UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH

DESARROLLO DE UNA PLATAFORMA PARA LA DISTRIBUCIÓN DE CONTENIDOS AUDIOVISUALES UTILIZANDO MPEG-DASH

Trabajo de Fin de Grado

Presentado en la Facultad

Escola Tècnica d'Enginyeria de Telecomunicació de Barcelona

Universitat Politècnica de Catalunya

por

Marc Macià Roca

En cumplimiento parcial

de los requisitos para el grado en

INGENIERÍA DE SISTEMAS AUDIOVISUALES

Dirigido por: Jorge Mata

Barcelona, enero 2016

Abstract

The growth in the consumption of audiovisual content over the Internet continues to grow year after year, thus users from around the world need that content is available for their connection and device needs.

The concept of adaptive bitrate streaming appeared a few years ago and is a technique that allows to adjust the audiovisual content flow to each user. MPEG-DASH is the standard that was created to unify the various existing adaptive streaming proprietary techniques.

This work consists firstly of a study of the standard, allowing subsequently creating a closed platform for distribution of MPEG-DASH audiovisual content. This platform, which will be formed by different software as a packager and several clients, allow us to evaluate the quality stream-switching algorithms.

The platform will be created, allowing us to generate MPEG-DASH effective content from any original video. Moreover an evaluation of the quality stream-switching algorithms that implements the clients according to different network conditions and providing improvements to some of the situations will be performed.

Resum

El creixement en el consum de contingut audiovisual a través d'Internet no para de créixer any rere any, per això usuaris d'arreu del món necessiten que aquest contingut estigui disponible a les seves necessitats de connexió i dispositiu.

El concepte de streaming amb bitrate adaptatiu va aparèixer ja fa uns anys i és una tècnica que permet ajustar el flux del contingut audiovisual a cada usuari. MPEG-DASH es l'estàndard que va néixer amb l'objectiu d'unificar les diferents tècniques propietàries de streming adaptatiu ja existents.

Aquest treball consisteix primerament estudi de l'estàndard, permetent a posteriori la creació d'una plataforma tancada de distribució de continguts audiovisuals MPEG-DASH. Aquesta plataforma, que estarà formada per diferent software com un segmetador i diversos clients, ens permetrà l'avaluació dels algoritmes de canvi de qualitat.

S'aconseguirà crear aquesta plataforma, podent crear contingut MPEG-DASH eficaç a partir de qualsevol vídeo original, realitzant després una avaluació dels algoritmes de canvi de qualitat que implementen els clients segons diferents condicions de xarxa i aportant millores per a alguna de les situacions.

Resumen

El crecimiento en el consumo de contenido audiovisual a través de Internet no para de crecer año tras año, con ello usuarios de todas partes del mundo necesitan que ese contenido esté disponible a sus necesidades de conexión y dispositivo.

El concepto de streaming con bitrate adaptativo apareció ya hace unos años y es una técnica que permite ajustar el flujo del contenido audiovisual a cada usuario. MPEG-DASH es el estándar que nació con el objetivo de unificar las distintas técnicas propietarias de streaming adaptativo ya existentes.

Este trabajo consiste primeramente en un estudio del estándar, permitiendo a posteriori la creación de una plataforma cerrada de distribución de contenidos audiovisuales MPEG-DASH. Esta plataforma, que estará formada por diferente software como un segmentador y varios clientes, nos permitirá la evaluación de los algoritmos de cambio de calidad.

Se conseguirá crear dicha plataforma, pudiendo crear contenido MPEG-DASH eficaz a partir de cualquier vídeo original, realizando después una evaluación de los algoritmos de cambio de calidad que implementan los clientes según distintas condiciones de red y aportando mejoras para alguna de las situaciones.

Este trabajo se lo dedico a mi madre y a mi hermano.

Agradecimientos

En primer lugar, me gustaría darle las gracias a mi tutor, Jorge Mata, por haberme ofrecido la posibilidad de realizar este proyecto, por todos los consejos, por toda la documentación aportada y por haber estado disponible en todo momento. Tu apoyo me ha ayudado mucho en el desarrollo del proyecto.

También me gustaría agradecer a mi familia el apoyo e inspiración que me han dado desde que empezó el grado, haciendo que la superación de cada uno de los retos haya sido especialmente agradable. Sin vosotros no habría sido posible.

¡Muchas gracias a todos!

Historial y aprobación de revisiones

Revisión	Fecha	Propósito
0	23/12/2015	Creación del documento
1	04/01/2015	Revisión del documento
2	18/01/2015	Revisión del documento
2	25/01/2015	Revisión del documento

LISTA DE DISTRIBUCIÓN DEL DOCUMENTO

Nombre	e-mail
Marc Macià	marc.macia.roca@alu-etsetb.upc.edu
Jorge Mata	jorge.mata@entel.upc.edu

Escrito por:		Revisado y aprobado por:	
Fecha	23/12/2015	Fecha	25/01/2016
Nombre	Marc Macià	Nombre	Jorge Mata
Posición	Autor del proyecto	Posición	Supervisor del proyecto

Tabla de contenidos

Abstract 1
Resum 2
Resumen 3
Agradecimientos 5
Historial y aprobación de revisiones 6
Tabla de contenidos 7
Lista de Figuras 9
Lista de Tablas 11
1. Introducción 12
1.1. Objetivos 12
1.2. Requisitos y especificaciones 13
1.3. Plan de trabajo 14
1.3.1. Work Packages, Tareas e Hitos 14
1.3.2. Diagrama Gantt 15
2. Estado del arte de la tecnología utilizada 16
2.1. Antecedentes 16
2.2. Adaptive Bitrate Streaming 16
2.3. MPEG-DASH 17
2.3.1. Media Presentation Description (MPD) 18
2.3.1.1. Estructura del MPD 18
2.3.1.2. Ejemplo de un MPD 18
2.4. H.264/Advanced Video Coding 19
2.5. AAC/Advanced Audio Coding 19
3. Metodología / desarrollo del proyecto 20
3.1. Entorno de trabajo 20
3.2. Generación del contenido 21
3.2.1. Codificación del vídeo 22
3.2.2. Codificación del audio 24
3.2.3. Segmentación de las representaciones 24
3.2.4. Automatización y especificaciones 25
3.2.4.1. Archivo de configuración 25

3.2.4.2. Funcionamiento 26
3.2.4.3. Especificaciones 27
3.3. Algoritmos de adaptación del bitrate 29
3.3.1. Propósito y funcionamiento genérico 29
3.3.2. Clientes estudiados 31
3.3.2.1. TAPAS 31
3.3.2.2. MP4Client 32
3.3.2.3. Aportaciones 33
4. Resultados 34
4.1. Definición de las pruebas 34
4.2. Definiciones de las condiciones de red para las pruebas 34
4.3. TAPAS 35
4.4. MP4Client 39
5. Presupuesto 41
6. Conclusiones y trabajo futuro 42
Bibliografía 43
Apéndices 45
A: Configuración del entorno de trabajo 45
A.1: Host 46
A.2: Servidores 46
A.3: WANem 48
B: Programa para automatizar la generación del contenido DASH 49
C: Instalación y utilización de los clientes DASH 58
D: Scripts para las pruebas 60
D.1: TAPAS 60
D.1: MP4Client 60
Glosario 62

Lista de Figuras

Figura 1: Composición del tráfico en período de máxima actividad. América del Norte [4] 12
Figura 2: Diagrama Gantt 15
Figura 3: Ejemplo de funcionamiento del streaming adaptativo para 3 calidades 16
Figura 4: Visión general del streaming con bitrate adaptativo 17
Figura 5: Modelo streaming MPEG-DASH 18
Figura 6: Entorno de pruebas controlado 20
Figura 7: Configuración de red 21
Figura 8: Diagrama general de la generación del contenido 22
Figura 9: Diagrama Programa 27
Figura 10: Niveles de calidad del vídeo en escala logarítmica 28
Figura 11: Niveles de calidad del audio 29
Figura 12: Patrón on-off de la velocidad de descarga 30
Figura 13: Solicitud-Respuesta entre servidor y cliente en los estados Buffering y Steady 30
Figura 14: Diagrama de flujo general del algoritmo para TAPAS 32
Figura 15: Diagrama de flujo general del algoritmo para MP4Client 33
Figura 16: Comparación de los niveles de calidad para la 1ª prueba, segmentos de 1 segundo, utilizando 2 formas de estimar el ancho de banda 35
Figura 17: Ancho de banda estimado y bitrate de la calidad para la 1ª prueba, segmentos de 5 segundos 35
Figura 18: Niveles de calidad para la 1ª prueba, segmentos de 5 segundos 36
Figura 19: Ocupación del buffer para la 1ª prueba, segmentos de 5 segundos 36
Figura 20: Ancho de banda estimado y bitrate de la calidad para la 2ª prueba, segmentos de 1s 37
Figura 21: Ancho de banda estimado y bitrate de la calidad para la 2ª prueba, segmentos de 15s 37
Figura 22: Ocupación del buffer para la 2ª prueba, segmentos de 1 segundo 37
Figura 23: Prueba 3, amigabilidad entre 2 clientes TAPAS, segmentos 1s. Captura de Wireshark 38
Figura 24: Prueba 3, amigabilidad entre 2 clientes TAPAS, segmentos 5s. Captura de Wireshark 38
Figura 25: Comparación del ancho de banda estimado y el bitrate de la calidad entre ambos clientes, con segmentos de 1s 39

Figura 26: Comparación del ancho de banda estimado y el bitrate de la calidad entre ambos clientes, con segmentos de 5s 39
Figura 27: Prueba 1, con segmentos de 5s. Captura de Wireshark 39
Figura 28: Prueba 3, amigabilidad entre 2 clientes MP4Client, segmentos 5s 40
Figura 29: Prueba 3, amigabilidad entre 2 clientes MP4Client, segmentos 15s 40
Figura 30: Configuración de la eth1 de ambos servidores 45
Figura 31: Configuración de la eth1 y la eth2 de WANem 46

Lista de Tablas

Tabla 1: Tareas e hitos del plan de trabajo 14
Tabla 2: Parámetros del fichero de configuración 26
Tabla 3: Representaciones del vídeo con bitrates y resoluciones asociados 28
Tabla 4: Representaciones del audio 29
Tabla 5: Configuraciones de red asignadas a cada prueba 34
Tabla 6: Coste total del proyecto 41
Tabla 7: Tabla de rutas del Host 46
Tabla 8: Tabla de rutas del servidor 1 47
Tabla 9: Tabla de rutas del servidor 2 47
Tabla 10: Tabla de rutas de WANem 48

1. Introducción

Los servicios de streaming de vídeo no han parado de crecer en los últimos años y eso ha llevado a un aumento en el tráfico de Internet, seguramente debido al crecimiento de los dispositivos con acceso a la red, como móviles y tabletas. Hoy en día servicios como Youtube [1] o Netflix [2] entre otros dominan en gran parte del mundo como puede observarse en el informe "Global Internet Phenomena" que realiza sandvine [3] con períodos de máxima actividad de descarga de hasta un 25% del tráfico total para Youtube en Europa durante julio 2015.

A continuación, en la Figura 1, podemos observar el porcentaje del tráfico total de Norteamérica desinado al streaming de audio y vídeo tanto para acceso móvil como para acceso fijo durante septiembre y octubre 2015 (en la gráfica marcado como Real-Time Entertainment) donde observamos que es del 70,4% y del 40,89% respectivamente.

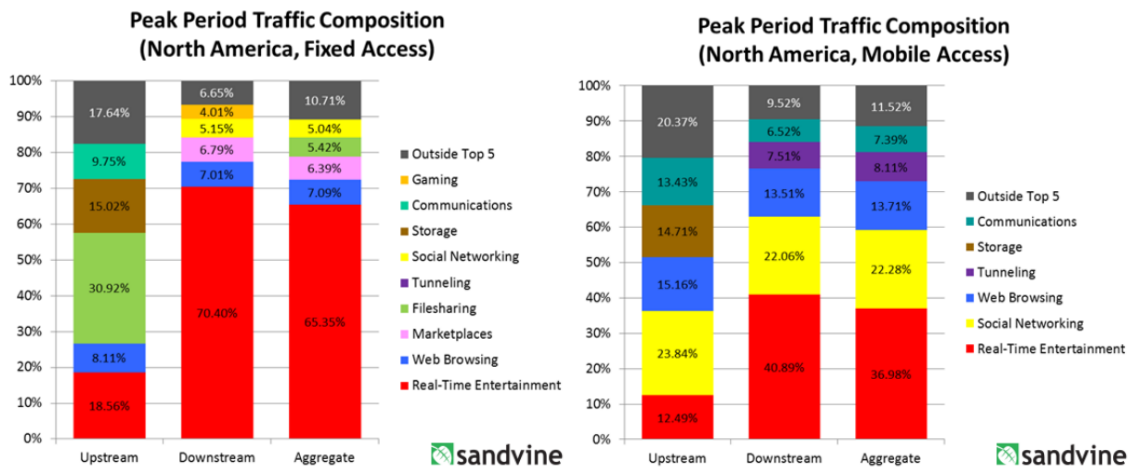

Figura 1: Composición del tráfico en período de máxima actividad. América del Norte [4]

Debido a que la demanda es tan alta y que cada usuario necesita que el contenido que se ofrece esté disponible a sus necesidades de conexión y dispositivo (por ejemplo, una televisión de 720p no necesita redimensionar el contenido a 1080 sino que puede conseguir la resolución adecuada directamente) nace el streaming adaptativo, con el máximo objetivo de mejorar la calidad de la experiencia del usuario (QoE) y permitiendo así que el que usuario obtenga un stream personalizado y pueda disfrutar del contenido independientemente de su conexión.

No sólo eso, sino que debe adaptarse a las fluctuaciones de la conexión o al reparto de los recursos de la misma, por ejemplo amortiguando el impacto que ocurre cuando varios usuarios comparten el mismo ancho de banda y evitando así que el vídeo se detenga a causa de ello.

1.1. Objetivos

El propósito de este proyecto es desarrollar una plataforma que nos permita estudiar principalmente los algoritmos que realizan el cambio de calidades, la adaptación del bitrate. Utilizando el estándar MPEG-DASH [5], que surgió con el objetivo de unificar las distintas técnicas propietarias de streaming adaptativo ya existentes, y distintas herramientas disponibles, experimentando con distintas configuraciones de red y duraciones de segmento para poder sacar conclusiones.

La plataforma deberá ser capaz de generar contenido audiovisual tal y como se define en el estándar, para ello se creará un programa que automatiza la generación de contenido y que hará uso de los programas de codificación y segmentación correspondientes. Se codificará el contenido en H.264 [6] para cada uno de los diferentes bitrate y resoluciones, y se segmentarán en trozos para la obtención del archivo *manifest*, que describe el contenido.

Se necesitará la puesta en marcha de un entorno de trabajo que incluya un par de servidores HTTP [7] con páginas de acceso a los diferentes contenidos y al menos un host para realizar distintas pruebas de funcionamiento con los clientes que soportan MPEG-DASH, así como un conformador de ancho de banda para emular un escenario de WAN en la que puedan ajustarse libremente distintas condiciones de red.

Se realizarán pruebas para la obtención de figuras de mérito y se valorará la calidad de experiencia del usuario según los distintos algoritmos de adaptación del bitrate a fin de determinar su funcionalidad según las configuraciones de red y en caso de tener la oportunidad modificándolos para mejorar su rendimiento.

