vez un mensaje ALERT Termination al actuador para después responder al cliente con otro mensaje CANCEL.

En nuestro desarrollo, como se puede ver en la figura 4.1, se ha incluido una etapa más al inicio llamada *Init*, en la cual se inicializa el programa. En esta etapa, además, se definen tres procesos diferentes en paralelo, correspondientes con las tres ramas de la figura. El proceso principal procesará los mensajes recibidos, así como también los enviará, gestionará la máquina de estados del protocolo y se comunicará con el actuador. Los otros dos hilos recibirán los mensajes, uno de ellos los mensajes TCP y el otro los mensajes UDP. El hilo principal, cuando recibe un mensaje TCP, automáticamente deja de leer los mensajes UDP hasta que el proceso requiera de nuevo su lectura. Por ejemplo, en la fase 1 de la etapa *Continuity*, una vez que el servidor ha recibido el mensaje TCP tipo PING con el resultado de las medidas, elimina los mensajes UDP recibidos después, hasta, o bien alcanzar la fase 1 y realizar las mediciones de ancho de banda, o bien volver a la fase 0. Como consecuencia de este borrado de mensajes, se asumirá un error por el cual mensajes que han llegado con una latencia muy grande serán tomados como paquetes perdidos. Por otro lado, como protección para este posible exceso de latencia, a la hora de realizar los cálculos, se empiezan a leer los últimos mensajes recibidos en primer lugar, de forma que queden descartados los mensajes repetidos recibidos antes.

Adicionalmente, se ha introducido la posibilidad de variar el tamaño del paquete a mandar durante la medida del ancho de banda en la etapa *Negotiation*. La posibilidad de variar esta longitud de mensaje, permite poder medir el ancho de banda y las pérdidas para el tamaño de paquete que utilizará la aplicación. Como se ha explicado en la sección referencia a factores críticos 2.1.1, la longitud de los paquetes representa un factor crítico de cara a caracterizar la red dependiendo de en qué escenario se esté trabajando.

Por otro lado, el protocolo Q4S desarrollado para este proyecto ofrece tres posibilidades para la presentación de los resultados. Estas posibilidades son:

- Guardar las medidas realizadas, así como la etapa en la que se encuentra y el timestamp de la toma de la medida.
- Imprimir por pantalla las medidas tomadas, así como las alertas enviadas y una marca que se activa cuando las medidas superan los parámetros de umbral.
- Indexar los datos extraídos en una base de datos Influxdb. Dicha base de datos es utilizada por Grafana para mostrar la información requerida. Los datos son enviados a la base de datos Influxdb enviando desde el Q4S una petición HTTP POST al final de las etapas *Negotiation* y *Continuity*.

4.1.2. Análisis de rendimiento

Como se explica al principio de esta sección, el Q4S es un protocolo diseñado para funcionar en segundo plano junto a otra aplicación con ciertos requisitos de red. El hecho de funcionar en un segundo plano conlleva que el rendimiento del Q4S debe estar bastante optimizado, y que no

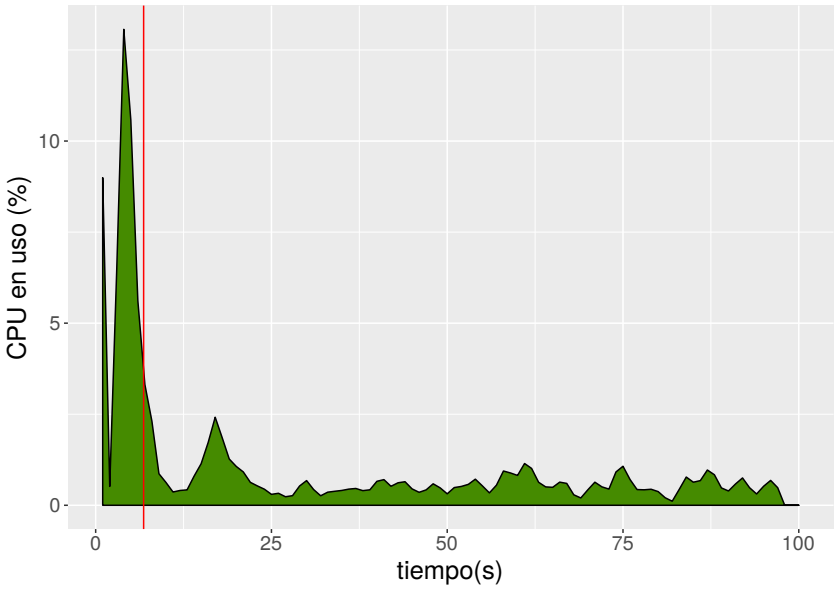

Figura 4.3: Uso de la CPU por el Q4S con respecto al tiempo.

sea muy intrusivo a nivel de red, sobre todo en la etapa de *Continuity* que es cuando se ejecuta en paralelo la aplicación.

Por ello, para comprobar que el rendimiento era el correcto se realizaron diferentes pruebas que monitorizaban el uso de la CPU. Para realizar estas pruebas se contó con un programa que leía la ocupación de la CPU de un equipo. Posteriormente, se implementó un programa que corría el protocolo Q4S entre dos terminales diferentes. Para estas pruebas se utilizaron una Zynqberry a modo de servidor y un PC a modo de cliente. Se forzó que ambos extremos corrieran sobre un sólo núcleo. A continuación, el programa capturaba el uso de CPU de la Zynqberry cada segundo, y repetía esta misma operación 100 veces para así poder calcular una estimación representativa del uso medio de la CPU. Se monitorizó el uso de CPU de la Zynqberry ya que se ha considerado que será el factor limitante de esta comunicación, asumiendo así que el uso de CPU en el PC será menor. Una vez obtenidas estas 100 medidas, para calcular la estimación, se hizo una interpolación lineal entre ellas para así cuadrar el timestamp y el número de muestras, con el fin de realizar la media aritmética de todas las medidas tomadas.

El resultado obtenido en consecuencia de dicho proceso es el observado en la figura 4.3. En ésta se pueden observar dos picos. El primero de ellos es debido a que el servidor, cuando está a la escucha esperando la conexión de un cliente, consume en torno al 10 % de CPU del núcleo. El segundo se corresponde con el uso de CPU en la fase 1 de la etapa *Negotiation*, durante la medida del ancho de banda, ocupando algo más del 10 % de CPU del núcleo. Sin embargo, una vez superada la etapa *Negotiation*, el rendimiento del sistema es bueno, ya que el uso de CPU desciende hasta aproximadamente cero.

Además de controlar el uso de CPU, también se estudió el impacto que suponía a nivel de red. Se estableció una conexión cableada entre la Zynqberry y el PC, sobre la que se ejecutó el protocolo Q4S, siendo el PC el servidor y la Zynqberry, el cliente. Los requisitos de ancho de banda que se fijaron fueron de 2MB en ambas direcciones. Se realizaron 25 capturas del tráfico generado por el Q4S durante 95 segundos. Después, igual que en la prueba de uso de CPU, se

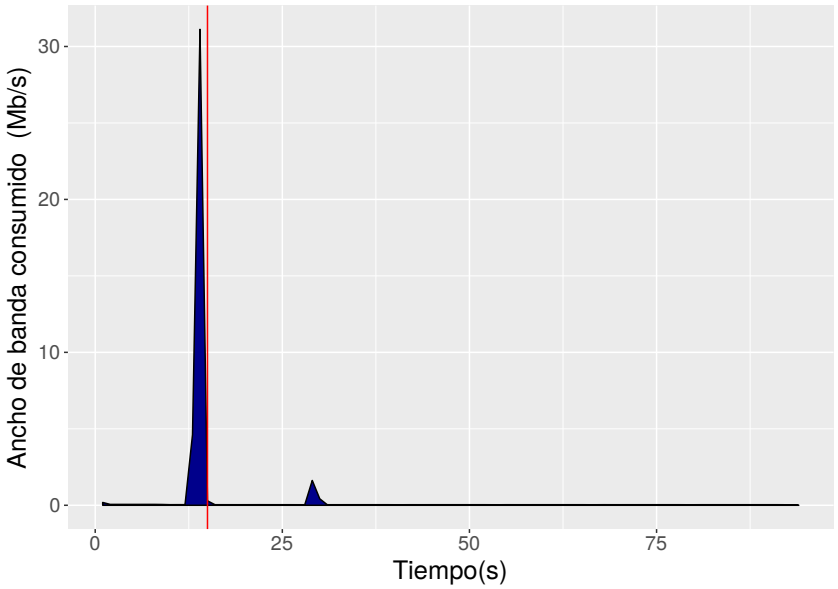

Figura 4.4: Ancho de banda medio consumido por el Q4S con respecto al tiempo.

hizo una interpolación lineal de todas las muestras y se calculó la media de estas medidas.

El resultado obtenido es el que se puede observar en la figura 4.4. En ésta se puede apreciar claramente cuál es el pico en el que mide el ancho de banda, en el que alcanza los 4MB de tráfico generado, que se corresponde a la suma del tráfico generado en ambas direcciones, fruto de los requisitos fijados que demandaban 2MB del servidor al cliente y 2MB del cliente al servidor. Una vez superada la medida de ancho de banda, se observa cómo el tráfico generado en la etapa *Continuity* es despreciable, de tal manera que genera un tráfico cruzado inexistente que es lo idóneo para poder ejecutar la aplicación con normalidad.

En conclusión, se puede afirmar que el protocolo Q4S que se ha implementado no hace un uso alto de la CPU, así como tampoco genera un tráfico excesivo de red, reduciéndose bastante en la etapa *Continuity*, que es en la que se va a ejecutar la aplicación. Por tanto, el rendimiento es bastante óptimo pudiendo funcionar en segundo plano sin perturbar otros servicios y aplicaciones que se deseen utilizar de manera principal.

4.2. Actuador

Este módulo se encarga de ajustar las propiedades de la aplicación en función del tráfico que ésta genera y de las características QoS de la red sobre la que va a funcionar. A partir de estas restricciones, el actuador tratará de que la aplicación ofrezca la calidad de experiencia máxima. Para ello, se encargará de dos tareas principales. La primera de ellas será fijar las características iniciales del contenido de la aplicación, en función de los parámetros tomados por el protocolo Q4S en la etapa de negociación. La segunda tarea será reaccionar a posibles variaciones de la red, aplicando cambios en caliente durante la ejecución del servicio.

Dado que este TFM esta enmarcado en el proyecto Racing Drones, la aplicación para la cual se va a diseñar este actuador será para la transmisión de un vídeo codificado con el codificador

LHE y enviado sobre RTP. Por ello, es conveniente realizar un estudio del tráfico generado por dicha aplicación para marcar cuáles son dichas características de QoS que debe cumplir la red. Por otro lado, como el actuador tiene la función de maximizar la QoE en función de los parámetros QoS medidos, se va a realizar un estudio de cómo afectan diferentes problemas de red a la QoE, y qué características de la aplicación ofrecen una mayor QoE en función de ciertos parámetros de QoS. A continuación se detallarán sendos estudios realizados.

4.2.1. QoS

Para realizar este estudio de la calidad de servicio, se han llevado a cabo unas medidas pasivas del tráfico de red correspondiente a la transmisión de un vídeo LHE. Para esto, se creó un escenario compuesto por la Zynqberry conectada por cable a un router (router 1), que mediante un puente, se conectaba de manera inalámbrica al otro router (router 2), el cual se conectaba por cable al PC. Además, en el enlace cableado entre la Zynqberry y el router 1, se ha añadido un μTAP[1]. Se trata de un dispositivo útil para analizar el tráfico de un enlace. En este escenario, se ha conectado el μTAP al PC, donde se analizaba la trama y se comparaba con la recibida en el PC. Dado que el PC sólo contaba con una interfaz de red, se utilizó un adaptador ethernet a USB, el cual podía introducir un poco de ruido a la señal recibida.

La cámara Zynqberry captaba las imágenes gracias a una Raspbery cam, las cuales eran codificadas con el códec LHE, empaquetadas con un formato específico, detallado en la sección 3.1, y transmitidas. En el lado del PC, se recibía la trama con las imágenes, las decodificaba y las reproducía. Estas imágenes eran grabadas en un entorno de oficina, con la cámara estática, lo que no debía suponer ningún problema ya que este codificador no utiliza información temporal. En las pruebas realizadas se realizaban transmisiones de 5 segundos, con diferente tasa de refresco, perfil de color y tamaño de bloque LHE. Para capturar el tráfico se utilizó la herramienta *tcpdump*. Posteriormente, se filtraba con la herramienta tshark para guardar sólo los paquetes pertenecientes a una sesión RTP.

