

UNIVERSIDAD DE MÁLAGA
ESCUELA TÉCNICA SUPERIOR DE
INGENIERÍA DE TELECOMUNICACIÓN



UNIVERSIDAD
DE MÁLAGA



TRABAJO FIN DE GRADO

EVALUACIÓN DE PRESTACIONES DEL
SERVICIO DE MULTI-STREAMING DE
VÍDEO «RESTREAM»

GRADO EN INGENIERÍA TELEMÁTICA

JOSÉ LUIS PUENTE BODOQUE
MÁLAGA, 2020

EVALUACIÓN DE PRESTACIONES DEL SERVICIO DE MULTI-STREAMING DE VÍDEO «RESTREAM»

Autor: José Luis Puente Bodoque

Tutor: Francisco Javier López Martínez

Departamento: Ingeniería de Comunicaciones (IC)

Titulación: Grado en Ingeniería Telemática

Palabras clave: Restream, retardo, latencia, *jitter*, calidad de vídeo percibido, ancho de banda, tasa de pérdidas, *streaming*, creador de contenido, usuario final.

Resumen

Los servicios de *multistreaming* han irrumpido con fuerza como soporte a creadores de contenido multimedia, posibilitando la emisión simultánea a múltiples plataformas de *streaming*. Las diferentes tecnologías empleadas por los distintos servicios de *streaming* de vídeo, y sus variados requisitos en términos de latencia o calidad de experiencia, hacen que surjan dudas razonables sobre si las prestaciones del servicio se verán degradadas al emplear un servicio de *multistreaming*.

En este trabajo, se analizan las prestaciones de la plataforma de *multistreaming* Restream, con el objeto de evaluar su funcionamiento respecto al caso de emitir directamente hacia plataformas de *streaming* convencionales como Twitch, YouTube o Facebook Live. Para ello, se diseña e implementa una plataforma de pruebas siguiendo las recomendaciones técnicas de Cisco y Amazon, y que permite evaluar el efecto de condiciones no ideales de red en la latencia extremo a extremo y la calidad de experiencia del usuario final.

Los resultados demuestran que el uso de Restream tiene un impacto limitado aunque no despreciable en la latencia extremo a extremo, si bien este difiere para las diferentes plataformas empleadas. En condiciones de *jitter* elevado en la red, el uso de Restream permite amortiguar mejor este efecto que en el caso de realizar *streaming* de manera directa hacia YouTube.

PERFORMANCE EVALUATION OF THE MULTISTREAMING SERVICE “RESTREAM”

Author: José Luis Puente Bodoque

Supervisor: Francisco Javier López Martínez

Department: Ingeniería de Comunicaciones (IC)

Degree: Grado en Ingeniería Telemática

Keywords: Restream, delay, latency, quality of experience, bandwidth, loss rate, streaming, streamer, end user.

Abstract

Multistreaming services have recently emerged as a support for multimedia content creators, enabling the simultaneous transmission of data to different commercial streaming platforms. The dissimilar technologies used for these video streaming services, and their different requisites and constraints in terms of latency and quality of experience, make a legitimate case for questioning whether the streaming performance would be degraded because of multistreaming.

In this work, we investigate the performance of the multistreaming platform Restream, with the ultimate goal of evaluating its operation with respect to the case of direct streaming towards conventional platforms like Twitch, YouTube o Facebook Live. We design and implement a testbed based on the technical recommendations by Cisco and Amazon, that allows us to evaluate the effect of non-ideal network conditions on the end-to-end latency and the quality of experience.

Results show that the use of Restream has a limited yet not negligible impact on the end-to-end latency, although such impact differs depending on the specific video streaming platform. In the presence of a high jitter, the use of Restream allows for a better buffering performance compared to the case of directly streaming to YouTube.

A Javier, mi tutor,
por facilitarme el trabajo con su generosidad y empatía.

A mis padres, de corazón,
por preocuparse por mí durante toda mi etapa universitaria.

José Luis Puente Bodoque

Acrónimos

ABR	Adaptive Bit Rate
ACK	Acknowledgment
AMF	Action Message Format
CBR	Constant Bit Rate
CDN	Content Delivery Network
DASH	Dynamic Adaptive Streaming over HTTP
HTTP	HyperText Markup Language
MPD	Media Presentation Description
NetEm	Network Emulator
QoE	Quality of Experience
QoS	Quality of Service
QUIC	Quick UDP Internet Connections
RTMP	Real Time Message Protocol
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
TFG	Trabajo Fin de Grado

Índice

Resumen	III
Abstract	v
Acrónimos	IX
I Introducción	1
1 Introducción y visión general	3
1.1 Motivación	3
1.2 Objetivos	4
1.3 Estructura del documento	4
II Desarrollo del proyecto	7
2 Fundamentos previos	9
2.1 Revisión del concepto de <i>streaming</i> de vídeo	10
2.2 Protocolos	12
2.2.1 RTMP	12
2.2.2 DASH	15
2.2.3 Google QUIC	18
2.3 Plataformas de <i>streaming</i>	19
2.3.1 Twitch	19

2.3.2	Facebook Live	20
2.3.3	YouTube	21
2.4	Plataforma Restream.io	22
2.5	Herramientas <i>software</i> empleadas	23
2.5.1	OBS	23
2.5.2	Wireshark	24
2.5.3	NetEm	24
2.5.4	Wondershaper	25
3	Plataforma de pruebas	27
3.1	Descripción del escenario de pruebas	28
3.1.1	Sin Restream	28
3.1.2	Con Restream	29
3.1.3	Método directo de medida de latencia	30
3.1.4	Elección del PC del creador de contenido	30
3.1.5	Información del archivo de vídeo	31
3.1.6	Ajuste del modo baja latencia	31
3.2	Descripción de casos de pruebas	32
4	Evaluación de prestaciones	35
4.1	Análisis de errores TCP	36
4.1.1	Enlace ascendente	36
4.1.2	Sin restricciones	37
4.1.3	Con restricciones	44
4.2	Medida de latencia media	51
4.2.1	Sin restricciones	51
4.2.2	Retardo incremental de 100 ms	51
4.2.3	Adición de pérdidas con granularidad 5 %	55
4.3	Efecto del <i>jitter</i> en casos extremos	58
4.4	Cálculo de la tasa de descarga límite	62
4.4.1	Procedimiento con Twitch	62

4.4.2 Resultados	66
4.5 Análisis y discusión de resultados	67
5 Conclusiones y líneas futuras	71
Bibliografía	74

Índice de figuras

2.1	Ejemplo de conversación en el puerto TCP 1935 entre cliente y servidor de ingesta	12
2.2	Establecimiento de la conexión RTMP (1): <i>handshake</i>	13
2.3	Establecimiento de la conexión RTMP (2): conexión	14
2.4	Diagrama de flujo TCP entre cliente y servidor de ingesta	15
2.5	Composición del MPD [9, Fig. 18]	16
2.6	Adaptación de tasa de descarga en DASH [10, Fig. 3]	17
2.7	Descarga progresiva en YouTube hasta 2018 [8, Fig. 8]	17
2.8	Plataforma de twitch.tv	20
2.9	RTMPS para subir el vídeo al servidor de Facebook Live	20
2.10	Plataforma de Facebook Live	21
2.11	Plataforma de Restream.io	23
3.1	Escenario de pruebas sin Restream	28
3.2	Escenario de pruebas con Restream	29
3.3	Marcas de tiempo en PC productor y consumidor	30
3.4	Activación del modo baja latencia en Twitch	32
3.5	Activación del modo baja latencia en YouTube	32
4.1	Ráfagas RTMP en sentido ascendente	36
4.2	Errores TCP en la máquina del usuario final en Twitch	37
4.3	Conversación entre <i>streamer</i> y usuario final en Twitch	38
4.4	Porcentaje de errores TCP en Twitch	38

4.5 Errores TCP (%) respecto al total de paquetes en Twitch	39
4.6 Conversación entre <i>streamer</i> y usuario final en YouTube	39
4.7 Errores TCP en el <i>stream</i> del usuario final en Facebook Live	40
4.8 Porcentaje de errores TCP en Facebook Live	40
4.9 Errores TCP (%) respecto al total de paquetes en Facebook Live .	41
4.10 Errores TCP en el <i>stream</i> del usuario final en Twitch con Restream	41
4.11 Porcentaje de errores TCP en Twitch con Restream	42
4.12 Errores TCP (%) respecto al total de paquetes en Twitch con Res-tream	42
4.13 Errores TCP en Facebook Live con Restream	43
4.14 Errores TCP (%) respecto al total de paquetes en Facebook Live con Restream	43
4.15 Errores TCP con restricciones en Twitch	44
4.16 Porcentaje de errores TCP con restricciones en Twitch	45
4.17 Errores TCP (%) respecto al total de paquetes con restricciones en Twitch	45
4.18 Errores TCP con restricciones en Facebook Live	46
4.19 Porcentaje de errores TCP con restricciones en Facebook Live .	46
4.20 Errores TCP (%) respecto al total de paquetes con restricciones en Facebook Live	47
4.21 Errores TCP con restricciones en Twitch con Restream	47
4.22 Porcentaje de errores TCP con restricciones en Twitch con Restream	48
4.23 Errores TCP (%) respecto al total de paquetes con restricciones en Twitch con Restream	48
4.24 Errores TCP con restricciones en Facebook Live con Restream .	49
4.25 Porcentaje de errores TCP con restricciones en Facebook Live con Restream	50
4.26 Errores TCP (%) respecto al total de paquetes con restricciones en Facebook Live con Restream	50
4.27 Evolución de la latencia en función del retardo en Twitch	53
4.28 Evolución de la latencia en función del retardo en YouTube	53
4.29 Evolución de la desviación típica en función del retardo en Twitch .	54

4.30 Evolución de la desviación típica en función del retardo en YouTube	54
4.31 Evolución de la latencia en función del retardo más pérdidas en Twitch	56
4.32 Evolución de la latencia en función del retardo más pérdidas en YouTube	56
4.33 Evolución de la desviación típica en función del retardo más pérdidas en Twitch	57
4.34 Evolución de la desviación típica en función del retardo más pérdidas en YouTube	57
4.35 Evolución de la latencia en función del <i>jitter</i> en Twitch	59
4.36 Evolución de la latencia en función del <i>jitter</i> en YouTube	59
4.37 Evolución de la desviación típica en función del <i>jitter</i> en Twitch	60
4.38 Evolución de la desviación típica en función del <i>jitter</i> en YouTube	61
4.39 <i>Throughput</i> en sentido ascendente en Twitch	63
4.40 <i>Throughput</i> en sentido descendente en Twitch	64
4.41 <i>Throughput</i> en sentido descendente en Twitch	64
4.42 <i>Goodput</i> en sentido descendente en Twitch	65

Índice de Tablas

4.1	Latencia media sin restricciones	51
4.2	Latencia con retardo incremental de 100 ms	52
4.3	Latencia con adición de pérdidas con granularidad 5 %	55
4.4	Latencia con el efecto del <i>jitter</i>	58
4.5	Tasas de descarga límite	66

Parte I

Introducción

Capítulo 1

Introducción y visión general

Contenido

1.1	Motivación	3
1.2	Objetivos	4
1.3	Estructura del documento	4

1.1. Motivación

En la última década, el *boom* experimentado por los servicios de *streaming* de video ha venido de la mano de diversos factores: por una parte, la rápida evolución de las redes móviles y cableadas ha posibilitado el soporte de una ingente cantidad de tráfico. Además, el avance en técnicas de codificación y compresión multimedia ha permitido mejorar la calidad de experiencia de los usuarios y a la vez reducir el volumen de datos generado. Por último, pero no por ello menos importante, el desarrollo de nuevos casos de uso diferentes de los originalmente planteados en el ámbito del *streaming* de video (videoconferencia y vídeo bajo demanda). Así, ha sido clave el desarrollo de nuevas plataformas de *streaming* que sirven como soporte a creadores de contenido que desarrollan su actividad profesional en ámbitos tan dispares como los deportes electrónicos o el *lifestyle*. Algunos ejemplos de dichas plataformas son Twitch, Stadia, Facebook Live, YouTube o TikTok.

En general, la mayoría de creadores de contenido transmiten directamente a una única plataforma, y, en consecuencia, generan una única fuente de ingresos. Con el fin de posibilitar la emisión de contenido hacia múltiples plataformas a la

vez, han surgido plataformas de *multistreaming* que permiten ampliar la audiencia de un creador de contenido y con ello sus ingresos.

En este contexto, diversas plataformas como Restream o Castr permiten retransmitir los medios procedentes del creador de contenido hacia varias plataformas de transmisión o *streaming* simultáneamente. Sin embargo, la gran heterogeneidad de plataformas de *streaming* disponibles lleva asociada una gran diversidad de técnicas de transmisión y codificación, así como diferentes requisitos en términos de latencia o calidad de experiencia.

En la literatura, no existe un estudio detallado del efecto de emplear plataformas de *multistreaming* en relación con la emisión tradicional. Por tanto, la respuesta a importantes preguntas claves está en el aire: ¿notaría el usuario final una diferencia o sería inapreciable? ¿Se degradaría la calidad de audio o vídeo? ¿Cuál es el impacto en la latencia, que afecta de manera decisiva a la interacción entre el *streamer* y el usuario final? ¿En qué medida afecta a la variación del retardo de paquetes de vídeo? ¿Sirve la actual tasa de codificación o debe adaptarse? ¿Cómo afecta a la calidad del vídeo percibido por el usuario?

1.2. Objetivos

El objetivo principal de este trabajo es evaluar las prestaciones de la plataforma de *multistreaming* Restream sobre la calidad del vídeo percibido por el usuario final y determinar en qué condiciones es recomendable su uso. Para ello, se realizará un análisis de sus prestaciones y de su robustez ante errores TCP durante la transmisión, latencia, *jitter*, pérdida de paquetes y ancho de banda de la red.

1.3. Estructura del documento

El presente documento se estructura en cinco capítulos. Después de este primer capítulo introductorio que nos ocupa, el segundo de ellos se dedica a los *Fundamentos previos*, y proporciona una introducción al concepto de transmisión o *streaming* de vídeo, así como el de retransmisión o *multistreaming* de vídeo. También presenta brevemente la historia de las principales plataformas, el servicio de *multistreaming* Restream, así como los protocolos que estas utilizan. Por último, se describirán las herramientas informáticas empleadas para realizar nuestro análisis.

A continuación, el capítulo de *Plataformas de pruebas* describe el escenario

de pruebas de transmisión directa y con Restream, expone el procedimiento de medida de latencia y señala otros aspectos importantes para la puesta a punto del escenario de pruebas.

En cuarto lugar se encuentra *Evaluación de prestaciones*, donde se llevarán a cabo los cuatro casos de prueba ideados para evaluar Restream, presentando datos organizados en tablas y gráficos y realizando comentarios detallados. Al final de este capítulo se analizará y relacionarán los resultados de forma más transversal.

Por último, en el capítulo de *Conclusiones y líneas futuras*, se discutirá en qué condiciones la plataforma Restream es una buena opción para un servicio de *multistreaming*, teniendo en cuenta los resultados arrojados por los casos de prueba. Finalmente, se identificarán algunas líneas posibles de trabajo futuro.

Parte II

Desarrollo del proyecto

Capítulo 2

Fundamentos previos

Contenido

2.1 Revisión del concepto de <i>streaming</i> de vídeo	10
2.2 Protocolos	12
2.2.1 RTMP	12
2.2.2 DASH	15
2.2.3 Google QUIC	18
2.3 Plataformas de <i>streaming</i>	19
2.3.1 Twitch	19
2.3.2 Facebook Live	20
2.3.3 YouTube	21
2.4 Plataforma Restream.io	22
2.5 Herramientas <i>software</i> empleadas	23
2.5.1 OBS	23
2.5.2 Wireshark	24
2.5.3 NetEm	24
2.5.4 Wondershaper	25

2.1. Revisión del concepto de *streaming* de vídeo

Los servicios de *streaming* o emisión continua de datos, usualmente vídeo y audio, se caracterizan por que el usuario reproduce el contenido a la vez que lo está recibiendo; esto quiere decir que la descarga de los datos también es continua. Tanto para servicios de *streaming* de contenidos almacenados (como Netflix), como de manera fundamental para servicios de *streaming* en vivo (como Twitch), es deseable que este flujo continuo sea ininterrumpido. Sin embargo, las condiciones de una red *best-effort* como es Internet fluctúan más de lo deseable, por lo que es preciso utilizar una serie de elementos que nos ayuden a mitigarlos.

La transmisión de contenido multimedia en vivo requiere de una fuente de grabación de vídeo y audio, un codificador que digitalice el vídeo, un editor y una red de distribución de contenido o CDN para su distribución y entrega.

En cuanto a los componentes necesarios para un acceso claro y continuo, la transmisión multimedia se apoya en:

- Códigos: aplicación que codifica el audio y el vídeo digital utilizando una serie de funciones algorítmicas para comprimir el fichero multimedia. Pueden ser con pérdidas (desecha la información imperceptible) o sin pérdidas (desecha solo la redundancia).
- Transporte (nivel de aplicación): la información multimedia se distribuye desde el servidor de retransmisión a un cliente utilizando un determinado protocolo de la capa de aplicación, como RTMP, RTP o incluso HTTP.
- Control (nivel de aplicación): realimentación desde el usuario para mejorar la calidad percibida, como RTSP o RTCP. El *streaming* adaptativo, orientado a baja latencia, no usa protocolos destinados a este fin y en su lugar se utilizan mecanismos de adaptación de la tasa de descarga controlados por el cliente.

En cuanto a los protocolos, estos han de aligerar el tamaño del paquete al máximo con el fin de entregarlos más rápido. En este sentido, UDP y RTSP son más rápidos que TCP y HTTP porque si los primeros sufren alguna pérdida no solicitan ninguna retransmisión. En un servicio continuo casi en tiempo real, la pérdida de algún paquete es inapreciable para el usuario final; pero si la cantidad de pérdidas excede un determinado umbral, la QoE (calidad de experiencia) del usuario sí se vería afectada. En este caso es útil y preciso utilizar TCP para solicitar retransmisiones a costa de aumentar el tiempo inicial de llenado del *buffer*. Los protocolos que utilizan las plataformas de *streaming* que se tratarán en este TFG son:

- Facebook: TLS 1.2/TCP.
- Twitch: TLS 1.3/TCP.
- YouTube: QUIC/UDP.
- Restream: TLS 1.2/TCP.

Un componente fundamental en *streaming* es el *buffer* de datos. Almacena el flujo de descarga en la memoria RAM del usuario para inmediatamente entregarlo al proceso consumidor de la aplicación que reproduce el *streaming*. Cuando la aplicación consume un fragmento de flujo de memoria, este se borra de ella para dejar espacio al flujo restante.

Por otro lado, es necesario conocer los parámetros de calidad de servicio (QoS), los cuales vamos a redefinir para ajustarlos a nuestro análisis.

- **Tasa de transferencia efectiva o *throughput*.** Velocidad real de transferencia de datos (kbps o Mbps). Debe ser ligeramente superior a la tasa de codificación del vídeo en transmisión de vídeo grabado.
- **Retardo o *delay*.** En el entorno de este TFG, es el tiempo de retención de los paquetes en la cola de salida.
- **Latencia** (velocidad de interacción). En el entorno de este TFG, es el retardo extremo a extremo (tiempo de ida) del vídeo desde que se genera hasta que se visualiza por el usuario.
- **Jitter.** Variación del retardo extremo a extremo.
- **Pérdidas.** Tráfico que se pierde en Internet desde la máquina productora hasta la consumidora del *streaming*.

Para medir el rendimiento de la transmisión hay que sumar a la lista anterior el ancho de banda (BW) de la conexión a Internet, que es la velocidad máxima a la que la información puede ser transferida¹, y el *throughput* (Th), velocidad real a la que la información finalmente es transferida. Y si, además, descartamos la sobrecarga que suponen las cabeceras de los protocolos y paquetes transmitidos obtenemos el *goodput* (Gp). En cualquier red telemática se cumple la expresión 2.1 a la que aludiremos en uno de los casos de prueba.

$$BW > Th > Gp \quad (2.1)$$

¹Se puede medir realizando un test de velocidad con cable UTP categoría 5e.

2.2. Protocolos

A continuación, se explicarán los protocolos de nivel de aplicación que entran en juego en *streaming* de vídeo en Internet.

2.2.1. RTMP

El protocolo RTMP (Real Time Messaging Protocol), propietario de Adobe y situado sobre TCP, permite conexiones persistentes y de baja latencia. Es el encargado de entregar el contenido audiovisual desde el codificador hasta el servidor de transmisión (como Twitch o Facebook Live) o *multistreaming* (Restream.io). Está definido de acuerdo a la mejor práctica actual [4] para definir un protocolo de ingesta. Estos protocolos se encargan de hacer llegar los medios y metadatos de la fuente de medios en vivo hacia los servidores de ingesta de una entidad de procesamiento (Twitch, Facebook, YouTube) o intermedia (Restream.io).

RTMP utiliza el puerto TCP 1935 por defecto. El codificador de fuentes utilizado en este TFG utiliza la variante RTMPT, que encapsula RTMP dentro de una solicitud HTTP. Por lo tanto, dirigirá los medios al puerto 80, bien conocido, para evitar que la mayoría de los filtros de tráfico de las redes corporativas bloquen el tráfico al puerto 1935. También utiliza el puerto 443 para proporcionar seguridad TLS a cierto tráfico. La figura 2.1 muestra una captura de Wireshark que ejemplifica las conversaciones TCP generadas al transmitir una fuente multimedia hacia el servidor de ingesta.