1.2. Requisitos y especificaciones

Requisitos del proyecto:

- Estudiar el funcionamiento de MPEG-DASH y la gran utilidad de las técnicas de streaming con bitrate adaptativo en el mundo actual.
- Diseñar un entorno de trabajo cerrado que deberá ser estable y funcional para la realización del proyecto que permita moldear a voluntad las condiciones de red entre un servidor y un cliente.
- La plataforma deberá ser capaz de generar contenido MPEG-DASH correctamente de manera fácil y rápida.
- Estudio de las distintas herramientas ya diseñadas que permiten la distribución del contenido, tanto herramientas que permiten la segmentación de un vídeo hasta los clientes capaces de reproducir dicho contenido.
- Distintas duraciones de segmentación deberán ser analizadas durante la fase de segmentación del contenido multimedia original para poder evaluar las diferencias observadas.
- Evaluación de los algoritmos de cambio de calidad, modificándolos en el caso de poder aportar alguna mejora.

Especificaciones del proyecto:

- El contenido multimedia que sea reproducido por el cliente debe ser de la mejor calidad posible dado el ancho de banda disponible.
- La continuidad de la reproducción debe ser muy elevada, se deben evitar las interrupciones debidas a amortiguar la insuficiencia de datos y se debe reducir al mínimo la frecuencia de conmutación de las distintas calidades.
- También debe reducirse al mínimo el tiempo transcurrido para iniciar la reproducción del flujo de vídeo.

- Deben reducirse el mínimo el tiempo en el que el cliente debe adaptarse a una configuración de red distinta.

- La compartición de recursos entre dos clientes debe ser equitativa, repartiendo el ancho de banda disponible de manera que ambos dispongan de una calidad parecida y en la menor cantidad de tiempo.

1.3. Plan de trabajo

La realización del Trabajo de Fin de Grado se ha realizado de acuerdo con el plan de trabajo establecido inicialmente en la proposición del proyecto, a excepción de algunas modificaciones realizadas en la revisión crítica. A continuación se detallan a modo resumen los distintos Work Packages, tareas, hitos y el diagrama Gantt (en la Figura 2), estando toda la documentación del plan de trabajo detallada en los documentos mencionados anteriormente.

1.3.1. Work Packages, Tareas e Hitos

WP1: Investigación

WP2: Implementación

WP3: Pruebas

WP4: Mejoras

WP5: Documentación

Tarea e Hitos

WP#	Tarea#	Título corto	Hito/Entregable	Fecha
1	1	Descripción del estándar MPEG-DASH	MPEG-DASH	25/09/15
2	1	Descripción del entorno de trabajo	Entorno	06/10/15
1	2	Análisis de los Segmentadores	Segmentadores	30/10/15
1	3	Análisis de los Clientes	Clientes	30/10/15
2	3	Generador de contenido	Generador	20/11/15
5	1	Revisión Crítica	Revisión Crítica	01/12/15
2	2	Código de los distintos controladores	Controladores	11/12/15
3	1	Evaluación de las pruebas efectuadas	Recopilación de Tests	14/01/16
5	2	Memoria	Memoria	25/01/16

Tabla 1: Tareas e hitos del plan de trabajo

1.3.2. Diagrama Gantt

DIAGRAMA DE GANTT

	Nombre de tarea	Duración	Comienzo	Fin
1	**Proyecto MPEG-DASH**	**115 días**	**mar 01/09/15**	**lun 08/02/16**
2	**Investigación**	**44 días**	**mar 01/09/15**	**vie 30/10/15**
3	MPEG-DASH Theorical Backgroun	19 días	mar 01/09/15	vie 25/09/15
4	Herramientas: Segmentadores	31 días	vie 18/09/15	vie 30/10/15
5	Herramientas: Clientes	31 días	vie 18/09/15	vie 30/10/15
6	**Implementación**	**71 días**	**vie 04/09/15**	**vie 11/12/15**
7	Creación Entorno de Trabajo	23 días	vie 04/09/15	mar 06/10/15
8	Diseño de Controladores	37 días	jue 08/10/15	vie 11/12/15
9	Generador DASH	14 días	mar 03/11/15	vie 20/11/15
10	**Pruebas**	**81 días**	**jue 24/09/15**	**jue 14/01/16**
11	Funcionamiento Entorno	30 días	jue 24/09/15	lun 07/12/15
12	Tests con Segmentadores	23 días	mar 06/10/15	jue 05/11/15
13	Tests con Clientes	22 días	lun 26/10/15	mar 24/11/15
14	Funcionamiento Controladores	28 días	lun 16/11/15	mié 23/12/15
15	Tests Plataforma completa	29 días	lun 07/12/15	jue 14/01/16
16	**Mejoras**	22 días	vie 25/12/15	lun 25/01/16
17	**Documentación**	**85 días**	**mar 29/09/15**	**lun 25/01/16**
18	Propuesta y Plan de Trabajo	5 días	mar 29/09/15	lun 05/10/15
19	Revisión Crítica	41 días	mar 06/10/15	mar 01/12/15
20	Memoria	39 días	mié 02/12/15	lun 25/01/16
21	**Preparación Presentación**	10 días	mar 26/01/16	lun 08/02/16

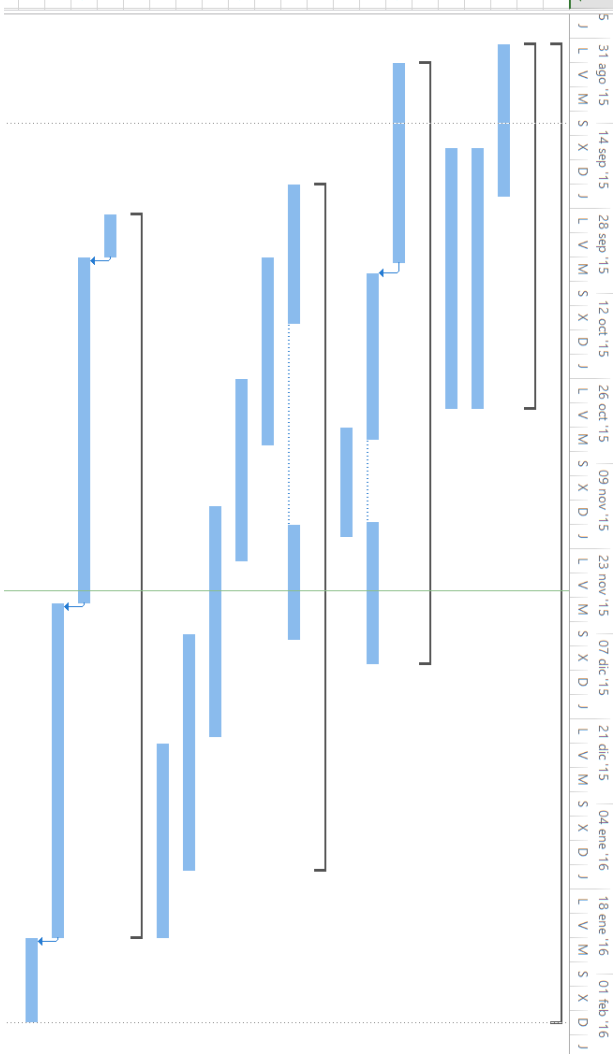

Figura 2: Diagrama Gantt

2. Estado del arte de la tecnología utilizada

El streaming de vídeo ha recorrido un largo camino desde su inicios a principios de los 90. Hoy en día la técnica predominante para transmitir vídeo en vivo y bajo demanda es el streaming adaptativo mediante segmentos por HTTP, las razones que lo permitieron fueron su sencillo despliegue y gestión ya que la infraestructura de HTTP (proxies, cachés y redes de entrega de contenidos, CDN) ya estaba desplegada. Siendo HTTP además un protocolo estándar para Internet.

2.1. Antecedentes

Los métodos de streaming de vídeo han ido evolucionando a lo largo del tiempo, el uso del protocolo UDP [8] para la entrega de vídeo en vivo y protocolos de transporte fiables como TCP [9] para la entrega de vídeos bajo demanda ha ido disminuyendo y actualmente la transmisión tanto de vídeo en vivo como de video bajo demanda (VoD) se realiza principalmente a través de HTTP, por encima del protocolo TCP.

La principal razón del uso de HTTP es la posibilidad de reutilizar su infraestructura, ya que es amigable con los dispositivos NAT, los cuales permiten intercambiar paquetes entre dos redes que asignan mutuamente direcciones incompatibles, y además permite pasar a través de la mayoría de cortafuegos por el puerto 80. Asimismo, como el flujo HTTP es dirigido por el cliente, éste soporta toda la lógica de adaptación, reduciendo las conexiones persistentes servidor-aplicación.

2.2. Adaptive Bitrate Streaming

El Streaming adaptativo es una técnica utilizada para la distribución de flujo multimedia y nace como un avance primordial respecto a la descarga progresiva del flujo a fin de optimizar la experiencia del usuario mediante la distribución del mejor bitrate posible basado en el ancho de banda disponible del usuario final. Para ello se codifica un vídeo en diferentes bitrates y se segmentan en trozos de pocos segundos, permitiendo que cada segmento esté disponible para las distintas velocidades de bits.

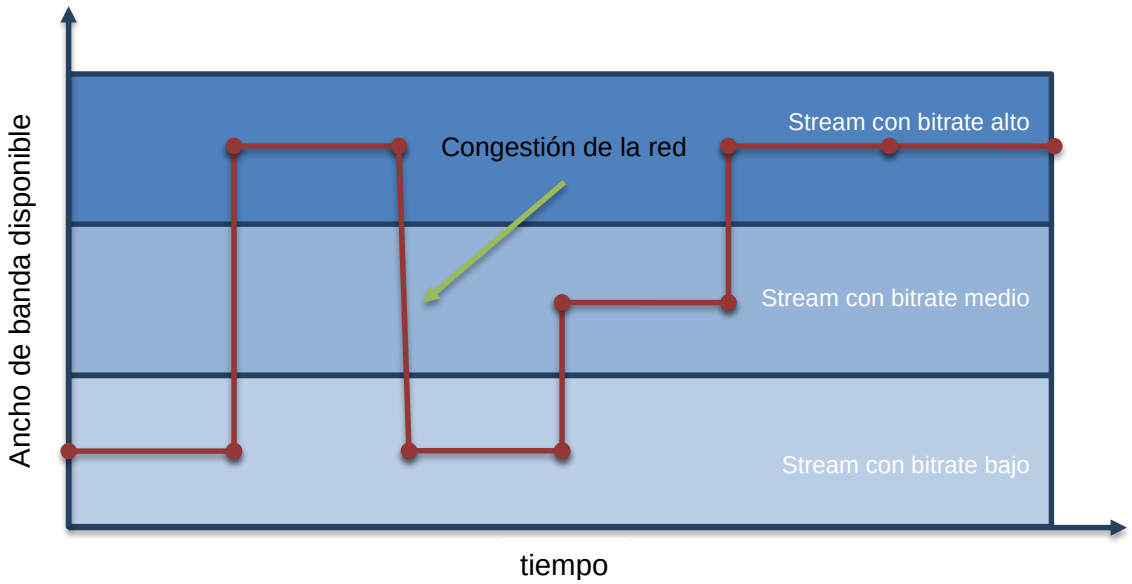

Figura 3: Ejemplo de funcionamiento del streaming adaptativo para 3 calidades

De este modo, durante la reproducción del contenido, el cliente se encargará de adaptarse a las condiciones de red o a la disponibilidad de los recursos, tal y como se puede observar en la Figura 3, solicitando los segmentos de una calidad u otra en función de su algoritmo de adaptación del bitrate (ABR). Esos segmentos serán distribuidos a través de un servidor web o a través de una CDN. En la Figura 4 podemos observar la visión general.

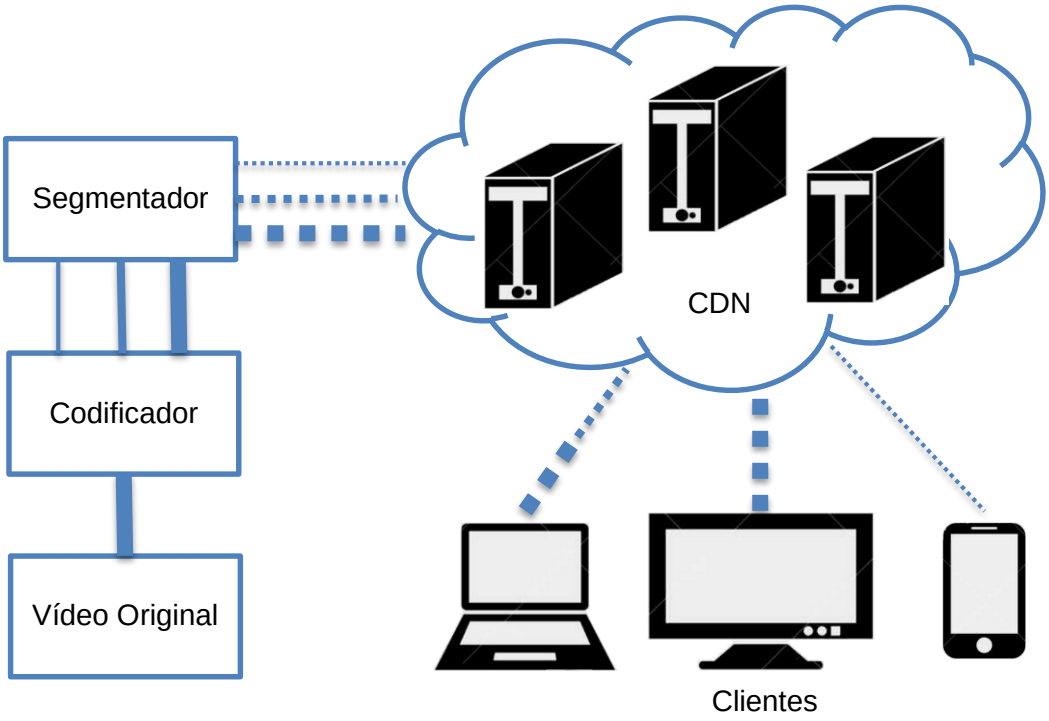

Figura 4: Visión general del streaming con bitrate adaptativo

Como se ha comentado anteriormente, aparecieron varias implementaciones propietarias de streaming adaptativo antes no naciera MPEG-DASH, estas son HTTP Dynamic Streaming (HDS) [10] de Adobe, HTTP Live Streaming (HLS) [11] de Apple y Smooth Streaming [12] de Microsoft. Todas ellas son aún utilizadas y aunque se parecen entre ellas existen bastantes diferencias [13].

2.3. MPEG-DASH

Dynamic Adaptive Streaming over HTTP (DASH) [5] es un estándar para streaming adaptativo sobre HTTP desarrollado por MPEG que se convirtió en estándar internacional en noviembre de 2011 y fue publicado como ISO/IEC 23009-1:2012 en abril 2012. Una segunda edición fue publicada en mayo 2014 como ISO/IEC 23009-1:2014.

MPEG-DASH posee gran cantidad de características, permite contenido tanto en vivo como bajo demanda, múltiples canales de audio para la utilización de distintos lenguajes, puede ser usado en cualquier servidor HTTP estándar sin necesidad de una lógica adicional, permite la inserción eficiente de anuncios, soporte para HTML5 gracias a Media Source Extensions y muchas características más.

El modelo de streaming que define MPEG-DASH ofrece el control exclusivamente al cliente, el cual solicita la información mediante el protocolo HTTP al servidor. Se puede observar dicho modelo en la Figura 5.