El primer parámetro que se quería obtener era el ancho de banda. En la tabla 4.1 se pueden ver los resultados obtenidos. Estos resultados muestran cómo a medida que desciende el número de cuadros por segundo, desciende significativamente el ancho de banda consumido. También, cuando se envía el frame con un perfil de color YUV 4:2:0, consume más ancho de banda que sólo transmitiendo con un perfil de color de YUV 4:0:0. A la hora de comparar el tamaño de bloques LHE, se aprecia cómo el ancho de banda disminuye cuando se trata de bloques y no de líneas. Esto encuentra explicación, en que si se aprovecha tanto la redundancia vertical, como la redundancia horizontal, la codificación será más eficiente. Adicionalmente, se observa que por lo general, los paquetes más grandes suelen acarrear menor ancho de banda, lo que puede deberse a que se añaden menos cabeceras de bloque y, también, dependiendo de la imagen, que cuanto mayor es un bloque mayor información no necesaria se puede eliminar, al tratarse de un códec con codificación espacial. Otro aspecto a destacar en la tabla es que, para tasas de refresco bajas, la variación de ancho de banda entre las tasas de refresco medidas es menor que la variación de ancho de banda para los dos perfiles de color a una misma tasa de cuadros por segundo.

Otra medida que se ha realizado ha sido el tiempo transcurrido entre paquetes consecutivos. De esta medida se extraen dos tipos de información que pueden ser de utilidad, una de ellas

[1] `http://cubro.org/index.php/component/zoo/item/cubro-utap`

Tabla 4.1: Medidas de ancho de banda en función de las características del códec.

fps	perfil de color	Altura del bloque	Ancho del bloque	Ancho de banda
55	YUV420p	1	320	5.97 MB/s
55	YUV420p	1	640	5.89 MB/s
55	YUV420p	6	160	5.66 MB/s
55	YUV420p	8	128	5.43 MB/s
55	YUV400p	1	320	4.51 MB/s
55	YUV400p	1	640	4.61 MB/s
55	YUV400p	6	160	4.43 MB/s
55	YUV400p	8	128	4.44 MB/s
36	YUV420p	1	320	4.00 MB/s
36	YUV420p	1	640	3.98 MB/s
36	YUV420p	6	160	3.85 MB/s
36	YUV420p	8	128	3.8 MB/s
36	YUV400p	1	320	3.18 MB/s
36	YUV400p	1	640	3.07 MB/s
36	YUV400p	6	160	2.95 MB/s
36	YUV400p	8	128	2.96 MB/s
18	YUV420p	1	320	2.07 MB/s
18	YUV420p	1	640	2.04 MB/s
18	YUV420p	6	160	1.97 MB/s
18	YUV420p	8	128	1.97 MB/s
18	YUV400p	1	320	1.60 MB/s
18	YUV400p	1	640	1.59 MB/s
18	YUV400p	6	160	1.54 MB/s
18	YUV400p	8	128	1.54 MB/s
13	YUV420p	1	320	1.57 MB/s
13	YUV420p	1	640	1.56 MB/s
13	YUV420p	6	160	1.51 MB/s
13	YUV420p	8	128	1.50 MB/s
13	YUV400p	1	320	1.22 MB/s
13	YUV400p	1	640	1.21 MB/s
13	YUV400p	6	160	1.16 MB/s
13	YUV400p	8	128	1.08 MB/s

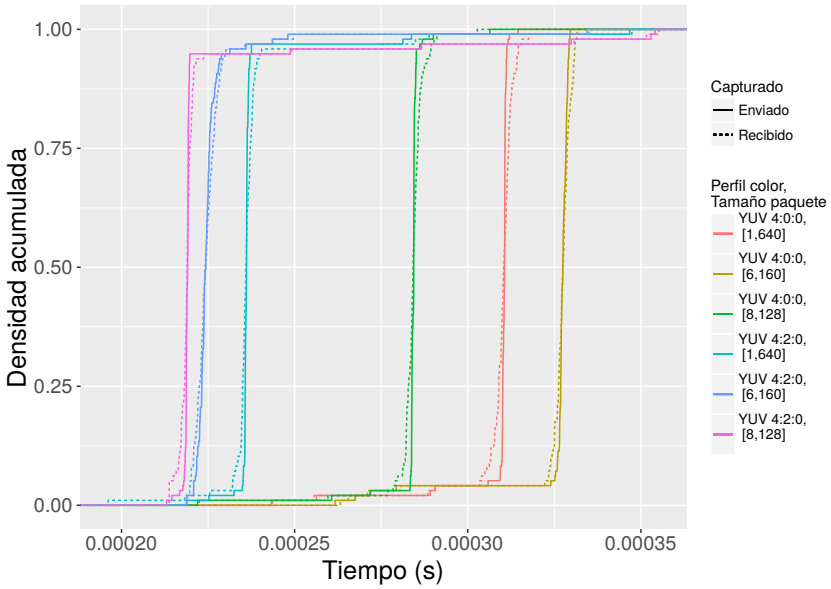

Figura 4.5: ECDF del tiempo entre llegadas de paquetes para una misma tasa de cuadros por segundo.

es el tiempo transcurrido entre dos paquetes consecutivos pertenecientes a un mismo frame, los *interarrivals*, y la otra es el tiempo transcurrido entre dos cuadros consecutivos, el tiempo *intraframe*. Ambos se han extraído gracias a las características tanto de la cabecera LHE, como de la cabecera RTP, sin necesidad de inspeccionar el paquete. También se han utilizado las medidas pasivas tomadas por el μTAP, tanto el tiempo de envío de frames consecutivos, como el tiempo entre paquetes consecutivos. Las medidas tomadas en el envío y en la recepción serán comparadas con el fin de visualizar cómo varía cuando se introduce a la red. Esta medida tiene especial repercusión en el jitter, ya que si la diferencia de tiempos entre el envío de dos paquetes consecutivos de un mismo cuadro es muy alta, y el jitter de la red es alto, puede provocar que los paquetes vengan desordenados, lo que puede suponer que se descarten estos paquetes o incluso el cuadro entero.

En la función de distribución empírica de los tiempos entre envío/recepción de secuencias con una misma tasa de refresco de la gráfica 4.5, se puede ver cómo afecta el tamaño de los bloques y el perfil de color sobre dichos tiempos entre paquetes. Se ve cómo para el perfil de color YUV 4:2:0 el tiempo medio entre salidas es menor que para el YUV 4:0:0. También se puede observar que con respecto a la emisión, la ECDF de la recepción se suaviza, lo que significa que los paquetes enviados con una diferencia entre envíos idéntica, en la recepción se observa que estos tiempos han variado levemente. Así mismo, es destacable cuándo ocurre la recepción de todos los paquetes del cuadro. Por ejemplo, para el perfil de color YUV 4:2:0, sobre todo los codificados por bloques LHE, aunque la mayoría de los paquetes se envían con una diferencia de tiempo muy pequeña, el 100 % de los paquetes se envían de manera muy tardía, habiendo entre estos tiempos una gran varianza.

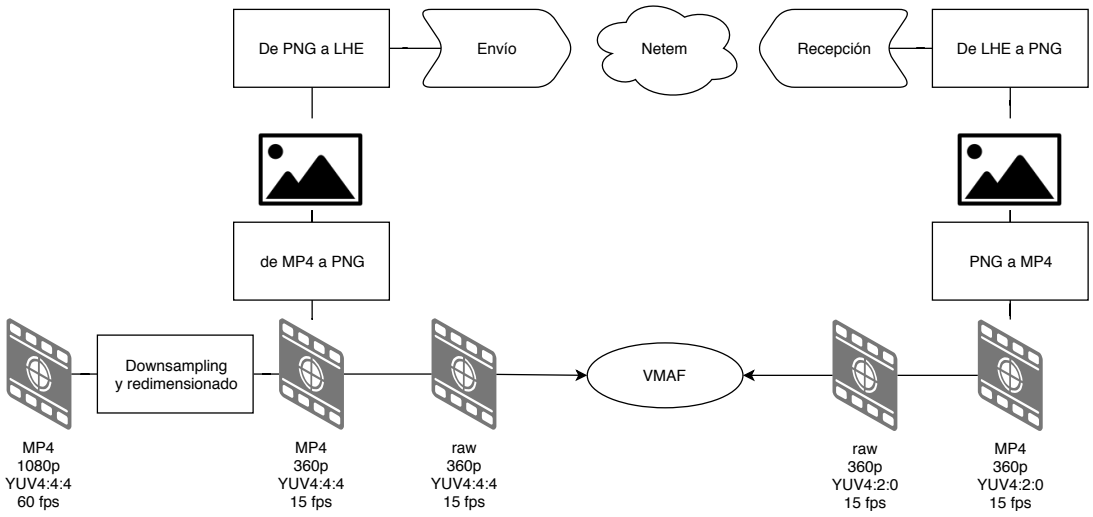

Figura 4.6: Funcionamiento del VMAF.

4.2.2. QoE

Para el análisis de la calidad de experiencia se ha utilizado la herramienta VMAF. Como ya se ha expuesto en el capítulo 2, esta herramienta compara dos vídeos que toma como entradas. Los vídeos que se han comparado han sido el vídeo original, y el vídeo transmitido en un entorno local. Para realizar este estudio se ha hecho uso de un programa realizado por el grupo HPCN, sobre el que se realizaron modificaciones para ajustarlo a los efectos que se querían estudiar.

El esquema de funcionamiento es el mostrado en la figura 4.6. Inicialmente se realiza una reducción de la tasa de cuadros por segundo del video original, para posteriormente codificar el vídeo a un formato LHE para transmitirse sobre un entorno local (*localhost*) y, por otro lado, convertir el vídeo original a un formato aceptado como entrada de VMAF para tomarlo como vídeo de referencia. A su recepción se decodifica y se convierte al formato de entrada de VMAF para poder ser comparado. Cabe destacar que VMAF no permite realizar comparaciones entre dos vídeos con diferente tasa de cuadros por segundo.

En el análisis de vídeo desarrollado por el grupo HPCN, se midió el efecto del tamaño del bloque LHE sobre la puntuación de VMAF cuando en la transmisión hay pérdidas. También se incluyó el resultado del efecto del tamaño de bloque LHE en la medida anterior. Los resultados se pueden ver en la figura 4.7, en ella se puede apreciar cómo las pérdidas en la red afectan claramente sobre la calidad percibida, independientemente del tamaño del bloque, que no afecta significativamente a la calidad percibida, ya que la varianza de las puntuaciones de una misma clase es muy similar a la varianza entre clases.

Para este trabajo, realizamos un segundo análisis, en el que se asumía que el tamaño de bloque y de paquete no influía en la calidad en vista de los resultados previamente obtenidos. Por ello, para un tamaño de bloque LHE fijo, se probó cómo afectaba la variación de diferentes parámetros de calidad de servicio con respecto a la calidad subjetiva. Estos parámetros se han probado a través de la herramienta netem. Se trata de una mejora de Linux que permite añadir retardos, agregar pérdidas, duplicar paquetes y demás características a los paquetes salientes de una interfaz de red seleccionada. Los parámetros que se probaron fueron:

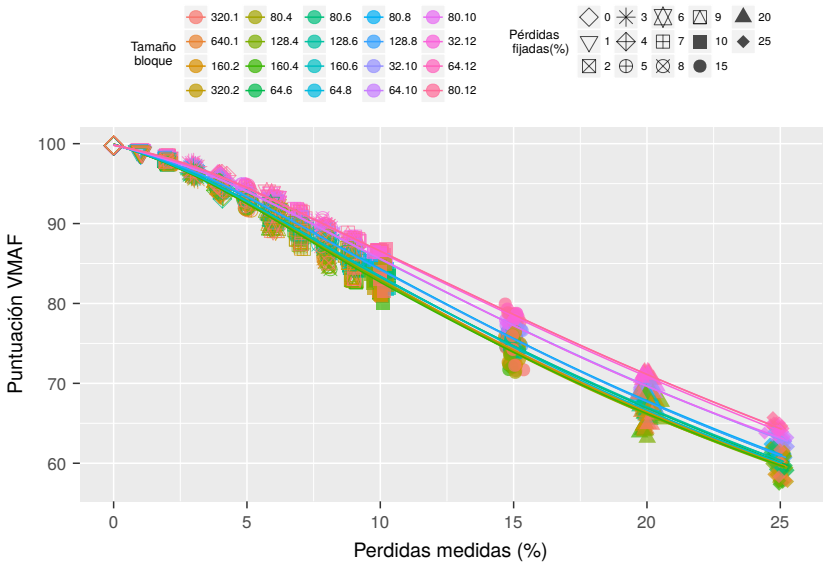

Figura 4.7: Puntuación VMAF en función de las pérdidas en la red y del tamaño del bloque LHE.