Address A	Port A	Address B	Port B
192.168.1.137	49383	52.223.197.202	1935
192.168.1.137	49384	88.221.52.137	80
192.168.1.137	49266	52.38.241.141	443
192.168.1.137	49264	54.186.246.131	443
192.168.1.137	49265	54.186.246.131	443
192.168.1.137	49271	151.101.134.167	443

Figura 2.1: Ejemplo de conversación en el puerto TCP 1935 entre cliente y servidor de ingesta

Según la documentación oficial de Adobe [5], una vez que RTMP establece una conexión TCP completa estos tres pasos:

1. Handshake

Se efectúa el establecimiento de la conexión mediante el intercambio de tres paquetes o *chunks* (fragmentos del vídeo). El proceso sigue así:

- El cliente inicializa la conexión enviando el paquete C0 con valor constante de 0x03 que representa la versión actual.
- El cliente envía C1, sin recibir todavía S0, con 1436 bytes de los cuales los cuatro primeros representan la época, los siguientes cuatro valen 0 y el resto son aleatorios.
- El servidor envía S0 y S1 que son ecos de C0 y C1, respectivamente, más S2 que es una copia de C1.
- El cliente envía C2, copia de S1. En este punto la conexión se da por establecida.

La figura 2.2 muestra los *chunks* correspondientes al *handshake*.

No.	Time	Source	Destination	Protocol	Length	Info
13	4.333768	192.168.1.137	52.223.197.202	TCP	66	49383 → 1935 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=256 SACK_PERM=1
14	4.347427	52.223.197.202	192.168.1.137	TCP	60	1935 → 49383 [SYN, ACK] Seq=0 Ack=1 Win=0 Len=0 MSS=1460
15	4.347497	192.168.1.137	52.223.197.202	TCP	54	49383 → 1935 [ACK] Seq=1 Ack=1 Win=64240 Len=0
16	4.360355	52.223.197.202	192.168.1.137	TCP	60	[TCP Window Update] 1935 → 49383 [ACK] Seq=1 Ack=1 Win=29200 Len=0
17	4.360387	192.168.1.137	52.223.197.202	TCP	1514	49383 → 1935 [ACK] Seq=1 Ack=1 Win=64240 Len=1460
18	4.360387	192.168.1.137	52.223.197.202	RTMP	131	Handshake C0+C1
19	4.373891	52.223.197.202	192.168.1.137	TCP	60	1935 → 49383 [ACK] Seq=1 Ack=1538 Win=32120 Len=0
20	4.404973	52.223.197.202	192.168.1.137	TCP	60	1935 → 49383 [PSH, ACK] Seq=1 Ack=1538 Win=32120 Len=1
21	4.405745	52.223.197.202	192.168.1.137	TCP	1514	1935 → 49383 [ACK] Seq=2 Ack=1538 Win=32120 Len=1460
22	4.405789	192.168.1.137	52.223.197.202	TCP	54	49383 → 1935 [ACK] Seq=1538 Ack=1462 Win=64240 Len=0
23	4.405829	52.223.197.202	192.168.1.137	TCP	1490	1935 → 49383 [PSH, ACK] Seq=1462 Ack=1538 Win=32120 Len=1436
24	4.405873	52.223.197.202	192.168.1.137	RTMP	230	Handshake S0+S1+S2
25	4.405896	192.168.1.137	52.223.197.202	TCP	54	49383 → 1935 [ACK] Seq=1538 Ack=3074 Win=64240 Len=0
26	4.405924	192.168.1.137	52.223.197.202	TCP	1514	49383 → 1935 [ACK] Seq=1538 Ack=3074 Win=64240 Len=1460
27	4.405924	192.168.1.137	52.223.197.202	RTMP	130	Handshake C2
28	4.419156	52.223.197.202	192.168.1.137	TCP	60	1935 → 49383 [ACK] Seq=3074 Ack=3074 Win=35040 Len=0

Figura 2.2: Establecimiento de la conexión RTMP (1): *handshake*

2. Conexión

El cliente envía objetos ActionScript al servidor como campos del formato de mensaje AMF (Action Message Format) por el que se solicita la conexión comunicando sus capacidades. El servidor le responde con el código `NetConnection.Connect.Success` si tuvo éxito y las capacidades de la conexión. La figura 2.3 muestra los mensajes que ambos se envían.



Figura 2.3: Establecimiento de la conexión RTMP (2): conexión

3. Flujo

Por último, tras el paquete `onStatus('NetStream.Publish.Start')` comienza el flujo de vídeo.

La figura 2.4 muestra, en forma de diagrama de flujo, la parte inicial del intercambio de mensajes entre la máquina cliente y el servidor de ingesta. En él podemos ver los tres pasos del establecimiento de la conexión y a continuación el envío de los medios.

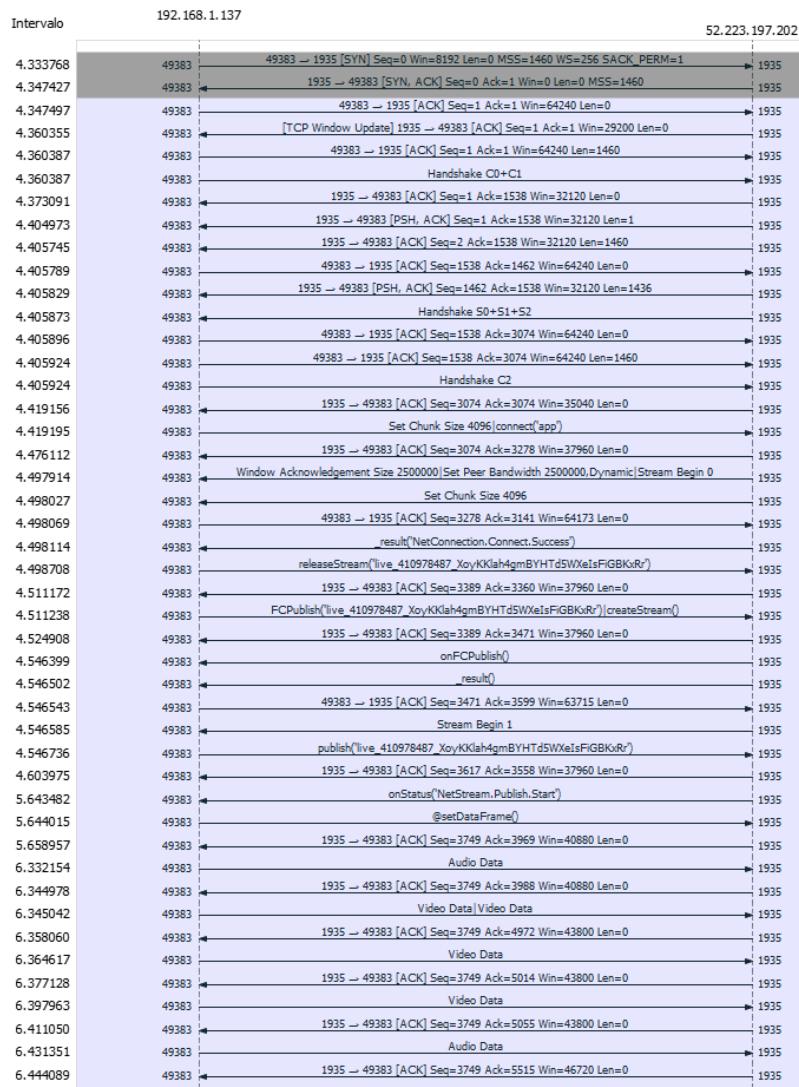


Figura 2.4: Diagrama de flujo TCP entre cliente y servidor de ingestión

2.2.2. DASH

El protocolo DASH (Dynamic Adaptive Streaming over HTTP) es una tecnología que trocea el fichero multimedia en una secuencia de trozos de video de una duración fija, que se denominan segmentos o *chunks* (pedazos). Estos trozos de vídeo se encuentran disponibles en diferentes calidades, lo que permite al cliente descargarlos de forma progresiva de acuerdo con la tasa ofrecida por la red en cada momento (ABR, adaptive bit rate). En principio, funciona directamente sobre HTTP.

mente sobre HTTP/TCP con el método GET (no usa UDP). También se lo conoce como MPEG-DASH porque MPEG fue el grupo de expertos que lo desarrolló y estandarizó para el *streaming* sobre HTTP.

DASH usa un modelo cliente-servidor en el que el cliente DASH solicita al servidor HTTP la descripción de los medios, MPD (Media Presentation Description). Se trata de una especie de carta para saber dónde y cómo realizar las solicitudes de los fragmentos del *streaming* al servidor. En ella se incluyen referencias temporales relativas al flujo de vídeo y se compone de múltiples *Periods*, los cuales pueden incluir varios *AdaptationSets*, que a su vez pueden contener un número variable de *Representations*, y cada uno de estos pueden incluir múltiples *Segments* o *chunks*. La figura 2.5 ilustra un ejemplo de cómo se compone el MPD.

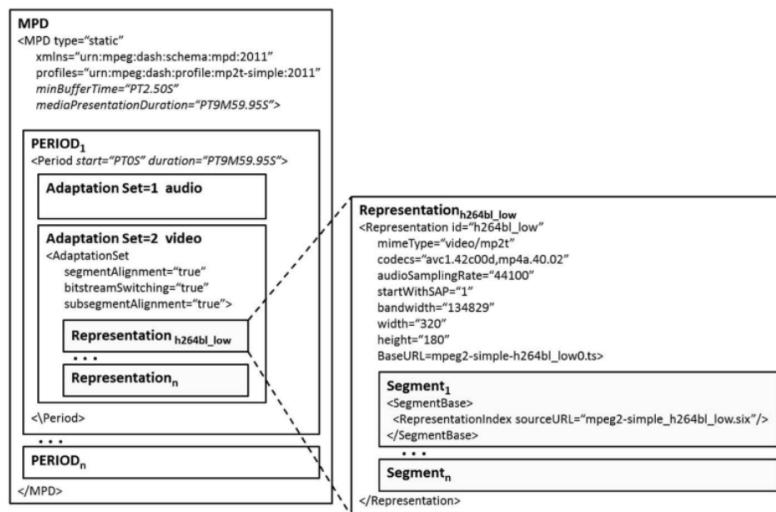


Figura 2.5: Composición del MPD [9, Fig. 18]

DASH funciona según los siguientes aspectos clave:

- Duración de fases ON/OFF + periodo de *chunk*.
 - Elige la calidad del próximo *chunk* a partir de cómo se recibieron los últimos.
 - Si el rendimiento, R , que ofrece la red es alto, ABR puede seleccionar la calidad más alta para proporcionar la mejor QoE al usuario. Si es bajo, se elige una calidad más baja para evitar pausas durante la reproducción.
 - El servidor ofrece un conjunto de calidades finitas hasta una R máxima, pudiendo el cliente soportar una R' todavía mayor.

La figura 2.6 muestra un ejemplo de adaptación de la tasa ofrecida.

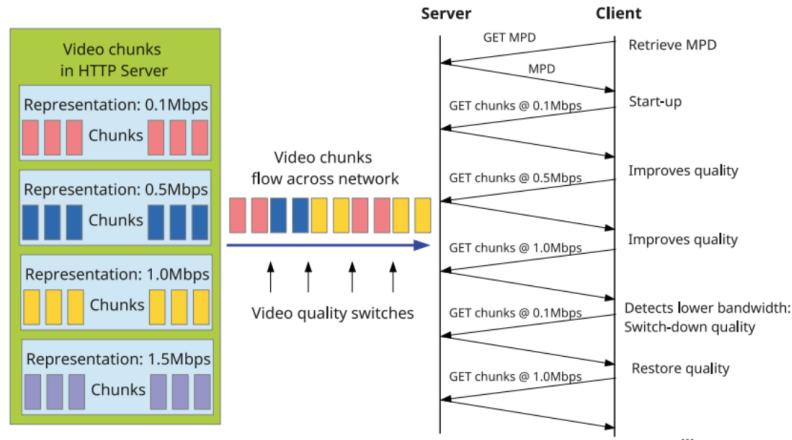


Figura 2.6: Adaptación de tasa de descarga en DASH [10, Fig. 3]

YouTube utilizaba hasta 2018 este modo de descarga progresiva que es representativo de cómo funciona hasta el día de hoy cualquier plataforma de servicios de vídeo bajo demanda. En la figura 2.7 se muestra una gráfica de la descarga de un vídeo para su visualización en el navegador de escritorio. En ella se representa los megabytes descargados a lo largo del tiempo. El trazo azul es el vídeo recibido, el trazo rojo discontinuo es el reproducido y el naranja punteado es el almacenado en el *buffer*. Este último es la diferencia entre los bytes recibidos y reproducidos.

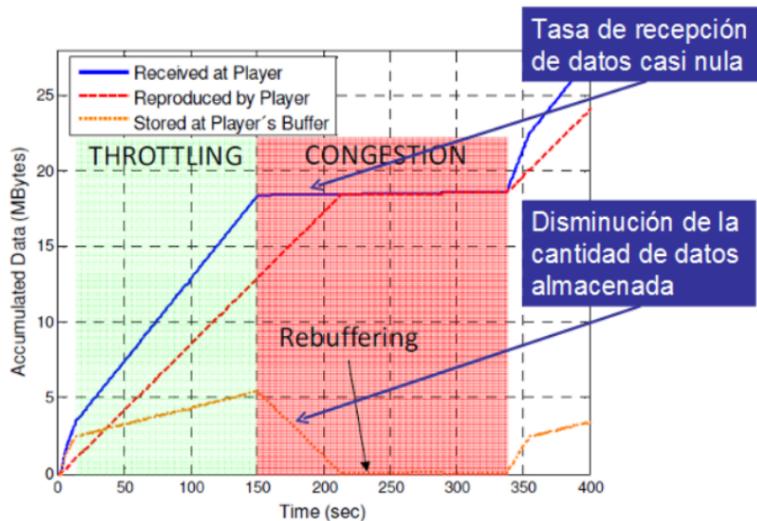


Figura 2.7: Descarga progresiva en YouTube hasta 2018 [8, Fig. 8]

Al principio del trazo azul, de muy corta duración, apreciamos una pendiente mayor que la del resto de la línea en la que el *buffer* se llena muy deprisa con vídeo a calidad baja, media o incluso alta o la más alta si la conexión y la red son favorables. El propósito de este llenado inicial es el de transportar datos lo más rápido posible al reproductor para tener al usuario esperando el menor tiempo posible. El trazo rojo discontinuo, que empieza en el instante cero o muy cercano a cero, es indicativo de que el usuario espera nada o un tiempo muy cercano a cero para ver algo. Esto sería lo ideal, aunque en la práctica siempre existía una espera de entre nada y apenas un decisegundo –a día de hoy, en 2020, esta espera es inapreciable con la implantación de QUIC y las superconexiones de fibrá óptica–. Los pedazos de vídeo descargados se almacenan en un *buffer*. Una tendencia ascendente de llenado del *buffer* es buena señal y significa que el *buffer* se llena a una velocidad mayor que la de consumición por el reproductor. Cuanto más vídeo haya almacenado, más vídeo podrá ver el usuario en caso de que se produzca un hipotética congestión en la red que detenga la descarga. Si la congestión persiste demasiado tiempo, el *buffer* se vacía y la reproducción se interrumpe. En el caso de la figura, tenemos que en el segundo 150 se produce un evento que congestionó la red y la curva azul se aplana, lo que quiere decir que la tasa de descarga es aproximadamente cero. Como ya no hay aporte de pedazos de vídeo, el reproductor vacía el *buffer* poco a poco hasta que en el segundo 215 ya no queda ningún byte y el vídeo se detiene –ahora el usuario ve una rueda girar sobre la imagen congelada que simboliza el *rebuffering*–. Cuando la red se recupera de la congestión en el segundo 335, la tasa de descarga vuelve al valor previo a la congestión y el *buffer* recupera su tendencia ascendente de manera que el usuario vuelve a poder visualizar el vídeo tan pronto como haya vídeo suficiente almacenado para ello. Al inicio de la recuperación, notamos otra vez un llenado rápido del *buffer* para minimizar el tiempo de espera al usuario.

2.2.3. Google QUIC

Motivado por el aumento de la latencia que supone incrementar la seguridad de la web, Google diseña QUIC en 2013, un protocolo experimental de baja latencia. Apunta a ser el equivalente de DASH con TCP, es decir, similar a TCP+TLS+HTTPS, pero sobre UDP y objetivo 0-RTT en el establecimiento de la conexión (aunque hoy es igual a 1-RTT todavía).

Los protocolos IP y TCP/UDP están implementados en el espacio del *kernel* y son muy difíciles de modificar. Tanto que los diseñadores del SO tendrían que lanzar varias actualizaciones del sistema en un proceso que duraría varios meses e incluso años para que llegase finalmente a todas las máquinas con dicho SO. Por este motivo, QUIC se implementa encima del nivel de aplicación y por debajo

de la API HTTP/2.

Utiliza un mecanismo de multiplexación de flujos, como explica [11], que entrelaza los datos de varios flujos y minimiza las pérdidas por posibles ráfagas de errores. También evita el *head-of-line blocking* de TCP que impide el deslizamiento de la ventana de recepción para recibir más datos si falta aún el segmento en la primera posición.

En la actualidad, QUIC es el protocolo por defecto para los servicios de *streaming* de Google, como los integrados dentro de Chrome o Gmail, o la plataforma YouTube.

2.3. Plataformas de *streaming*

A continuación se introducirán las plataformas de *streaming* que serán analizadas en el marco de este TFG.

2.3.1. Twitch

En el año 2007, los programadores Justin Kan y Emmet Shear crean la plataforma Justin.tv. Kan decide convertirla en su *espectáculo de realidad* personal para transmitir, en el canal que lleva su nombre, su vida diaria 24 horas al día, 7 días a la semana. Como carecía de popularidad, dejó que otras personas también pudieran transmitir a través de la plataforma para obtener mayores visitas.

Redes corporativas como Microsoft y un número significativo de jugadores empezaron a usar la plataforma atribuyéndola gran popularidad. En el año 2011 crearon Twitch, de manera paralela a Justin.tv, destinada a alojar los deportes electrónicos. El nombre está inspirado en el término *twitch gameplay* que significa el tiempo de reacción de los jugadores. Como la mayoría de la comunidad de Justin.tv eran jugadores acabaron migrando a Twitch y en 2014 acabó cerrando para destinar todos los recursos a Twitch.

En 2013, Twitch tenía un promedio de 43 millones de espectadores y en febrero de 2014 se convierte en la cuarta mayor fuente de tráfico en Internet en Estados Unidos. La plataforma mejoró técnicamente durante la primera mitad de 2014 y en septiembre Amazon adquiere Twitch por 970 millones de dólares.

Las grandes prestaciones que ofrece la convierten en la plataforma más popular para *streaming* de video en vivo en la categoría de videojuegos (*gaming*). En este TFG justificaremos experimentalmente por qué es una de las más consistentes respecto a la latencia.

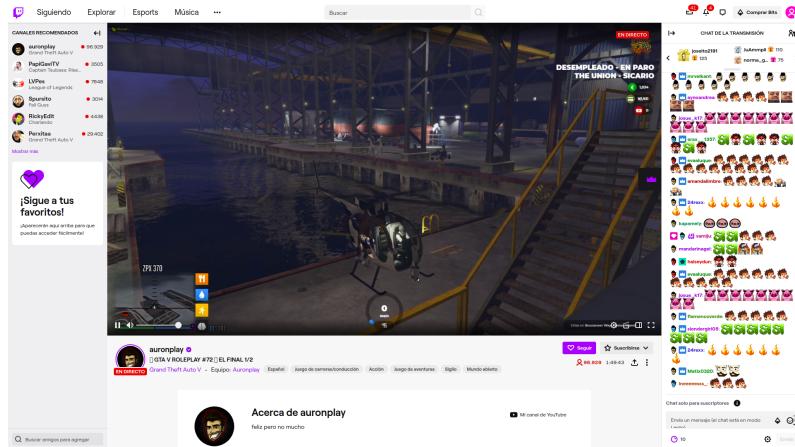


Figura 2.8: Plataforma de twitch.tv

2.3.2. Facebook Live

Empieza a funcionar a partir de 2016. A diferencia de las otras dos plataformas, Facebook Live no es una plataforma independiente sino que está integrada dentro de Facebook.

Facebook Live utiliza el protocolo RTMPT en enlace ascendente de la transmisión, pero la sesión que encapsula lleva paquetes RTMPS (RTMP sobre conexión TLS/SSL), como muestra la figura 2.9.

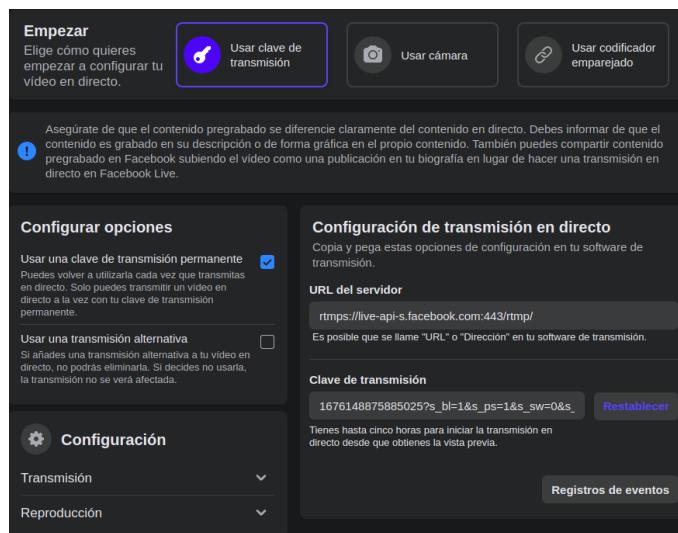


Figura 2.9: RTMPS para subir el vídeo al servidor de Facebook Live

La figura 2.10 muestra una transmisión en vivo desde el navegador de escritorio.



Figura 2.10: Plataforma de Facebook Live

Las estadísticas que convierten a Facebook Live en objeto de análisis son:

- Facebook es la red social con más usuarios activos mensuales.
- Los usuarios pasan más tiempo en Facebook que en ninguna otra red social.

2.3.3. YouTube

En febrero de 2005, Chad Hurley, Steve Chen y Jawed Karim, tres antiguos empleados de PayPal, fundan YouTube ante las dificultades de compartir videos grabados por ellos en una fiesta. La idea original de Karim era crear una página de citas donde las personas pudieran calificarse según sus videos.