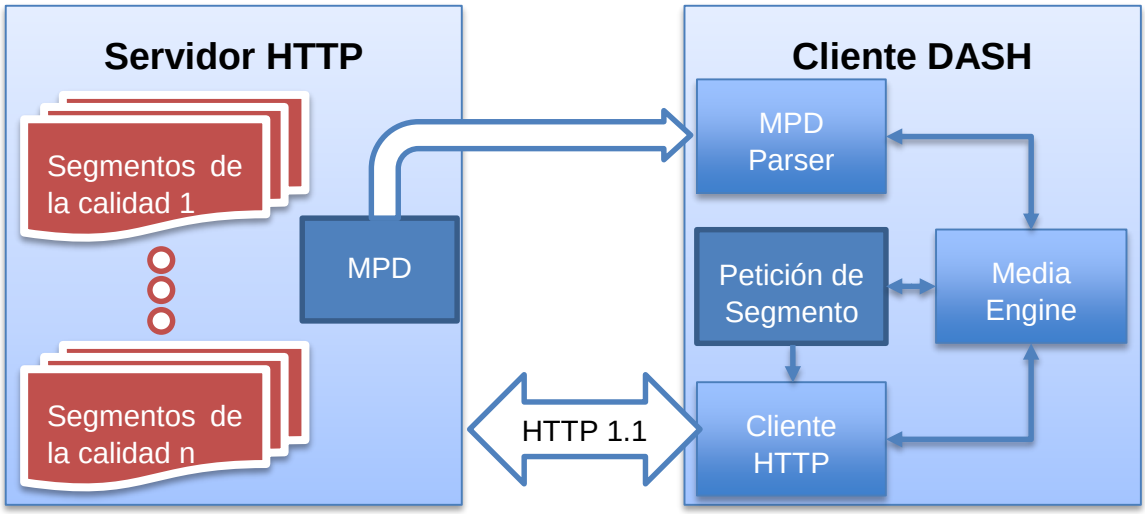

Figura 5: Modelo streaming MPEG-DASH

2.3.1. Media Presentation Description (MPD)

El MPD es un documento XML que contiene todos metadatos necesarios para que un cliente DASH pueda acceder a los segmentos y proporcionar un servicio de streaming al usuario. El estándar MPEG-DAH se centra básicamente en definir la estructura de este documento.

2.3.1.1. Estructura del MPD

El MPD sigue una estructura de datos jerárquica de cuatro capas [14]:

Periods: Describen un contenido o parte de un contenido a través de un intervalo, con una hora de inicio y una duración. Varios períodos se pueden utilizar para separar escenas, capítulos o para separar los anuncios del contenido principal.

Adaptation Sets: Contienen uno o un conjunto de streams multimedia. En el caso más simple un Period podía tener un único Adaptation Set que contuviera todo el vídeo y el audio del contenido. Se utilizaría por ejemplo varios Adaptation Sets para separar el vídeo, los audios en distintos idiomas y los subtítulos.

Representations: Permiten a un Adaptation Set contener el mismo contenido codificado de maneras distintas, en la mayoría de los casos proporcionaran el mismo contenido en múltiples anchos de banda, pero también se puede dar el caso de codificarlo con distintos codecs permitiendo dar soporte a todo tipo de clientes. El intercambio entre representaciones se suele hacer automáticamente por parte del cliente.

Segments: Los segmentos son los archivos que los clientes DASH reproducirán uno tras otro como si formaran un mismo archivo. Los segmentos pueden estar en archivos separados (común en el streaming en vivo) o pueden estar dentro del mismo archivo delimitados por rangos de bytes (común en el streaming bajo demanda).

2.3.1.2. Ejemplo de un MPD

A continuación se presenta a modo de ejemplo un MPD generado con dos representaciones de vídeo de 500 kbit/s y 1500 kbit/s respectivamente y de una representación de audio de 128 kbit/s.

```xml
<?xml version="1.0"?>
<!-- MPD file Generated with GPAC version 0.5.1-DEV-rev5619M -->
<MPD xmlns="urn:mpeg:dash:schema:mpd:2011" minBufferTime="PT1.500000S"
type="static" mediaPresentationDuration="PT0H3M0.00S"
profiles="urn:mpeg:dash:profile:isoff-on-demand:2011">
 <ProgramInformation moreInformationURL="http://gpac.sourceforge.net">
  <Title>/home/bbb/5sec/ondemand/bbb_onDemand_dash.mpd generated by GPAC</Title>
 </ProgramInformation>

 <Period duration="PT0H3M0.00S">
  <AdaptationSet segmentAlignment="true" maxWidth="1920" maxHeight="1080"
maxFrameRate="30" par="16:9" lang="und" subsegmentStartsWithSAP="1">
   <Representation id="1" mimeType="video/mp4" codecs="avc1.4d401f" width="854"
height="480" frameRate="30" sar="1:1" startWithSAP="1" bandwidth="500153">
    <BaseURL>bbb_sunflower_500k_dashinit.mp4</BaseURL>
    <SegmentBase indexRangeExact="true" indexRange="853-1316">
      <Initialization range="0-852"/>
    </SegmentBase>
   </Representation>
   <Representation id="2" mimeType="video/mp4" codecs="avc1.4d401f" width="1280"
height="720" frameRate="30" sar="1:1" startWithSAP="1" bandwidth="1488009">
    <BaseURL>bbb_sunflower_1500k_dashinit.mp4</BaseURL>
    <SegmentBase indexRangeExact="true" indexRange="850-1313">
      <Initialization range="0-849"/>
    </SegmentBase>
   </Representation>
  </AdaptationSet>
  <AdaptationSet segmentAlignment="true" lang="und" subsegmentStartsWithSAP="1">
   <Representation id="3" mimeType="audio/mp4" codecs="mp4a.40.2"
audioSamplingRate="48000" startWithSAP="1" bandwidth="129699">
    <AudioChannelConfiguration
schemeIdUri="urn:mpeg:dash:23003:3:audio_channel_configuration:2011" value="2"/>
    <BaseURL>bbb_sunflower_audio_128k_dashinit.mp4</BaseURL>
    <SegmentBase indexRangeExact="true" indexRange="1130-1605">
      <Initialization range="0-1129"/>
    </SegmentBase>
   </Representation>
  </AdaptationSet>
 </Period>
</MPD>
```


2.4. H.264/Advanced Video Coding

H.264 o MPEG-4 Part 10, Advanced Video Coding (MPEG-4 AVC) es un formato de codificación de vídeo que actualmente es uno de los formatos más utilizados en la grabación, compresión y distribución de contenido. Fue desarrollado por la ITU-T Video Coding Experts Group (VCEG) juntamente con la ISO/IEC JTC1 Moving Picture Experts Group (MPEG).

Se usa típicamente para compresión con pérdidas, aunque éstas a veces son imperceptibles. Nació con el objetivo de crear un estándar capaz de proporcionar una buena calidad de vídeo con el uso de menores bitrates que los estándares previos.

2.5. AAC/Advanced Audio Coding

Advanced Audio Coding (AAC) es un estándar de codificación de audio para la compresión con pérdidas del audio digital. Fue diseñado para ser el substituto del formato MP3, obteniendo generalmente mejor calidad de sonido que MP3 a bitrates similares. AAC fue estandarizado por ISO e IEC como parte de las especificaciones de MPEG-2 y MPEG-4.

3. Metodología / desarrollo del proyecto

Dividiremos el desarrollo del proyecto en tres apartados. Primero vamos a detallar el entorno de trabajo en el que vamos a trabajar, después se hará hincapié en el proceso de generación de contenido DASH, desde el vídeo original hasta conseguir el archivo *manifest* correspondiente. Finalmente hablaremos de los algoritmos de hacen la adaptación del bitrate y de las mejoras propuestas, y definiremos las pruebas a realizar para comprobar si su rendimiento es los suficiente satisfactorio.

3.1. Entorno de trabajo

Para la realización de los correspondientes tests vamos a necesitar un entorno controlado y cerrado que nos permita llevar a cabo la evaluación del algoritmo de control de intercambio de calidades del streaming. El entorno de pruebas lo implementaremos utilizando cuatro máquinas, tal y como se muestra en la Figura 6. La primera será el Host, el cual poseerá distintos clientes capaces de solicitar y reproducir los segmentos del vídeo accesibles desde un servidor HTTP estándar. La segunda y tercera máquina serán dos servidores exactamente iguales, el segundo será un "clon" del primero. Esto nos permitirá testear dos clientes en el Host, ya que cada uno se comunicará con un servidor diferente y así podremos filtrar con comodidad ambos tráficos porque irán asociados a distintas IPs. La última máquina será un conformador de ancho de banda que permitirá emular un escenario WAN y donde podremos ajustar libremente el ancho de banda, el RTT o las pérdidas.

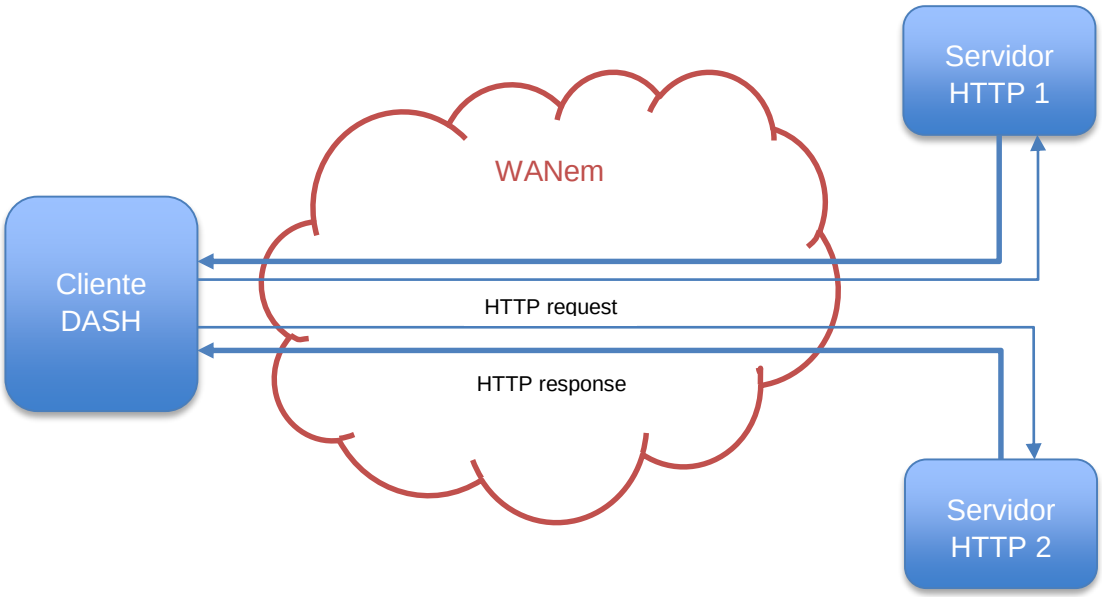

Figura 6: Entorno de pruebas controlado

Para la creación de la plataforma se utilizará un Ubuntu 14.04 LTS que hará de Host. Usaremos Virtualbox [15] para virtualizar las otras 3 máquinas desde el mismo Host, la primera será WANem [16] y las otras dos serán servidores que utilizarán también Ubuntu 14.04 LTS y tendrán instalado un servidor HTTP Apache [17], el cual es un servidor web HTTP de código abierto y multiplataforma, con el fin de poder acceder a los contenidos DASH desde el Host.

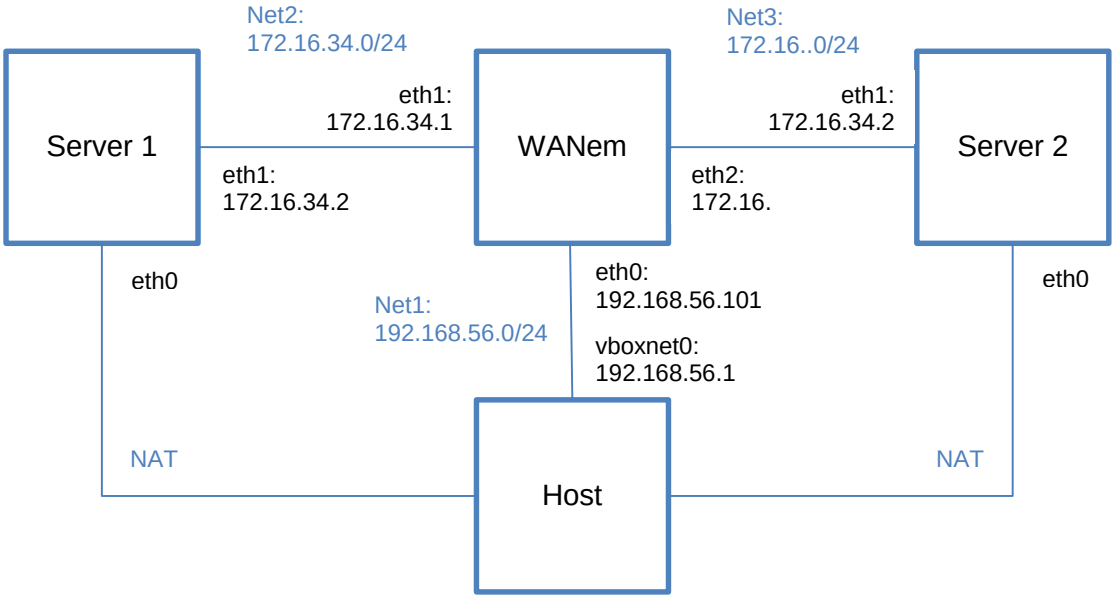

Figura 7: Configuración de red

WANem es un software que corre sobre Knoppix 6.7.1, una distribución basada en Debian, el cual nos servirá para emular redes de área amplia. El objetivo de WANem será emular calidades de enlace específicas para probar el comportamiento de la plataforma de distribución de contenidos DASH. Se pueden configurar con facilidad distintos ajustes de red tales como el ancho de banda, pérdida de paquetes, jitter, latencia...

La configuración de red seguirá el esquema mostrado en la Figura 7 y en el Apéndice A se puede ver con detalle cómo se ha configurado el entorno de trabajo.

3.2. Generación del contenido

La segunda parte de la plataforma consiste en la creación del contenido multimedia. Para ello se precisa de un vídeo en formato raw o, sencillamente, un vídeo ya comprimido que tenga la suficiente calidad como para ser recodificado en distintas versiones del mismo con menor bitrate. En el caso de trabajar con un vídeo raw necesitaremos los archivos de audio y vídeo por separado, por ejemplo:

> **.Y4M** para la secuencia de imágenes: archivo en formato YUV4MPEG donde se almacena una secuencia de frames sin comprimir YCbCr.
>
> **.FLAC** para el audio: archivo de audio comprimido sin pérdidas.

De este modo tendremos que recodificar el vídeo y el audio por separado ya que cada representación sólo puede contener un único componente multimedia según las directrices de DASH-IF [18]. En la Figura 8 se puede observar el esquema general utilizado para generar contenido MPEG-DASH a partir de un vídeo de entrada.

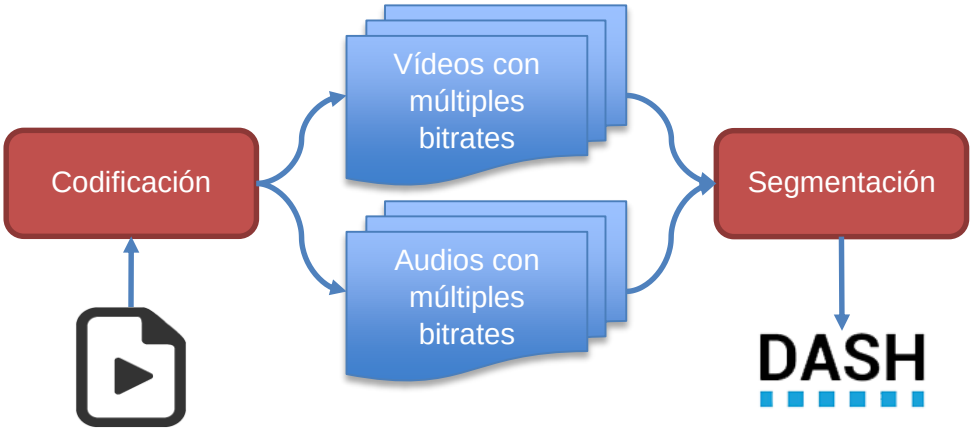

Figura 8: Diagrama general de la generación del contenido

3.2.1. Codificación del vídeo

El primer paso consiste en codificar el vídeo (la secuencia de imágenes) fuente para obtener las distintas representaciones del contenido original, para ello utilizaremos x264 [19], que es una biblioteca libre que permite codificar vídeo en formato H.264/MPEG-4 AVC.