1. Latencia: se probaron los valores de 0, 8 y 15 milisegundos.
2. Jitter: Se utilizaba el cociente entre la latencia y los coeficientes 2, 3, 4.
3. Ancho de banda: Se tomaron los anchos de banda de 100Mbits/s 30Mbits/s y 10MBits/s.
4. Pérdida de paquetes: se introdujeron pérdidas de 0 %, 10 % y 20 %.

De estas medidas se extrajeron diferentes conclusiones. La primera de ellas, que ya se había mencionado anteriormente, es el hecho de que las pérdidas son las que más afectan a la calidad percibida. La segunda conclusión es, que además de las pérdidas fijadas por netem, si un ancho de banda es suficientemente pequeño, se traduce en pérdidas, lo que se refleja a su vez en una disminución de la calidad percibida, como se puede ver en la figura 4.8.

No obstante, existen algunos resultados de dicha gráfica que no se ajustan a lo comentado hasta el momento. Estos valores se corresponden a los situados en la esquina inferior izquierda, en los que las pérdidas son iguales, pero sin embargo la puntuación de VMAF es distinta en función de las pérdidas fijadas. Para indagar acerca de qué parámetro provocaba este descenso de la puntuación VMAF, se hizo una gráfica que incluyese la influencia de la latencia y del jitter. Los resultados son los que se ven en la figura 4.9. En ella se puede ver, que para retardos distintos de cero, las pérdidas medidas aumentaban cuando el ancho de banda era bajo. Sin embargo, no se aprecia ninguna influencia del jitter en las medidas realizadas.

Por otro lado, un factor que se ha tenido muy en cuenta en este trabajo ha sido la influencia de la latencia en la jugabilidad. En la subsección 2.2.2, se han analizado diferentes investigaciones al respecto, llegando a la conclusión de que la latencia máxima admisible para juegos en primera persona, como es el caso de Racing Drones es 100 milisegundos. Sin embargo, dado que se trata de un juego cuyo contenido no es continuo, sino que lo que se ve es un muestro de lo observado

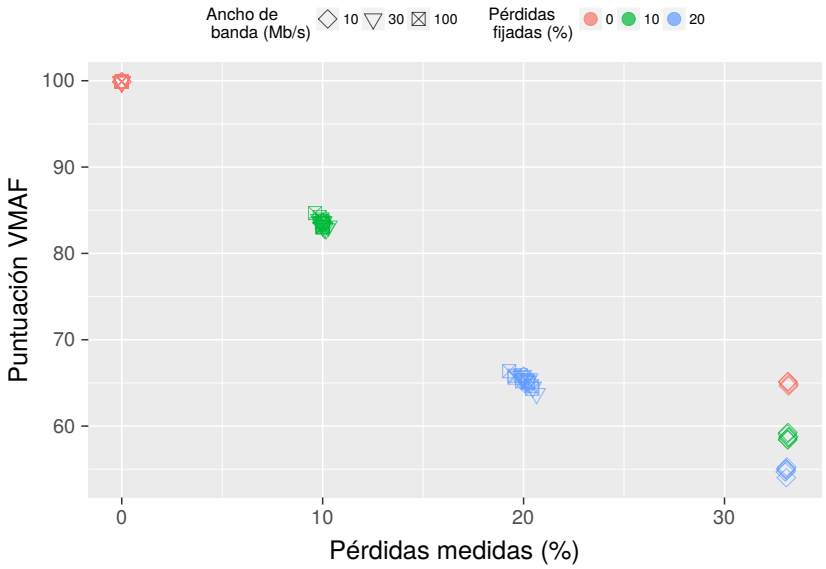

Figura 4.8: Puntuación VMAF en función de las pérdidas medidas, las pérdidas fijadas, y el ancho de banda fijado.

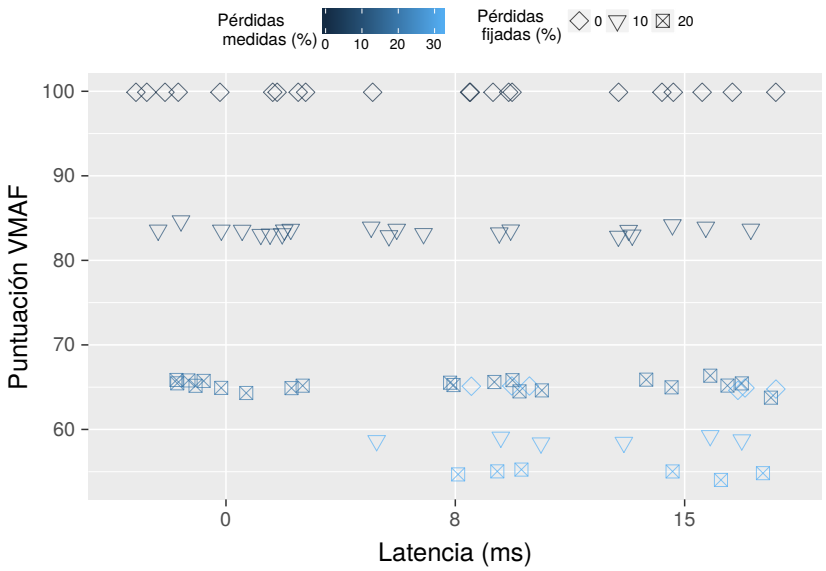

Figura 4.9: Puntuación VMAF en función de la latencia fijada y las pérdidas medidas y fijadas.

Tabla 4.2: Latencia permitida según tasa de cuadros por segundo.

fps	latencia
55	41ms
36	36ms
27	32ms
22	27ms
18	23ms
13	14ms
9	0ms

por la cámara, la latencia permitida se verá afectada por esa tasa de muestreo. Si un jugador debe tener constancia del movimiento que ha ejecutado en menos de 100ms, significa que desde que el jugador ejecutó la orden, hasta que recibe la imagen, teniendo en cuenta el tiempo de codificación y decodificación y la tasa de refresco de cuadro (fps), pasan como máximo 100ms. Por tanto, suponiendo que el tiempo de codificación y de decodificación es despreciable, la latencia del sistema deberá cumplir:

$$RTT_{red} + \frac{1000ms}{fps} \leq 100ms \tag{4.1}$$

$$latencia_{red} \leq \frac{100ms - \frac{1000ms}{fps}}{2} \tag{4.2}$$

Con ese requisito, la máxima latencia de red posible para las diferentes tasas de cuadros por segundo que ofrece el códec LHE, es la reflejada en la tabla 4.2. Donde se puede ver que para la tasa de refresco de 9 cuadros por segundo, la latencia permitida es 0ms, ya que esta tasa entrega un cuadro cada 111 milisegundos.

4.3. Integración del sistema

En este apartado se realizó la integración del sistema. Para ello, se contaba con dos partes, una primera parte que consistía en incluir el protocolo Q4S en ambos dispositivos y hacer que ambos módulos se comuniquen, y un segundo apartado que consistía en el diseño de la máquina de estados del actuador, y la comunicación de este actuador a su vez con la aplicación, de forma que pueda modificarla como considere oportuno. Hay que tener en cuenta que, según el dispositvo (Zynqberry o Raspberry), varía el codificador y el paquetizador implementados en el mismo, lo que conlleva que, dependiendo del dispositivo, se podrán modificar unos parámetros u otros. Por tanto, se va a explicar la integración del sistema implementado en función del dispositivo.

4.3.1. Zynqberry

Este dispositivo cuenta con un codificador LHE, que codifica las imágenes que le van llegando de la cámara Raspberry Pi v2.1. Este codificador, es capaz de codificar bloques con un tamaño

ajustable. Posteriormente, el paquetizador implementado en este dispositivo, empaqueta cada bloque LHE con una cabecera específica, que a su vez se empaqueta en un paquete tipo LHE, el cual se envía sobre RTP. El sistema completo tiene la posibilidad de realizar cambios en caliente de los frames por segundo, el tamaño de los bloques LHE y sobre el perfil de color.

Posteriormente se instaló el servidor del protocolo desarrollado. El protocolo media la calidad de la red y cuando no se cumplían las restricciones, enviaba un mensaje tipo ALERT al puerto 27017.

El actuador estaba a la escucha de mensajes tipo ALERT en ese mismo puerto. Una vez que recibía el ALERT, analizaba los parámetros recibidos y decidía su próximo salto en la máquina de estados que se explicará más tarde. Una vez realizado dicho salto, el actuador implementado enviaba un petición HTTP POST con los cambios a realizar en el codificador a un servidor de aplicación web que se encargaba de ello. Este servidor de aplicación web creado tiene dos funciones principales, aplicar los cambios que recibe del actuador en el codificador y, por otro lado, actualizar los requisitos de red demandados por la aplicación, que en función de las características de los cambios en caliente realizados tendrán distintos valores. Los requisitos de red son leídos por el protocolo Q4S, siendo éstos los nuevos umbrales demandados por el mismo.

Adicionalmente, se han realizado cambios sobre el paquetizador, haciendo que el tamaño máximo de los paquetes sea modificable. Esta modificación se realizará antes de comenzar la transmisión y se fijará el mismo tamaño que el utilizado en el Q4S en la fase 1 de la etapa *Negotiation* durante la medida del ancho de banda.

Con todo ello, se ha realizado el diagrama de estados que se puede ver en la figura 4.10. En él se pueden distinguir dos etapas. La primera de ellas correspondiente a la etapa *Negotiation*, y la segunda etapa la correspondiente con la etapa *Continuity*. Para ambas etapas los parámetros comunes a modificar serán el número de cuadros por segundo y el perfil de color, mientras que el parámetro en el que difieren es, en la etapa *Negotiation*, el tamaño del paquete que se enviará, y en la etapa *Continuity*, el tamaño del bloque LHE.

Lo que se persigue con este actuador es que en función de la calidad de servicio medida se apliquen cambios que no afecten a la calidad de experiencia percibida, y en caso de que afecte que sea lo mínimo posible. Por ello, se han tenido en cuenta los siguientes factores.

- Se buscará que la jugabilidad sea máxima, intentando que la latencia desde la cámara hasta la pantalla del usuario final sea menor que 100ms.
- La calidad percibida es mayor con perfil de color YUV 4:2:0 que con perfil de color YUV 4:0:0, sin embargo, el perfil de color 4:0:0 es menos restrictivo que el YUV 4:2:0.
- El tamaño de los bloques apenas afecta a la calidad percibida.
- Una reducción de los cuadros por segundo disminuye los requisitos tanto de ancho de banda, como de latencia y jitter.
- Según el escenario en el que se esté trabajando, el tamaño de paquete afectará de forma diferente, siendo, como se ha visto en el capítulo 2, más beneficioso para disminuir las pérdidas de paquetes pequeños para ambos escenarios, y por otro lado, los paquetes grandes consumen menos ancho de banda. Sin embargo, para disminuir la latencia, es más