El 23 de abril de 2005 se sube el primer video a la plataforma titulado *Me at the zoo* y en poco tiempo los creadores se percataron de que la comunidad subía todo tipo de videos en contra de la idea original. El tráfico se disparó en muy poco tiempo por la curiosidad que despertaba esta nueva plataforma y grandes compañías como Nike también la utilizaron para colgar sus anuncios.

En diciembre de 2005 la página ya era visitada unas 50 millones de veces y tras publicar el video *Lazy Sunday* del programa *Saturday Night Live*, las visitas se dispararon meteóricamente.

En octubre de 2006 se rumoreó que Google podría comprar la compañía por 1600 millones de dólares. Finalmente, ese mismo mes Google compra la compañía por 1650 millones de dólares.

Curiosamente, YouTube todavía no ha logrado reportar ganancias a Google, produciendo en un principio más gastos que beneficios aunque ha mejorado con los años hasta situarse en un balance nulo. Entre las causas de que no sea rentable se sitúa el gran coste de mantenimiento de los equipos que hacen posible la rápida reproducción del contenido.

La estadísticas que convierten a YouTube en objeto de análisis son:

- Tiene 2000 millones de usuarios activos en todo el mundo situándose la segunda, justo por detrás de Facebook.
- Los usuarios ven en promedio 1000 millones de horas diariamente, generando miles de millones de visitas.

2.4. Plataforma Restream.io

Es un servicio de retransmisión de contenidos multimedia hacia múltiples plataformas al mismo tiempo. Andrew Surzynskyi y Alexander Khuda la fundan en 2015 para ayudar a los creadores de contenido a ensanchar su audiencia a más de 30 plataformas y previsiblemente aumentar la monetización que reciben por cada retransmisión, como ilustra la figura 2.11.

Este servicio cuenta con más de 8 millones de retransmisiones mensuales y más de 600 millones de usuarios que consumen estos contenidos a través de esta plataforma de *multistreaming*.

Restream utiliza el códec HE-AAC (High-Efficiency Advanced Audio Coding): compresión con pérdidas definido como un perfil MPEG-4 Parte 3 de audio en la ISO/IEC 14495-3 basado en la replicación de banda espectral y el estéreo paramétrico. Para el vídeo utiliza H.264/MPEG-4 Parte 10.

Entre las utilidades que se infieren del servicio destacan las siguientes:

- **Multistreaming:** capacidad para transmitir simultáneamente a múltiples plataformas.
- **Chat global** para leer y responder comentarios en una sola ventana.
- **Analítica de datos:** medida de parámetros relacionados con la calidad de servicio.

- **Vídeos en la nube:** posibilidad de guardar las retransmisiones en la nube durante 15 o 30 días. Se trata de un servicio de pago.
- **La audiencia decide la plataforma que más le convenza.** Por ejemplo, en dispositivo móvil por YouTube y en ordenador por Twitch.

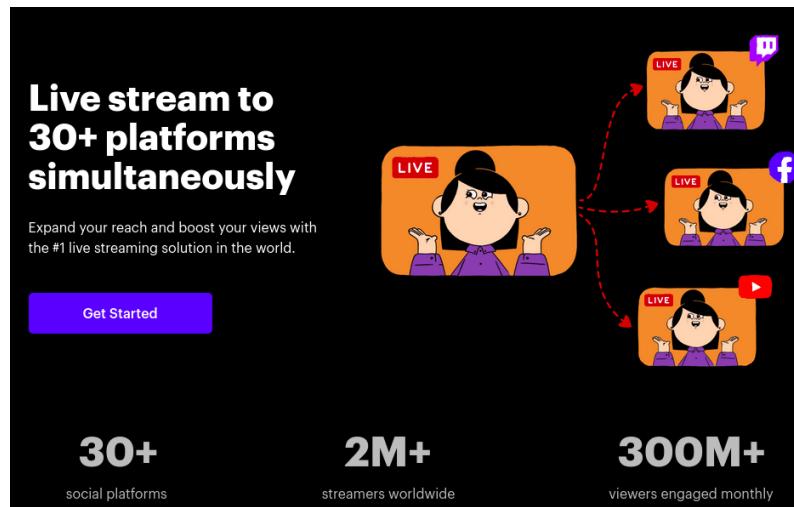


Figura 2.11: Plataforma de Restream.io

2.5. Herramientas *software* empleadas

En este apartado se presentan las herramientas y aplicaciones empleadas en la realización de las pruebas desarrolladas en el capítulo 4.

2.5.1. OBS

Se trata de un programa gratuito y de código abierto para la grabación y transmisión de vídeo en vivo por Internet. Permite la captura y mezcla de vídeo/audio en tiempo real, composición de escenas, codificación, retransmisión y grabación de vídeo.

Para la emisión de vídeo se ha utilizado el codificador x264, que prioriza el uso de la CPU en vez de las tarjetas gráficas, modo CBR a 3800 kbps, un intervalo de fotogramas clave de 0 segundos (para minimizar la latencia), un perfil de uso de CPU *veryfast* y perfil *main*. La frecuencia de muestreo del audio es 44,1 kHz, canales estéreo y codificación de audio a un *bitrate* de 160 kbps.

Para la configuración de OBS se han seguido las guías en línea [3] y [2].

2.5.2. Wireshark

Wireshark es un analizador de tráfico ampliamente utilizado en redes de comunicaciones que permite capturar el tráfico que pasa a través de una interfaz de red con el fin solucionar algún problema en la red.

Vamos a utilizarlo para comprobar que en la práctica las plataformas utilizan los protocolos que esperamos que utilicen (RTMPT, HTTP/2, UDP, TCP) y vamos a utilizar los filtros para seleccionar el flujo de interés y aplicarle las herramientas de estadísticas: conversaciones, IO/Graph, diagramas de flujo y de gráfica de flujo TCP.

2.5.3. NetEm

NetEm (Network Emulator) es un emulador de red preinstalado en plataformas Linux. Se trata de una mejora de las facilidades del control de tráfico que permite añadir retardo, pérdida de paquetes, duplicación y muchas otras características. NetEm está implementado usando las facilidades de calidad de servicio (QoS) ya presentes en el *kernel* de Linux.

Las opciones que utilizaremos en las pruebas realizadas en el capítulo 4 son:

- **delay, jitter y correlación:** añade un retardo fijo (ms) a los paquetes salientes de la interfaz de red seleccionada. Los parámetros opcionales permiten introducir una variación del retardo (ms) y una correlación (%) con el valor del paquete anterior.
- **loss:** emula pérdida de paquetes (%). El parámetro opcional permite introducir una correlación (%) con el valor del paquete anterior.

Para lanzar una orden es necesario escribir al principio la palabra `sudo` para dar privilegios de administrador. También es necesario escribir el nombre de la interfaz de red (`enp3s0` en nuestro caso) y asignar un nombre a la nueva disciplina de cola, `root`. Por ejemplo, para introducir un retardo fijo de 100 ms con variación de 20 ms y distribución normal:

```
$sudo tc qdisc add dev enp3s0 root netem delay 100ms 20ms
distribution normal
```

El manual de Linux y [6] han servido como guía de uso de la herramienta.

2.5.4. Wondershaper

Wondershaper es un conformador de tráfico para Linux que permite limitar el ancho de banda de un adaptador de red seleccionado. Al igual que NetEm, lo hace desde la terminal con comandos tc, pero simplificando la orden, y además también trabaja directamente sobre el *kernel* del SO.

La sintaxis es muy sencilla; por ejemplo, para limitar el ancho de banda de la interfaz enp3s0 a 3 Mbps de bajada y 1 Mbps de subida se emplea la instrucción:

```
$sudo wondershaper -a enp3s0 -d 3000 -u 1000
```

Para modificar los límites hay que eliminar antes cualquier disciplina de cola ya creada.

```
$sudo wondershaper -a enp3s0 -c
```

Se puede modificar también una sola de las tasas.

```
$sudo wondershaper -a enp3s0 -d 2000
```

Una vez hayamos terminado las pruebas, basta con eliminar la disciplina de cola para restaurar el ancho de banda del ordenador. Para instalar esta aplicación se han seguido los pasos de la guía en línea [7].

Capítulo 3

Plataforma de pruebas

Contenido

3.1 Descripción del escenario de pruebas	28
3.1.1 Sin Restream	28
3.1.2 Con Restream	29
3.1.3 Método directo de medida de latencia	30
3.1.4 Elección del PC del creador de contenido	30
3.1.5 Información del archivo de vídeo	31
3.1.6 Ajuste del modo baja latencia	31
3.2 Descripción de casos de pruebas	32

Sinopsis

En este capítulo, se ilustrará el entorno de trabajo y las aplicaciones que emplea cada lado del *streaming*, se explicará el método de medida de latencia empleado en los casos de prueba y se describirá brevemente los propios casos de pruebas para las diferentes plataformas de *streaming* de manera directa, y a través de Restream que desarrollaremos en detalle en el capítulo 4.

3.1. Descripción del escenario de pruebas

En primer lugar, se describirán los elementos básicos que componen la plataforma de pruebas que se empleará en este TFG. Dicha plataforma consta de dos tipos de escenarios de interés: emisión del flujo multimedia directamente a la plataforma de *streaming* considerada, y emisión del flujo a través de la plataforma Restream. Por simplicidad, nos referiremos en adelante a ambos escenarios como sin y con Restream, respectivamente.

3.1.1. Sin Restream

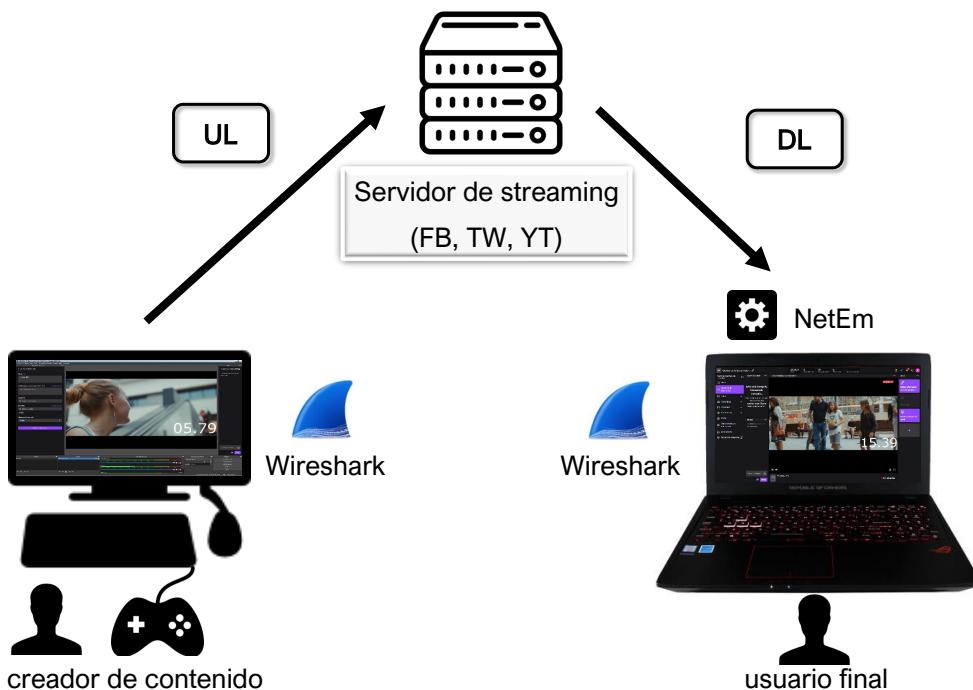


Figura 3.1: Escenario de pruebas sin Restream

La figura 3.1 muestra el escenario de pruebas sin Restream. El creador de contenido utiliza un ordenador de sobremesa con una tarjeta adaptadora de red Gigabit Ethernet, más que suficiente para nuestras pruebas. También tiene instalado Wireshark para analizar el tráfico en el sentido ascendente.

Una vez el flujo de vídeo en TCP o UDP, dependiendo de la plataforma, llegue al servidor, este lo transcodifica hacia el usuario final al proceso en ejecución de la plataforma de *streaming*. Tanto el creador de contenido como el usuario usarán un cable UTP Cat. 5e para garantizar unas velocidades de subida y de bajada estables. Sobre dicha conexión pondremos trabas de manera controlada para realizar las pruebas: intervendremos el plano de datos en el *kernel* del sistema operativo del ordenador portátil con NetEm para provocar situaciones desfavorables e interesantes y estudiar cómo afectan a la latencia y calidad de experiencia.

3.1.2. Con Restream

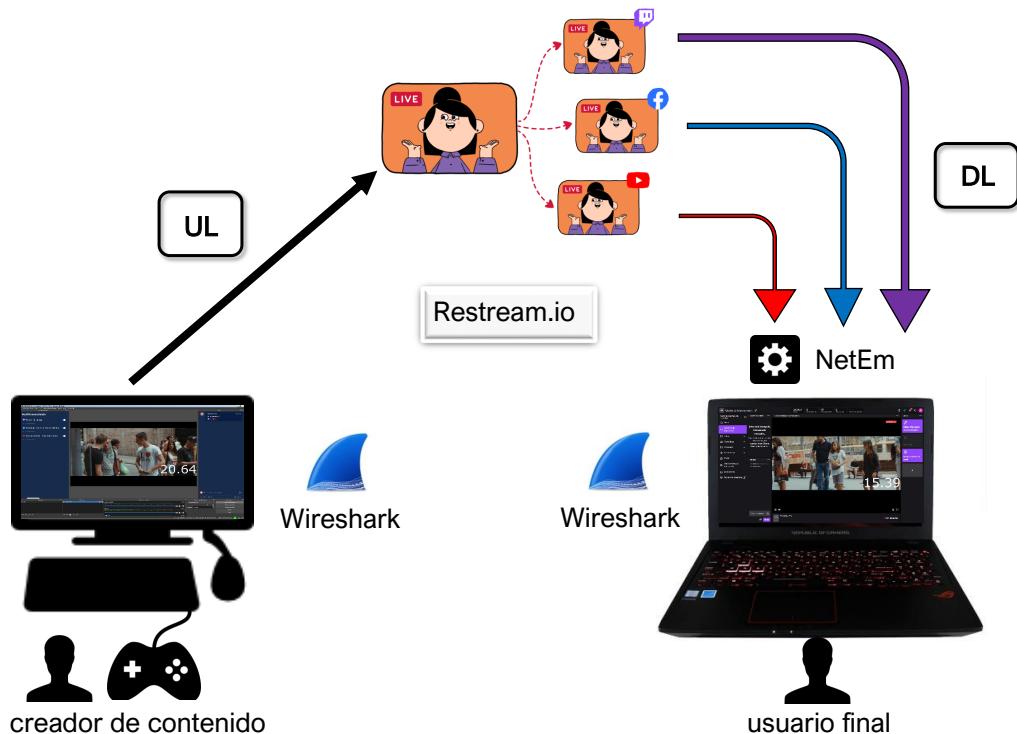


Figura 3.2: Escenario de pruebas con Restream

La figura 3.2 muestra el escenario de pruebas con Restream, es decir, multiplicando el flujo recibido desde la máquina del creador de contenido. En este caso, adelantamos que la plataforma Restream introducirá siempre un retardo al flujo que habrá que medir porque variará en cada caso de prueba. Con ayuda de Wireshark, se verá en qué grado afecta a la calidad del vídeo percibido por el

usuario final.

3.1.3. Método directo de medida de latencia

Existen dos formas de medir la latencia de un vídeo: directa y estimada. Utilizaremos la forma directa porque aborda directamente la evaluación de la calidad percibida por el usuario, objeto de este trabajo. Además, está aconsejada por Amazon en el artículo [1] y es ampliamente utilizada.

Para medir la latencia de forma directa es necesario realizar 10 fotografías de ambas pantallas donde se vea la marca de tiempo, como muestra la figura 3.3. La marca de tiempo es, en realidad, un contador insertado en una esquina del vídeo mediante un editor de vídeo. Después, realizamos la diferencia entre la marca de tiempo del PC productor y consumidor y promediamos esos valores. El resultado es la latencia media.

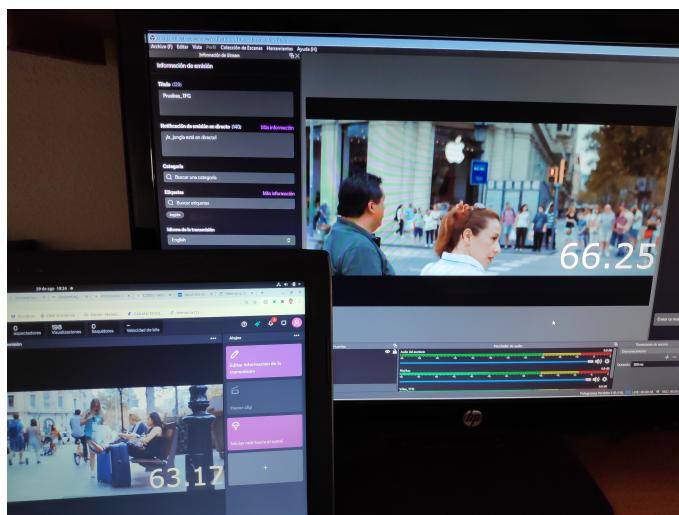


Figura 3.3: Marcas de tiempo en PC productor y consumidor

3.1.4. Elección del PC del creador de contenido

Tanto el PC de sobremesa como el portátil tienen ambos una tarjeta GBE que permite velocidades de hasta 1 Gbps y, hasta aquí, ambos valen sobradamente para transmitir. Sin embargo, existe una causa que provoca latencia y tiene que ver con los componentes físicos de la máquina cliente: si la potencia y la memoria RAM es insuficiente incluso con la conexión más rápida, la máquina podría

experimentar dificultades para procesar el flujo continuo que se está transmitiendo, agravado si varias aplicaciones están corriendo al mismo tiempo. Como el ordenador portátil tiene 15.5 GiB de memoria RAM y un procesador Intel Core i7-7700HQ @2.8 GHz × 8, más potente y con mayor memoria principal que el de sobremesa, emplearemos el portátil para consumir el *streaming* y el de sobremesa para producirlo.

3.1.5. Información del archivo de vídeo

Se trata del vídeo *Breathing Barcelona* de Davide Quatela (2014) descargado de un banco de vídeos sin derechos de autor. Sus características son:

- Duración: 2 min 23 s
- Velocidad de bits total: 3800 kbps
- Tamaño de fotograma: 800 × 1920
- Tamaño: 57.1 MB
- Velocidad de fotograma: 23 fotogramas/segundo
- Frecuencia de muestreo de audio: 48 kHz
- Velocidad de bits de audio: 256 kbps

3.1.6. Ajuste del modo baja latencia

Todos los casos de pruebas requieren disponer de un canal de Twitch, YouTube (se creará a partir de la cuenta de Google), Restream.io y una cuenta de Facebook. Por otro lado, cada una de ellas tendrá activada la pestaña que habilite el modo de más baja latencia posible desde su panel de control.

En Twitch. Hay que dirigirse a *Estudio de vídeo*. Una vez dentro, a la izquierda, en el *Panel de control de creador*, desplegar *Preferencias* y pinchar *Canal*. Bajo el primer encabezado, *Preferencias y clave de transmisión*, buscar la opción *Modo de baja latencia* y seleccionar la opción *Baja latencia: ideal para interactuar casi en tiempo real con los espectadores* (figura 3.4).

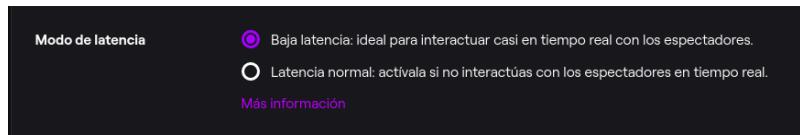


Figura 3.4: Activación del modo baja latencia en Twitch

En YouTube. Una vez hayamos iniciado sesión, desde la vista general de la plataforma, pinchamos en el ícono junto a nuestra foto de perfil para crear un vídeo y seleccionamos *Emitir en directo* para ir al estudio de control. Desde esta vista ya podremos ver bajo el encabezado *Latencia de emisiones en directo* la opción *Latencia extremadamente baja* (figura 3.5) y la marcamos.

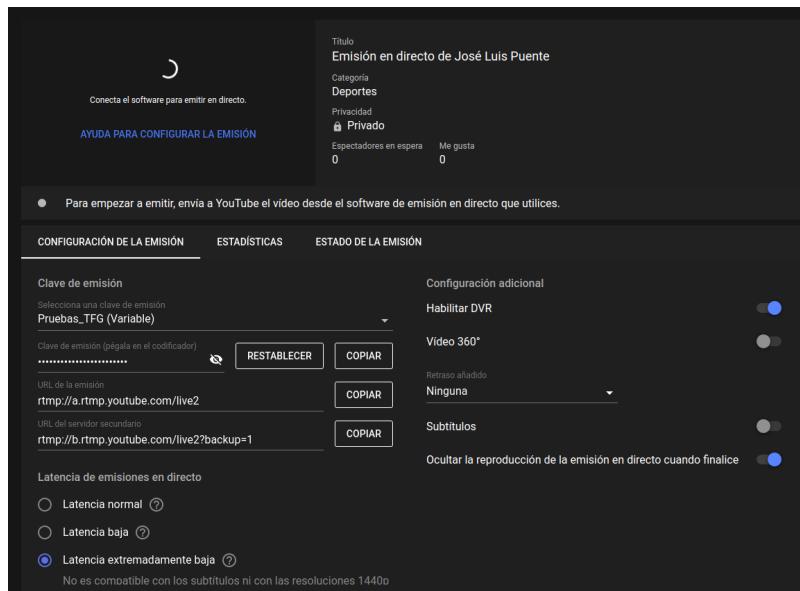


Figura 3.5: Activación del modo baja latencia en YouTube

En Facebook y Restream. No se permite.