Es importante remarcar que todas las calidades deben tener alineación de key frames. En el lenguaje de codificación, los key frames, Intra-frames o I-frames son aquellas imágenes que pueden ser reconstruidas sin tener ninguna referencia de las otras imágenes del vídeo. Incorporan toda la información acerca de los píxeles de la imagen y por lo tanto son más pesadas y menos frecuentes que los otros tipos de frames (imágenes con codificación predictiva y codificación predictiva bidireccional). El GOP (Group of Pictures) especifica el orden de estos frames. Y el número de imágenes existente entre Intra-frames es conocido como longitud del GOP.

Además, después de la codificación, tendremos que dividir cada uno de los vídeos en segmentos de corta duración, donde cada segmento tendrá que empezar con un I-frame, permitiendo que el decodificador no necesite referencias a frames de segmentos anteriores y por lo tanto cada segmento pueda tener una resolución, un bitrate o un framerate diferente. Tal como se detallará en la sección de segmentación, las longitudes de los segmentos no coincidirán en caso de que los I-frames no estén alineados entre las diferentes calidades y hará imposible el intercambio de calidades.

A continuación se muestra un ejemplo de cómo se codificaría un vídeo con x264 y seguidamente se describirá detalladamente el uso de los distintos parámetros:

```
$ x264 --output outputvideo_BR_k.264 --fps fps --preset slow --bitrate BR --
vbv-maxrate BR --vbv-bufsize 2*BR --min-keyint seg_dur*fps --keyint
seg_dur*fps --no-scenecut --video-filter "resize:width=W,height=H"
inputvideo.y4m
```

Para conseguir la alineación de los key frames se debe especificar debidamente. Utilizamos la opción *keyint* para indicar el máximo tamaño del GOP. Así como *min-keyint* para indicar el tamaño mínimo del GOP y *no-scenecut* para desactivar la inclusión de manera adaptativa de I-frames extra en los cambios de escena, dependiendo de lo diferentes que sean dos frames consecutivos.

Utilizaremos el mismo valor tanto en *keyint* como en *min-keyint* para garantizar que insertamos los I-frames en intervalos regulares en todas las calidades. Para escoger la longitud del GOP se deberá tener en cuenta la longitud de los segmentos a generar a posteriori. Como queremos que cada segmento empiece por un I-frame, para asegurar que el cliente es capaz de cambiar entre las diferentes representaciones en cualquiera de los segmentos, definiremos la longitud del GOP dela siguiente manera, asumiendo una duración del segmento entera mayor o igual a uno:

$$máxima\ longitud\ del\ GOP = framerate * duración\ del\ segmento$$

$$minima\ longitud\ del\ GOP = framerate$$

Por ejemplo, si el framerate es de 30 y se quieren segmentos de 3 segundos, la longitud de GOP deberá ser 30, 45 o 90, o sea, un divisor de la máxima longitud de GOP mayor o igual que la mínima longitud de GOP, asegurando un mínimo de un I-frame por segmento. En la medida de lo posible se utiliza la máxima longitud de GOP para reducir el número de I-frames de la codificación y aprovechar así la redundancia temporal, ya que la duración de los segmentos es de pocos segundos.

Con los argumentos *bitrate*, *vbv-maxrate, vbv-bufsize* controlaremos la tasa de bits por segundo. Haciendo uso del parámetro *bitrate*, x264 realizará la codificación con el objetivo de alcanzar la tasa de bits dada como promedio; *vbv-maxrate* y *vbv-bufsize* permiten al codificador minimizar los efectos de las fluctuaciones del bitrate. Pese a ello, son parámetros relacionados con el buffer del descodificador (cliente), *vbv-maxrate* es la velocidad máxima a la cual dicho buffer puede ser rellenado, es decir, el máximo ancho de banda de la red y *vbv-bufsize* es el tamaño del buffer. La configuración que se utilizará y que es funcional para la mayoría de reproductores de streaming es definir *vbv-maxrate* igual que el *bitrate* y un *vbv-bufsize el doble de grande que vbv-maxrate.* El cociente entre *vbv-maxrate y vbv-bufsize* determina los segundos que el decodificador debe almacenar antes de empezar a reproducir, es decir, 2 segundos.

Con el parámetro *video-filter* indicaremos la resolución, con el parámetro *fps* especificaremos el framerate del fichero creado, el cual será el mismo que en el vídeo original y finalmente con *preset* establecemos el compromiso entre la velocidad de codificación y la calidad obtenida, siendo slow una opción que ofrece tanto calidad como velocidad.

Una vez codificadas las distintas calidades ya podremos segmentarlas, para ello primero vamos a tener que añadir cada uno de los vídeos h264 creados en un contenedor mp4 para que MP4Box sea capaz de segmentarlos:

```
$ MP4Box -add outputvideo_BR_k.264 -fps fps outputvideo_BR_k.mp4
```

Para comprobar la correcta alineación de los segmentos, MP4Box dispone de unos comandos que permiten asegurar que las longitudes entre key frames son las correctas:

```
$ MP4Box -info TRACK_ID outputvideo_BR_k.mp4 2>&1 | grep GOP
```

Este comando devuelve el promedio de las longitudes de los GOPs del fichero. Todas las calidades deberán tener el mismo promedio. Pero incluso así, se podría dar el caso de que dos ficheros tuvieran el mismo promedio y no estuvieran exactamente alineados.

```
$ MP4Box -std -diso outputvideo_BR_k.mp4 2>&1 | grep SyncSampleEntry >
GOP_BR_k.txt
```

Con este obtenemos la lista completa de los índices de los key-fames. Estos índices deberán coincidir para cada una de las calidades, asegurando la alineación al completo.

3.2.2. Codificación del audio

Paralelamente a la codificación del vídeo podemos proceder a re-codificar el audio fuente para obtener las representaciones requeridas. Para ello utilizaremos FFmpeg [20], que es un software libre que produce librerías y programas para manejar datos multimedia. Especialmente haremos uso de su librería libfaac para codificar el audio en formato AAC.

A continuación se muestra un ejemplo de cómo se codificaría un audio con FFmpeg y seguidamente se describirá el uso de los distintos parámetros:

```
$ ffmpeg –i inputaudio.flac –map 0:0 -acodec codec –ac channels –ar fm -b:a
BR outputaudio_BR_k.mp4
```

Utilizamos *acodec* para definir la librería utilizada para la codificación, libfaac, *ac* para especificar los canales de audio, *ar* para indicar la frecuencia de muestreo, *map* para la pista de audio y *b:a* para establecer el bitrate.

3.2.3. Segmentación de las representaciones

Una vez obtenidas todas las representaciones deseadas tanto de audio como de vídeo ya podemos proceder a crear los segmentos para cada una de ellas y el correspondiente MPD. Para ello utilizaremos el programa MP4Box de GPAC [21] de la siguiente manera:

```
$ MP4Box -dash seg_dur –frag seg_dur -rap -frag-rap -segment-name segment_ -
profile PROFILE outputvideo_BR1_k.mp4 … outputvideo_BRN_k.mp4
outputaudio_BR1_k.mp4 … outputaudio_BRN_k.mp4 –out /PATH/video.mpd
```

Con el argumento *dash* indicaremos exactamente la duración de los segmentos en los cuales será dividido el vídeo, la opción *frag* nos permite crear sub-segmentos dentro de los propios segmentos, indicaremos la misma duración que la duración del segmento para obtener sólo un sub-segmento por segmento.

Tal y como se ha mencionado en el apartado de codificación de vídeo, que no existiera alineación de keyframes provocaría que el tamaño de los segmentos variara para las distintas representaciones. El parámetro *rap* y *frag-rap* fuerza a los segmentos a empezar en un Random Acces Point, es decir, en un key frame, pero si estos no están debidamente colocados haría que la duración variara debido a donde los key-frames están en el vídeo, haciendo que las duraciones introducidas en los argumentos *dash* y frag no sea exacta y obteniendo segmentos no alineados entre las calidades.

Otra opción importante es la del *profile*, que especifica el perfil DASH, garantizando la conformidad estableciendo los valores por defecto de la opción seleccionada. Como veremos más adelante algunos de los clientes sólo serán capaces de reproducir algunos profiles en específico.

segment-name asigna un nombre a los segmentos generados. Para el profile onDemand no utilizaremos dicho parámetro con el fin de lograr que los segmentos sean concadenados en un único fichero de salida, para el profile live utilizaremos nombre '$RepresentationID$/video_name_seg_durs_seg', lo que nos permitirá que cada conjunto de segmentos de cada representación se guarde en una carpeta distinta para tenerlo mejor organizado y no se sobrescriban los ficheros.

Una vez segmentado el contenido se puede subir a un servidor HTTP y comprobar que el streaming de vídeo es satisfactorio mediante los distintos clientes.

3.2.4. Automatización y especificaciones

Para conseguir crear contenido MPEG-DASH con facilidad, de manera que a través de un fichero de configuración podamos modificar los distintos parámetros requeridos para la generación, como la duración de los segmentos, el bitrate de las representaciones, la resolución asociada… se desarrolló un software en C++ que automatizara y unificara los distintos procesos, permitiendo obtener contenido ya preparado para subirlo a un servidor HTTP a partir de cualquier vídeo. Dicho programa está basado en DASHEncoder [22], una herramienta de mismo uso que quedó obsoleta al sólo recibir soporte para las versiones de MP4Box más antiguas.

3.2.4.1. Archivo de configuración

El programa hace uso de un fichero de configuración llamado *DASHGen.config* donde el usuario puede modificar los distintos parámetros para adaptarse a las distintas necesidades. Se utiliza clase *AnyOption* desarrollada por hackorama [23], que facilita la lectura de las opciones del programa tanto por la línea de comandos como por un fichero de configuración con formato de valores par, usando ":" como elemento separador (Opción : Valor).

Los parámetros indicados en el archivo de configuración van desde el nombre del vídeo original y el directorio de destino hasta la duración de los segmentos, permitiendo así crear contenido para distintas duraciones al mismo tiempo. En la Tabla 2 de detallan todas las variables del archivo de configuración.

Parámetro	Descripción
Directorio de entrada	Ubicación donde se encuentran el/los ficheros de entrada
Directorio de salida	Será donde se almacene todo el contenido generado
Archivo de vídeo	Nombre del archivo de vídeo original
Archivo de audio	Nombre del archivo de audio original, en caso de que el audio se encuentre dentro del mismo archivo que el vídeo, pero en otra pista, se debe repetir el nombre
Framerate	Numero de imágenes por segundo del vídeo original Ejemplo, fps : 30

Canales de audio	Número de canales del audio, estéreo = 2
Frecuencia de muestreo	Número de muestras por segundo del audio en Hz Ejemplo: 48000
Pista de audio	Para evitar problemas durante la codificación del audio especificamos en que pista se encuentra el audio a codificar. Ejemplos: pista = 0:0 (Para archivos FLAC) pista = 0:1, 0:2, etc (Para archivos MP4, donde puede haber distintas pistas de audio, siendo 0:0 la de vídeo)
Calidades de vídeo	Se indica el bitrate en kbit/s y la resolución en píxeles de ancho y alto de cada una de las calidades Ejemplo de 6 niveles de calidad: bitrate : 50,200,500,1200,2500,5000 width : 320,640,960,1280,1280,1920 height : 180,360,540,720,720,1080
Duración de los segmentos	Se indica la duración de los segmentos en segundos. Se generará contenido DASH para cada una de las duraciones. Deben estar ordenados de menor a mayor. Ejemplo con segmentos de 2 y 10 segundos: Ejemplo, segment_dur : 2,10
Flag	Variable utilizada para controlar el funcionamiento del programa, permitiendo ejecutar o no el largo proceso que se va a realizar a continuación. Flag : 0 (No ejecuta, sólo printa las ordenes) Flag : 1 (Sí se ejecuta)

Tabla 2: Parámetros del fichero de configuración

3.2.4.2. Funcionamiento

El programa engloba los procesos descritos en las secciones anteriores. Utiliza un fichero de configuración para personalizar el contenido, x264 para codificar el vídeo, FFmpeg para

codificar el audio, MP4Box para segmentar las representaciones. Además hace uso de los comandos Unix mkdir y sed. El primero para crear los nuevos directorios y el segundo para hacer una modificación en el MPD resultante del profile *live*.

El la Figura 9 se muestra un esquema general del funcionamiento del programa y en el Apéndice B se muestra el código al completo.

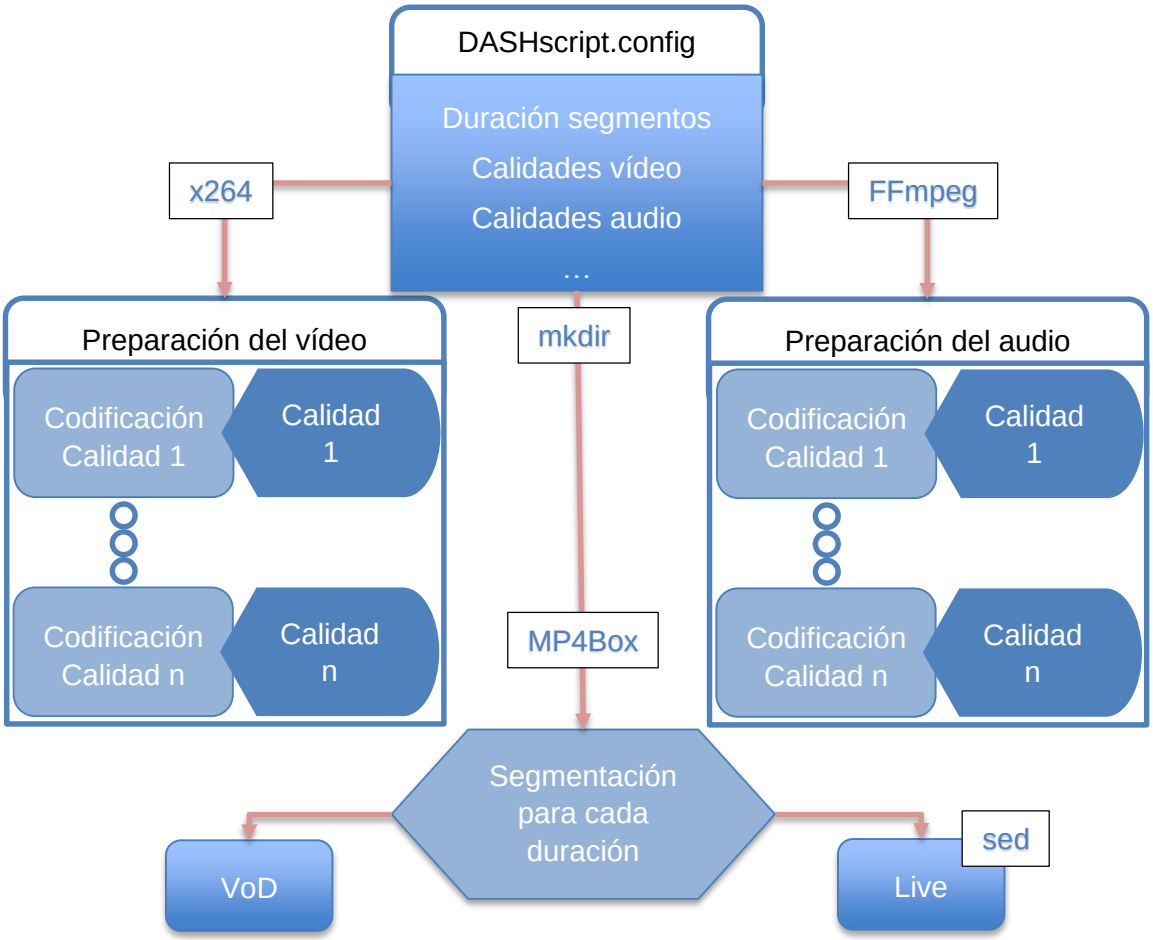

Figura 9: Diagrama Programa

En el caso de que se quiera generar segmentos de varias duraciones a la vez, el programa calcula el máximo común divisor de la máxima longitud del GOP de cada una de las duraciones. Eso garantiza la alineación de los key frames haciendo falta codificar el contenido una sola vez, en caso contrario haría falta codificar el contenido específicamente para cada una de las duraciones de segmento (podemos hacerlo igualmente ejecutando el programa varias veces, variando la duración del segmento en el archivo de configuración). Como queremos generar un contenido generoso, tanto en calidades como en duraciones, esta característica nos ahorrará tiempo y espacio.