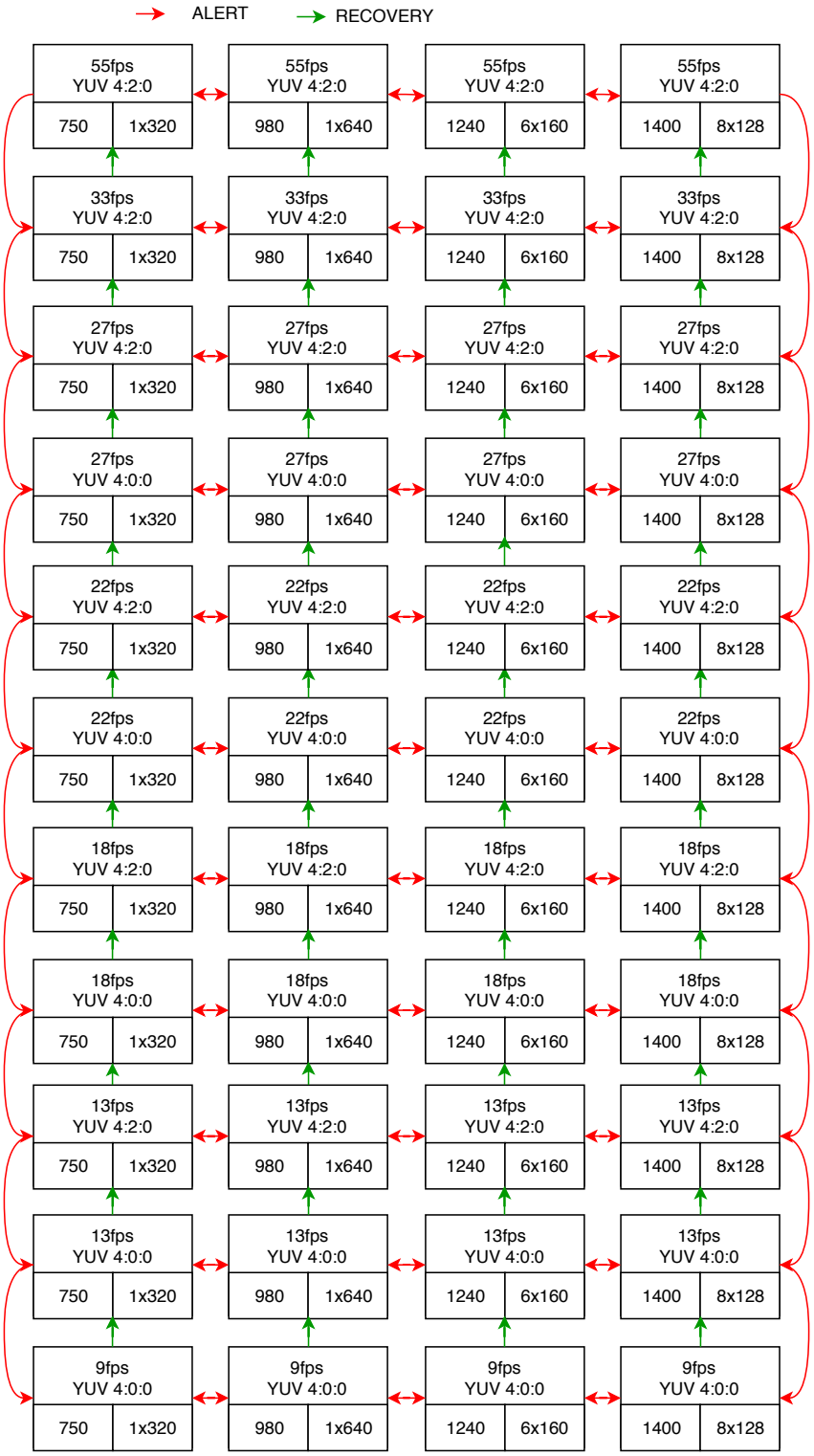

Figura 4.10: Máquina de estados diseñado para la Zynqberry.

beneficioso reducir el tamaño del paquete para los enlaces inalámbricos, mientras que en el caso de los escenarios LAN, una reducción del número de paquetes enviados (paquetes más grandes) puede hacer que la latencia sea menor.

- Cuanto más alta es la tasa de refresco, menor es la tasa de llegadas entre cuadros. Además, el tiempo de llegadas entre paquetes de un mismo cuadro varía, haciendo que puedan llegar desordenados, mezclándose con las llegadas del cuadro siguiente, lo que puede provocar pérdidas.

Por tanto, el diagrama de estados cambia según el escenario en el que se esté trabajando, ambos tendrán los mismos estados, sin embargo variará el salto entre estados en los que solo se altera el tamaño del paquete y del bloque LHE. Para el escenario inalámbrico, tal y como se ha visto en los factores expuestos anteriormente, tanto cuando haya pérdidas en la red como cuando haya una latencia demasiado alta, se disminuirá el tamaño del paquete y del bloque LHE. Cuando haya problemas de ancho de banda se aumentará el tamaño del paquete. Para enlaces por cable, con respecto a problemas en la red con el ancho de banda disponible y con problemas de latencia, se aumentará el tamaño del paquete o de bloque LHE, mientras que para problemas con las restricciones de pérdidas de paquetes se tomará la decisión de enviar paquetes más pequeños.

Una vez que el tamaño del paquete y el tamaño del bloque LHE no se puedan aumentar o diminuir más, se pasa a modificar el resto de parámetros que sí que afectan a la calidad de experiencia. En las primeras fases, se tratará de reducir el número de cuadros por segundo antes que el perfil de color, hasta llegar en torno a los 30 fps, ya que como se ha comentado antes, apenas disminuye la calidad. En el caso de este actuador, aplicará esta política hasta alcanzar los 27 frames por segundo, manteniendo el perfil de color YUV 4:2:0. Una vez alcanzada esa tasa, se pasará a variar el perfil de color y la tasa de cuadros por segundo de manera intercalada. Es decir, primero se pasará de perfil de color YUV 4:2:0 a YUV 4:0:0, y en el siguiente salto se disminuirá el número de cuadros por segundo, restableciendo de nuevo el perfil de color YUV 4:2:0.

A medida que se disminuye la tasa de cuadros por segundo, las medidas de latencia se van haciendo más restrictivas. Aun así, se irá disminuyendo esta tasa para que el tráfico generado sea menor, ya que es la única acción posible que se puede ejecutar en la aplicación para que disminuya esta latencia.

Estos saltos se darán cada vez que se reciba un mensaje tipo ALERT del servidor Q4S, a excepción del ALERT Continuity y del ALERT Termination. No obstante, cuando se recibe por parte del servidor un mensaje RECOVERY, no se deshace el camino, sino que lo que se busca es maximizar la calidad de experiencia, es decir, cuando se recibe un RECOVERY se obvia el tamaño del bloque LHE, y se pasa a un estado en el que, o bien se aumenta el número de cuadros por segundo, o bien se pasa de perfil de color YUV 4:2:0 a YUV 4:0:0.

Cuando se recibe un mensaje tipo ALERT Continuity, el actuador inicia la sesión de transmisión del vídeo con las características negociadas en la primera etapa. Esto quiere decir que cuando se pasa de la etapa *Negotiation* a la etapa *Continuity*, el actuador permanece en el mismo estado. Esta sesión RTP es finalizada por el actuador cuando recibe un ALERT Termination.

Por último, el módulo que ejecuta los cambios en la aplicación es común a ambos actuadores, tanto el del escenario con conexión cableada, como para el del escenario con conexión wifi.

Tabla 4.3: Umbrales de los parámetros QoS para la Zynqberry.

fps	Perfil de color	Latencia	Jitter	Ancho de banda	Pérdidas(%)
55	YUV420p	41ms	20ms	6 MB/s	30 %
36	YUV420p	36ms	40ms	4.00 MB/s	30 %
27	YUV420p	32ms	70ms	3.10 MB/s	30 %
27	YUV400p	32ms	80ms	2.40 MB/s	30 %
22	YUV420p	27ms	100ms	2.40 MB/s	30 %
22	YUV400p	27ms	100ms	2.0 MB/s	30 %
18	YUV420p	23ms	130ms	2.1 MB/s	30 %
18	YUV400p	23ms	130ms	1.6 MB/s	30 %
13	YUV420p	14ms	200ms	1.6 MB/s	30 %
13	YUV400p	14ms	200ms	1.3 MB/s	30 %
9	YUV400p	5ms	200ms	850 KB/s	30 %

Además, este módulo, como se ha mencionado antes, es el encargado de modificar los requisitos de red a demandar por el protocolo Q4S. Estos requisitos de red han sido fijados en función de la tabla 4.1 y de la tabla 4.2, así como en función de los factores vistos al inicio de esta subsección. Los requisitos tan sólo varían cuando se cambia la tasa de refresco o el perfil de color. Se asume que los cambios en los parámetros QoS que provocan los diferentes tamaños de bloque LHE no son suficientes como para cambiar los umbrales, y además, lo que se persigue realizando cambios en el bloque es que el tráfico generado se adapte a la red, lo cual no se puede medir por el Q4S, pero si se puede observar el efecto provocado. Las restricciones impuestas para cada nivel son las de la tabla 4.3. Las restricciones se fijaron de la siguiente manera:

- Latencia: la latencia calculada en la tabla 4.2. Para el último nivel la latencia demandada es 5ms ya que una latencia de 0ms es inalcanzable.
- Jitter: el jitter máximo permitido es el tiempo entre llegadas entre el último paquete de un cuadro y el primer paquete del cuadro siguiente.
- Ancho de banda: el más restrictivo calculado en la tabla 4.1.
- Pérdidas: el porcentaje de pérdidas estimado en función del estudio de la puntuación VMAF obtenido en la figura 4.7.

4.3.2. Raspberry

El sistema con el que cuenta la Raspberry consta de un codificador LHE, que codifica línea a línea, desaprovechando la redundancia vertical, por lo que los bloques de codificación no serán modificables. La salida de este codificador se empaqueta con la cabecera de un mensaje H264 en lugar de empaquetarlo con una cabecera específica como era el caso anterior. Por otro lado, este codificador cuenta con un servidor HTTP que recibe peticiones para poder realizar cambios en caliente sobre la codificación del vídeo. El primero de estos cambios es *discard*, que se corresponde al nivel de descarte de líneas, es decir, tira líneas en función del nivel elegido, para luego reconstruir la imagen en recepción. Este nivel va del 0 al 5, y cuanto más alto sea,

más líneas descartadas. Con este método se asume una perdida de calidad en la imagen recibida. El segundo cambio que se puede realizar es *skip*, siendo este el número de frames que se van a saltar.

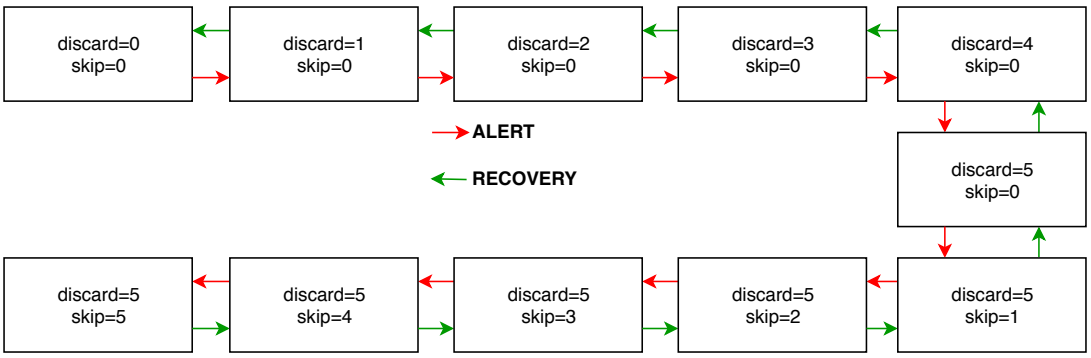

Figura 4.11: Máquina de estados diseñada para la Raspberry.

Por ello, el actuador realizado, sigue el diagrama de estados correspondiente con la figura 4.11. En el que en cada estado se va aumentando poco a poco el nivel de Discard, y cuando su valor es máximo se empiezan a descartar frames. A medida que se cambia de estado disminuye la calidad de experiencia. Los saltos entre un estado y otro tienen lugar cuando se recibe un ALERT, en el cual se salta de un estado al más próximo con una calidad inferior, mientras que si se trata de un mensaje RECOVERY, se pasa al estado que aumenta la calidad percibida.

Dados los únicos cambios que se pueden realizar para este dispositivo, no es posible realizar dos actuadores distinguiendo de si se trata de un enlace cableado o un enlace inalámbrico.

Además, como en el caso anterior, cuando recibe del Q4S un mensaje ALERT Continuity inicia la recepción del vídeo, el cual está continuamente enviándose desde el emisor, y cuando recibe un mensaje ALERT Termination pone fin a esta recepción.

4.4. Conclusiones

En esta sección se ha explicado todo el desarrollo llevado a cabo, así como la integración de todos sus módulos en sendos dispositivos. En primer lugar, se ha realizado un protocolo Q4S que mide latencia, jitter, ancho de banda y pérdidas para la monitorización de la red. Cuando detecta un problema, el Q4S notifica al actuador, el cual realiza cambios en la aplicación. El actuador depende de la aplicación desarrollada, por lo que dependerá en gran medida de la aplicación sobre la que se trabaja. Para este proyecto se trabajó con dos códec LHE, pero implementados de manera diferente, uno para la Zynqberry (por hardware) y otro para la Raspberry (por software).

Para la primera implementación del códec LHE se realizó un estudio para determinar cómo poder maximizar la QoE respetando la QoS de la que se disponía. En este estudio, se llegaron a las siguiente conclusiones:

- El parámetro QoS que más influye en la calidad percibida son las pérdidas. Otros factores como un ancho de banda muy limitado también supone un empobrecimiento de la QoE. De

igual forma pasa con la latencia, la cual si es muy elevada reduce la calidad de experiencia.

- El tamaño de bloque LHE no es relevante para la QoE, aunque éste sí que influye en el ancho de banda consumido por la aplicación.
- El tiempo de llegadas entre paquetes en una sesión RTP puede dar infomación del jitter a partir del cual se pueden empezar a detectar problemas.
- A medida que se disminuye la tasa de cuadros por segundo, se hace más restrictiva la latencia de red permitida.