3.2. Descripción de casos de pruebas

Los casos de prueba que se desarrollarán en el capítulo 4 son:

1. Análisis de errores TCP (retransmisiones, DUP ACK, segmentos fuera de orden) en Twitch y Facebook Live sin y con Restream mediante la herramienta IO/Graph. Compararemos cada una de las gráficas con porcentajes

relativos a los errores TCP y el total de paquetes intercambiados durante la transmisión.

2. Medida de latencia media sin y con Restream variando retardo y tasa de pérdidas de paquetes.
3. Efecto del *jitter* en la latencia en situaciones extremas de retardo. Estudaremos el efecto sobre la latencia.
4. Cálculo de la tasa de descarga límite. Buscaremos un valor de tasa de descarga a partir del cual la calidad de experiencia del *streaming* siempre sea excelente, es decir, con latencia mínima.

Capítulo 4

Evaluación de prestaciones

Contenido

4.1 Análisis de errores TCP	36
4.1.1 Enlace ascendente	36
4.1.2 Sin restricciones	37
4.1.3 Con restricciones	44
4.2 Medida de latencia media	51
4.2.1 Sin restricciones	51
4.2.2 Retardo incremental de 100 ms	51
4.2.3 Adición de pérdidas con granularidad 5 %	55
4.3 Efecto del <i>jitter</i> en casos extremos	58
4.4 Cálculo de la tasa de descarga límite	62
4.4.1 Procedimiento con Twitch	62
4.4.2 Resultados	66
4.5 Análisis y discusión de resultados	67

4.1. Análisis de errores TCP

En esta sección se analizará el impacto que el retardo y la tasa de pérdidas tienen sobre la latencia sin y con Restream.

4.1.1. Enlace ascendente

Cuando pulsamos en *Iniciar transmisión* la máquina del creador de contenido establece una conexión TCP con el servidor de ingesta seleccionado, sea Facebook, Twitch, YouTube o Restream. Al inicio, se establece la conexión TCP y después tiene lugar la conexión RTMP. Sea cual sea la plataforma de *streaming*, TCP transporta siempre los fragmentos de vídeo al servidor de ingesta en el sentido ascendente de la transmisión. Wireshark muestra el comienzo de una ráfaga de paquetes de audio o vídeo marcando el primer paquete de esta con el protocolo de nivel de aplicación RTMP, como muestra la figura 4.1.

No.	Time	Source	Destination	Protocol	Length	Info
44865	83.915703	192.168.1.137	52.223.197.242	RTMP	506	Audio Data
44866	83.925836	192.168.1.137	52.223.197.242	TCP	1514	49366 → 1935 [ACK] Seq=37560609 Ack=4309 Win=63005 Len=1460
44867	83.925836	192.168.1.137	52.223.197.242	TCP	1514	49366 → 1935 [ACK] Seq=37562069 Ack=4309 Win=63005 Len=1460
44868	83.925836	192.168.1.137	52.223.197.242	RTMP	380	Video Data
44869	83.929233	52.223.197.242	192.168.1.137	TCP	60	1935 → 49366 [ACK] Seq=4309 Ack=37560609 Win=65535 Len=0
44870	83.939639	52.223.197.242	192.168.1.137	TCP	60	1935 → 49366 [ACK] Seq=4309 Ack=37563855 Win=65535 Len=0
44871	83.948375	192.168.1.137	52.223.197.242	RTMP	515	Audio Data
44872	83.949046	192.168.1.137	52.223.197.242	TCP	1514	49366 → 1935 [ACK] Seq=37564316 Ack=4309 Win=63005 Len=1460
44873	83.949046	192.168.1.137	52.223.197.242	TCP	1514	49366 → 1935 [ACK] Seq=37565776 Ack=4309 Win=63005 Len=1460
44874	83.949046	192.168.1.137	52.223.197.242	TCP	1238	49366 → 1935 [PSH, ACK] Seq=37567236 Ack=4309 Win=63005 Len=1184
44875	83.949119	192.168.1.137	52.223.197.242	TCP	1514	49366 → 1935 [ACK] Seq=37568420 Ack=4309 Win=63005 Len=1460
44876	83.949119	192.168.1.137	52.223.197.242	TCP	1514	49366 → 1935 [ACK] Seq=37569880 Ack=4309 Win=63005 Len=1460
44877	83.949119	192.168.1.137	52.223.197.242	TCP	1231	49366 → 1935 [PSH, ACK] Seq=37571340 Ack=4309 Win=63005 Len=1177
44878	83.949163	192.168.1.137	52.223.197.242	TCP	1514	49366 → 1935 [ACK] Seq=37572517 Ack=4309 Win=63005 Len=1460
44879	83.949163	192.168.1.137	52.223.197.242	TCP	1514	49366 → 1935 [ACK] Seq=37573977 Ack=4309 Win=63005 Len=1460
44880	83.949163	192.168.1.137	52.223.197.242	TCP	1231	49366 → 1935 [PSH, ACK] Seq=37575437 Ack=4309 Win=63005 Len=1177
44881	83.949208	192.168.1.137	52.223.197.242	TCP	1514	49366 → 1935 [ACK] Seq=37576614 Ack=4309 Win=63005 Len=1460
44882	83.949208	192.168.1.137	52.223.197.242	TCP	1514	49366 → 1935 [ACK] Seq=37578074 Ack=4309 Win=63005 Len=1460
44883	83.949208	192.168.1.137	52.223.197.242	TCP	1231	49366 → 1935 [PSH, ACK] Seq=37579534 Ack=4309 Win=63005 Len=1177
44884	83.949252	192.168.1.137	52.223.197.242	TCP	1514	49366 → 1935 [ACK] Seq=37580711 Ack=4309 Win=63005 Len=1460
44885	83.949252	192.168.1.137	52.223.197.242	TCP	1514	49366 → 1935 [ACK] Seq=37582171 Ack=4309 Win=63005 Len=1460
44886	83.949252	192.168.1.137	52.223.197.242	TCP	1231	49366 → 1935 [PSH, ACK] Seq=37583631 Ack=4309 Win=63005 Len=1177
44887	83.949294	192.168.1.137	52.223.197.242	TCP	1514	49366 → 1935 [ACK] Seq=37584808 Ack=4309 Win=63005 Len=1460
44888	83.949294	192.168.1.137	52.223.197.242	TCP	1514	49366 → 1935 [ACK] Seq=37584868 Ack=4309 Win=63005 Len=1460
44889	83.949294	192.168.1.137	52.223.197.242	TCP	1231	49366 → 1935 [PSH, ACK] Seq=37587278 Ack=4309 Win=63005 Len=1177
44890	83.949334	192.168.1.137	52.223.197.242	TCP	1514	49366 → 1935 [ACK] Seq=37588965 Ack=4309 Win=63005 Len=1460
44891	83.949334	192.168.1.137	52.223.197.242	RTMP	456	Video Data
44892	83.961574	52.223.197.242	192.168.1.137	TCP	60	1935 → 49366 [ACK] Seq=4309 Ack=37564316 Win=65535 Len=0
44893	83.962575	52.223.197.242	192.168.1.137	TCP	60	1935 → 49366 [ACK] Seq=4309 Ack=37567236 Win=65535 Len=0
44894	83.963627	52.223.197.242	192.168.1.137	TCP	60	1935 → 49366 [ACK] Seq=4309 Ack=37569880 Win=65535 Len=0
44895	83.963659	52.223.197.242	192.168.1.137	TCP	60	1935 → 49366 [ACK] Seq=4309 Ack=37572517 Win=65535 Len=0
44896	83.963679	52.223.197.242	192.168.1.137	TCP	60	1935 → 49366 [ACK] Seq=4309 Ack=37578074 Win=65535 Len=0
44897	83.964410	52.223.197.242	192.168.1.137	TCP	60	1935 → 49366 [ACK] Seq=4309 Ack=37580711 Win=65535 Len=0
44898	83.965235	52.223.197.242	192.168.1.137	TCP	60	1935 → 49366 [ACK] Seq=4309 Ack=37584808 Win=65535 Len=0
44899	83.965266	52.223.197.242	192.168.1.137	TCP	60	1935 → 49366 [ACK] Seq=4309 Ack=37590767 Win=65535 Len=0
44900	83.972879	192.168.1.137	52.223.197.242	RTMP	533	Audio Data
44901	83.979734	192.168.1.137	52.223.197.242	TCP	1514	49366 → 1935 [ACK] Seq=37591246 Ack=4309 Win=63005 Len=1460
44902	83.979734	192.168.1.137	52.223.197.242	TCP	1514	49366 → 1935 [ACK] Seq=37592706 Ack=4309 Win=63005 Len=1460
44903	83.979734	192.168.1.137	52.223.197.242	TCP	1238	49366 → 1935 [PSH, ACK] Seq=37594166 Ack=4309 Win=63005 Len=1184
44904	83.979823	192.168.1.137	52.223.197.242	TCP	1514	49366 → 1935 [ACK] Seq=37595350 Ack=4309 Win=63005 Len=1460
44905	83.979823	192.168.1.137	52.223.197.242	RTMP	639	Video Data
44906	83.985617	52.223.197.242	192.168.1.137	TCP	60	1935 → 49366 [ACK] Seq=4309 Ack=37591246 Win=65535 Len=0
44907	83.992639	52.223.197.242	192.168.1.137	TCP	60	1935 → 49366 [ACK] Seq=4309 Ack=37594166 Win=65535 Len=0

Figura 4.1: Ráfagas RTMP en sentido ascendente

4.1.2. Sin restricciones

En esta sección, se evaluarán los errores TCP sin restricciones de retardo ni pérdida de paquetes aplicada. Ambas restricciones se incorporarán más adelante a través de la herramienta NetEm.

Twitch

La figura 4.2 es una captura IO/Graph de Wireshark que ilustra los errores TCP en número de paquetes por segundo. Como se trata del enlace aún sin condicionar, esta figura muestra la cantidad base de errores TCP sobre la que se sumarán más cuando condicionemos el enlace.

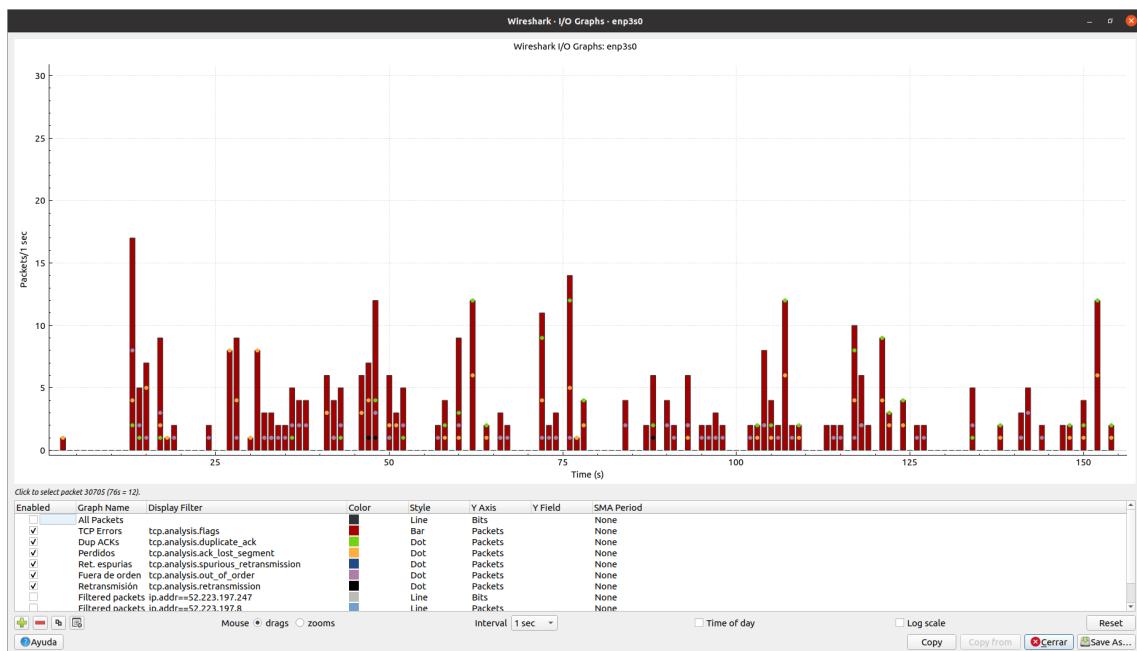


Figura 4.2: Errores TCP en la máquina del usuario final en Twitch

La figura 4.3 muestra la conversación entre el *streamer* y el usuario final. Hemos filtrado todas las conversaciones por número de paquetes en ambas direcciones para situar el *stream* en los primeros lugares. Sabemos que es la primera que aparece con dirección remota 185.42.206.41 por esos 76 millones de bytes intercambiados y buscando la procedencia de esa dirección IP en www.who.is. El vídeo tiene un tamaño de 57,1 MB, pero hay que tener en cuenta el tráfico en ambos sentidos más el destinado al control de flujo. Por otro lado, podemos ver que el flujo se envía por el puerto TCP 443 para proporcionar seguridad TLSv1.3.

El flujo ha generado un tráfico en la red de 59 888 paquetes.

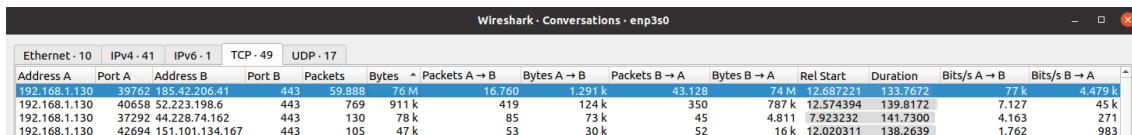


Figura 4.3: Conversación entre *streamer* y usuario final en Twitch

La figura 4.4 representa el porcentaje de errores TCP de cada tipo con respecto al total de errores TCP desde el *buffering* inicial hasta el final del vídeo. En ella se observa que predominan los paquetes perdidos y los DUP ACK sobre los paquetes fuera de orden y las retransmisiones.

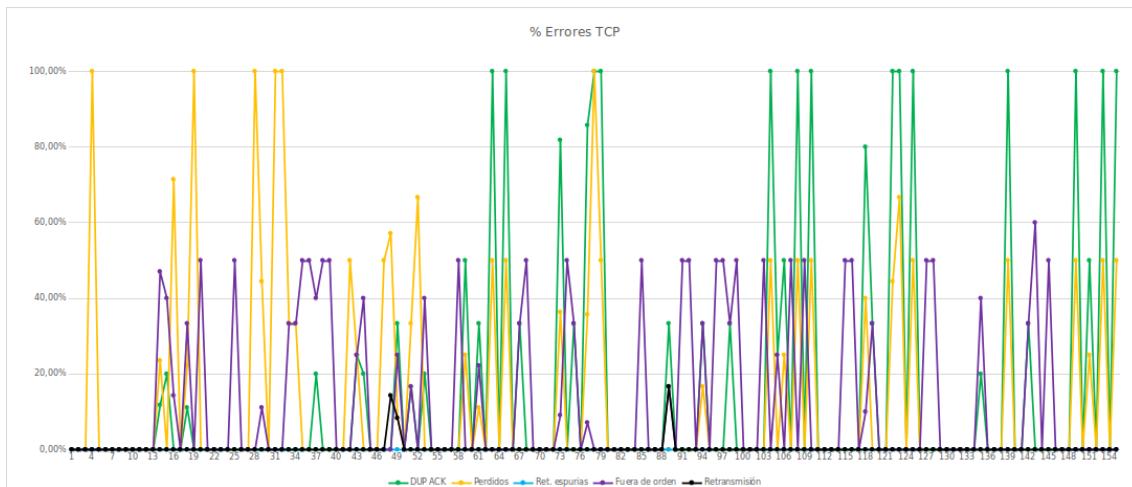
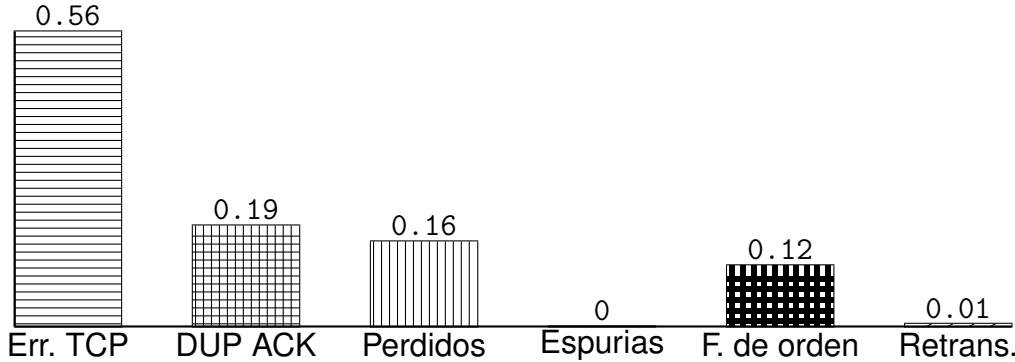


Figura 4.4: Porcentaje de errores TCP en Twitch

En la figura 4.5 se muestra el porcentaje de errores TCP durante la transmisión respecto a los 59 888 paquetes que se intercambiaron. De ellos, tan solo el 0.56 % son notificaciones de errores TCP, de los cuales el 0.19 % son DUP ACK, 0.16 % perdidos, 0 retransmisiones espurias, 0.12 % fuera de orden y 0.01 % retransmisiones. Los paquetes fuera de orden no son estrictamente errores TCP porque el TCP destino los guarda a la espera de llenar los huecos; pero la recepción de segmentos fuera de orden sí que provoca retransmisiones tras excluir el RTO (*retransmission timeout*).

Figura 4.5: Errores TCP (%) respecto al total de paquetes en Twitch



YouTube

La figura 4.6 muestra todas las conversaciones entre servidor de ingesta y la máquina del usuario final. YouTube utiliza el protocolo QUIC que multiplexa vídeo a través de varias conexiones lógicas de modo que el vídeo procede de varios destinos. Todas las conversaciones con número de puerto 443 pertenecen a YouTube así que todas ellas han contribuido a transportar el vídeo hasta el usuario final. Al usar QUIC, YouTube no genera tráfico TCP y por ello en los próximos subcasos de esta sección no se volverá a mencionar nada más al respecto de esta plataforma.

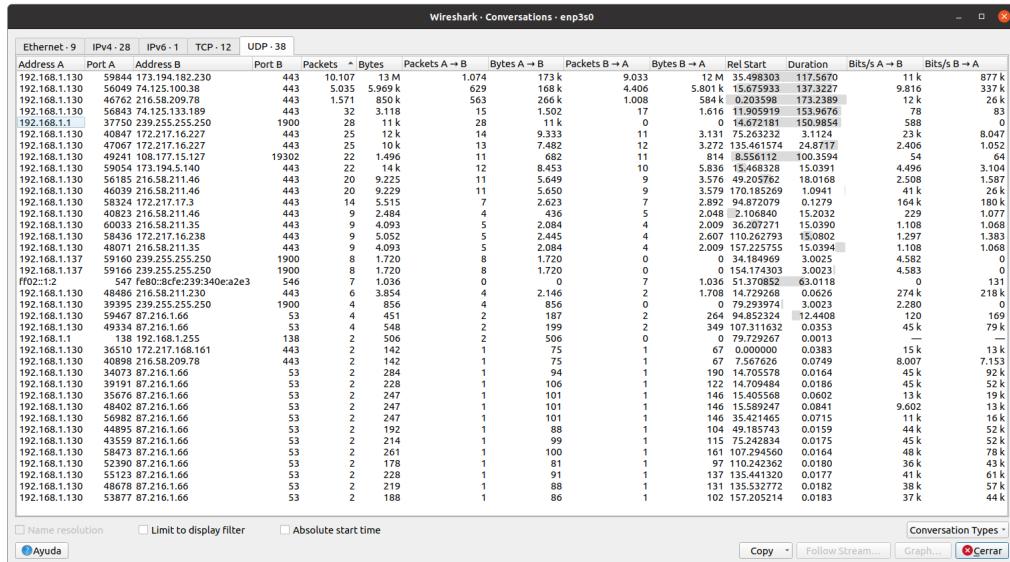


Figura 4.6: Conversación entre streamer y usuario final en YouTube

Facebook Live

La figura 4.7 muestra los errores TCP en la transmisión de Facebook Live.

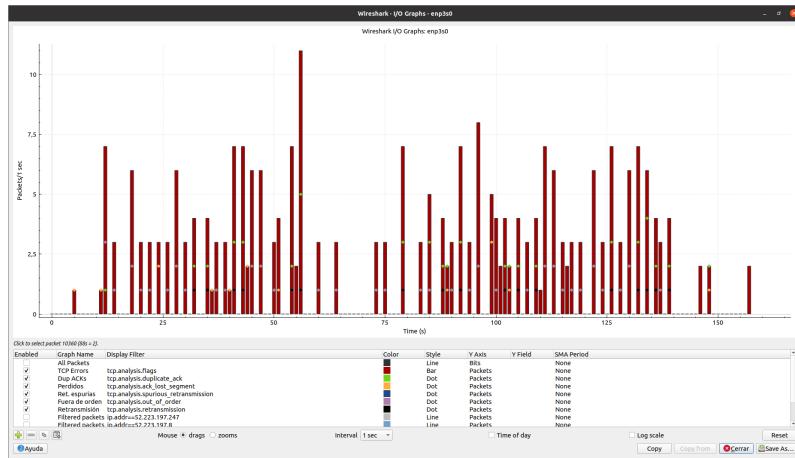


Figura 4.7: Errores TCP en el *stream* del usuario final en Facebook Live

La figura 4.8 muestra que los paquetes fuera de orden son el 30 % de los errores TCP casi durante toda la retransmisión, lo que da lugar a la notificación de paquetes perdidos y en el peor caso de ACK duplicados.