3.2.4.3. Especificaciones

Una vez expuesto como se genera el contenido y habiendo desarrollado la herramienta que nos simplifica el trabajo podemos definir exactamente el contenido DASH a generar para la realización de las pruebas. En este proyecto se hace uso de 21 representaciones debido a que necesitamos las suficientes como para poder realizar un estudio adecuado

de los algoritmos de adaptación del bitrate de los clientes. Con un gran número de representaciones conseguimos cubrir una gran variedad de situaciones. Cada una de las calidades tendrá un bitrate y una resolución asociados tal y como podemos ver en la Tabla 3 y en la Figura 10.

Nivel de calidad	Bitrate BR (kbp/s)	Resolución WxH (px)	Relación de aspecto
0	50	320x240	4:3
1	100	320x240	4:3
2	150	320x240	4:3
3	200	480x360	4:3
4	250	480x360	4:3
5	300	480x360	4:3
6	400	480x360	4:3
7	500	854x480	16:9
8	600	854x480	16:9
9	700	854x480	16:9
10	900	854x480	16:9
11	1200	1280x720	16:9
12	1500	1280x720	16:9
13	2000	1280x720	16:9
14	2500	1280x720	16:9
15	3000	1920x1080	16:9
16	4000	1920x1080	16:9
17	5000	1920x1080	16:9
18	6000	1920x1080	16:9
19	7000	1920x1080	16:9
20	8000	1920x1080	16:9

Tabla 3: Representaciones del vídeo con bitrates y resoluciones asociados

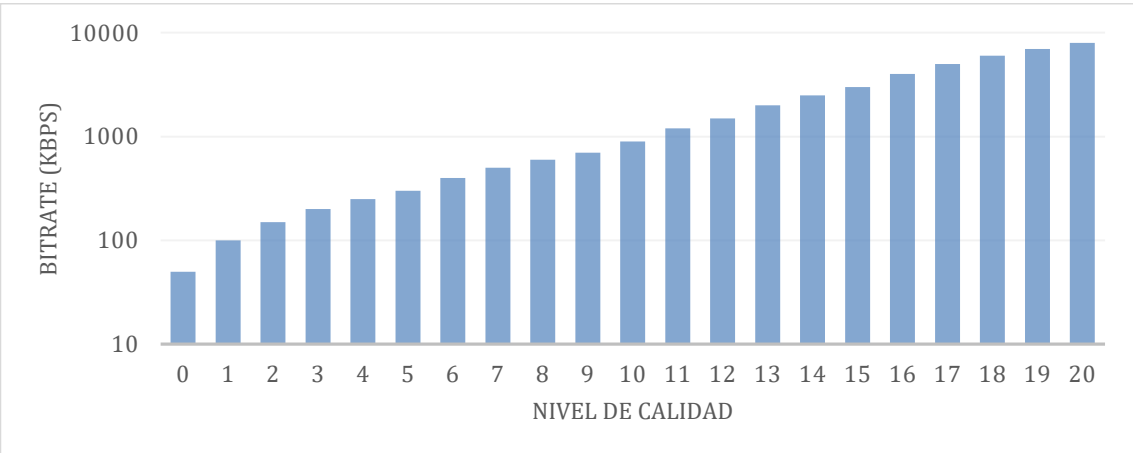

Figura 10: Niveles de calidad del vídeo en escala logarítmica

En el caso del streaming de vídeo, el audio no es un factor tan determínate como lo es el vídeo ya que ocupa un tamaño mucho menor, es por ese motivo que no tiene demasiado sentido crear muchas representaciones para el audio. En este proyecto trabajaremos con cuatro representaciones de audio tal y como se ve en la Tabla 4 y la Figura 11:

Nivel de calidad	Bitrate BR (kbp/s)	Frecuencia (kHz)
0	32	48
1	64	48
2	128	48
3	192	48

Tabla 4: Representaciones del audio

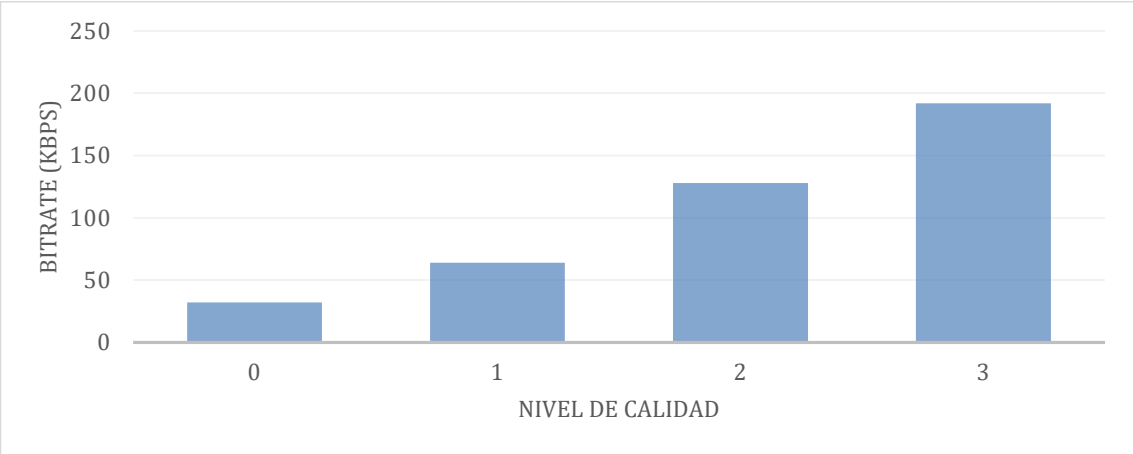

Figura 11: Niveles de calidad del audio

Definiremos 3 duraciones de segmento para poder comprobar cómo afecta el parámetro en las pruebas: 1, 5 y 15 segundos, siendo las duraciones de 1 y 15 segundos las duraciones límite de las principales datasets de MPEG-DASH [24].

Vamos a generar contenido DASH para dos de los profiles que admite el estándar: *onDemand* para vídeos VOD y *live* para vídeos en vivo, pese a que este último no sea un caso real al no realizar el proceso de compresión y segmentación en vivo mientras se va generando el vídeo.

3.3. Algoritmos de adaptación del bitrate

3.3.1. Propósito y funcionamiento genérico

El controlador es el componente central de un sistema de vídeo streaming adaptativo. Su objetivo es decidir la calidad del vídeo entre todas las que ofrece el archivo *manifest*, basándose en la retroalimentación de diversos factores como pueden ser el ancho de banda estimado, la longitud del buffer de reproducción o el estado Buffering/Steady de la transmisión, que se explica a continuación.

El algoritmo debe ser capaz de cambiar a una calidad inferior cuando el tiempo que demora la descarga es superior a la duración del segmento, es decir, la velocidad de descarga

debe ser mayor que la velocidad de reproducción, de lo contrario el buffer del cliente podría vaciarse eventualmente produciendo una pausa en la reproducción.

Por lo tanto, como la velocidad de la descarga de un segmento deberá ser superior a la velocidad de reproducción del mismo, una vez el buffer del cliente se llene con los segmentos descargados pasado un tiempo, se creará un patrón on-off, donde el cliente deberá posponer la descarga del siguiente segmento hasta que uno de los segmentos en la cola sea consumido por el reproductor, tal como se ilustra en la Figura 12.

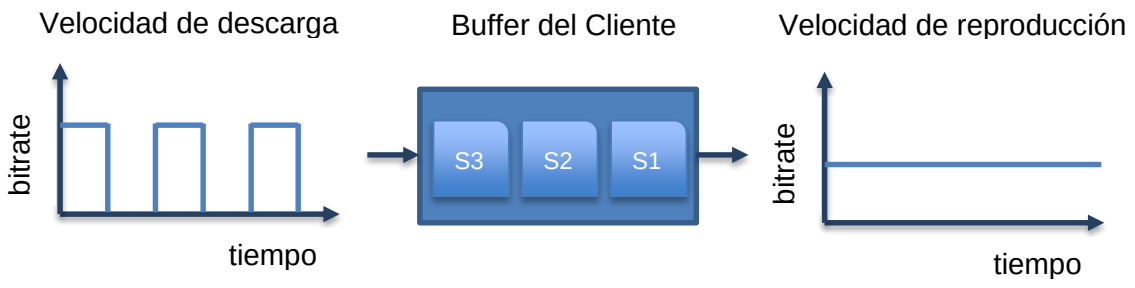

Figura 12: Patrón on-off de la velocidad de descarga

La longitud de estos períodos on y off se rige por la velocidad de reproducción, que es constante, la velocidad de la descarga y de la disponibilidad del segmento en el caso de transmisión en vivo. Esto da como resultado unos períodos de on-off casi constantes a menos que el ancho de banda o el bitrate cambien. En caso de transmisión en vivo se solicita un nuevo segmento tan pronto como esté disponible. A pesar de que, en general, las solicitudes de los clientes toman tiempo diferente para llegar al servidor debido a, por ejemplo, diferentes RTT.

El estado Buffering es aquel donde el cliente pide un nuevo segmento justo después de haber descargado el anterior con el fin de rellenar rápidamente el buffer del reproductor. En el estado Steady, en cambio, el cliente debe esperarse a descargar un nuevo segmento debido a que el buffer ya está lleno, produciendo los períodos off o de inactividad de descarga. En la Figura 13 se puede observar la comunicación entre cliente y servidor durante ambos estados.

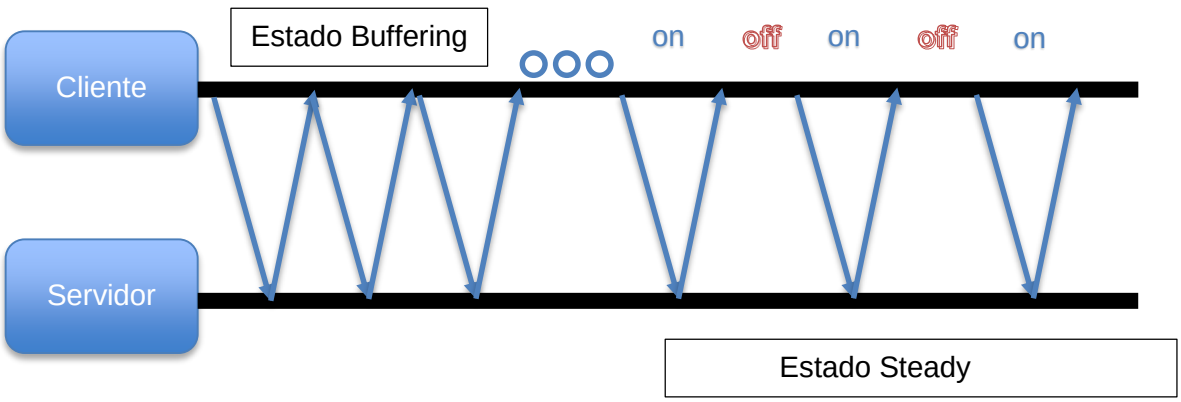

Figura 13: Solicitud-Respuesta entre servidor y cliente en los estados Buffering y Steady

3.3.2. Clientes estudiados

La primera idea al comenzar el proyecto era hacer uso del reproductor VLC [25] y de la librería libdash [26], la cual contiene un reproductor, para testear el uso de la plataforma. VLC se descartó debido a que utilizaba un plugin con una sintaxis antigua de MPD, haciendo que el contenido creado por MP4Box no fuera reproducible, además de no ofrecer un registro detallado del cambio de calidades. La idea con libdash era aprovechar su reproductor, el cual permite el intercambio de calidades manual, para desarrollar un algoritmo de intercambio de calidades al completo. Se descartó debido a la dificultad que suponía estudiar toda su densa librería en C++ y que hacía tiempo que no recibía soporte.

Se prefirió buscar algún cliente que nos facilitara el estudio del cambio de calidades y se optó por dos opciones. La primera fue el reproductor de GPAC, MP4Client/Osmo4, ya que implementa un algoritmo de adaptación de calidades en una sola función en C, permitiendo que el estudio sólo se centrara en una pequeña parte del código del reproductor.

La segunda fue TAPAS [27], la cual es una herramienta desarrollada en Python que permite el prototipado rápido y la evaluación experimental de los algoritmos de control del streaming adaptativo. Además implementa un algoritmo convencional, que será el que estudiaremos. Pese a todo, esta herramienta no utiliza las representaciones de audio y sólo permite reproducir contenido MPEG-DASH VOD.

También se utilizó *dash.js* [28], ya que permite reproducir contenido DASH muy fácilmente mediante un navegador. Está programado en JavaScript y hace uso de las Media Source Extensions, pero finalmente se descartó el estudio de su algoritmo de adaptación debido a su similitud con los otros dos clientes.

3.3.2.1. TAPAS

TAPAS define una clase *BaseController* que debe ser heredada para cualquier controlador que se quiera implementar. Este controlador deberá constar de 2 métodos obligatorios, *caclControlAction()* e *isBuffering()*.

calcControlAction() debe retornar el ancho de banda estimado y será llamado cada vez que se descargue un segmento para determinar la calidad del siguiente. Para ello, justo antes de llamar la función se actualizan los diccionarios de datos con la información obtenida. El controlador convencional que implementa TAPAS define la estimación del ancho de banda de la siguiente manera:

$$r_{K-1} = bitrate\ asociado\ al\ último\ segmento\ descargado$$

$$t = duración\ del\ segmento\ en\ segundos\ (s)$$

$$T_{K-1} = tiempo\ empleado\ para\ descargar\ el\ último\ segmento\ \ (s)$$

$$x_k = \frac{t * r_{K-1}}{T_{K-1}}$$

$$estimación\ del\ ancho\ de\ banda = bwe_k = EWMA(x)$$

$$= bwe_{k-1} - T_{k-1} * a(bwe_{k-1} - x_k)\ para\ a = 0.2$$

Donde EWMA, media móvil ponderada exponencialmente, es un tipo de filtro IIR que aplica factores de ponderación que disminuyen exponencialmente.

Esta estimación del ancho de banda pasará a través de una función de cuantificación, la cual nos retornará el índice del nivel de calidad del próximo segmento a descargar. Esta función sólo compara el ancho de banda estimado con el diccionario de bitrates de las representaciones y escoge el nivel justo inferior a dicha estimación.