Gracias a esta información y a la vista en el estado del arte, se realizó el actuador para esta implementación. En concreto se realizaron dos, uno para cada escenario, en el que cambiaba la política a la hora de variar el tamaño de los paquetes. Con respecto a la segunda implementación del códec LHE, se realizó un actuador que determinaba el comportamiento de la aplicación descartando líneas y frames.

5

Pruebas y Resultados

La simulación de este trabajo se centra principalmente en validar el protocolo y el actuador implementado. Dicha validación se ha realizado mediante dos pruebas diferentes, tal y como se muestra en la figura 5.1. En la primera de ellas se cuenta con una red estática, es decir, una red con unos parámetros fijos de latencia, jitter, ancho de banda y pérdidas. En la segunda prueba se configura una red cuyos parámetros de QoS varían a lo largo del tiempo. Por tanto, en el primer supuesto, se comprobará si el funcionamiento del protocolo Q4S en su fase *Negotiation* es correcto. En la segunda prueba se tratará de probar la fase *Continuity*. Además, se probará el funcionamiento del Q4S de manera conjunta con el módulo actuador.

5.1. Preparación de las pruebas

Para la realización de sendas validaciones, se ha utilizado la herramienta netem. Esta herramienta sólo ha sido utilizada sobre la interfaz de red del ordenador, no en el dispositivo (Zynqberry o Raspberry), por lo que este filtro sólo se aplicarán en un sentido. Los valores de los parámetros de QoS de la red que se han utilizado en sendas pruebas, han sido los siguientes:

<table>
<tr><th colspan="6">Pruebas</th></tr>
<tr><td colspan="2">Pruebas estáticas</td><td colspan="4">Pruebas dinámicas</td></tr>
<tr><td rowspan="2">WLAN</td><td rowspan="2">LAN</td><td colspan="2">WLAN</td><td colspan="2">LAN</td></tr>
<tr><td>Sin actuador</td><td>Con actuador</td><td>Sin actuador</td><td>Con actuador</td></tr>
</table>

Figura 5.1: Estructura de las pruebas realizadas.

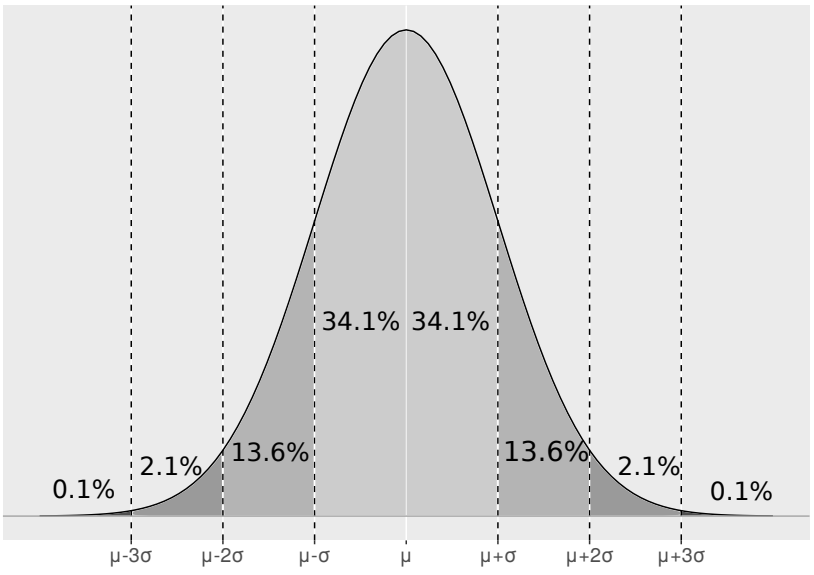

Figura 5.2: Densidad de probabilidad cubierta por cada desviación estándar en una distribución normal.

1. Latencia: los retardos añadidos a la red en las pruebas estáticas han sido 0ms, 5ms y 10ms. Estas medidas se han elegido con el fin de probar si el comportamiento en la fase *Negotiation* es correcto. Para ello, el umbral de latencia elegido para realizar los experimentos ha sido 5ms. De esta manera, en el primer caso, el Q4S debería superar la fase 0 de la etapa *Negotiation*, en el segundo caso puede superarla o no (dependiendo del jitter), y en el tercer caso no debería superarla de ningún modo. Para las pruebas estáticas se ha variado de manera aleatoria la latencia en un intervalo entre 0 y 10.

2. Jitter: como se ha demostrado en la subsección 2.1.1, se puede definir el jitter como la varianza de la latencia. Además, sabiendo que la herramienta netem genera el jitter siguiendo una distribución normal, y que la latencia mínima está acotada inferiormente en 0, se ha llevado el siguiente razonamiento:

   Dada una distribución normal como la de la figura 5.2, en la que existe la propiedad que dicta que en el intervalo $\mu - 2\sigma, \mu + 2\sigma]$ se encuentra comprendida, aproximadamente el 95,6 % de la distribución, y que el espacio muestral en el que se llevan a cabo las pruebas está acotado entre $[0, \text{inf})$, y debido a que se quiere que la distribución normal respete sus características de simetría; se debe cumplir que $\mu - 2\sigma = 0$ por lo que el jitter máximo que se propone será $\sigma = \frac{\mu}{2}$.

   Por ello, para que las medidas realizadas estén centradas en la latencia, se tomará como valor máximo de jitter aquel que incluya la mayor cantidad de muestras posibles, en nuestro caso, $fraclatencia2$. Además de este coeficiente, también se realizaron pruebas con los coeficientes de jitter 3 y 4. El resultado entre la latencia y el coeficiente de jitter se redondea al número inferior. Es decir, que para la latencia de 0ms no se va a introducir jitter, para la de 5ms el jitter que se introducirá será $jt = |\frac{5ms}{2}| = 2$, $jt = |\frac{5ms}{3}| = 1$.

3. Ancho de banda: el ancho de banda elegido para limitar la salida de la interfaz de red ha

Tabla 5.1: Parámetros QoS por defecto de las redes utilizadas.

Escenario	Latencia(ms)	Jitter(ms)	Pérdidas(%)
LAN	0.53	0.057	0 %
WLAN	2.24	2.37	0 %

sido 50 Megabits, 28 Megabits y 20 Megabits. Por otra parte, el umbral del Q4S en una primera ronda es 28Mbits, mientras que en los niveles de QoS posteriores, es 20Mbits. A priori, debería superar la fase 1 siempre, aunque para las pruebas con el ancho de banda limitado a 20 Megabits deberá pasar por la fase 1 dos veces, de forma que alcance el nivel 1 de QoS.

4. Pérdida de paquetes: la pérdida de paquetes está implementada por netem mediante el descarte aleatorio de un porcentaje de paquetes antes de que estos sean encolados. Para estas pruebas, los valores elegido como la probabilidad de que un paquete se pierda han sido 10 %, 5 % y 1 %. Al incluir la bandera *random* en netem, la probabilidad de pérdidas será algún valor próximo a los valores seleccionados. El umbral de pérdidas que está configurado en el Q4S en ambos sentidos es de 5 %.

Estos experimentos han sido llevados a cabo en dos escenarios distintos. El primero de ellos se trata de una conexión cableada entre el PC y la Zynqberry, pasando por el router. Sobre ellos se corría el protocolo Q4S, siendo el PC el cliente y la Zynqberry el servidor. El segundo escenario trataba de una conexión inalámbrica entre la Zynqberry y el router, que a su vez estaba conectado con un cable Ethernet al ordenador. El acceso wifi de la Zynqberry se obtuvo gracias a un adaptador wifi a USB. En este escenario, como en el anterior, se probó el protocolo Q4S, situando el servidor en la Zynqberry y el cliente en el PC.

Conviene tener en cuenta los parámetros de ambas conexiones para dar validez a los experimentos realizados. Por ello, utilizando la herramienta *ping*, se ha medido la latencia, el jitter, y las pérdidas de ambos escenarios. Para que las medidas fueran válidas, se debía de contar con un número suficiente de muestras, y por tanto se enviaron en torno a 1000 PINGs, tomando como resultados la media de la latencia, la media de jitter, y el porcentaje de pérdidas con respecto el total. Estos parámetros se encuentran en la tabla 5.1.

5.2. Pruebas estáticas

Estas pruebas tenían como finalidad comprobar el correcto funcionamiento del Q4S en la etapa *Negotiation*. Se debía comprobar que las medidas que realizaba se correspondían con la realidad. Por tanto, lo que se ideó fue un programa que fijaba un filtro de netem a la interfaz de red del PC, combinando las características de latencia, jitter, pérdidas y ancho de banda previamente descritas, y posteriormente dando inicio al protocolo Q4S entre el PC y la Zynqberry. Estas medidas se guardaban para así dar lugar a los resultados mostrados a continuación. Las medidas del Q4S que se guardaban eran las siguientes: etapa en la que se encontraba(donde 0 es *Negotiation* y 1 es *Continuity*), marca temporal (timestamp) de la realización de la medida, latencia de subida y de bajada, jitter de subida y de bajada, ancho de banda de subida y de bajada, pérdidas de subida y de bajada, nivel de QoS, y una bandera que se levantaba cuando

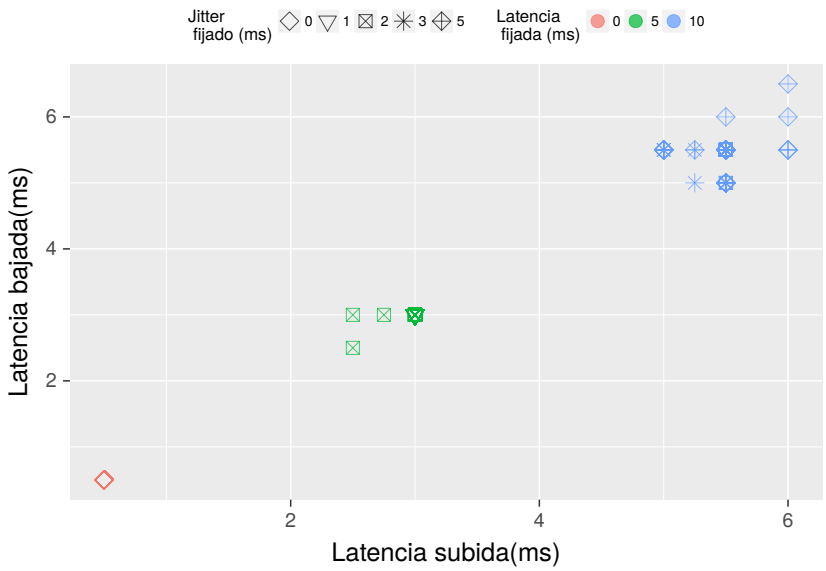

Figura 5.3: Latencia de subida vs. Latencia de bajada en LAN.

las medidas superaban el umbral. Se entiende por subida la medida tomada por el servidor, y viceversa. En lo que resta de esta sección se van a exponer estos resultados, dividiéndolos entre latencia, jitter, ancho de banda y pérdidas.

5.2.1. Latencia

Para comprobar que las medidas de latencia realizadas son correctas, se enfrentó en una gráfica las medidas de latencia realizadas por el cliente y las medidas de latencia realizadas por el servidor. La gráfica resultante se puede ver en la figura 5.3 y en la figura 5.4

En la figura 5.3 se puede observar cómo las medidas pintadas están bastante centradas en la diagonal, lo que significa que las medidas tomadas de latencia de subida respecto de bajada coinciden. A medida que la latencia va aumentando se empieza a apreciar una dispersión generada por el jitter. Por ello, cuando el jitter es 0, no se aprecia ninguna distorsión. Nótese también, que a pesar de que se ha introducido una latencia de 0ms, 5ms y 10ms, en ambas gráficas la latencia medida aparece centrada en torno a 0ms, 2.5ms y 5ms. Esto es debido a que, como se ha especificado en la subsección anterior, la herramienta netem solo ha sido utilizada en la salida de la interfaz de red del PC, actuando sólo en los paquetes de salida por lo que el retardo incluido se reflejará afectando al *RTT*. Con lo cual, el resultado es el obtenido al dividir entre dos la latencia añadida.

La figura 5.4 se corresponde a la gráfica del escenario inalámbrico. En esta se pueden apreciar las mismas afirmaciones con respecto a que las medidas se encuentran centradas en la diagonal. Sin embargo, al tratarse de una red wifi, con las características por defecto señaladas en la tabla 5.1, se observa una mayor dispersión de dichas muestras, añadiendo un jitter mayor y una latencia más elevada.