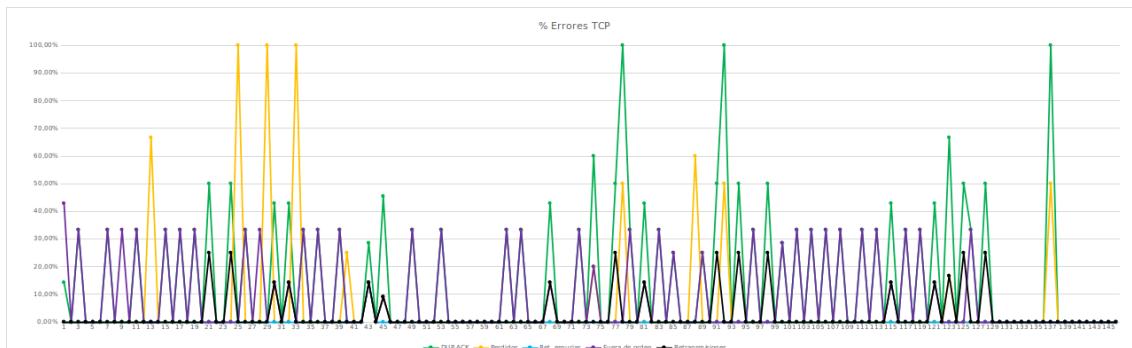
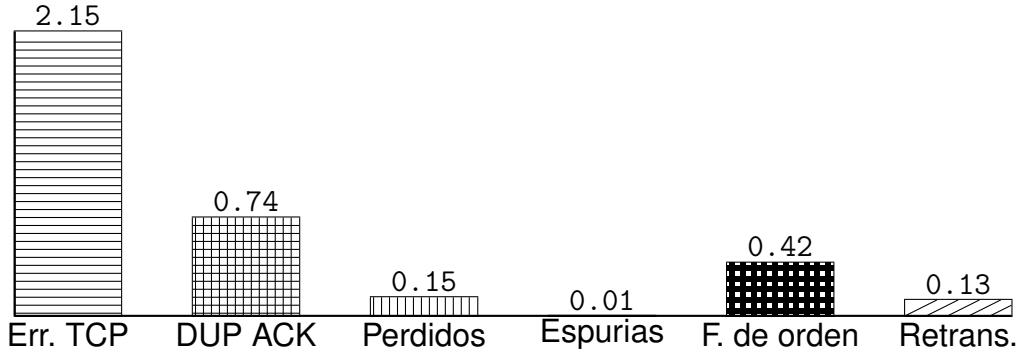


Figura 4.8: Porcentaje de errores TCP en Facebook Live

La tabla 4.9 muestra que de un total de 12 660 paquetes, solo el 2.15 % son errores TCP, el 0.74 % ACK duplicados y en menor proporción el resto.

Figura 4.9: Errores TCP (%) respecto al total de paquetes en Facebook Live



Restream a Twitch

La figura 4.10 muestra los errores TCP en la retransmisión a Twitch, donde no se aprecian picos de errores que comprometan la fluidez de la reproducción.

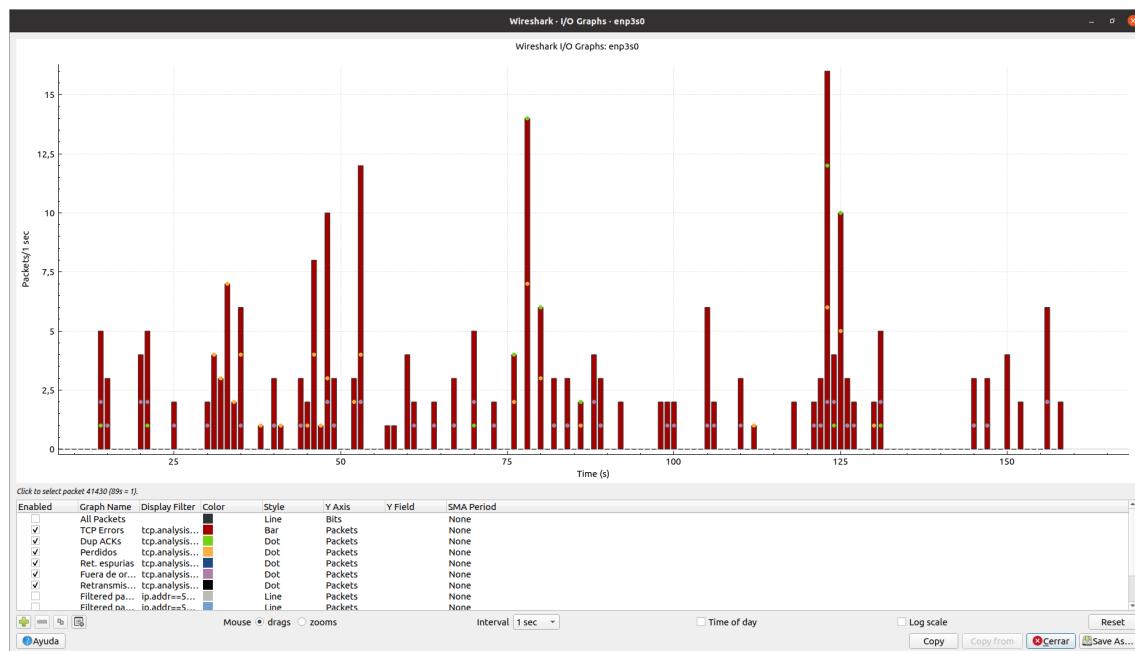


Figura 4.10: Errores TCP en el *stream* del usuario final en Twitch con Restream

En la figura 4.11 vemos que predominan los segmentos perdidos en algunos intervalos de la reproducción o los ACK duplicados. Los segmentos fuera de orden suponen aproximadamente un 50 % de las notificaciones TCP.

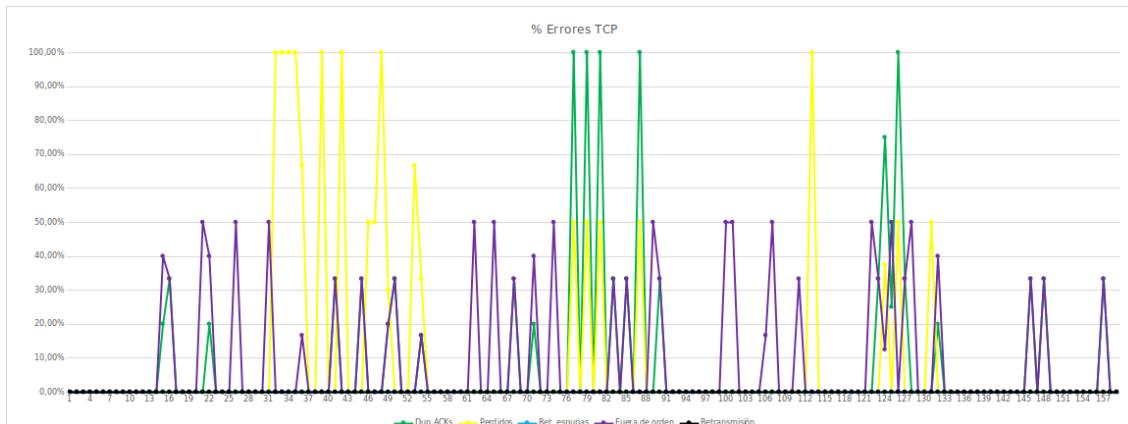


Figura 4.11: Porcentaje de errores TCP en Twitch con Restream

Como se observa en la figura 4.12, de un total de 68 526 paquetes, solo el 0.35 % fueron errores TCP frente al 0.56 % del caso de transmisión directa. Restream disminuye el porcentaje de errores TCP.

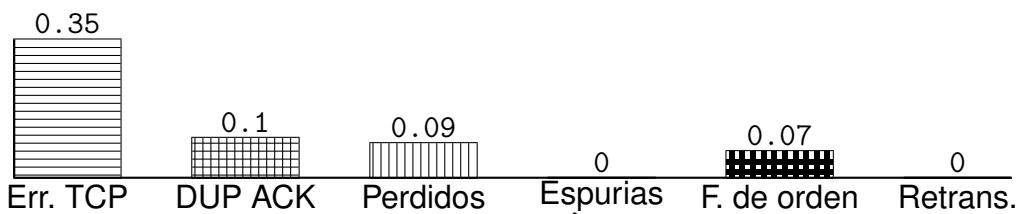


Figura 4.12: Errores TCP (%) respecto al total de paquetes en Twitch con Restream

Restream a Facebook Live

La figura 4.13 es muy interesante porque muestra un pico de errores aislado a los 28 segundos de más de 30 paquetes/s que no causó una pausa. Esto es debido a la virtud del *buffer*. Prueba de ello es la normalidad de las sucesivas columnas de errores. Esta gráfica pone de manifiesto dos puntos: el primero, la naturaleza *best-effort* de Internet, en la que aun con la mejor conexión pueden darse casos de eventos de congestión inesperados e impredecibles, y la segunda, la importancia de dimensionar correctamente el *buffer* de recepción por Facebook Live.

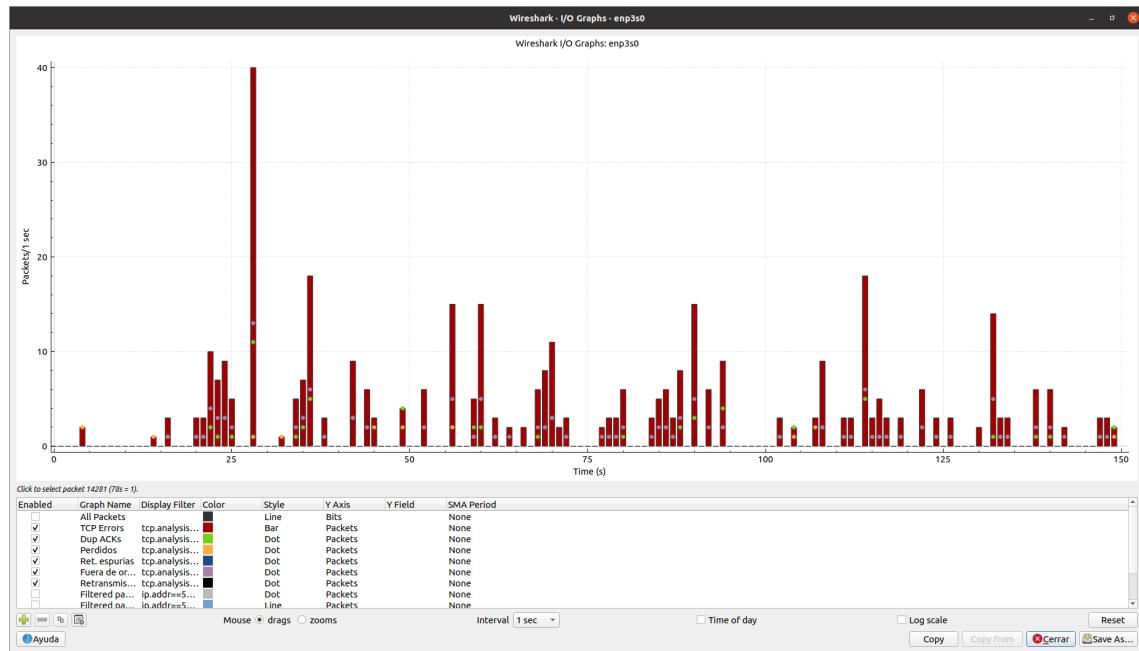
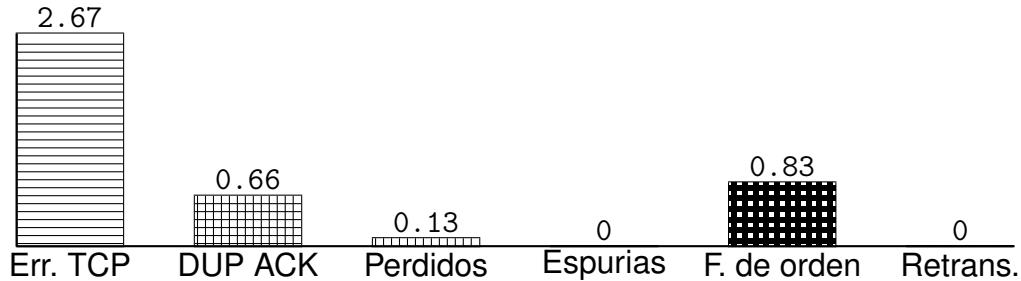


Figura 4.13: Errores TCP en Facebook Live con Restream

De un total de 14 890 paquetes, el 2.67 % son errores TCP, el 0.83 % son segmentos fuera de orden y el 0.66 % son ACK duplicados, como muestra la figura 4.14.

Figura 4.14: Errores TCP (%) respecto al total de paquetes en Facebook Live con Restream



En el caso sin restricciones, la calidad percibida por el usuario siempre es excelente, siendo los errores TCP originados insignificantes en el caso directo y con Restream.

4.1.3. Con restricciones

A continuación, se realizará el mismo análisis añadiendo un retardo de 500 ms y un 40 % de pérdidas de paquetes con 25 % correlación¹.

Twitch

La figura 4.21 muestra picos de errores de más de 30 paquetes/s. Se trata de pausas de *rebuffering* para llenar el *buffer* de pedazos de vídeo y poder continuar con la reproducción. Cada pausa acumula una latencia de unos 30 segundos al final del vídeo.

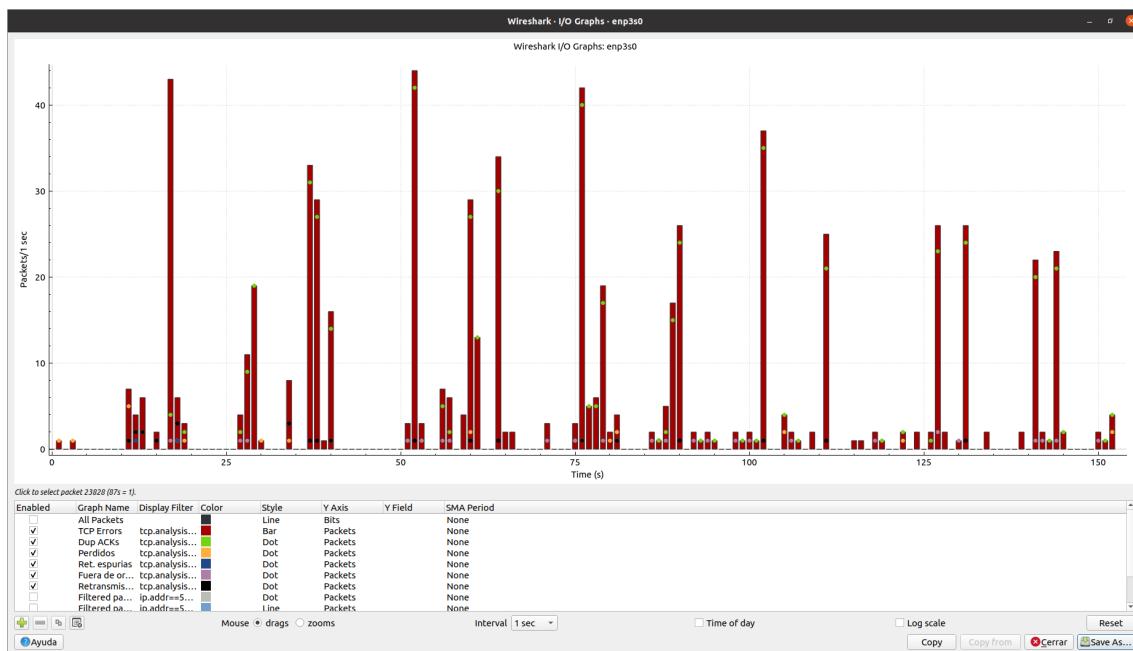


Figura 4.15: Errores TCP con restricciones en Twitch

En la figura 4.16 vemos cómo se dispara el número de ACK duplicados debido al incremento del retardo aplicado a los paquetes.

¹ Es preciso recordar que debido a que NetEm actúa únicamente sobre los paquetes de salida, estas pérdidas se corresponden a los mensajes TCP desde el equipo de usuario hacia el servidor de streaming correspondiente.

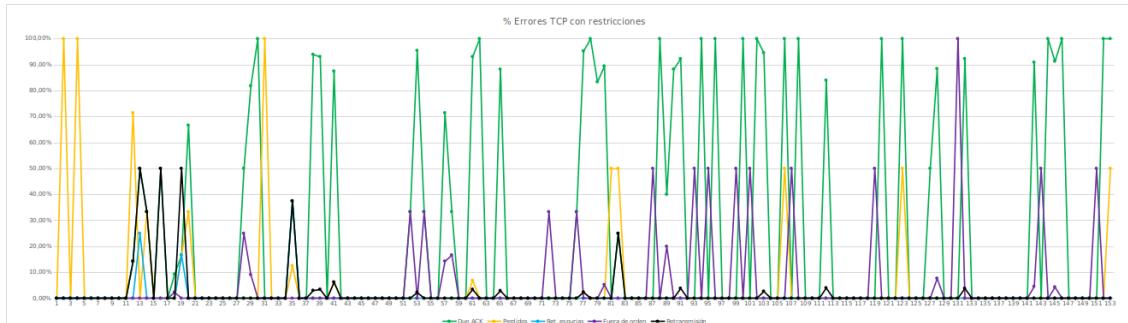
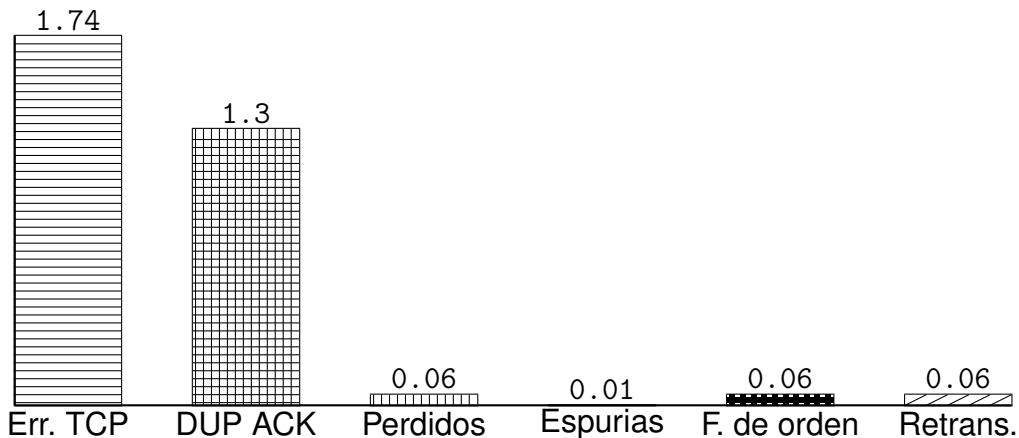


Figura 4.16: Porcentaje de errores TCP con restricciones en Twitch

Respecto a un total de paquetes de 39 586, el 1.74 % son errores TCP y el 1.3 % son de ACK duplicados. Cuando hay retardo los paquetes tardan más en llegar y propician un escenario de disparos de ACK duplicados.

Figura 4.17: Errores TCP (%) respecto al total de paquetes con restricciones en Twitch



Facebook Live

La figura 4.18 se caracteriza tanto por su densidad de errores como por sus altos picos. Sin embargo, al no superar ninguno de ellos los 30 paquetes/s, no se produjo ninguna pausa que afectara a la calidad de experiencia del usuario. La gran densidad de errores tampoco propició pausas que contribuyeran a la latencia acumulada. Sin embargo sí se produjeron distorsiones que alteran la atención.

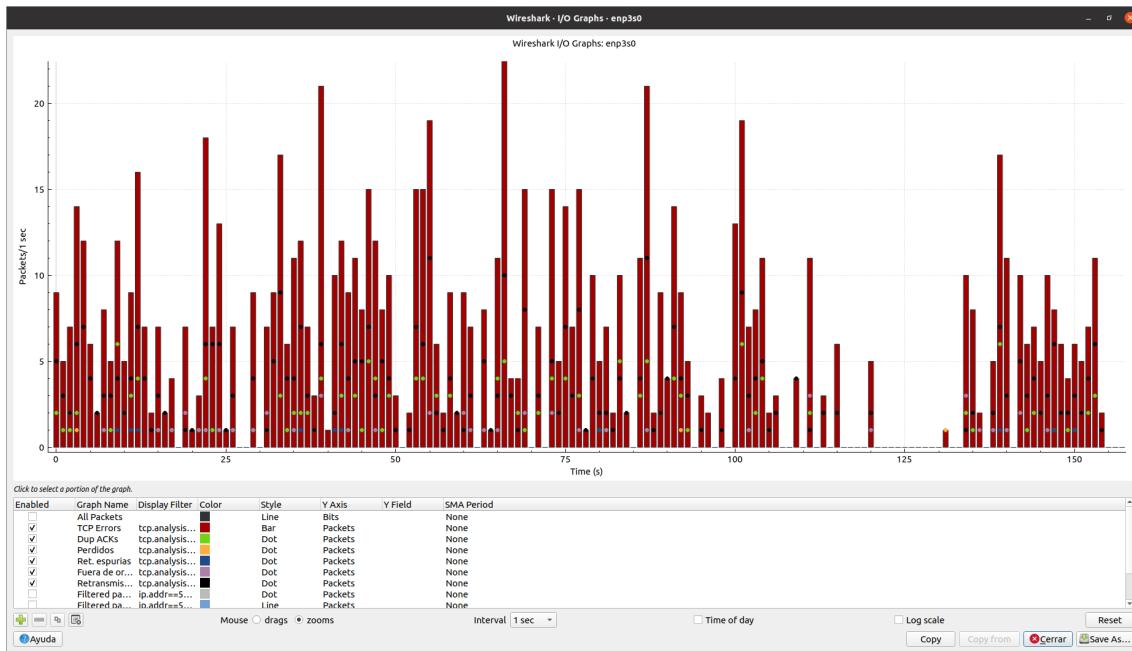


Figura 4.18: Errores TCP con restricciones en Facebook Live

Como consecuencia de esa alta densidad de errores, observamos en la figura 4.19 por primera vez un aumento significativo de las retransmisiones debido a que expira el RTO.