Durante la invocación del método deberemos definir el tiempo de inactividad entre la última descarga y la siguiente mediante la función *setIdleDuration(), es decir,* el tiempo que debe de transcurrir entre la descarga de dos segmentos consecutivos durante el estado Steady. En esta implementación se define el tiempo de inactividad de la siguiente forma:

$$IdleDuration = t - T_{k-1}$$

O sea, la diferencia existente entre la duración del segmento y el tiempo empleado para descargarlo.

isBuffering() es el otro método obligatorio del controlador de TAPAS, éste retornará *True* si se considera que el sistema está en estado de Buffering, o sea que todavía tiene espacio para almacenar segmentos. En el estado de Buffering el período de inactividad es igual a cero. En TAPAS la función devuelve *True* si la longitud buffer de reproducción está por debajo de 15 segundos, obteniendo la longitud del buffer mediante el framework multimedia GStreamer [29] para Python a través de la función *get_property.*

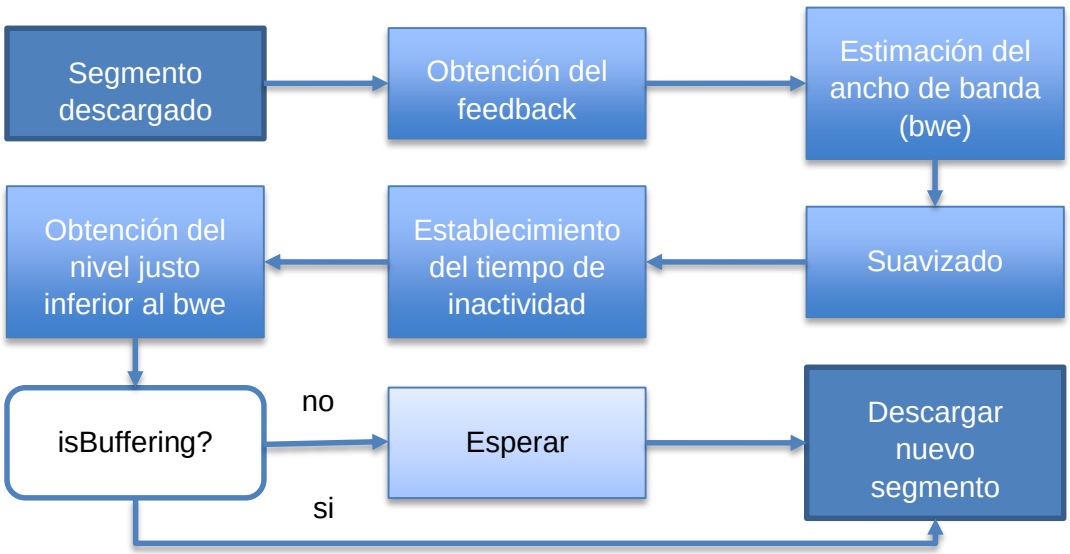

Figura 14: Diagrama de flujo general del algoritmo para TAPAS

3.3.2.2. MP4Client

El algoritmo de control del bitrate de MP4client se encuentra en archivo *dash_client.c,* exactamente en la función *dash_do_rate_adaptation().*

A diferencia que con TAPAS, aquí se utiliza el directamente el rate de la descarga para estimar el ancho de banda, gracias al cálculo del tiempo que se ha demorado la descarga y a su tamaño en bytes.

Si el rate de la descarga, *dl_rate,* es mayor que el bitrate de la representación actual, se define la variable *go_up* como *True.* A partir de este punto se compara de manera recursiva el valor de dl_rate con los bitrates de cada una de las representaciones disponibles en el MPD, empezando por la primera.

La nueva representación seleccionada será aquella justo por debajo del valor de *dl_rate*. Además, para evitar fluctuaciones, se comprueba si la nueva representación escogida es la inmediatamente superior, haciendo uso de go_up y del contador de representaciones subidas. En caso de serlo, no se cambiará la calidad durante ese segmento y en el caso de que el siguiente vuelva a subir un nivel de calidad sí que se realizará la adaptación.

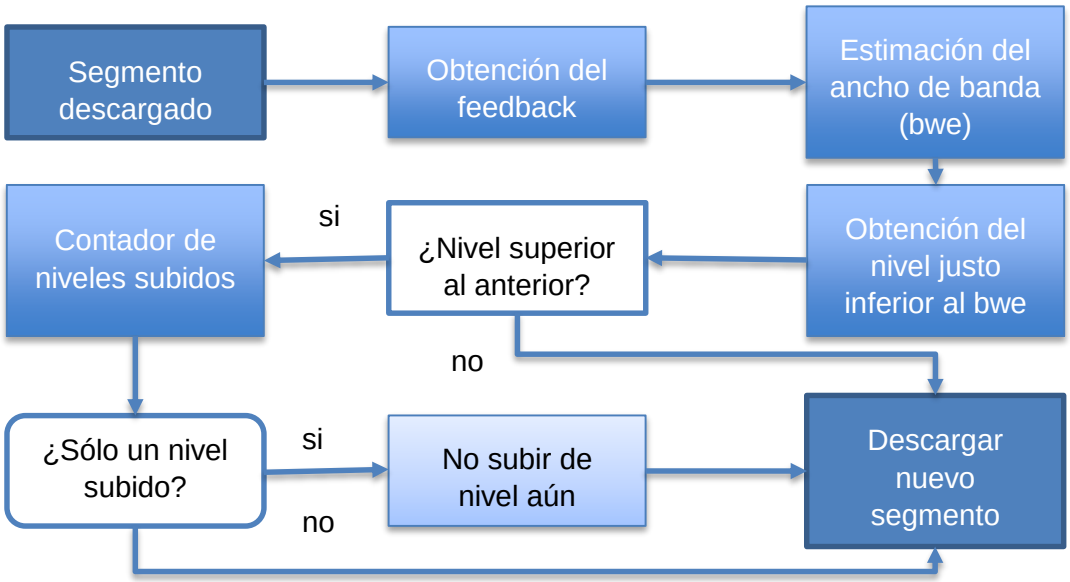

Figura 15: Diagrama de flujo general del algoritmo para MP4Client

3.3.2.3. Aportaciones

Se harán un par de variaciones el controlador que implementa TAPAS para comprobar si su funcionamiento es satisfactorio. La primera de ellas será un cambio en la parte de la estimación del bitrate, prescindiendo de toda la parte de suavizado mediante el filtro y calculando el ancho de banda sencillamente como se hace en MP4Client. Tal y como lo realiza TAPAS se asume un tamaño de segmento constante para todos los segmentos de una misma calidad, y no es así, sobre todo para segmentos pequeños.

La segunda será un control del tamaño del buffer, permitiendo así que según si el buffer está muy lleno o muy poco lleno se realice un cambio de calidad al nivel inferior o superior.

Con la primera mejora esperamos mejorar el tiempo que el reproductor tarda en alcanzar la máxima calidad posible, ya que debido al suavizado este tiempo es demasiado grande. Pese a ello perderemos las ventajas que nos ofrece dicho suavizado y hará el intercambio de calidades quizás demasiado variante si las condiciones cambian en exceso.

Con la segunda esperemos mejorar el rendimiento en los casos donde el buffer esté al límite.

No se aportó ninguna modificación para MP4Client debido a que la función no da mucho margen de maniobra y el resto de parámetros como por ejemplo el buffer se controlan a lo largo de las 6000 líneas de código de *dash_client.c,* haciendo muy difícil modificar alguna cosa.

4. Resultados

4.1. Definición de las pruebas

Definimos 3 pruebas:

Prueba tiempo de respuesta: Se calculará el tiempo que tarda el algoritmo en subir a la calidad por debajo a la máxima calidad teórica posible que permite el ancho de banda de la configuración de red. También se calculará el tiempo que tarda en bajar a la calidad adecuada una vez el ancho de banda se reduce significativamente, a un tercio. Queremos que estos tiempos sean lo más pequeños posibles y una vez descargado el primer segmento ya deberíamos ser capaces de estimar el ancho de banda disponible.

Prueba con pérdidas/RTT: Se trata de comprobar cómo afecta al intercambio de calidades la introducción de pérdidas y de RTT (tiempo que tarda un paquete en volver al emisor habiendo pasado por el receptor) a la conexión, teóricamente se deberá cumplir la ecuación de Mathis. En el caso de esta prueba utilizaremos 4% de pérdidas y 250 ms de RTT en la configuración de la red. Obteniendo un throughput máximo de 279,4 kbps.

$$Th <= \frac{MSS}{RTT}\sqrt{\frac{1{,}5}{p}} = \frac{1460 * 8}{250ms}\sqrt{\frac{1{,}5}{0{,}04}} = 286100\ bps = 279{,}4\ kbps$$

$$MSS\ (Tamaño\ Máximo\ de\ Segmento) = MTU - header\ TCP - header\ IP = 1460 * 8\ bits$$

- $MTU\ =\ 1500\ bytes$
- $IP\ header\ =\ 20 - 60\ bytes$
- $TCP\ header\ =\ 20 - 60\ bytes$

Prueba Amigabilidad: Se medirá el tiempo que tardan dos clientes en compartir equitativamente el ancho de banda una vez un segundo cliente se pone en marcha después de que el primero lleve ya un tiempo activo. Se comprobará si realmente existe un reparto equitativo de los recursos de la red.

Durante la realización de todas las pruebas se requerirán las mínimas interrupciones, mínimos cambios de calidad (molestos) y máxima calidad posible.

4.2. Definiciones de las condiciones de red para las pruebas

Configuración de WANem para las pruebas 3 pruebas:

# prueba	Nombre de la conexión	Ancho de banda (Mbit/s)	RTT (ms)	Pérdidas (%)
1,3	Standard ADSL downstream	6,144	100	0
1	E-1, DS-1 Europe	2,048	100	0
2	E-2 Europe	8,448	250	4

Tabla 5: Configuraciones de red asignadas a cada prueba

4.3. TAPAS

Prueba tiempo de respuesta:

Inicializamos el vídeo y calculamos el tiempo que transcurre hasta que se llega a la calidad por debajo de la óptima, pasados 40 segundos reducimos el ancho de banda a un tercio y calculamos nuevamente el tiempo transcurrido hasta alcanzar la calidad por debajo de la óptima.

Debido a que el ancho de banda de la configuración de red elegida es de 6,144 Mbps, la calidad por debajo de la óptima (6000 kbps) será la de 5000 kbps, es decir, la calidad 17. Para la configuración de red de 2048 kbps, la calidad por debajo de la óptima (2000 kbps) será la de 1500kbps, es decir, la 12.

Primero compararemos los resultados que se obtienen antes y después de la realización de las modificaciones en el código, ya que el motivo principal del cambio era conseguir un mejor resultado en la prueba del tiempo de respuesta.

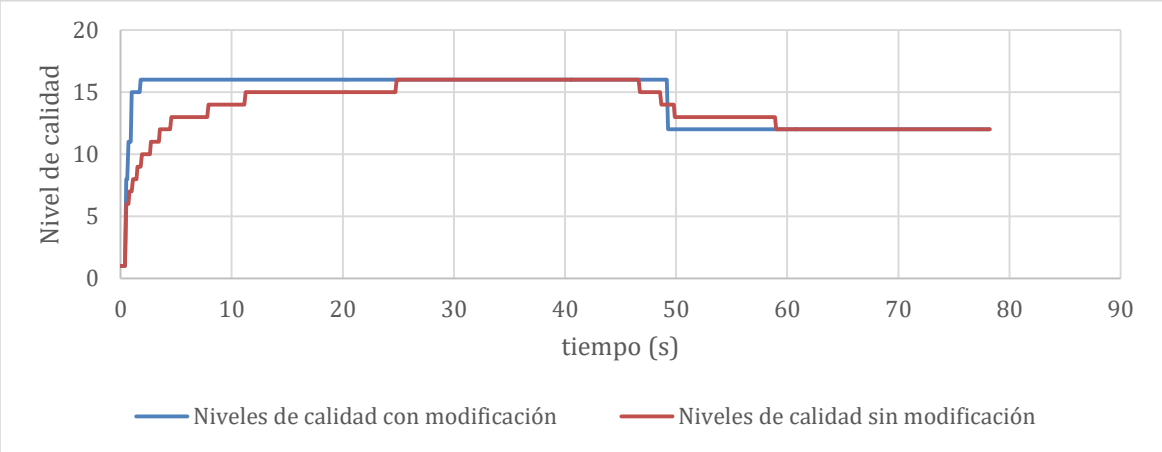

Figura 16: Comparación de los niveles de calidad para la 1ª prueba, segmentos de 1 segundo, utilizando 2 formas de estimar el ancho de banda

Tal y como se puede observar el suavizado del ancho de banda es perjudicial en el caso de desear una rápida adaptación a los cambios de red. Se tardan 22 segundos más en alcanzar el nivel de calidad 16 y 9 segundos más en alcanzar la calidad 12. Cuando mayores son los segmentos menos diferencia existe entre ambos resultados.

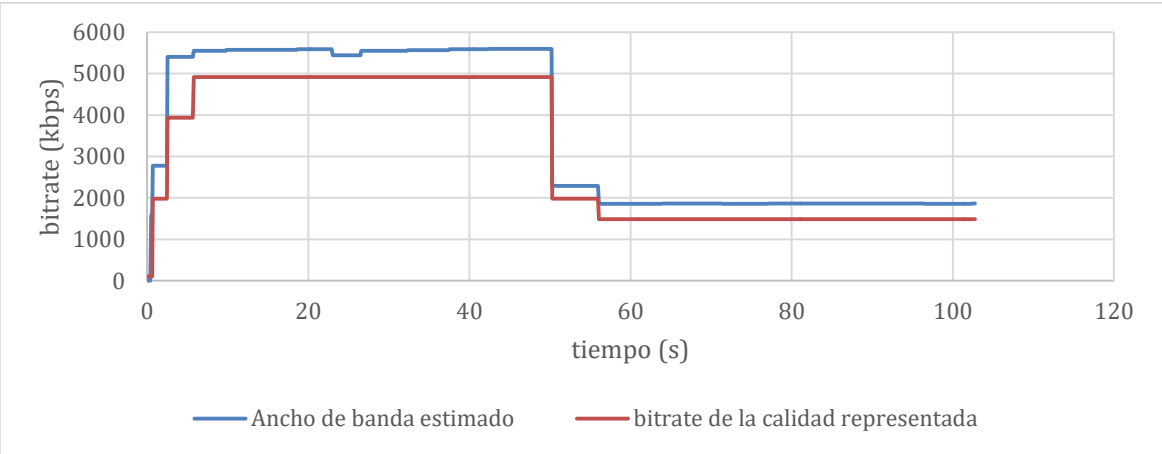

Figura 17: Ancho de banda estimado y bitrate de la calidad para la 1ª prueba, segmentos de 5 segundos

Utilizando segmentos de 5 segundos, el tiempo requerido en llegar a la calidad 17 es de casi 6 segundos, mientras que el tiempo que se requiere para que la calidad se estabilice en 1500 kbps, una vez realizado el cambio en el segundo 40, es de 15 segundos. Se puede observar que la estimación del ancho de banda es mejor cuando mayor es el segmento. Esto provoca que con segmentos de 1 segundo no se alcance la calidad 17.