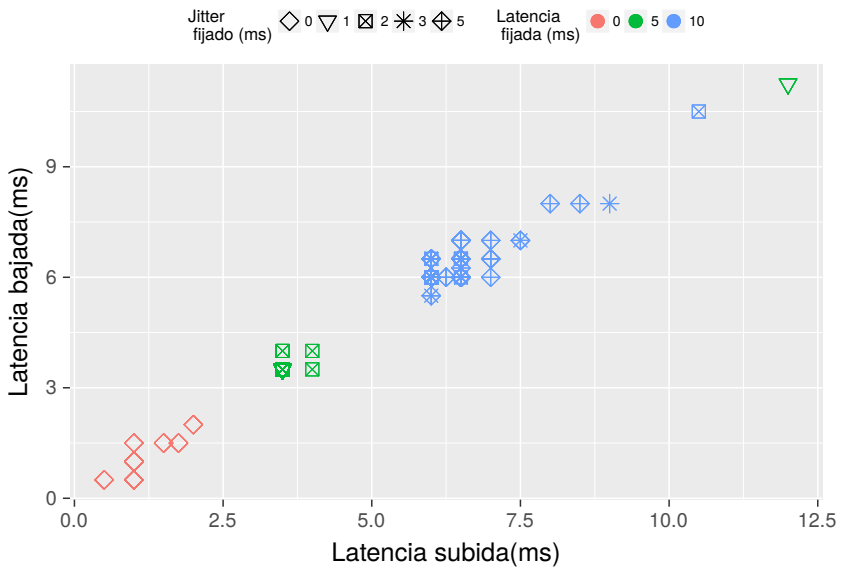

Figura 5.4: Latencia subida vs. Latencia de bajada en WLAN.

5.2.2. Jitter

El jitter se trata de una medida de varianza de la latencia. Este retardo viene añadido por el filtro creado por la herramienta netem, siguiendo lo explicado en la introducción, en el que cabe destacar que dicha herramienta añade un jitter aleatorio entre ciertos valores. Por ello, no tiene sentido enfrentar el jitter medido de subida y de bajada, ya que son medidas independientes entre sí. Por ello, se hará uso de las gráficas 5.3 y 5.4 para justificar su medida, ya que nuevamente esa dispersión de la latencia coincide con el jitter fijado.

5.2.3. Pérdida de paquetes

Por otro lado, se comprobó el correcto funcionamiento del protocolo a la hora de medir el porcentaje de pérdida de paquetes. Para comprobarlo, se ha dibujado una gráfica en la que se puede ver el porcentaje de pérdidas medido con respecto al porcentaje de pérdidas fijado en netem. Esta gráfica se corresponde con la figura 5.5.

En esta gráfica se puede ver cómo los valores medidos están centrados en la diagonal con respecto a los valores fijados. La dispersión observada se debe a la selección del filtro aleatorio en la herramienta netem, que hace que las pérdidas sean un valor aproximado de las deseadas. Por otro lado, se observan puntos muy alejados de estas modas. Como se puede ver en la escala de color, estos *outlayers* encuentran explicación en las limitaciones de ancho de banda fijadas, traduciéndose también en pérdidas en el sistema.

5.2.4. Ancho de banda

De nuevo, para el ancho de banda, se ha optado por enfrentar el ancho de banda medido en el cliente contra el ancho de banda medido en el servidor. El resultado es el mostrado en la

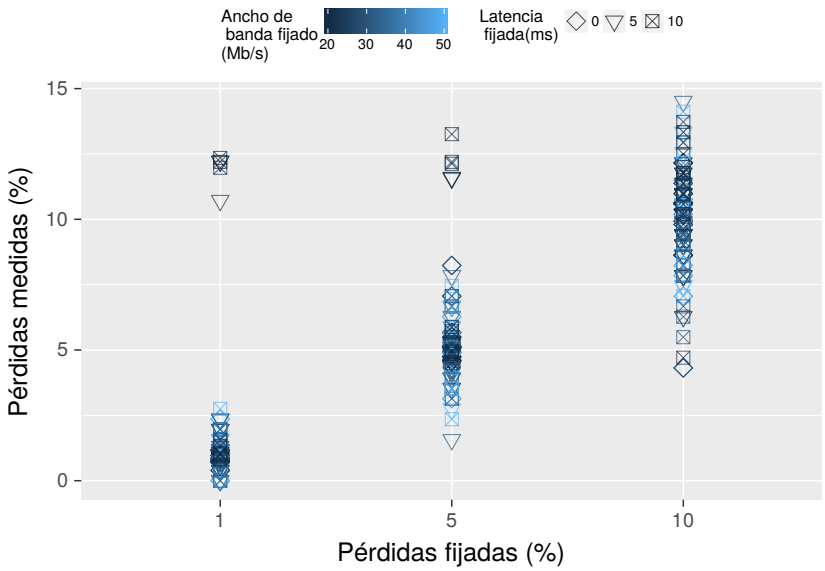

Figura 5.5: Pérdidas de subida vs. Pérdidas de bajada.

figura 5.6.

A modo de recordatorio, queda reseñar que el Q4S no mide el ancho de banda disponible, sino que mide si la red dispone de un ancho de banda solicitado. Los anchos de banda solicitados son 28Mbps y 20 Mbps. Por tanto, las restricciones que se ejerzan sobre la red, sólo afectaran en el caso de que los umbrales sean mayores que las restricciones.

Con todo ello, la figura muestra cómo el ancho de banda medido está situado alrededor del ancho de banda solicitado, tanto para subida como para bajada. Sin embargo, se puede ver que el ancho de banda de subida y de bajada medido no es igual siempre, ya que la herramienta netem solo interfiere en la tarjeta de red, por lo que en un sentido sí cumplirá las restricciones, mientras que hacia el otro, cuando haya restricciones de red, no cumplirá.

5.3. Pruebas dinámicas con Zynqberry

Las pruebas dinámicas han sido realizadas para comprobar el funcionamiento del protocolo Q4S en su etapa *Continuity*, es decir, ver cómo se comporta la aplicación frente a cambios. Para ello, nuevamente se han utilizado ambos escenarios explicados en la introducción del capítulo.

Por lo tanto, se diseñó un programa para automatizar las pruebas que lanzaba el Q4S, tanto en el cliente (PC) como en el servidor (Zynqberry), y aplicaba un filtro sobre la interfaz de red de salida del PC. Como lo que se buscaba en esta prueba era validar el funcionamiento de la etapa *Continuity*, se iniciaba la aplicación sin ningún filtro hasta que alcanzaba la etapa *Continuity*, una vez alcanzada dicha etapa se iba variando el jitter, la latencia y las pérdidas, dejando de lado el ancho de banda, ya que según se explica en el borrador de Q4S, no se hacen comprobaciones del ancho de banda en esta etapa. Además, se han eliminado las restricciones de red, de forma que no llegase a la etapa *Termination*, y no hubiera que volver a lanzar el

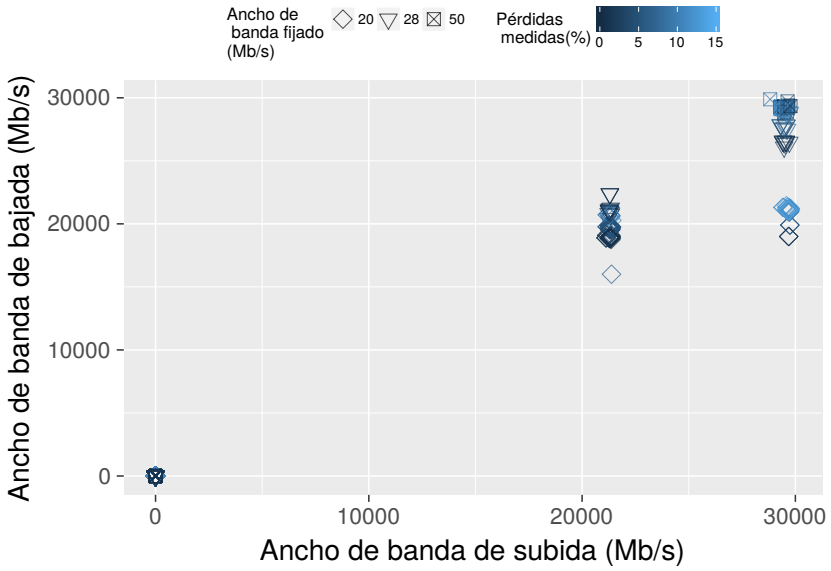

Figura 5.6: Ancho de banda de subida vs. Ancho de banda de bajada.

cliente de nuevo. El resultado, tanto para el escenario cableado como para el inalámbrico, es el que se puede observar en la figura 5.7 y 5.8.

Con respecto a la figura 5.7, se observa que a medida que se va variando la latencia de salida, las medidas realizadas por el Q4S van variando también. El efecto del jitter también se refleja, haciendo que las medidas tomadas disten más de sus valores teóricos cuando la latencia fijada es alta. El mismo efecto pero más acentuado se puede ver en la figura 5.8, debido al jitter que añade la red inalámbrica.

La figura 5.7 se corresponde con la ECDF tanto de los parámetros reales, como de los parámetros medidos en el escenario cableado. Se puede ver cómo ambas tienen la misma forma, pero se aprecia que la curva de las medidas está levemente desplazada a la derecha. Este desplazamiento es originado por el retardo que añade la red sobre los mensajes transmitidos.

Para probar el actuador se ha utilizado el mismo procedimiento que anteriormente, con la salvedad de que también se ha lanzado el Actuador y se han fijado nuevamente las restricciones. Para comprobar que tanto el Q4S como el actuador están coordinados, es decir, que ambos son conscientes del estado de la red, se ha dibujado el nivel QoS que ambos han obtenido de manera independiente, producto de los mensajes tipo ALERT y tipo RECOVERY intercambiados, y de los algoritmos de estimación del nivel de QoS presentes en el cliente, en el servidor y en el acuador. Además, también se debía apreciar como el protocolo volvía al estado inicial una vez que se alcanzaba la etapa *Termination*, fijando el nivel QoS a cero y restableciendo los valores iniciales.

Como se puede ver en la figura 5.10, el comportamiento es el esperado, siendo el nivel QoS del Actuador el mismo que el nivel QoS del protocolo del cliente y del servidor. El procedimiento que este sigue es el siguiente:

1. Se va aumentando el nivel QoS hasta que se llega a un acuerdo, en este caso en 6.

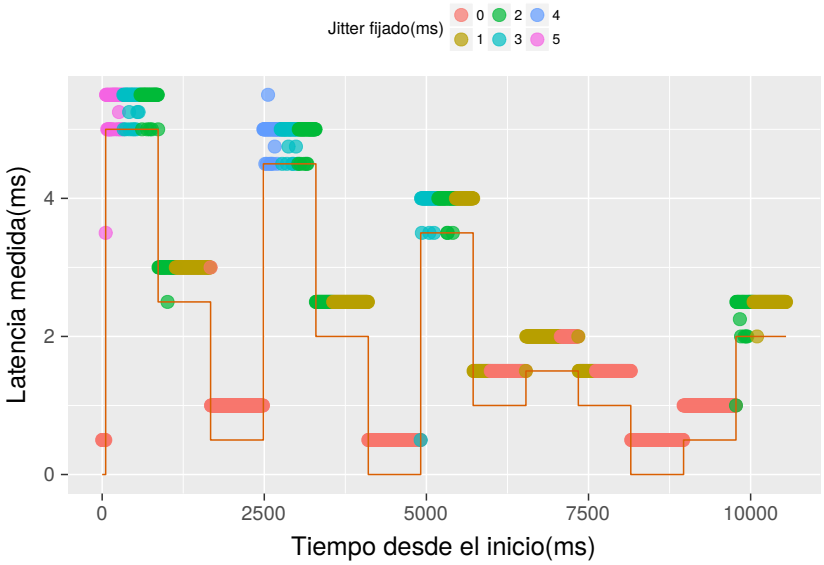

Figura 5.7: Latencia medida dinámicamente en LAN.

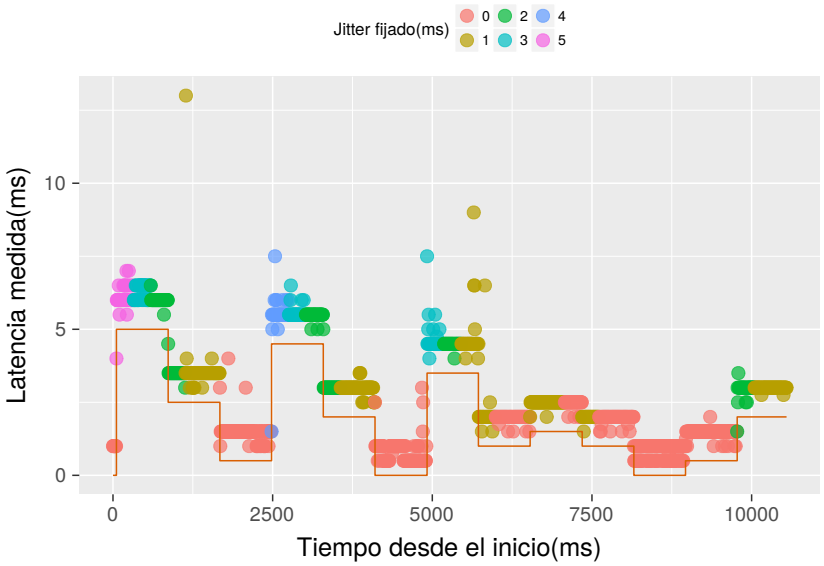

Figura 5.8: Latencia medida dinámicamente en WLAN.