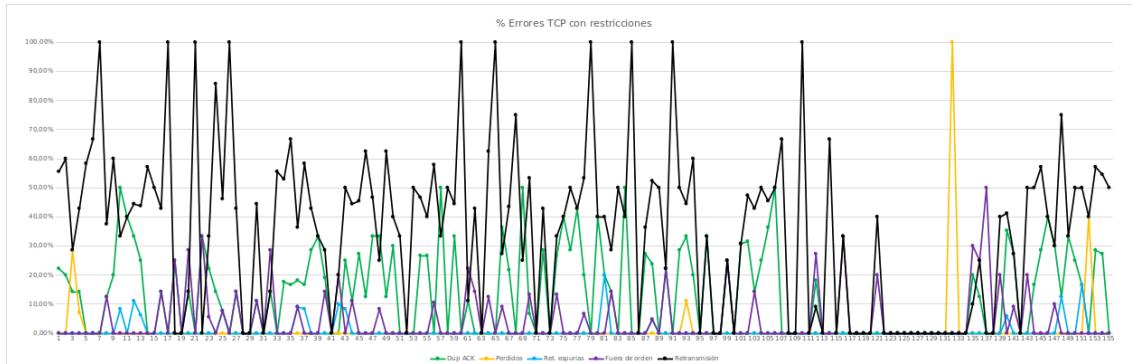
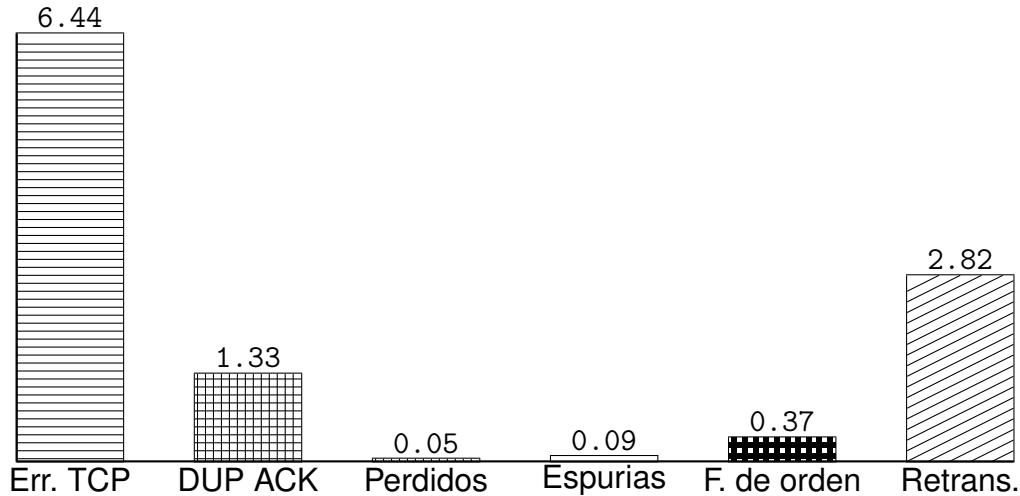


Figura 4.19: Porcentaje de errores TCP con restricciones en Facebook Live

La figura 4.20 muestra que de entre los 14 890 paquetes intercambiados en la transmisión, un 6.44 % fueron errores TCP, de los cuales un 2.82 % son retransmisiones (un aumento del 100 % frente al caso de transmisión sin restricciones).

Figura 4.20: Errores TCP (%) respecto al total de paquetes con restricciones en Facebook Live



Restream a Twitch

En la figura 4.21 vemos un aumento de la densidad de errores respecto a la transmisión directa con restricciones, pero sin efecto sobre la calidad del vídeo.

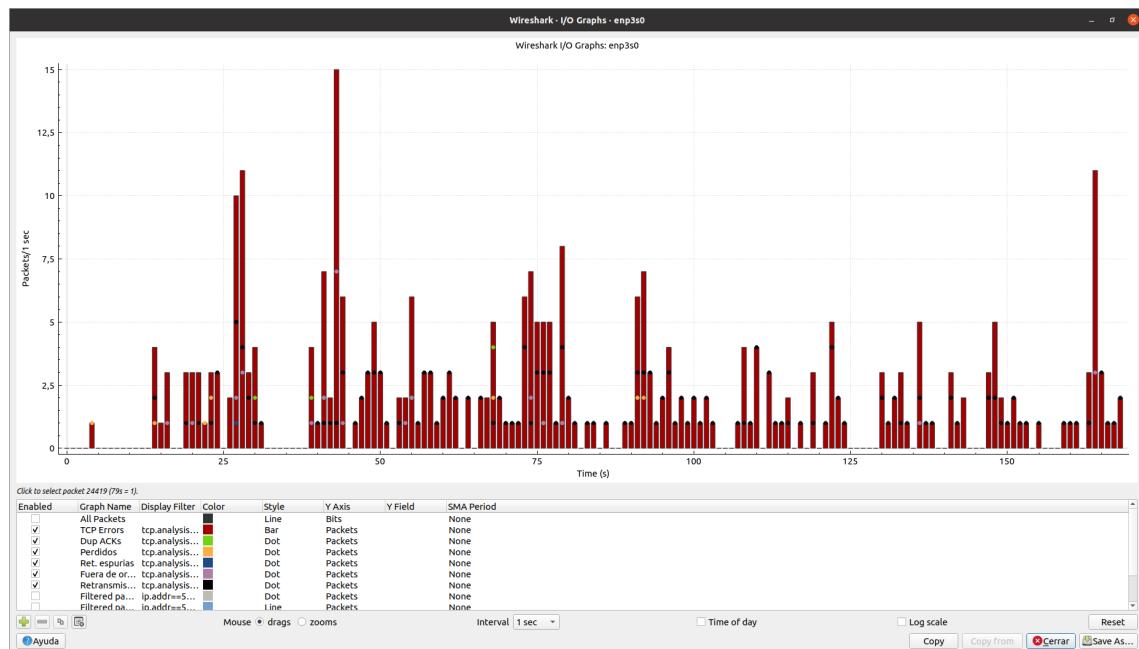


Figura 4.21: Errores TCP con restricciones en Twitch con Restream

En la figura 4.22 se observa cómo las retransmisiones ascienden hasta superar el 100 % de los errores TCP en muchos instantes de la retransmisión.

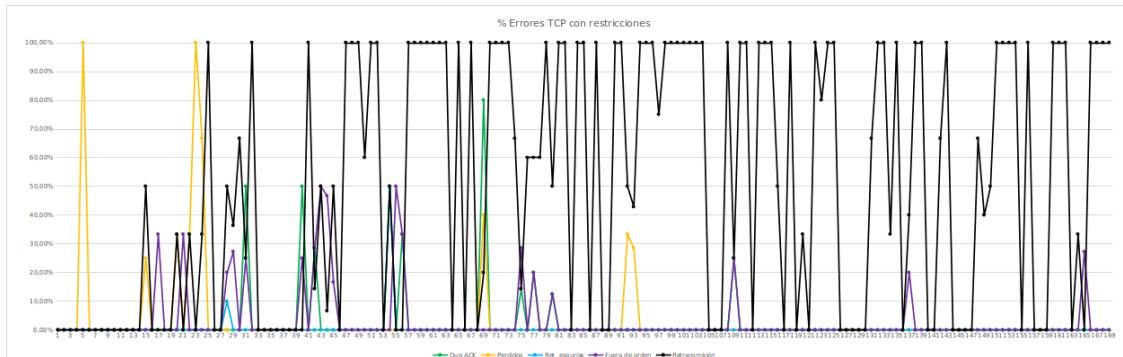
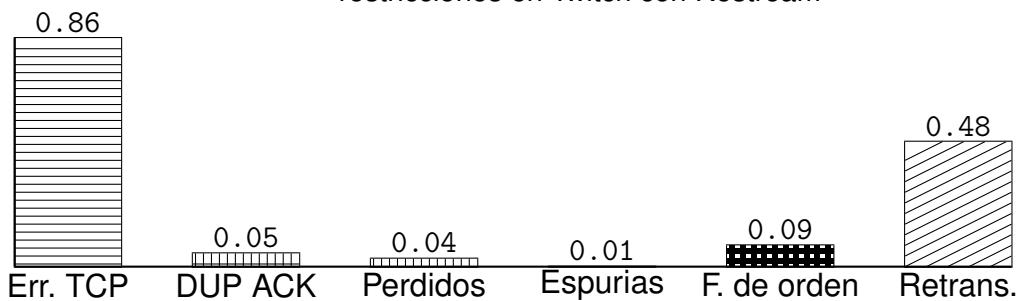


Figura 4.22: Porcentaje de errores TCP con restricciones en Twitch con Restream

La tabla 4.23 cuantifica las retransmisiones que se han producido en un 0.48 % de los 36 871 paquetes.

Figura 4.23: Errores TCP (%) respecto al total de paquetes con restricciones en Twitch con Restream



Restream a Facebook Live

La figura 4.24 muestra una densidad de errores parecida al caso de transmisión directa pero que presenta cinco picos mayores a los 30 paquetes/s que indican cinco pausas en el vídeo para llenar el *buffer* de datos.

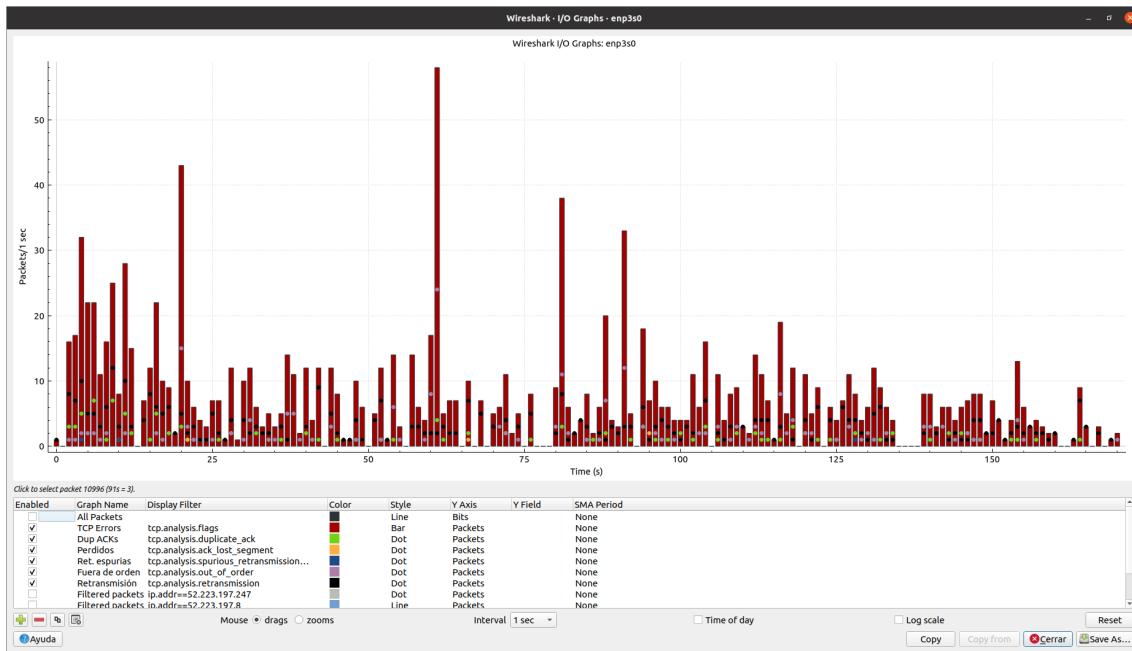


Figura 4.24: Errores TCP con restricciones en Facebook Live con Restream

La figura 4.25 es muy interesante porque pone de manifiesto varios aspectos importantes:

- Por cada pausa de *rebuffering* aumenta la latencia. Luego, si hay sucesivas pausas entonces la latencia se acumula a lo largo del tiempo.
- El tiempo extra de reproducción del vídeo es igual a la suma de incrementos de latencia provocados por las pausas.
- Las pausas disparan las retransmisiones, como se ve en la figura, al expirar el temporizador de retransmisión de la plataforma.
- El vídeo dura 143 segundos, pero el usuario pasa 28 más para lograr ver el vídeo completo. Ese tiempo extra es la latencia acumulada.

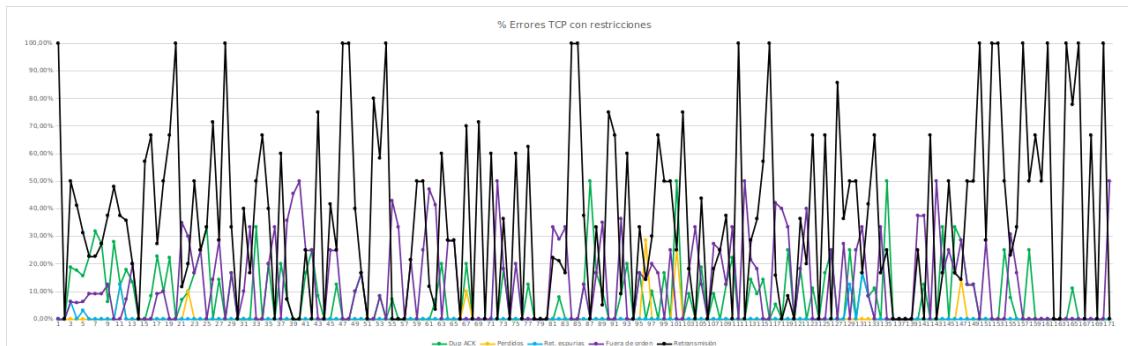
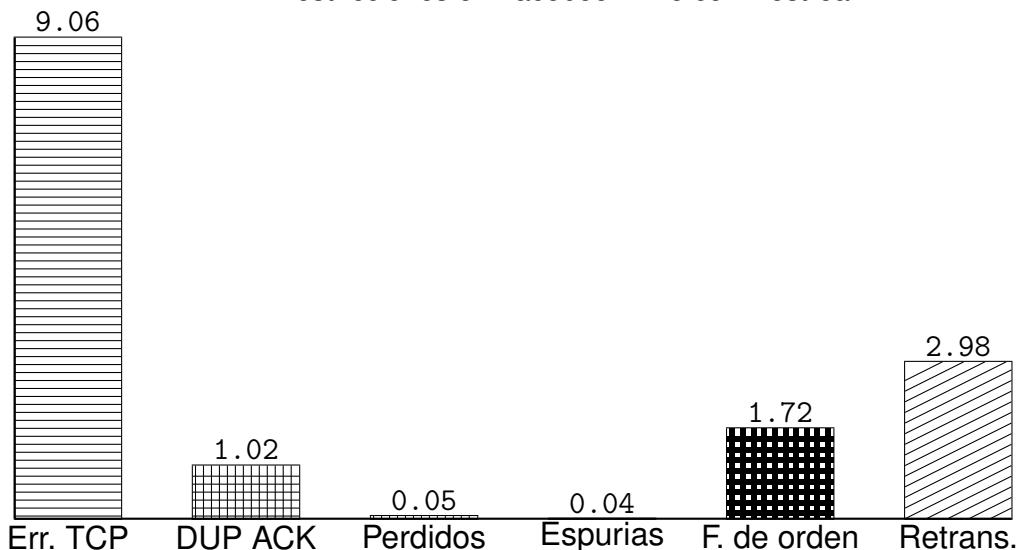


Figura 4.25: Porcentaje de errores TCP con restricciones en Facebook Live con Restream

La figura 4.26 muestra un 9.06 % de errores TCP respecto a los 14 209 paquetes durante la retransmisión.

Figura 4.26: Errores TCP (%) respecto al total de paquetes con restricciones en Facebook Live con Restream



Concluimos que para el caso con restricciones se disparan las retransmisiones y aumentan los ACK duplicados. Ello hace que existan pausas de *rebuffering* cada cierto tiempo para poder continuar con la reproducción. Además, aparecen saltos, disminuye la calidad de la imagen y aparece distorsión de audio que imposibilitan la visualización del vídeo. Con Restream esta bajada de la calidad de experiencia es todavía mayor en Facebook Live, aunque a Twitch lo penaliza más el caso directo.

4.2. Medida de latencia media

Vamos a emplear la medida directa de latencia sin restricciones, es decir, en el mejor de los escenarios posibles. Después, realizaremos la misma medida añadiendo tasas de pérdida de paquetes con granularidad fina.

Cada medida se tomará en una una nueva sesión del *streaming*, es decir, tras un refresco de la ventana del navegador. De esta manera, se obligará a la plataforma a dimensionar un nuevo *buffer* para obtener un abanico amplio de datos y evitar la suerte de enganchar una sesión con una latencia alta o baja.

Caracterizaremos la latencia como una distribución normal en función tanto del retardo, τ , como de la tasa de pérdidas, p (%). Calcularemos la latencia media y su desviación típica mediante 10 muestras en todos los subcasos de prueba.

4.2.1. Sin restricciones

La medida se ha realizado transmitiendo directamente a la plataforma y a través de Restream. Los resultados se muestran en la tabla 4.1.

Plataforma \ Vía	Directo	Restream
Plataforma		
Twitch	2.83 s	3.2 s
YouTube	2.75 s	6.42 s
Facebook	7.14 s	8.45 s

Tabla 4.1: Latencia media sin restricciones

4.2.2. Retardo incremental de 100 ms

En esta sección se añadirán retardos en incrementos de 100 ms y se medirá la latencia media y su desviación típica. Además, buscaremos el porcentaje crítico que dispare el encadenamiento de pausas e imposibilite ver el vídeo.

Tras realizar pruebas preliminares en la plataforma de Facebook Live, se ha decidido excluirla del análisis porque produce mediciones y conclusiones muy parecidas a Twitch. Ambas tienen en común que usan TCP, así que este apartado se centrará en comparar servicios TCP frente a UDP.

A continuación se presenta la tabla 4.2 que recoge las latencias medias junto a su desviación típica de cada subcaso de la prueba.

τ	Plat.	Twitch				YouTube			
		Directo		Restream		Directo		Restream	
		μ (s)	σ						
100 ms		3.36	0.17	4.93	0.35	3.11	0.36	9.23	0.5
200 ms		3.90	0.21	5.26	0.41	3.28	0.42	9.45	0.6
300 ms		4.48	0.25	5.46	0.47	3.77	0.58	9.66	0.75
400 ms		5.61	0.33	6.22	0.54	4.04	0.72	9.81	0.96
500 ms		6.66	0.44	7.18	0.62	4.21	0.91	9.84	1.19
600 ms		8.1	0.54	8.52	0.71	4.76	1.09	9.86	1.36
700 ms		9.84	0.66	10.34	0.82	4.78	1.21	9.92	1.57
800 ms		10.84	0.74	11.38	0.93	4.97	1.39	10.02	1.76
900 ms		12.59	0.81	13.46	1.02	5.25	1.56	11.15	1.92
1000 ms		15.46	1.17	16.13	1.32	5.37	1.74	13.27	2.07

Tabla 4.2: Latencia con retardo incremental de 100 ms

La figura 4.27 muestra que la tendencia sin y con Restream es ligeramente exponencial con la diferencia que en la segunda la curva está elevada 1.57 segundos respecto a la primera. Sin embargo, la tendencia en YouTube es ascendente de forma errática y con Restream presenta una tendencia lineal, casi plana hasta los 900 ms, donde pasa a ser exponencial de forma pronunciada, como muestra la figura 4.28.

A primera vista observamos que YouTube ofrece mejores prestaciones que Twitch, sin y con Restream. Los mecanismos de control de flujo de TCP en Twitch incrementan la latencia a lo largo del tiempo. Por otro lado, YouTube estabiliza la latencia entre 4 y 6 s sin Restream, pero con Restream se ve penalizada por una ubicación desfavorable de los servidores de ingesta.

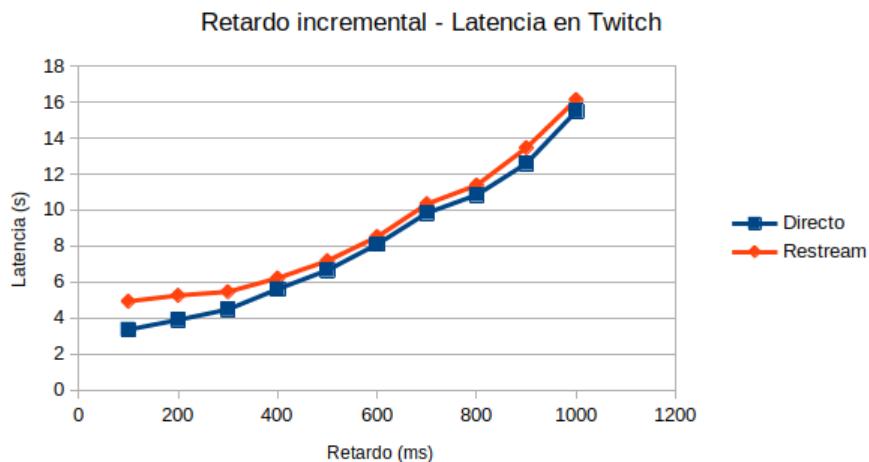


Figura 4.27: Evolución de la latencia en función del retardo en Twitch

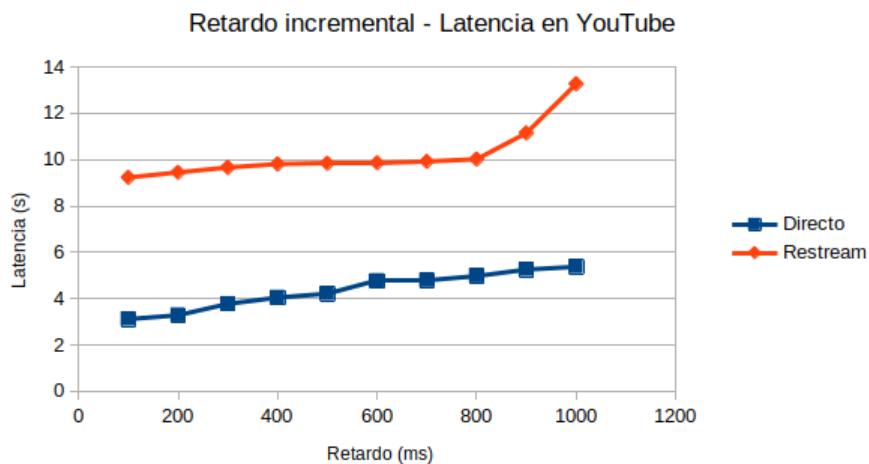


Figura 4.28: Evolución de la latencia en función del retardo en YouTube

Respecto a la desviación típica, la tendencia general de las gráficas es lineal y ascendente, aunque cabe señalar que para el caso de Twitch, tanto sin como con Restream, las gráficas presentan un aumento significativo de la pendiente, como muestra la figura 4.29. Para un retardo de 900 ms, el valor de latencia se vuelve cada vez más inestable, obteniendo valores de latencia cada vez más separados

entre sí, es decir, se alarga la cola de la función de distribución normal que suele usarse para modelar el retardo extremo a extremo.