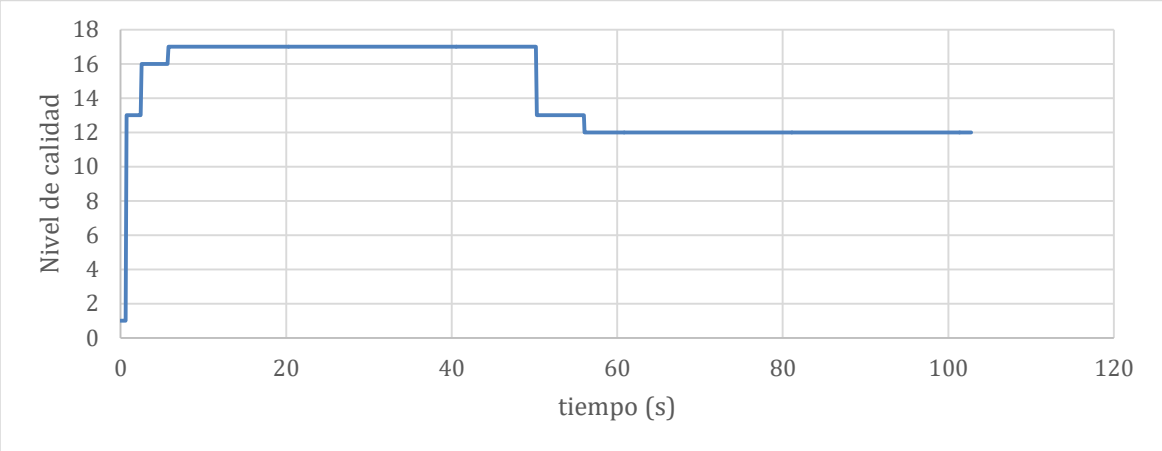

Figura 18: Niveles de calidad para la 1ª prueba, segmentos de 5 segundos

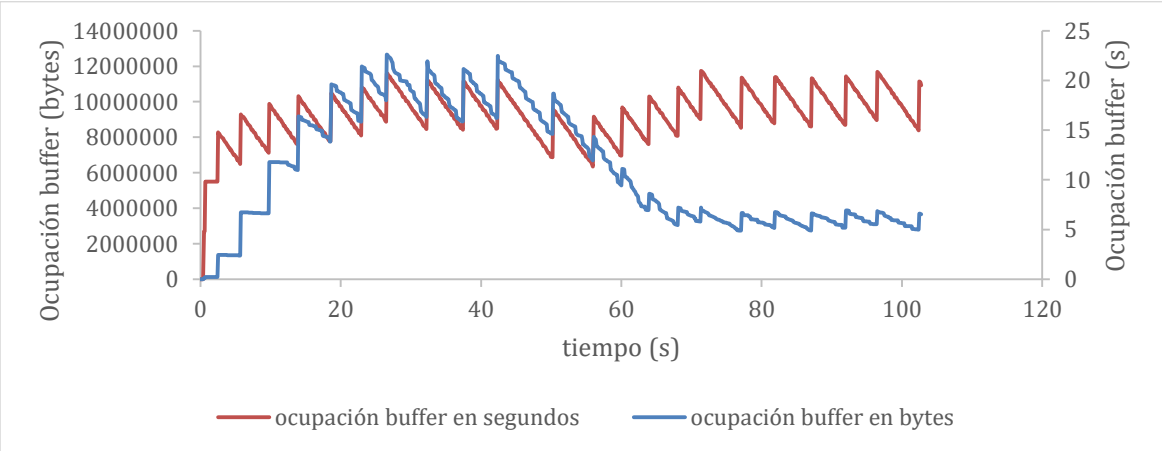

Figura 19: Ocupación del buffer para la 1ª prueba, segmentos de 5 segundos

En la figura del estado del buffer se puede observar los períodos Steady, donde el buffer debe consumir un segmento antes de poder volver a ser rellenado cuando se superan los 15 segundos de almacenaje.

En esta prueba el cliente TAPAS no es funcional para segmentos de 15 segundos debido a que el tamaño de los segmentos es demasiado grande para adaptarse a un cambio brusco del ancho de banda, provocando que el buffer se vacíe antes de que finalice la descarga del segmento y por lo tanto se interrumpa la reproducción.

<u>Prueba con pérdidas:</u>

Esta prueba consiste en verificar si se consigue satisfacer la ecuación de Mathis para las distintas duraciones de segmento. Pese a tener un ancho de banda de 8,448 Mbps, que sería capaz de satisfacer la máxima calidad, no deberíamos ser capaces de obtener una calidad superior a la que ofrece 300 kbps ya que el throughput máximo será de 279,4 kbps.

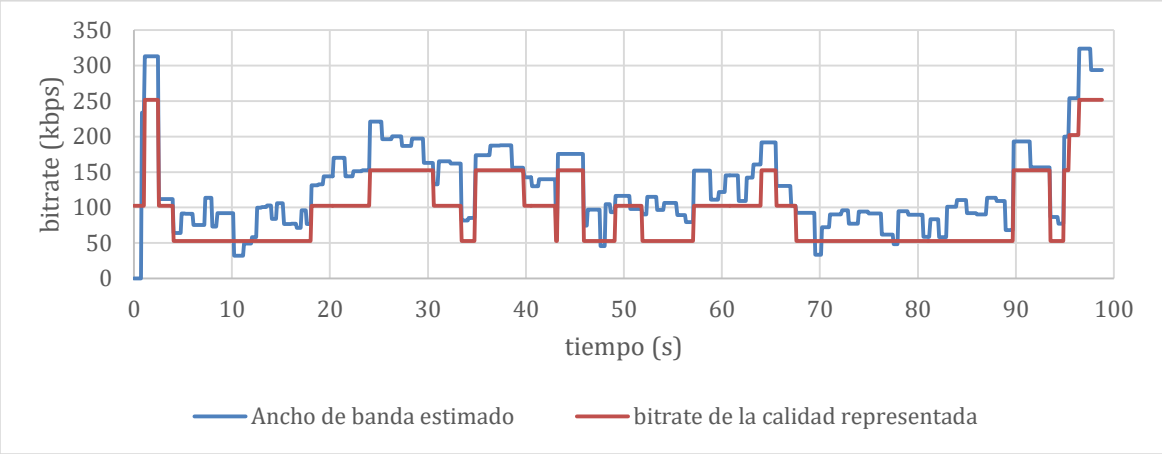

Figura 20: Ancho de banda estimado y bitrate de la calidad para la 2ª prueba, segmentos de 1s

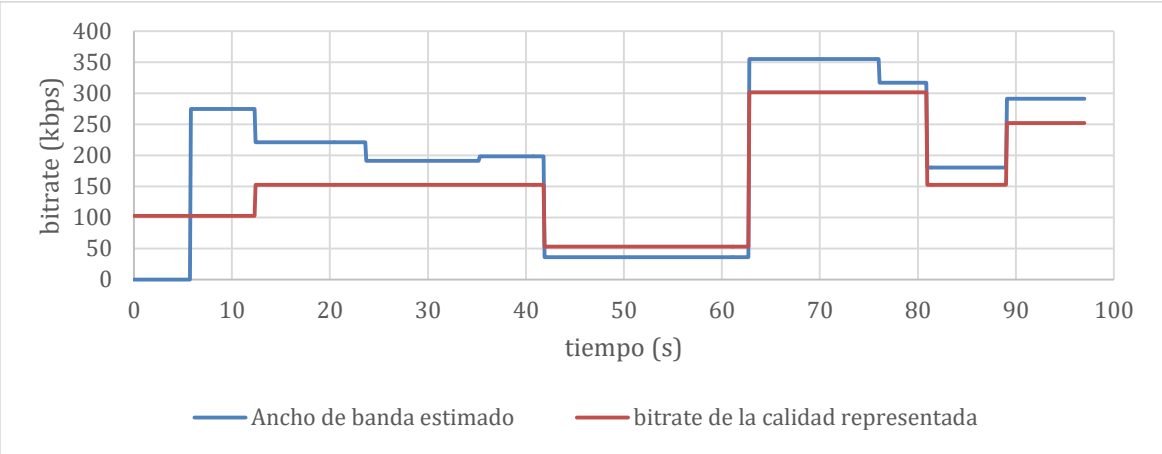

Figura 21: Ancho de banda estimado y bitrate de la calidad para la 2ª prueba, segmentos de 15s

Observamos para todas las 3 duraciones de segmento se verifica la ecuación, pese a que en algún punto se alcanzan los 400 kbps en ancho de banda estimado (para segmentos de 5 segundos) y los 300 kbps en la calidad representada. En caso de pérdidas y RTT el ancho de banda queda limitado y depende de éstos.

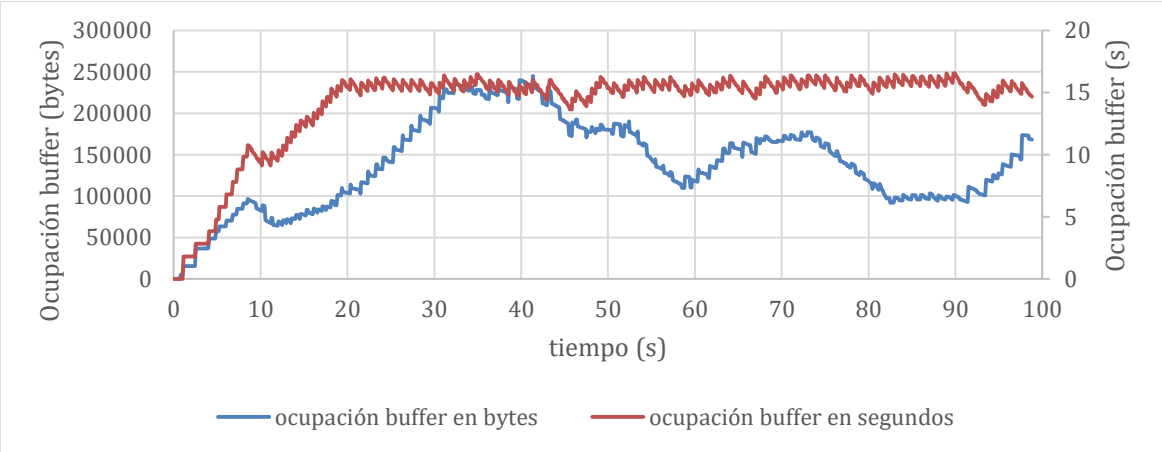

Figura 22: Ocupación del buffer para la 2ª prueba, segmentos de 1 segundo

Prueba amigabilidad:

Utilizando la configuración de red con un ancho de banda de 6,144 Mbps y un RTT de 100 ms realizamos la prueba de amigabilidad con TAPAS utilizando segmentos de 1 y 5 segundos, descartando los segmentos de 15 segundos por el mimo motivo que en la primera prueba. El primer cliente empezará un streaming de vídeo con el primer servidor y pasados 70 segundos un segundo cliente se conectará a al segundo servidor. Se capturará el tráfico a través de WireShark para filtrar ambos flujos.

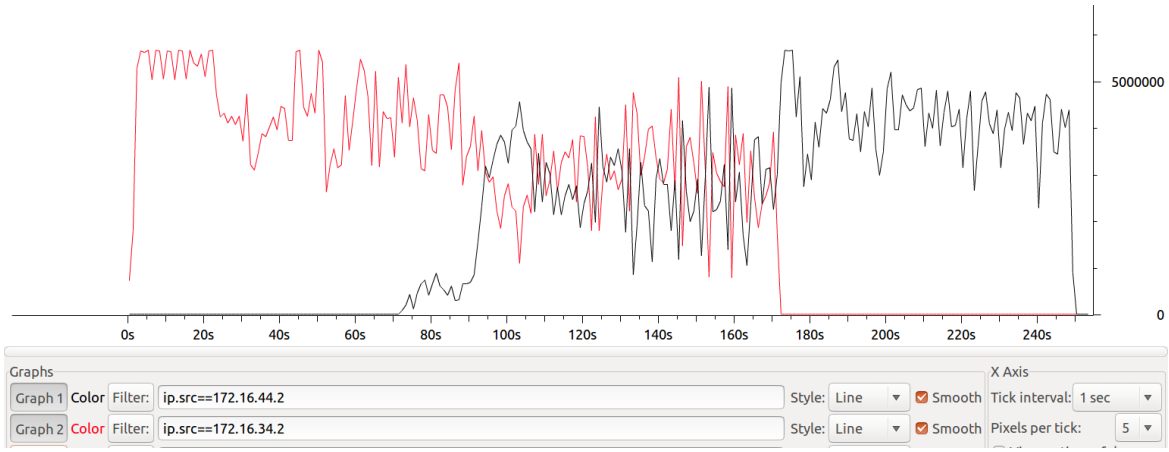

Figura 23: Prueba 3, amigabilidad entre 2 clientes TAPAS, segmentos 1s. Captura de Wireshark

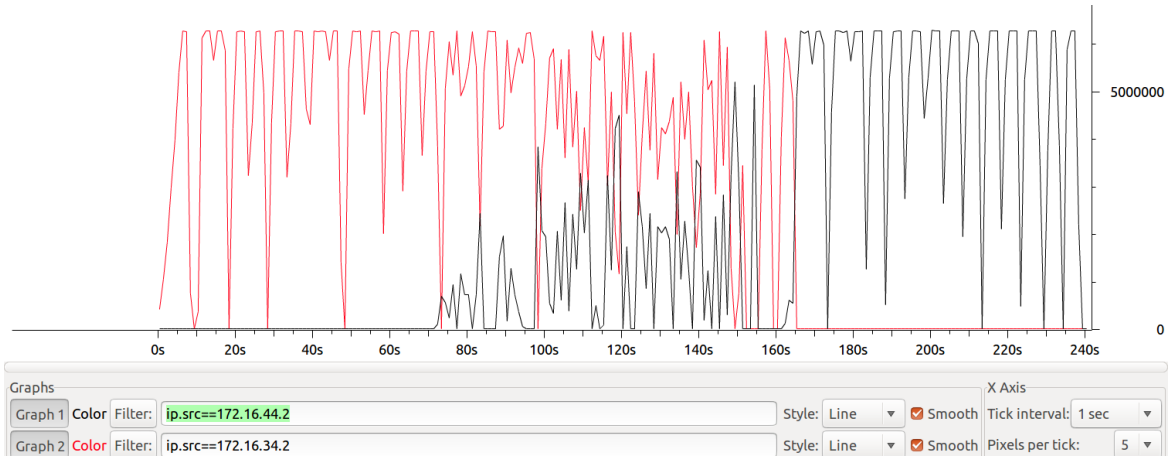

Figura 24: Prueba 3, amigabilidad entre 2 clientes TAPAS, segmentos 5s. Captura de Wireshark

Se puede observar claramente que el reparto de los recursos de la red es mucho mejor con los segmentos de 1 segundo, logrando prácticamente que ambos flujos sean iguales. En cambio, con segmentos de 5 segundos no se consigue un buen reparto, quedando ambos flujos descompensados.