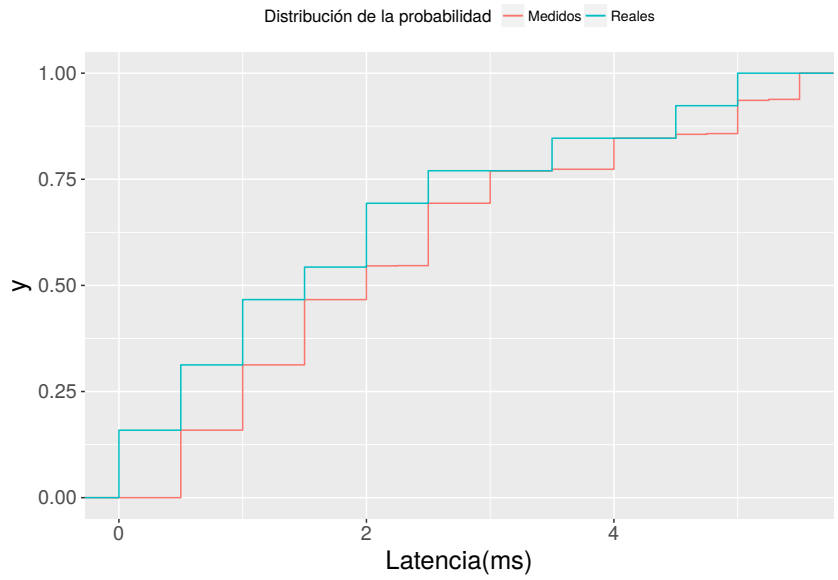

Figura 5.9: ECDF de la latencia medida y la latencia real.

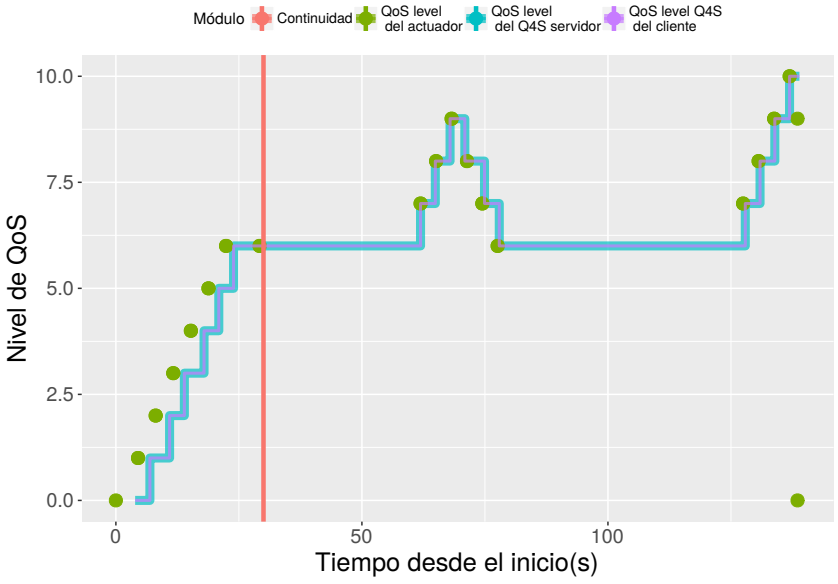

Figura 5.10: Nivel QoS del Actuador, Nivel QoS del servidor Q4S y nivel QoS del cliente Q4S con respecto al tiempo.

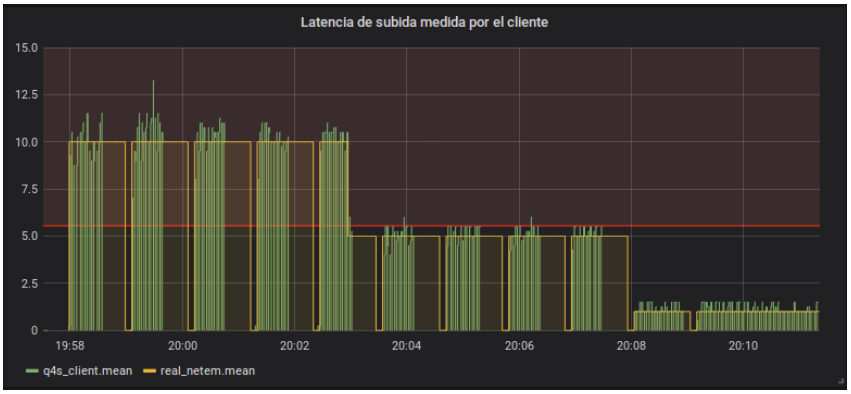

Figura 5.11: Latencia de subida medida por el cliente Q4S.

2. Una vez en la etapa *Continuity* se mantiene el nivel hasta que empieza a haber restricciones de red. Entonces es cuando se vuelve a aumentar el nivel QoS.

3. Cuando estas restricciones cesan, se reduce de nuevo el nivel QoS, pero sin superar el nivel QoS negociado al principio.

4. La red vuelve a empeorar, por lo que se incrementa sucesivamente el nivel QoS, pero esta no cumple con los requisitos establecidos ni en el nivel más alto, por lo que se envía un mensaje ALERT Termination, finalizando así la sesión y reestableciendo los valores iniciales en el actuador.

5.4. Pruebas dinámicas con Raspberry

En esta sección se han realizado pruebas que verifican el funcionamiento del sistema en la Raspberry. Sin embargo, para la visualización de esta información, se ha optado por la opción de visualización implementada haciendo uso de Grafana. Para realizar estas pruebas, se ha creado un programa de automatización de pruebas que mediante la herramienta netem se fijan filtros sobre la salida de la interfaz de red que van variando cada 30 segundos, y en caso de que se haya cerrado la aplicación por haber superado el nivel QoS máximo, llegando así a la etapa *Termination*, la lanza de nuevo eliminando los filtros de red hasta que alcanza la etapa *Continuity*. En las gráficas 5.11 y 5.12 se puede ver el *dashboard* generado. Al estar validando la etapa *Continuity*, las medidas que se han dibujado han sido latencia (figura 5.11) y jitter (5.12). En ambos se observa cómo las medidas tomadas se ajustan bastante bien a las fijadas. Además, también se aprecia que cuando las medidas superan el umbral establecido (franja roja), se acaba finalizando la comunicación.

5.5. Conclusiones

En este capítulo se han validado los diferentes módulos realizado durante el proyecto. En primer lugar, se ha validado el comportamiento del Q4S en su etapa *Negotiation* mediante unas pruebas estáticas y, posteriormente, se han realizado unas pruebas dinámicas para probar la

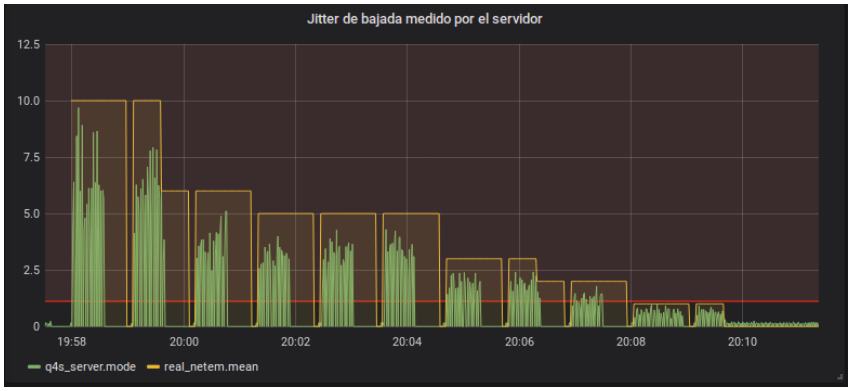

Figura 5.12: Jitter de bajada medido por el servidor Q4S.

etapa *Continuity*. De estas pruebas estáticas se ha concluido que la latencia medida se ajusta bastante a la latencia esperada, así como por otro lado, también se ha observado el efecto que el jitter puede tener en la latencia. Con respecto a las pérdidas, en este proyecto se han generado tirando paquetes de manera aleatoria, por lo que no siempre será la misma medida. Sobre las medidas de ancho de banda, se ve como muchas veces el ancho de banda medido se asemeja bastante al esperado. Por último, se han visualizado las medidas realizadas con la Raspberry, en la que se han obtenido resultados muy similares que para los de la Zynqberry.

La conclusión principal que se puede extraer de este capítulo es, que tanto el cliente Q4S, como el servidor Q4S, como el actuador, funcionan de manera coordinada, realizando las medidas de una manera bastante ajustada a la realidad. Ésto, junto con el hecho de que funciona correctamente para ambos dispositivos, permite afirmar que el sistema implementado es interoperable y funcional, independientemente de la máquina de estados que siga el actuador.

6

Conclusiones y trabajo futuro

En este apartado se va a realizar un breve resumen del trabajo realizado, exponiendo las ideas principales que se han extraído, y plantendo los aspectos más relevantes del análisis, tanto del sistema desarrollado, como de los resultados obtenidos. A continuación se relacionará el TFM con lo aprendido en el máster, y por último, se harán algunas propuestas de trabajo futuro que se consideren interesantes.

6.1. Conclusiones

En el capítulo 1, se expusieron los principales objetivos de este trabajo. Por ello en primer lugar se hará una descripción del estado de los objetivos marcados al inicio del proyecto:

- Se ha desarrollado un protocolo Q4S, capaz de realizar medidas de los diferentes parámetros de la QoS de la red en tiempo real, con una intrusión muy baja y un uso de la CPU despreciable. Además, este protocolo es capaz de notificar cuando no se cumplen las restricciones marcadas por una aplicación.
- Se ha implementado un actuador, que dadas las características del tráfico, envía un mensaje HTTP a un servidor de aplicación web encargado de la aplicación con las nuevas características del servicio.
- Se ha realizado un actuador para cada dispositivo y cada escenario de Racing Drones. Para ello, se ha personalizado este actuador en vista de los parámetros modificables de la aplicación de Racing Drones. Para realizarlo, se ha estudiado el tráfico generado por la aplicación las diferentes características de codificación del LHE y de paquetización llevada a cabo para esa información. También se ha hecho un estudio sobre como afectaban diferentes parámetros de QoS a la QoE calculada por VMAF para vídeos codificados con LHE.

- Se han realizado pruebas para validar los objetivos anteriores, utilizando diferentes escenarios y realizando diferentes simulaciones de red.

Por otro lado, se han cumplido las tareas expuestas en el capítulo 1 con sus respectivos entregables, fundamentales para la realización de este trabajo.

Para ser más concreto, se ha desarrollado un protocolo en C++, ideado para sistemas UNIX, en el que se medía la calidad de un enlace entre dos usuarios, en nuestro caso un cliente y un servidor. Además, este protocolo no solo debía funcionar sobre dispositivos terminales, sino que además tenía que funcionar para dispositivos cuya CPU podía ser un factor limitante, pudiendo funcionar en un segundo plano.

Asimismo, este protocolo tenía que ser resistente a redes con unas características muy desfavorables, como puede ser una latencia muy alta, jitter, un porcentaje de pérdidas alto, y una capacidad de red muy limitada, pero sin perder de vista su principal función de ser apto para entornos con restricciones de baja latencia. Adicionalmente, se han propuesto diferentes mejoras y aportaciones para dotarlo de interoperabilidad y hacerlo más eficiente.

Con respecto al desarrollo del actuador, al no poder hacerse un actuador estandarizado ya que cada aplicación requiere sus propias características, se han desarrollado tres actuadores diferentes, que a pesar de que siguen la misma estructura y persiguen el mismo objetivo, los medios para alcanzarlo eran bien distintos.

Para realizar el actuador de la Zynqberry, se ha realizado un análisis del tráfico generado por la aplicación y un estudio la QoE de la aplicación. Sobre el análisis del tráfico de red generado se han extraído conclusiones como que el tamaño de los bloques de LHE afecta al ancho de banda, así como que la latencia permitida depende en gran medida de la tasa de cuadros por segundo.