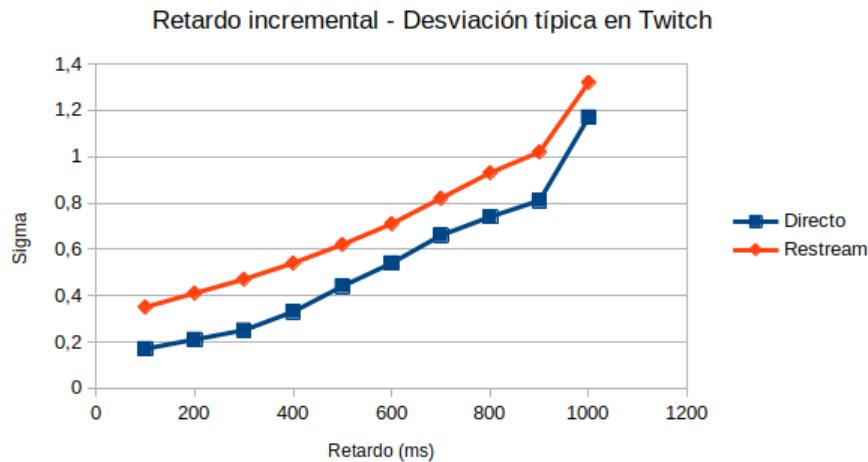


Figura 4.29: Evolución de la desviación típica en función del retardo en Twitch

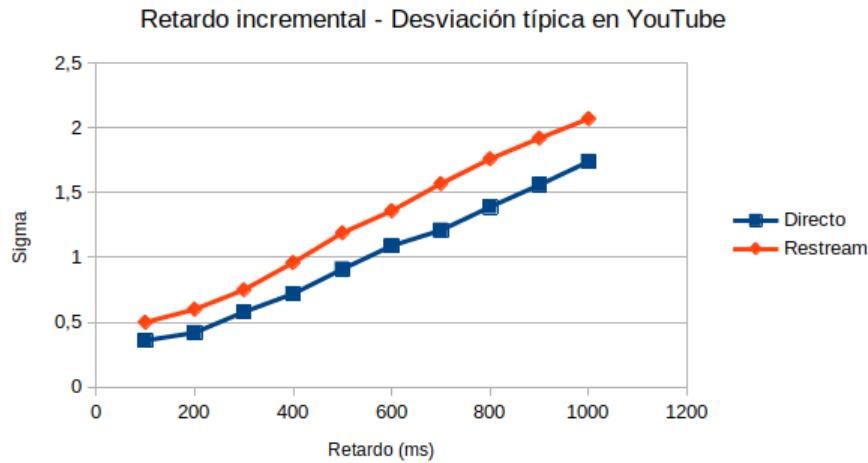


Figura 4.30: Evolución de la desviación típica en función del retardo en YouTube

4.2.3. Adición de pérdidas con granularidad 5 %

A continuación, se rehará el análisis con retardos de 100, 300 y 500 ms en Twitch y 500, 600 y 700 ms en YouTube añadiendo un porcentaje de pérdidas en incrementos de 5 %, es decir, 5 %, 10 %, 15 %, 20 %. Luego, buscaremos el porcentaje crítico que eleve la latencia por encima de 10 segundos (límite recomendado) e imposibilite ver el vídeo de forma aceptable.

Por la misma razón que en el apartado anterior, nos centraremos únicamente en el análisis de Twitch frente a YouTube. La tabla 4.3 recoge los valores de latencia obtenidos.

Plat. $\tau + p$		Twitch				Plat. $\tau + p$		YouTube					
		Directo		Restream				Directo		Restream			
		μ (s)	σ	μ (s)	σ			μ (s)	σ	μ (s)	σ		
100 ms	5 %	3.47	0.75	5.03	0.97	500 ms	5 %	5.14	0.75	9.96	0.92		
	10 %	3.92	0.83	5.23	1.07		10 %	5.62	0.79	10.07	1.07		
	15 %	4.4	0.99	5.73	1.24		15 %	6.31	0.87	10.19	1.23		
	20 %	4.46	1.02	5.88	1.29		20 %	7.04	0.92	10.26	1.25		
	33 %	4.64	1.14	5.96	1.34		33 %	7.92	1.13	10.97	1.49		
	40 %	7.3	1.33	9.03	1.88		40 %	11.04	1.33	13.4	1.78		
	50 %	15.77	2.17	16.45	2.56		50 %	15.79	2.26	19.79	2.84		
300 ms	5 %	4.57	0.88	5.54	1.09	600 ms	5 %	5.67	0.86	10.25	1		
	10 %	4.77	0.93	5.95	1.14		10 %	6.32	0.92	10.67	1.15		
	15 %	5.28	1.07	6.29	1.21		15 %	7.24	0.96	11.33	1.37		
	20 %	6.17	1.12	6.8	1.12		20 %	8.31	1.08	13.67	1.52		
	33 %	7.41	1.33	8.15	1.35		33 %	9.68	1.46	16.12	2.1		
	40 %	12.98	1.88	13.36	2.1		40 %	12.77	1.68	19.25	2.46		
	50 %	18.3	2.69	17.8	2.6		50 %	19.07	2.33	25.32	3.5		
500 ms	5 %	7.39	0.92	6.76	1.12	700 ms	5 %	6.1	0.92	10.57	1.27		
	10 %	7.74	1.21	7.26	1.36		10 %	6.66	1.12	11.77	1.35		
	15 %	8.02	1.57	9.2	1.56		15 %	7.71	1.34	13.89	1.68		
	20 %	8.68	2.33	11.01	1.86		20 %	9.22	1.68	17.23	2.3		
	33 %	11.6	2.11	13.43	2.5		33 %	10.83	1.99	21.05	3.6		
	40 %	16.61	3.02	18.55	3.34		40 %	14.49	2.4	28.3	4.2		
	50 %	21.5	3.46	23.56	3.76		50 %	23.11	3.2	32.25	5.7		

Tabla 4.3: Latencia con adición de pérdidas con granularidad 5 %

En adelante, utilizaremos en las gráficas el trazo continuo para el caso sin Restream y el trazo discontinuo para con Restream. La figura 4.31 muestra la representación gráfica del promedio en el caso de Twitch sin y con Restream, donde se ve que el retardo límite a partir del cual cambia claramente la tendencia es el 40 % de pérdidas de paquetes. Por otro lado, lo lógico es pensar que si bien a mayores pérdidas, mayor latencia, solo para Twitch hemos decidido probar los

efectos de una correlación, tan solo, del 10%. Se observa una diferencia de tendencia con YouTube: en Twitch, para los retardos de 100 y 300 ms, ambos casos de prueba ofrecen prestaciones razonablemente similares, siendo más notoria la diferencia para el caso de un retardo de 500 ms. Sin embargo, en el caso de YouTube es únicamente cuando se emplea Restream que el efecto de aumentar el retardo es más apreciable.

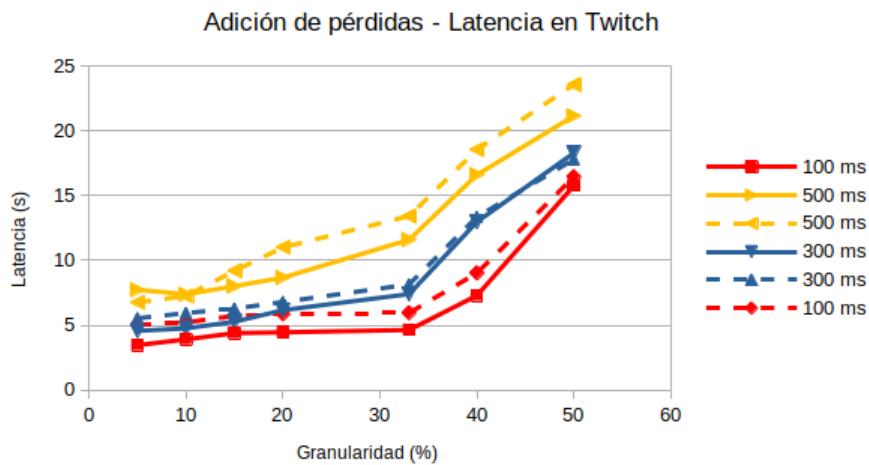


Figura 4.31: Evolución de la latencia en función del retardo más pérdidas en Twitch

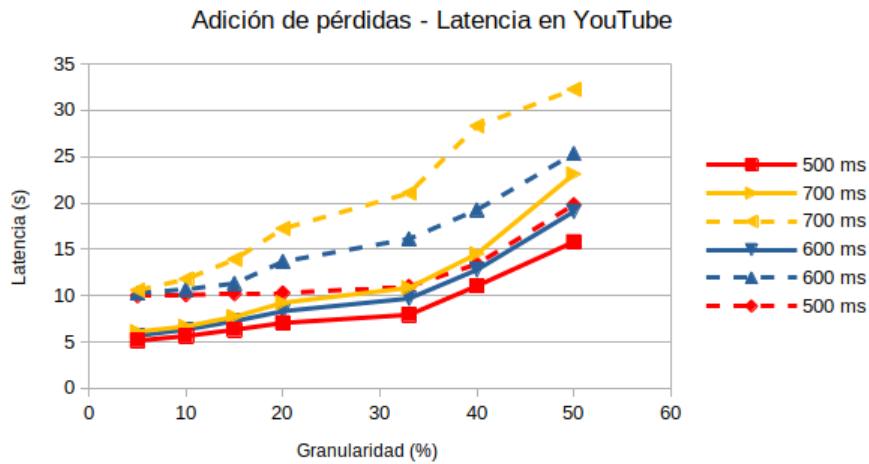


Figura 4.32: Evolución de la latencia en función del retardo más pérdidas en YouTube

Respecto a la desviación típica, en líneas generales la curva de 500 ms para Twitch y 700 ms para YouTube son claramente superiores a su inferior. Además, observamos que para una granularidad del 15 % se produce un cambio de pendiente en las curvas que las redirige decididamente hacia latencias más altas.

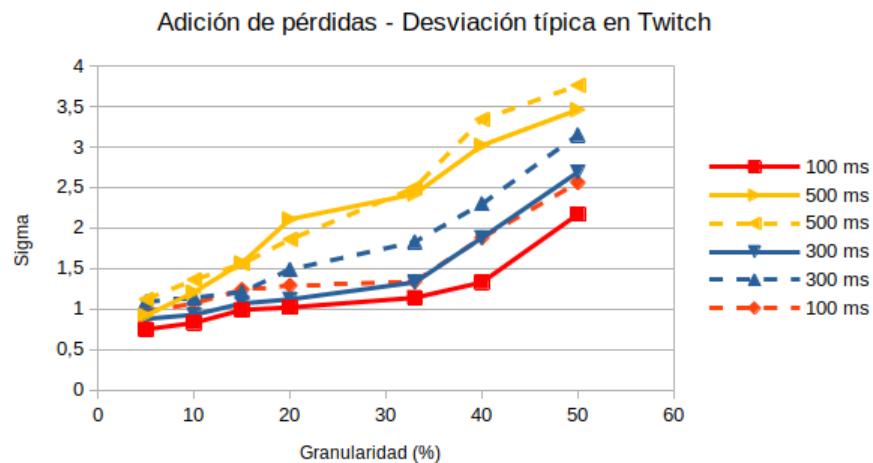


Figura 4.33: Evolución de la desviación típica en función del retardo más pérdidas en Twitch

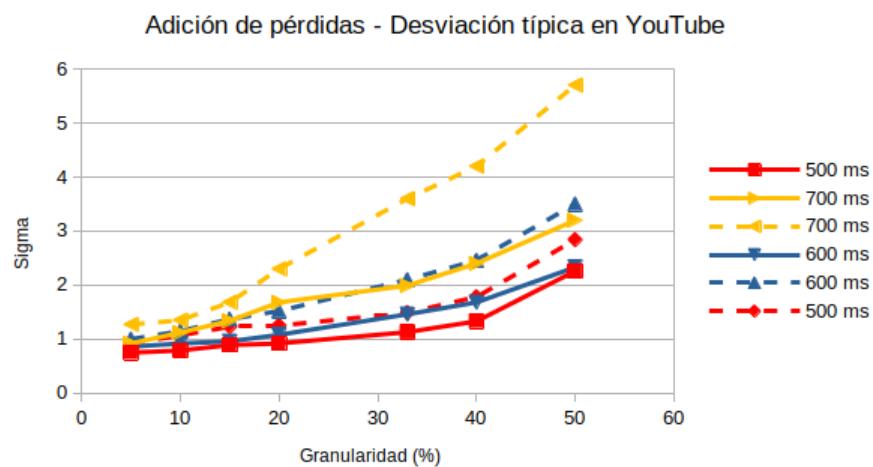


Figura 4.34: Evolución de la desviación típica en función del retardo más pérdidas en YouTube

4.3. Efecto del *jitter* en casos extremos

Vamos a medir la influencia del *jitter* en la latencia en casos desfavorables. Caracterizaremos la latencia como una distribución normal en función de retardos (τ) de 300, 500, 700 y 900 ms acompañados de un *jitter* (j) del 10, 30 y 100 % mediante 10 muestras para cada subcaso de prueba. La tabla 4.4 recoge las medidas de latencia para cada subcaso de la prueba.

$\tau + j$	Plat.	Twitch				YouTube			
		Directo		Restream		Directo		Restream	
		μ (s)	σ						
300 ms	10 %	4.46	0.33	6.13	0.67	5.55	0.23	10.33	0.49
	30 %	4.85	0.4	6.47	0.75	6.21	0.33	11.35	0.64
	100 %	6.28	0.83	8.13	0.91	7.13	0.81	13.66	1.13
500 ms	10 %	7.29	0.62	7.33	0.69	7.79	0.85	10.67	1.42
	30 %	7.82	0.66	8.01	0.89	8.23	0.9	12.01	1.6
	100 %	8.54	0.92	10.89	1.1	10.3	1.41	13.69	2.1
700 ms	10 %	10.27	0.74	10.97	1.2	9.2	0.9	10.95	1.56
	30 %	10.74	0.86	12.33	1.9	12.35	1.3	12.36	1.66
	100 %	11.75	1.23	15.61	2.5	17.36	2.7	14.01	3.33
900 ms	10 %	11.74	0.96	14.01	2.1	10.72	1.11	11.56	1.78
	30 %	12.29	1.39	16.69	2.7	15.86	2.5	13.48	3.12
	100 %	13.56	2.5	19.05	3.5	25.73	4.6	16.59	6.23

Tabla 4.4: Latencia con el efecto del *jitter*

La figura 4.35 muestra la evolución de la latencia conforme aumenta el porcentaje de *jitter* aplicado sobre los paquetes en Twitch sin y con Restream. Observamos que la tendencia es lineal y en aumento, es decir, la latencia aumenta proporcionalmente al *jitter*.

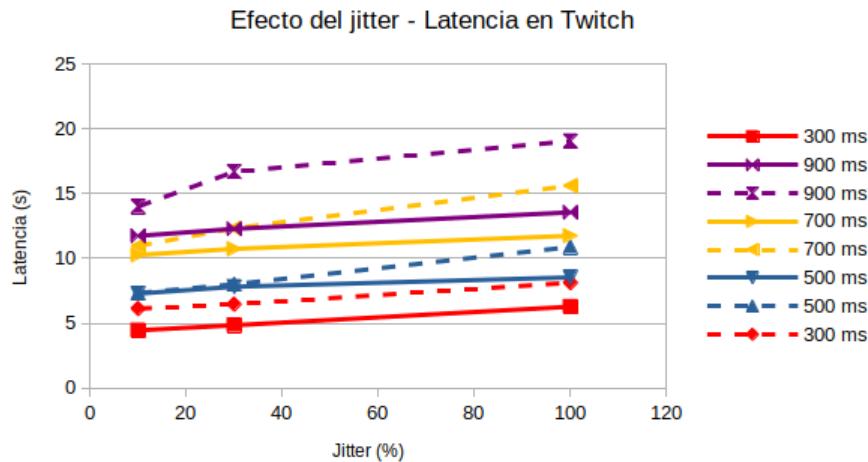


Figura 4.35: Evolución de la latencia en función del *jitter* en Twitch

En la figura 4.36 vemos cómo en Restream, para *jitter* bajo, siempre beneficia transmitir de manera directa. Sin embargo, aumenta la diferencia para *jitters* excesivos y puede darse incluso el caso contrario. La mayor latencia inicial que introduce Restream amortigua el efecto del *jitter* mejor que usar solamente YouTube a causa del *doble buffer* (Restream más YouTube).

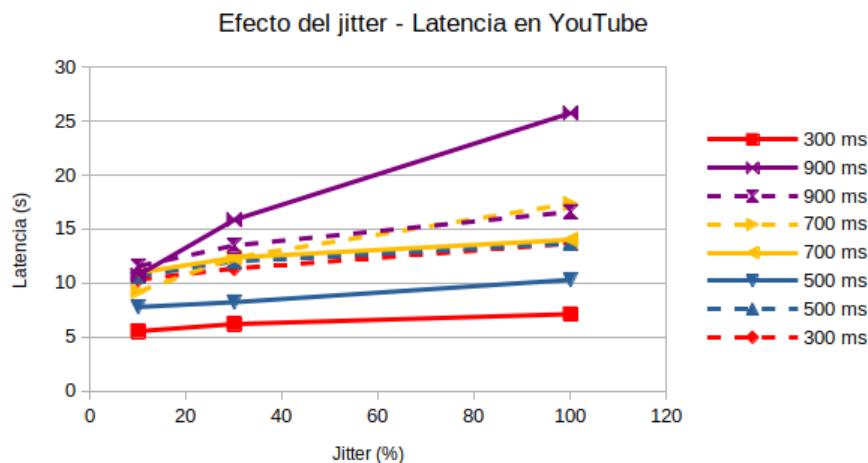


Figura 4.36: Evolución de la latencia en función del *jitter* en YouTube

Respecto a la desviación típica, en general todos los modos presentan una mayor velocidad de cambio conforme aumenta el retardo sobre el que se aplica

el *jitter*. Esto significa que a mayor retardo, mayor variabilidad de la latencia. Cuando una medida presenta demasiada variabilidad requiere ampliar el espacio muestral para extraer una desviación típica que se ajuste mejor a la realidad. En YouTube esta variabilidad es mayor que en Twitch, por lo que se tuvo que ampliar el número de muestras a 20 para calcular su media y desviación típica. En la figura 4.37 vemos que el incremento de la desviación típica es mayor cuanto mayor es el *jitter* aplicado. En cambio, esta diferencia es menor en el caso de YouTube que muestra la figura 4.38. La diferencia se debe al doble *buffer* entre Restream y la plataforma de YouTube mencionado anteriormente.

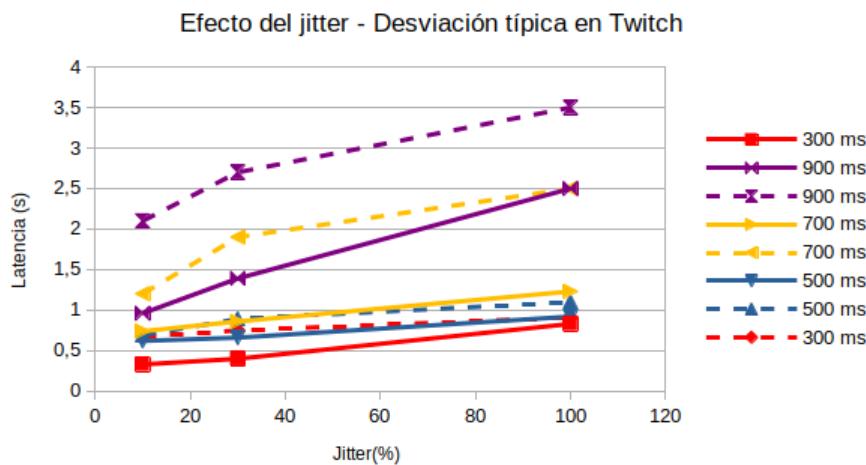


Figura 4.37: Evolución de la desviación típica en función del *jitter* en Twitch

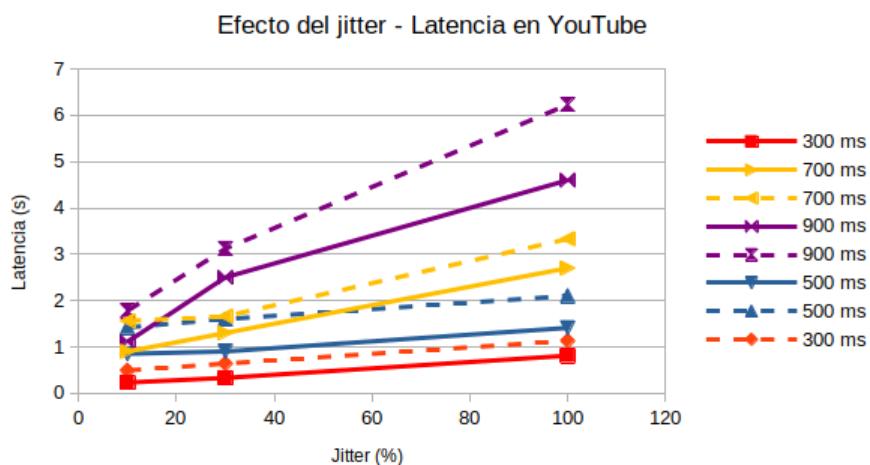


Figura 4.38: Evolución de la desviación típica en función del *jitter* en YouTube

4.4. Cálculo de la tasa de descarga límite

Se visualizará el vídeo modificando el ancho de banda de bajada, R_{des} , en el PC del usuario final. Se partirá de un valor menor que la tasa de codificación, $R_{cod} = 3800 \text{ kbps}$, para observar los efectos sobre la calidad y se subirá progresivamente hasta encontrar un valor límite, R_{lim} , a partir del cual la calidad de experiencia del usuario sea excelente. Para alcanzar un valor firme de tasa límite se realizarán cinco transmisiones.