Con respecto al estudio de la QoE, en vista de los resultados dados por la herramienta VMAF, se concluye que el tamaño de los bloques LHE y el tamaño de los paquetes que se elijan no suponen una mejora en la calidad relevante. Las pérdidas son totalmente determinantes en la QoE, a pesar de que el LHE tiene mayor resistencia a perdidas que otros codificadores. Además, un ancho de banda limitado también disminuye la QoE, ya que esta reducción de ancho de banda se traduce en un aumento de las pérdidas. La latencia también reduce significativamente la puntuación recibida por el VMAF, aunque afecta más gravemente a la jugabilidad.

Por último, para la realización de este Trabajo de Fin de Máster, se ha hecho uso de varios conocimientos y técnicas aprendidos en el máster. En concreto, destaca la influencia de siete asignaturas, como son Gestión de Redes, Planificación de Redes, Tecnologías y Servicios de Internet, Sistemas Electrónicos Integrados y Proyectos en Ingeniería de Telecomunicación y también, aunque en menor medida, Teoría de las Comunicaciones y Procesamiento Avanzado de Señales Multimedia. De estas asignaturas los conocimientos y aptitudes necesarias y reforzadas durante la ejecución del trabajo han sido:

- Gestión de Redes: se ha realizado una gestión de red integrada, en la que se han seguido las 4 fases para la monitorización de la red, como son la definición de la información que se monitoriza, el acceso a la información, el diseño de mecanismos y el procesado de ésta.
- Planificación de redes: el tema que más se ha tratado ha sido las medidas de red aprendidas.

Se han hecho tanto monitorizaciones activas como pasivas, así como medidas de la QoE y de la QoS.

- Tecnologías y Servicios de Internet: se han hecho uso de aplicaciones Web como es el caso de Grafana, y se ha creado un servidor de aplicación web como el módulo que controla el codificador LHE de la Zynqberry.
- Sistemas Electrónicos Integrados: se han manejado sistemas como Zynqberry y Raspberry.
- Proyectos en Ingeniería de Telecomunicación: se ha enfocado este trabajo como un proyecto real de ingeniería, ya que está enmarcado en uno.
- Teoría de las Comunicaciones: se ha trabajado con un codificador, que a pesar que no se ha estudiado durante la asignatura, el hecho de haber estudiado otros, ayuda a la comprensión del LHE.
- Procesamiento Avanzado de Señales Multimedia: desde el punto de vista del procesado de señal, este trabajo puede enfocarse como un clasificador. En este clasificador se introducen los parámetros QoS medidos, y se obtienen dos clasificaciones, si la red es buena o no, y en caso de no serlo, qué parámetros dan la QoE más alta.

6.2. Trabajo futuro

Tras la realización de este proyecto y el análisis de los resultados obtenidos, se proponen las siguientes mejoras:

- Sobre la monitorización, se propone realizar nuevas medidas más precisas. Por ejemplo, la latencia se mide como la mitad del tiempo de ida y vuelta, lo que no se correspondía con la realidad del caso estudiado.
- Aumentar la independencia entre las variables medidas, para así poder realizar un clasificador más fiable.
- Realizar un estudio sobre la influencia de los cuadros por segundo en la calidad de experiencia de jugabilidad para este tipo de juego.
- Crear una red que conozca el funcionamiento del protocolo Q4S, de forma que se pueda adaptar las preferencias de red, crear canales, etc.
- Simular un entorno con tráfico cruzado con herramientas como mininet-Wifi [35], para poder probar las máquinas de estado.
- Simular un entorno real, con pérdidas reales que no son fácilmente simulables por software.

Glosario de acrónimos

- RTT, *Round Trip Time*: tiempo de ida y vuelta.
- OWD, *One Way Delay*: retardo de un camino.
- SDP, *Session Description Protocol*: protocolo de descripción de sesión.
- Q4S, *Quality for Service*: calidad para el servicio.
- QoS, *Quality of Service*: calidad de servicio.
- QoE, *Quality of Experience*: calidad de experiencia.
- UDP, *User Datagram Protocol*: protocolo de datagramas de usuario.
- TCP, *Transmission Control Protocol*: protocolo de control de transmisión.
- ITU, *International Telecommunication Union*: unión internacional de telecomunicaciones.
- MOS, *Mean Opinion Score*: puntuación de opinión media.
- DMOS, *Diference MOS*: diferencia entre MOS.
- GOS, *Game Outcome Score*: puntuación media de un juego.
- VMAF, *Video Multimethod Assessment Fusion*: fusión multimétodo de la evaluación de vídeo.
- VIF, *Visual Information Fidelity*: fidelidad de la información visual.
- DLM, *Detail Loss Metric*: métrica de pérdida de detalle.
- FPS, *First Person Shooter*: tirador en primera persona.
- LHE, *Logarithmical Hop Encoding*: codificación de salto logarítmico.
- RTP, *Real time Transport*: protocolo en tiempo real.
- RTSP, *Real Time Streaming Transport*: protocolo de transmisión en tiempo real.
- HTTP, *Hypertext Transfer Protocol*: protocolo de transferencia de hipertexto.
- CPU *Central Processing Unit*: unidad central de procesamiento.
- IP *Internet protocol*: protocolo de internet.

- IETF *Internet Engineering Task Force*: grupo de trabajo de ingeniería de internet.
- ETSI *European Standards Institute*: instituto de estandarización europea.
- FR, *Full Reference*: referencia completa.
- NR, *No Reference*: sin referencia.
- FR, *Reduced Reference*: referencia reducida.
- ISP, *Internet Service Provider*: proveedor de servicios de internet.
- PC, *Personal Computer*: computador personal.
- USB, *Universal Serial Bus*: bus universal en serie.
- ECDF, *Empirical Cumulative Distribution Function*: función de densidad empírica.

Bibliografía

[1] Joaquin Salvachua, Jose Javier Garcia, Juan Quemada, Monica Cortes, Luis Vizcaino, Gonzalo Fernandez, Jacobo Lajo, and Carlos Barcenilla. The Quality for Service Protocol. 2015.

[2] Ricky KP Mok, Edmond WW Chan, and Rocky KC Chang. Measuring the quality of experience of HTTP video streaming. In *Integrated Network Management (IM), 2011 IFIP/IEEE International Symposium on*, pages 485–492. IEEE, 2011.

[3] International Telecommunication Union – Telecommunication standardization. Definitions of terms related to quality of service. *ITU-T Recommendation E.800*, 2008.

[4] Eric Crawley, H Sandick, R Nair, and B Rajagopalan. A Framework for QoS-based Routing in the Internet. 1998.

[5] Guy Almes, Sunil Kalidindi, Matthew Zekauskas, and Al Morton. A one-way delay metric for IP performance metrics (IPPM). Technical report, 2016.

[6] David Mills, Jim Martin, Jack Burbank, and William Kasch. Network time protocol version 4: Protocol and algorithms specification. Technical report, 2010.

[7] Eduardo Miravalls-Sierra, David Muelas, Jorge E López de Vergara, Javier Ramos, and Javier Aracil. On the use of affordable COTS hardware for network measurements: Limits and good practices. *Information*, 9(2):43, 2018.

[8] Carlo Demichelis and Philip Chimento. IP packet delay variation metric for IP performance metrics (IPPM). Technical report, 2002.

[9] Constantinos Dovrolis, Parameswaran Ramanathan, and David Moore. What do packet dispersion techniques measure? In *INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 2, pages 905–914. IEEE, 2001.

[10] Xiliang Liu, Kaliappa Ravindran, and Dmitri Loguinov. A queueing-theoretic foundation of available bandwidth estimation: single-hop analysis. *IEEE/ACM Transactions on Networking*, 15(4):918–931, 2007.

[11] G Almes, S Kalidindi, and M Zekauskas. RFC 2680: A one-way packet loss metric for IPPM. *Status: Standards Track*, 1999.

[12] Jose Saldana, Julian Fernandez-Navajas, Jose Ruiz-Mas, Eduardo Viruete Navarro, and Luis Casadesus. The utility of characterizing packet loss as a function of packet size in

commercial routers. In *Consumer Communications and Networking Conference (CCNC), 2012 IEEE*, pages 346–347. IEEE, 2012.

[13] Jari Korhonen and Ye Wang. Effect of packet size on loss rate and delay in wireless links. In *Wireless Communications and Networking Conference, 2005 IEEE*, volume 3, pages 1608–1613. IEEE, 2005.

[14] Javier Ramos. Proactive measurement techniques for network monitoring in heterogeneous environments. 2013.

[15] Ricardo de O Schmidt, Anna Sperotto, Ramin Sadre, and Aiko Pras. Towards bandwidth estimation using flow-level measurements. In *IFIP International Conference on Autonomous Infrastructure, Management and Security*, pages 127–138. Springer, 2012.

[16] EG ETSI. 202 057-4:'User related QoS parameter definitions and measurement', 2005.

[17] Constantinos Dovrolis, Parameswaran Ramanathan, and David Moore. Packet-dispersion techniques and a capacity-estimation methodology. *IEEE/ACM Transactions On Networking*, 12(6):963–977, 2004.

[18] Mario Ruiz, Gustavo Sutter, Sergio López-Buedo, Javier Ramos, JE López de Vergara, and Javier Aracil. Leveraging open source platforms and high-level synthesis for the design of FPGA-based 10 GbE active network probes. In *ReConFigurable Computing and FPGAs (ReConFig), 2015 International Conference on*, pages 1–6. IEEE, 2015.

[19] Andreas Johnsson. On the comparison of packet-pair and packet-train measurements. In *Proc. Swedish National Computer Networking Workshop*, pages 241–250, 2003.

[20] Recommendation P ITU-T. 10/g. 100 amendment 1 new appendix i-definition of quality of experience (qoe). *International Telecommunication Union*, 2007.

[21] Debajyoti Pal and Tuul Triyason. A Survey of Standardized Approaches towards the Quality of Experience Evaluation for Video Services: An ITU Perspective. *International Journal of Digital Multimedia Broadcasting*, 2018, 2018.

[22] Fernando Kuipers, Robert Kooij, Danny De Vleeschauwer, and Kjell Brunnström. Techniques for measuring Quality of Experience. In *International Conference on Wired/Wireless Internet Communications*, pages 216–227. Springer, 2010.

[23] ITUT Rec. P. 10: Vocabulary for performance and Quality of Service, Amendment 2: New definitions for inclusion in Recommendation ITU-T P. 10/G. 100. *Int. Telecomm. Union, Geneva*, 2008.

[24] G Armitage and Sebastian Zander. Empirically measuring the QoS sensitivity of interactive online game players. 2004.

[25] Matthias Dick, Oliver Wellnitz, and Lars Wolf. Analysis of factors affecting players' performance and perception in multiplayer games. In *Proceedings of 4th ACM SIGCOMM workshop on Network and system support for games*, pages 1–7. ACM, 2005.

[26] M Suznjevic and J Saldana. Delay limits for real-time services. *draft-suznjevic-dispatch-delay-limits-00 (work in progress)*, 2015.

[27] Mark Claypool and Kajal Claypool. Latency and player actions in online games. *Communications of the ACM*, 49(11):40–45, 2006.

[28] Yanjiao Chen, Kaishun Wu, and Qian Zhang. From qos to qoe: A tutorial on video quality assessment. *IEEE Communications Surveys & Tutorials*, 17(2):1126–1165, 2015.

[29] Markus Fiedler, Tobias Hossfeld, and Phuoc Tran-Gia. A generic quantitative relationship between quality of experience and quality of service. *IEEE Network*, 24(2), 2010.

[30] Akira Takahashi, David Hands, and Vincent Barriac. Standardization activities in the ITU for a QoE assessment of IPTV. *IEEE Communications Magazine*, 46(2), 2008.

[31] Zhi Li, Anne Aaron, Ioannis Katsavounidis, Anush Moorthy, and Megha Manohara. Toward a practical perceptual video quality metric. *The Netflix Tech Blog*, 6, 2016.

[32] Henning Schulzrinne, Stephen Casner, Ron Frederick, and Van Jacobson. Rtp: A transport protocol for real-time applications. Technical report, 2003.

[33] Jose Javier García Aranda, Marina González Casquete, Mario Cao Cueto, Joaquín Navarro Salmerón, and Francisco González Vidal. Logarithmical hopping encoding: a low computational complexity algorithm for image compression. *IET Image Processing*, 9(8):643–651, 2015.

[34] E.5.1.2 Estudio de la caracterización del tráfico de la aplicación. Technical report, Racing Drones, 2018.

[35] Ramon R Fontes, Samira Afzal, Samuel HB Brito, Mateus AS Santos, and Christian Esteve Rothenberg. Mininet-wifi: Emulating software-defined wireless networks. In *Network and Service Management (CNSM), 2015 11th International Conference on*, pages 384–389. IEEE, 2015.