Por motivos de repetitividad, solo se expondrá el procedimiento práctico para el caso de la transmisión directa a Twitch. El resto de casos se recogerán, junto a este último, en una tabla al final de la sección.

4.4.1. Procedimiento con Twitch

Partimos de una tasa de bajada de 2660 kbps, un 30 % inferior a R_{cod} , y observamos que el vídeo presenta múltiples pausas de *rebuffering* hasta acumular 40 s de latencia al final del vídeo, saltos y distorsión de audio. Es imposible ver el vídeo.

Subimos a 3000 kbps, 21.05 % inferior a R_{cod} . La distorsión de audio desaparece pero las múltiples pausas *rebuffering* y los saltos continúan. Es imposible ver el vídeo.

Subimos a 3300 kbps, 13.15 % inferior a R_{cod} . Desaparecen los saltos, pero las múltiples pausas *rebuffering* siguen presentes acumulando una latencia de unos 35 s al final del vídeo. Es imposible ver el vídeo.

Subimos a 3600 kbps, 5.25 % inferior a R_{cod} . Las múltiples pausas *rebuffering* acumulan un retardo al final del vídeo de 30 s. Sigue siendo imposible ver el vídeo.

Igualamos la tasas de descarga a la de codificación, 3800 kbps. Las pausas de *rebuffering* descienden a unas 7, pero siguen acumulando el retardo hasta los 30 s. Todavía no se puede ver el vídeo de forma aceptable.

Subimos a 4400 kbps, 15.78 % superior a R_{cod} . El vídeo tan solo presenta 3 pausas de *rebuffering* que acumulan una latencia de 9 s al final del vídeo. Aunque la calidad de experiencia mejora, todavía no se ha logrado el objetivo de cero pausas y latencia óptima.

Subimos a 4600 kbps, 21.05 % superior a R_{cod} . El vídeo no presenta ninguna pausa de *rebuffering* y la latencia es menor de 3 s, es decir, la óptima. Hemos encontrado el valor de descarga límite para el cual la QoE es excelente.

Vamos a expresar el incremento de la tasa límite por encima de la de codificación con la ecuación 4.1.

$$\Delta R = \left(\frac{R_{lim}^{TW}}{R_{cod}} - 1 \right) \cdot 100 \% = 21,05 \% \quad (4.1)$$

También vamos a obtener el *throughput* desde las gráficas TCP de Wireshark y vamos a expresarlo como el porcentaje de la tasa de descarga que en efecto es transferida en el *streaming* con la ecuación 4.2.

$$Th (\%) = \frac{Th}{R_{des}} \cdot 100 \% = 93,92 \% \quad (4.2)$$

La figura 4.39 muestra las peticiones del usuario a la plataforma. Al principio se observa que una de las ráfagas es más alta que las demás debido al llenado rápido inicial del *buffer* en el que se solicitan muchos datos del vídeo. Cuanto mejor es la conexión, más corto es el tren de peticiones inicial y, de hecho, a nuestra aplicación solo le ha bastado un elemento de petición.

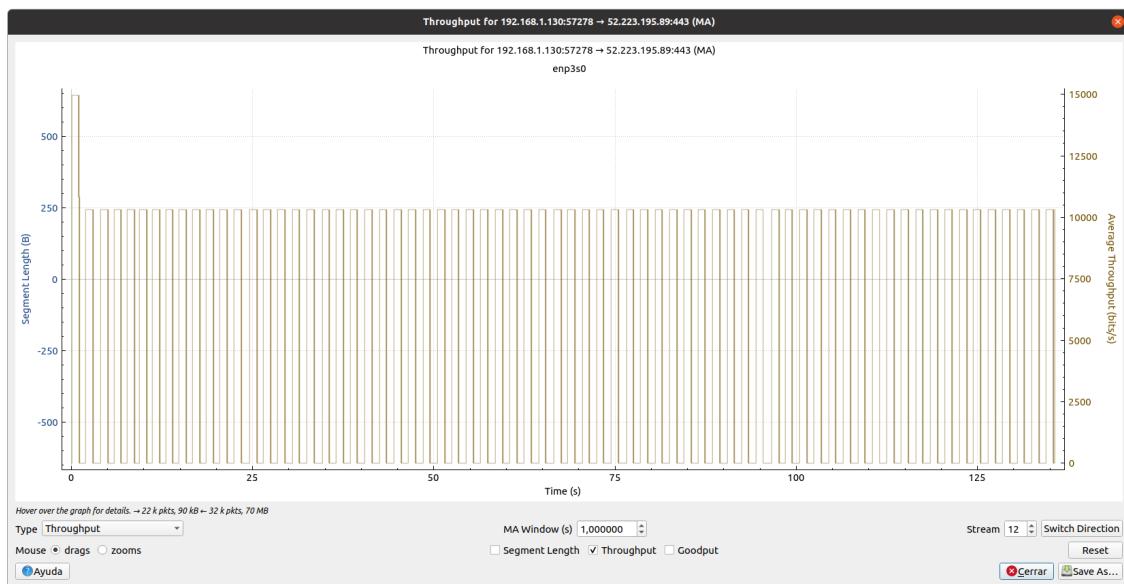


Figura 4.39: *Throughput* en sentido ascendente en Twitch

La figura 4.40 muestra la tasa efectiva de transferencia de la plataforma al usuario. La gráfica es una línea recta casi perfecta debido a que la calidad de vídeo percibido es excelente: sin pausas, saltos ni distorsiones que disparen las

retransmisiones. En tal caso, se puede calcular el *throughput* perfectamente como la pendiente de dicha recta escogiendo dos puntos cualquiera de ella y dividiendo la diferencia entre los números de secuencia y los instantes de tiempo.

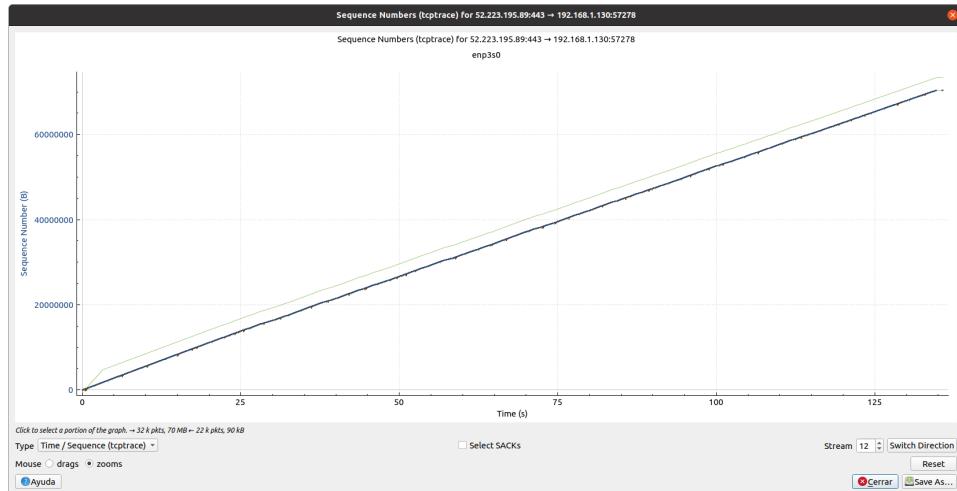


Figura 4.40: *Throughput* en sentido descendente en Twitch

La figura 4.41 es interesante porque muestra la evolución de la tasas de transferencia a lo largo del tiempo que dura el vídeo. Para calcular el valor promedio se ha guardado la gráfica como un fichero .csv, se ha abierto en una hoja de cálculo y allí se ha realizado el cálculo. El valor del *throughput* siempre será menor que R_{lim} .

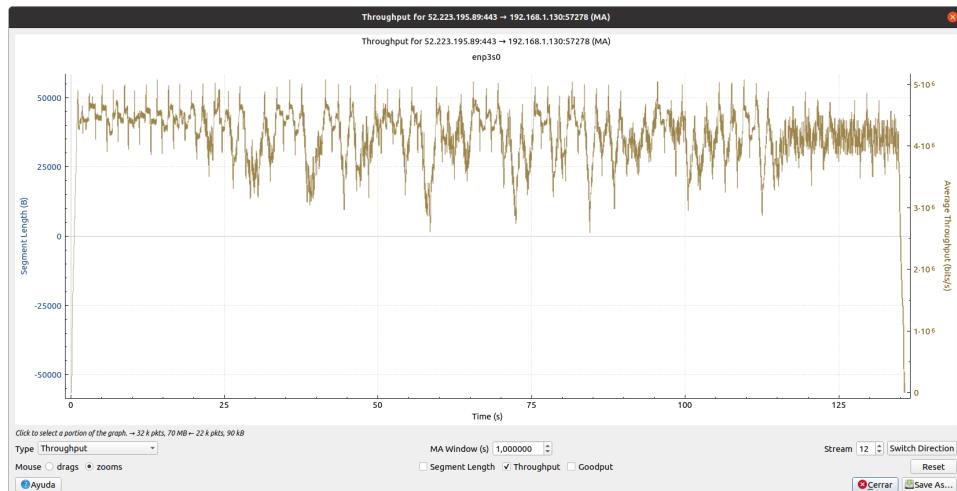


Figura 4.41: *Throughput* en sentido descendente en Twitch

Por último, la figura 4.42 muestra el *goodput* que es la tasa efectiva de transferencia excluyendo las cabeceras y el tráfico generado por retransmisiones. Este valor también se puede promediar y nos ha salido siempre igual o ligeramente inferior al *throughput*.

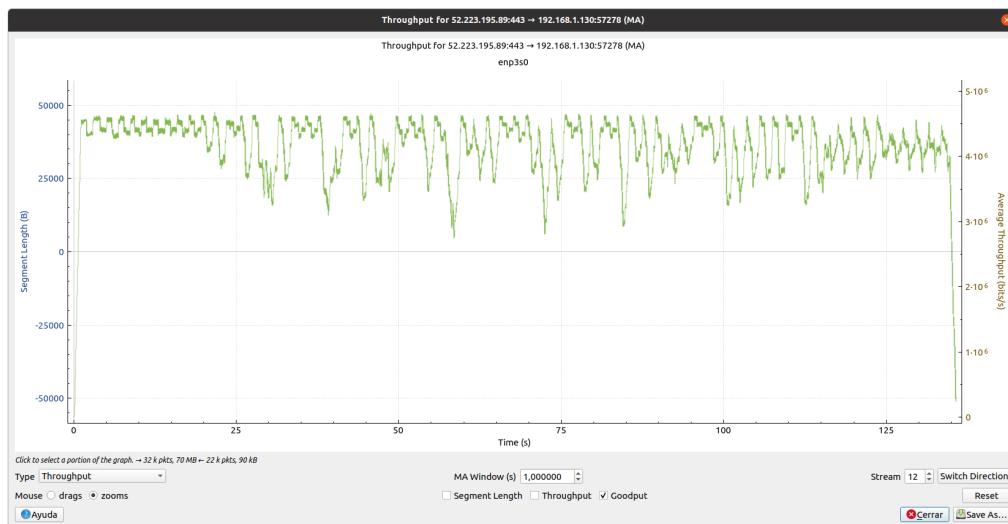


Figura 4.42: *Goodput* en sentido descendente en Twitch

4.4.2. Resultados

A continuación, la tabla 4.5 muestra los resultados tras encontrar R_{lim} en todos los escenarios, el *throughput* y los valores de ΔR y Th (%).

Plataforma \ Vía		Directo	Restream
Twitch	R_{lim}	4600 kbps	4600 kbps
	ΔR	21.05 %	21.05 %
	Th	4320.7 kbps	4412.6 kbps
	$Th\ %$	93.92 %	95.91 %
Facebook Live	R_{lim}	4400 kbps	4700 kbps
	ΔR	15.78 %	23.68 %
	Th	4150.2 kbps	4563.3 kbps
	$Th\ %$	94.32 %	97.09 %
YouTube	R_{lim}	4200 kbps	4100 kbps
	ΔR	10.52 %	7.89 %
	Th	3918.5 kbps	4018.6 kbps
	$Th\ %$	93.28 %	98.01 %

Tabla 4.5: Tasas de descarga límite

Twitch requiere un incremento del 21.05 %, sin y con Restream, por encima de la tasa de codificación del vídeo para que la calidad del vídeo percibido por el usuario sea excelente. Facebook Live requiere un incremento del 15.78 % superior a la tasa de codificación frente a un 23.68 % con Restream. Y YouTube requiere una tasa de descarga del 10.52 % superior a la de codificación frente a un 7.89 % con Restream.

Respecto al *throughput* vemos que con Restream mejora la eficiencia de la transmisión para las tres plataformas.

4.5. Análisis y discusión de resultados

En esta sección se analizarán los resultados obtenidos en los casos de prueba. En primer lugar, tras realizar el caso de prueba de errores TCP se han alcanzado las siguientes conclusiones:

- Cuando se introducen restricciones severas de retardos agravadas con pérdidas de paquetes, empiezan a llegar ráfagas de errores de más de 30 paquetes/segundo tanto en Twitch como Facebook Live. Se trata de pausas que restan calidad de experiencia al usuario: pausas, saltos y distorsiones de audio. Se vio como en Twitch estas ráfagas ocasionaron pausas; sin embargo, en Facebook Live no, por el buen dimensionamiento inicial del *buffer*. La explicación es que el codificador está ajustado con una estrategia CBR y la plataforma hace una reserva de recursos según el bit más pobre al inicio de la transmisión. Es decir, la plataforma no se adapta a los cambios en la tasa de bits que recibe y por eso las ráfagas de errores son más predecibles.
- Respecto al total de errores, se observa que los segmentos fuera de orden son un 25 % más en el caso sin restricciones que con ellos. Esto quiere decir que sin restricciones la plataforma ocupa una mayor parte de su tiempo enviando pedazos de vídeo que solicitando retransmisiones al *streamer*.
- Cuando hay restricciones en el caso de Facebook Live, observamos que la densidad de errores es significativamente mayor que en Twitch, y que se disparan las retransmisiones a causa de la expiración del temporizador de retransmisión.
- Respecto a la transmisión directa a la plataforma, el uso de Restream disminuye los errores en Twitch pero los aumenta en Facebook Live. Esto se debe a que la red de distribución de contenidos es tal que pone el servidor de ingesta más cerca del usuario final y, en cambio, lo pone más lejos en el caso de Facebook Live.

Para el caso de prueba de medida de latencia media, se han extraído los siguientes puntos:

- Respecto a la sección del retardo incremental, vemos que el aumento del retardo propicia las pausas de *rebuffering* que degradan la calidad de experiencia del usuario. Con Restream, vemos que el aumento de la latencia es significativo pero sin riesgo si el retardo es menor de 800 ms. Alcanzar tales

valores es muy difícil incluso con una conexión de datos móviles, por lo que el uso de Restream no supone un riesgo para la calidad de experiencia del usuario.

- Cuando entran en juego las pérdidas, el uso de Restream curiosamente no aumenta la latencia del vídeo sino que tiende a aplanar la curva respecto al caso directo. Para el caso de Youtube, observamos que la latencia de partida en retardo de 100 ms ya es muy próxima a latencia de 10 s con Restream. Sin embargo, incluso en un escenario desfavorable de muchas pérdidas si la plataforma es capaz de dimensionar bien el *buffer*, se comprueba que no genera pausas y no afecta a la calidad del vídeo percibido. La solución, en este caso, es una mayor espera inicial del usuario.
- Restream supone un riesgo para la calidad del vídeo percibido en YouTube cuando el retardo supera los 500 ms, pues la curva de latencia se pega a 10 s, que es el límite que hemos encontrado experimentalmente. Cuando se añaden pérdidas es peligroso cuando el retardo es de 700 ms y a partir de una granularidad del 15 % en Twitch y 5 % en YouTube. Con Restream, la plataforma de YouTube es más vulnerable que Twitch en presencia de retardo combinado con pérdidas.
- Al hilo del punto anterior, el riesgo de degradación de QoE coincide con una desviación típica aproximadamente mayor que 1.5. La desviación típica es, por tanto, un indicador de QoE relacionado con la latencia, como hemos comprobado experimentalmente.

Para la prueba del efecto del *jitter*, se pueden hacer los siguiente comentarios:

- Restream pone en riesgo el *streaming* en Twitch a partir de los 700 ms de retardo y 5 % de *jitter*, donde empiezan a aparecer pequeñas distorsiones de audio dando paso a pausas y tirones a medida que aumenta el *jitter*. Para el caso de YouTube, el riesgo también llega a partir de los 700 ms y *jitter* del 5 % pero, en este caso, como no hay mecanismos de control de flujo los paquetes que se pierdan afectan directamente a la calidad del vídeo sin posibilidad de recuperación. Por eso, la curva de latencia a 900 ms de YouTube es más empinada que la de Twitch. Por tanto, YouTube es más vulnerable a los efectos del *jitter* que Twitch por utilizar UDP, el cual no proporciona ninguna garantía.
- Respecto a la desviación típica, la gráfica de YouTube respalda lo comentado en el punto anterior para un retardo de 900 ms.

Para la prueba del cálculo de la tasa de descarga límite, podemos hacer los siguientes comentarios:

- El uso de Restream obliga a subir la tasa de codificación en Facebook Live pero se puede bajar en YouTube, mientras que en Twitch se debe mantener constante. Transmitir en Facebook Live con Restream sería la opción más eficiente si existieran problemas de ancho de banda.
- El doble *buffer* que tenemos, a costa de aumentar algo la latencia, permite ajustar mejor la tasa de codificación a la tasa de transmisión por la red.

Capítulo 5

Conclusiones y líneas futuras

El trabajo desarrollado permite concluir de manera general que Restream es un fantástico servicio de *multistreaming*, que permite retransmitir hacia múltiples plataformas sin degradar en exceso la calidad de experiencia del usuario. Se ha evaluado su comportamiento para la transmisión de vídeo hacia tres plataformas diferentes como Twitch, YouTube y Facebook Live, proporcionando buenas prestaciones en general respecto al caso de llevar a cabo la transmisión directa a la plataforma.

Las principales conclusiones que se han extraído en este trabajo pueden resumirse del siguiente modo:

- El efecto de la pérdida de paquetes se traduce en una degradación de la calidad de experiencia, debido principalmente al aumento de retransmisiones que se asocian con pausas en la reproducción de vídeo para servicios basados en TCP. El uso de Restream en este escenario penaliza más en la plataforma Facebook Live que en el caso de Twitch.
- En cuanto al efecto observado al aumentar el retardo en la comunicación, el uso de Restream en servicios basados en UDP (como YouTube) introduce una latencia adicional de mayor magnitud que en el caso de servicios basados en TCP (como Twitch). En ambos casos, el uso de Restream provoca un aumento del *jitter* medido extremo a extremo de similar magnitud.
- El efecto de aumentar el *jitter* provoca un incremento proporcional de la latencia extremo a extremo. Una conclusión interesante en este caso es que el uso de Restream permite amortiguar mejor el efecto del *jitter* en comparación con la transmisión directa hacia YouTube, gracias al doble *buffer* que se implementa en este caso.

- Por último, la tasa límite de descarga a partir de la cual la calidad de experiencia se ve comprometida no se modifica severamente por usar Restream.

En cuanto a las principales líneas futuras de trabajo, se proponen las siguientes:

- Comparar las prestaciones de Restream con otras plataformas incipientes de *multistreaming* como Castr, Mobcrush, StreamYard o SwitchBoard.
- Evaluación cuantitativa de la calidad de experiencia de servicios *multistreaming*, más allá de los resultados cualitativos analizados en este trabajo.
- Estudio del efecto de la latencia extremo a extremo en la interactividad para servicios de *gaming* como Twitch.

Bibliografía

- [1] Latencia de vídeos en streaming en directo. Último acceso: 31 de enero de 2023.
- [2] Configurar OBS Studio para hacer stream en Facebook. Último acceso: 31 de enero de 2023.
- [3] Configurar OBS Studio para hacer stream en Twitch. Último acceso: 31 de enero de 2023.
- [4] Live Media and Metadata Ingest Protocol. Último acceso: 31 de enero de 2023.
- [5] Adobe Real Time Messaging Protocol. Último acceso: 31 de enero de 2023.
- [6] Use Linux Traffic Control as impairment node in a test environment. Último acceso: 31 de enero de 2023.
- [7] Wondershaper – A tool to limit network bandwidth in Linux. Último acceso: 31 de enero de 2023.
- [8] Pablo Ameigeiras, Juan J. Ramos-Munoz, Jorge Navarro-Ortiz, and J.M. Lopez-Soler. Analysis and modelling of youtube traffic. *Transactions on Emerging Telecommunications Technologies*, 23(4):360–377, 2012.
- [9] L. Beloqui Yuste, F. Boronat, M. Montagud, and H. Melvin. Understanding timelines within mpeg standards. *IEEE Communications Surveys Tutorials*, 18(1):368–400, 2016.
- [10] J. Kua, G. Armitage, and P. Branch. A survey of rate adaptation techniques for dynamic adaptive streaming over http. *IEEE Communications Surveys Tutorials*, 19(3):1842–1866, 2017.
- [11] Adam Langley, Alistair Riddoch, Alyssa Wilk, Antonio Vicente, Charles Krasic, Dan Zhang, Fan Yang, Fedor Kouranov, Ian Swett, Janardhan Iyengar,

et al. The QUIC transport protocol: Design and internet-scale deployment. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pages 183–196, 2017.