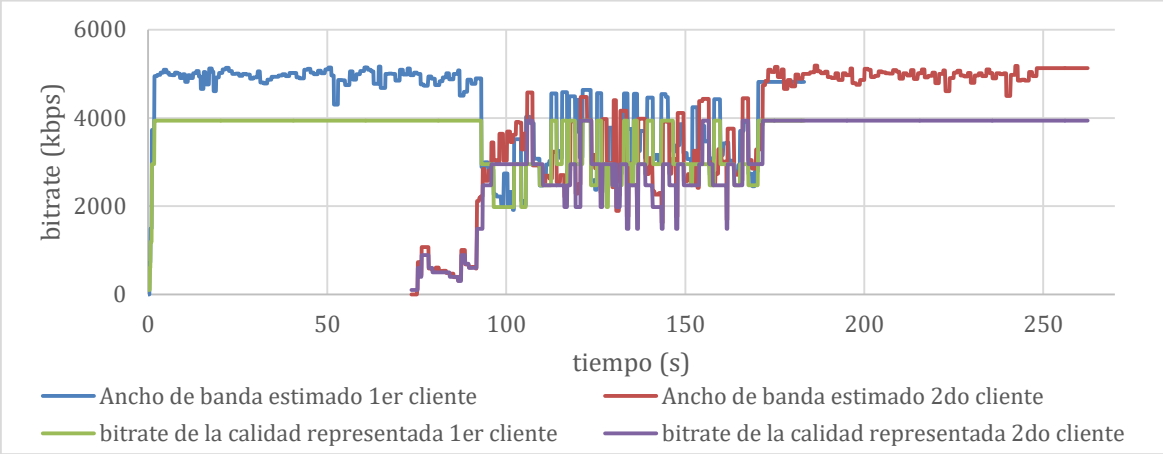

Figura 25: Comparación del ancho de banda estimado y el bitrate de la calidad entre ambos clientes, con segmentos de 1s

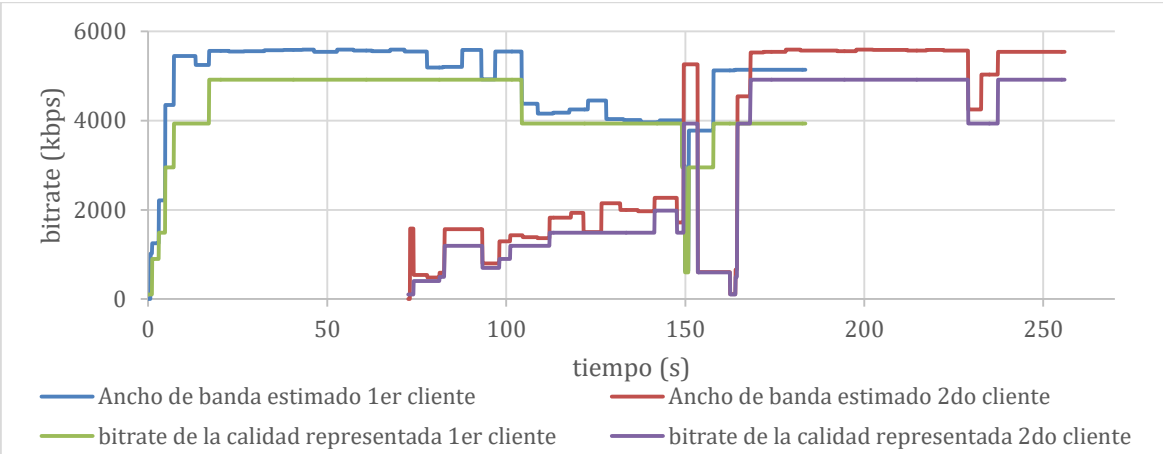

Figura 26: Comparación del ancho de banda estimado y el bitrate de la calidad entre ambos clientes, con segmentos de 5s

4.4. MP4Client

Se intentó modificar el código fuente del programa de cara a conseguir el log apropiado para poder realizar las mismas graficas que en TAPAS pero no se logró que quedaran unos logs semejantes. Para la realización del as pruebas con MP4Client se utilizará el contenido perteneciente al profile *live.*

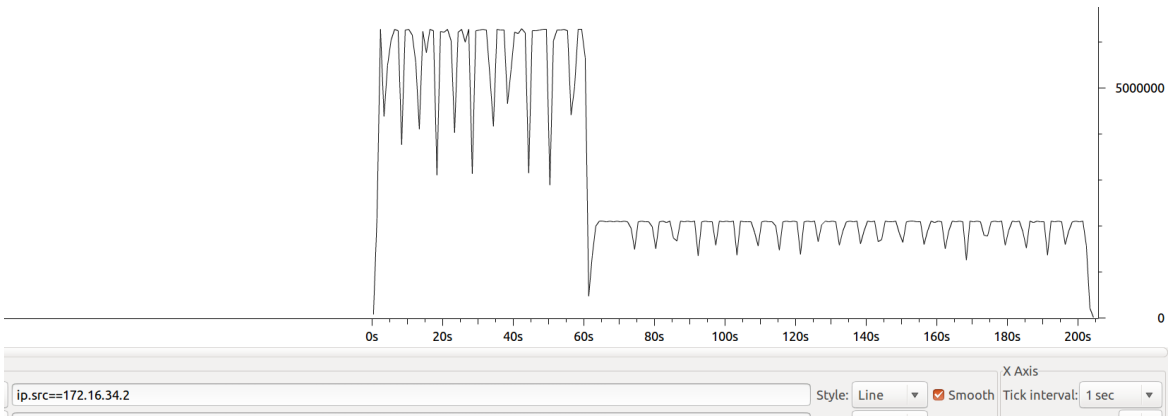

Figura 27: Prueba 1, con segmentos de 5s. Captura de Wireshark

Para la primera prueba se puede comprobar que los tiempos de respuesta son muy pequeños, mejores que con TAPAS. Esto puede observarse tanto en la captura de Wireshark como en las estadísticas que ofrece la GUI de MP4Client.

Se logró un log del ancho de banda estimado y del bitrate de la representación con cada descarga de un segmento, pero debido a que MP4Client si tiene en cuenta las representaciones de audio, es necesario hacer una diferenciación entre ambos flujos para lograr un log funcional.

Para la segunda prueba los resultados obtenidos para MP4Client son insatisfactorios para cualquiera de las duraciones de segmento, obteniendo bastantes interrupciones. Seguramente debido a las representaciones de audio, que cuando se dispone poco ancho de banda si juegan un papel destacable sobre el total.

<u>Prueba amigabilidad:</u>

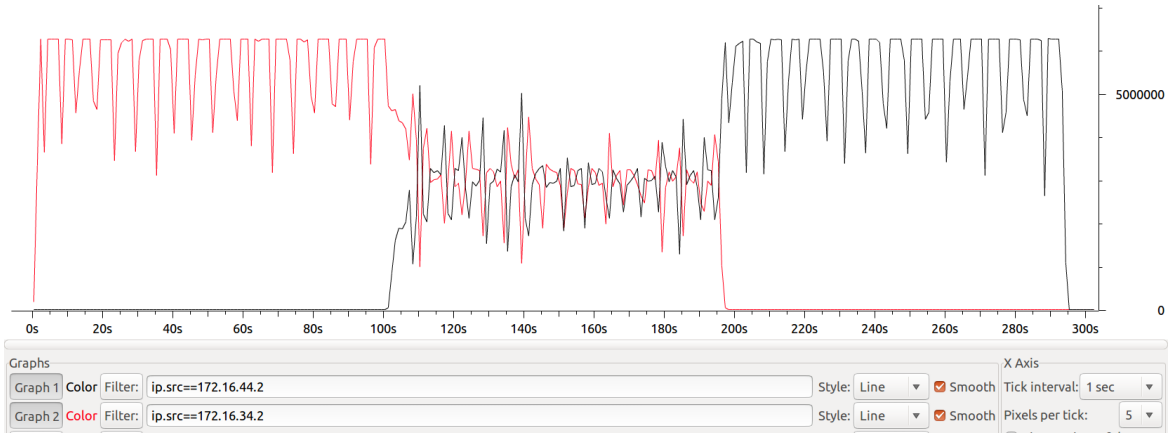

Figura 28: Prueba 3, amigabilidad entre 2 clientes MP4Client, segmentos 5s

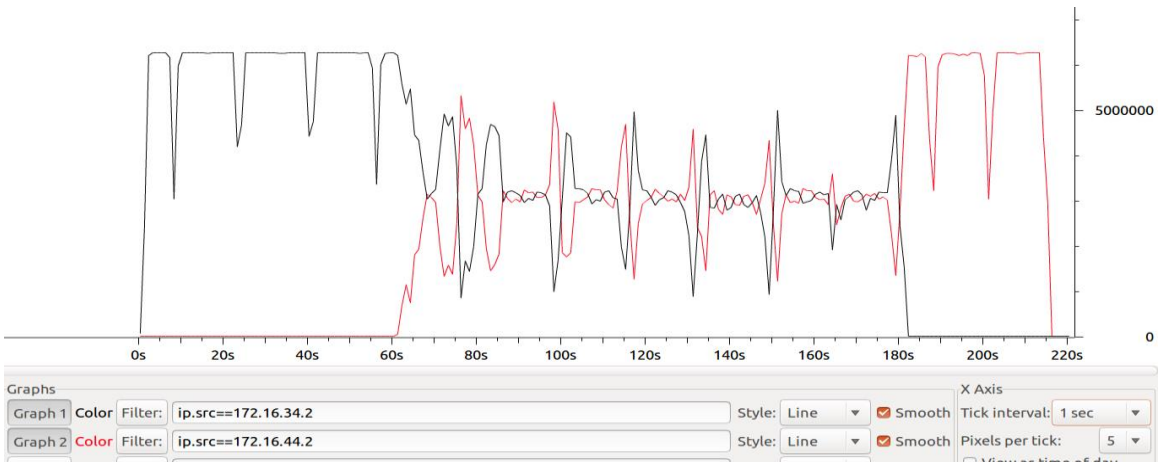

Figura 29: Prueba 3, amigabilidad entre 2 clientes MP4Client, segmentos 15s

Vemos que el resultado obtenido en el caso de MP4Box es mejor que el obtenido con TAPAS ya que el objetivo es conseguir que ambos flujos compartan equitativamente los recursos de la red y que se estabilicen lo más rápido posible. Se logra para cualquiera de las duraciones de segmento.

5. Presupuesto

Este apartado contiene una estimación del coste de este proyecto. Se tendrán en cuenta los costes procedentes del personal y también los costes en el hardware y el software utilizado.

Para calcular el salario del personal se tendrán en cuenta el número horas dedicadas al proyecto por el estudiante, evaluando el coste como el de un ingeniero junior.

Dado que todo el software utilizado para la realización del proyecto es de código abierto y/o libre: Ubuntu, TAPAS, herramientas de GPAC, VLC, Servidor Apache, WANem, VirtualBox, dashjs, librería libdash, Wireshark, etc. Sólo se tendrá en cuenta los costes producidos por el hardware, donde se ha precisado de un portátil lo suficientemente potente (8GB de RAM y un procesador i7) para permitir la virtualización de varios sistemas operativos a la vez y varios vídeos en reproducción simultánea.

Hardware	Cantidad	Precio/unidad	Coste
Portátil	1	900 €	900 €
Total			**900 €**

Personal	Precio/hora	Horas/mes	Salario/mes	Meses	Total
Ingeniero Junior	8 €	100	800 €	5	4.000 €
Total					**4.000 €**

TOTAL	**4.900 €**

Tabla 6: Coste total del proyecto

6. Conclusiones y trabajo futuro

Se puede decir que se han cumplido los objetivos, si bien es cierto que no se ha implementado un algoritmo controlador de calidades de cero sí que se han podido realizar las 3 partes en las que se dividía el desarrollo de manera satisfactoria. El entorno de trabajo nos permite controlar con comodidad la configuración de red, fiable y de gran uso para la realización de las pruebas.

La generación del contenido también ha llegado a su meta, llegando incluso a crear un software que nos permite crear contenido con comodidad. Finalmente, se han podido testear gran parte de los clientes DASH e incluso estudiar los algoritmos de algunos de ellos, permitiendo evaluarlos mediante a distintas pruebas.

Una vez llegados a este punto, hay muchas cosas que aún se pueden realizar para acabar de afianzar todos los conceptos pertenecientes a MPEG-DASH y profundizar a un nivel mucho más profesional.

La primera sería la inclusión de DRM en la plataforma creada, tal y como hace CableLabs con Demystified [30], el cual se puede hacer fácilmente con MP4Box ya que soporta el estándar Common Encryption y es altamente configurable. DRM hace referencia a las tecnologías de control de acceso que son utilizadas para restringir el uso del contenido. Otra cosa a introducir en la plataforma seria el uso de SVC (Scalable Video Coding) para estudiar los efectos podría tener en la adaptación del bitrate [31].

Finalmente, podría ser interesante estudiar también cómo afecta la latencia en la difusión de contenido en vivo y como se puede solucionar [32]. Debido a que el streaming HTTP basado en segmentos obliga a la inclusión de un keyframe cada segmento, sumado a la alineación de los keyframes a través de todas las calidades hace que la compresión de los vídeos sea menos eficiente. Además el contenido en vivo debe hacer frente a caches y buffers más pequeños. Estos deberían ser capaces de usar los segmentos incluso antes de que estuvieran completamente descargados.

Bibliografía

[1] YouTube - Broadcast Yourself: https://www.youtube.com/

[2] Netflix, Watch TV shows & movies anytime, anywhere: https://www.netflix.com/

[3] Global Internet Phenomena Report, September 2015, Asia-Pacific and Europe: https://www.sandvine.com/downloads/general/global-internet-phenomena/2015/global-internet-phenomena-report-apac-and-europe.pdf

[4] Global Internet Phenomena Report, December 2015, Africa, Middle East, and North America: https://www.sandvine.com/downloads/general/global-internet-phenomena/2015/global-internet-phenomena-africa-middle-east-and-north-america.pdf

[5] ISO/IEC 23009-1:2014. Information technology - Dynamic adaptive streaming over HTTP (DASH) – Part 1: Media presentation description and segment formats

[6] ISO/IEC 14496-10:2012. Information technology – Coding of audio-visual objects – Part 10: Advanced Video Coding.

[7] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, “Hypertext Transfer Protocol – HTTP/1.1.” RFC 2616 (Draft Standard), June 1999. Updated by RFCs 2817, 5785, 6266, 6585.

[8] J. Postel, “User Datagram Protocol.” RFC 768 (Standard), Aug. 1980.

[9] J. Postel, “Transmission Control Protocol.” RFC 793 (Standard), Sept. 1981. Updated by RFCs 1122, 3168, 6093, 6528.

[10] HTTP Dynamic Streaming Specification - Version 3.0 FINAL http://wwwimages.adobe.com/content/dam/Adobe/en/devnet/hds/pdfs/adobe-hds-specification.pdf

[11] R. Pantos (ed), “HTTP Live Streaming.” http://tools.ietf.org/html/draft-pantos-httplive-streaming-17, October 2015.

[12] A. Zambelli, “Smooth Streaming Technical Overview.” http://download.microsoft.com/download/4/2/4/4247C3AA-7105-4764-A8F9-321CB6C765EB/IIS_Smooth_Streaming_Technical_Overview.pdf, March 2009.

[13] MPEG-DASH VS. APPLE HLS VS. MICROSOFT SMOOTH STREAMING VS. ADOBE HDS: https://www.bitcodin.com/blog/2015/03/mpeg-dash-vs-apple-hls-vs-microsoft-smooth-streaming-vs-adobe-hds/

[14] MPEG-DASH IN A NUTSHELL: https://www.bitcodin.com/blog/2015/04/mpeg-dash/

[15] VirtualBox: https://www.virtualbox.org/

[16] WANem 2.0 Wide Area Network Emulator: http://downloads.sourceforge.net/project/wanem/Documents/wanemulator_all_about_v2.0.pdf

[17] Apache – HTTP Server Project: https://httpd.apache.org/

[18] DASH Industry Forum Guidelines: http://dashif.org/guidelines/

[19] VideoLAN - x264, the best H.264/AVC encoder: http://www.videolan.org/developers/x264.html

[20] FFmpeg - A complete, cross-platform solution to record, convert and stream audio and video: https://www.ffmpeg.org/

[21] GPAC Multimedia Open Source Project: https://gpac.wp.mines-telecom.fr/

[22] Stefan Lederer, Christopher Müller and Christian Timmerer, “Dynamic Adaptive Streaming over HTTP Dataset”, In Proceedings of the ACM Multimedia Systems Conference 2012, Chapel Hill, North Carolina, February 22-24, 2012.

[23] Clase C++ AnyOption: http://www.hackorama.com/anyoption/

[24] 10 Free & Public MPEG-DASH Teststreams and Datasets: http://www.dash-player.com/blog/2015/04/10-free-public-mpeg-dash-teststreams-and-datasets/

[25] VLC media player: http://www.videolan.org/vlc/index.html

[26] Christopher Mueller, Stefan Lederer, Joerg Poecher, and Christian Timmerer, “libdash – An Open Source Software Library for the MPEG-DASH Standard”, in Proceedings of the IEEE International Conference on Multimedia and Expo 2013, San Jose, USA, July, 2013

[27] Luca De Cicco, Vito Caldaralo, Vittorio Palmisano, Saverio Mascolo, "TAPAS: a Tool for rApid Prototyping of Adaptive Streaming algorithms", in Proc. of ACM VideoNext Workshop, Sydney, Australia, December 2014.

[28] dash.js: https://github.com/Dash-Industry-Forum/dash.js/wiki

[29] GStreamer open source multimedia framework: http://gstreamer.freedesktop.org/

[30] Content Creation Demystified: Open Source to the Rescue: http://www.cablelabs.com/content-creation-demystified-open-source-to-the-rescue/

[31] Bouzakaria, N.; Concolato, C.; Le Feuvre, J. "Overhead and performance of low latency live streaming using MPEG-DASH," IISA 2014, pp.92,97, 7-9 July 2014

[32] H. Kalva, V. Adzic, and B. Furht, “Comparing MPEG AVC and SVC for adaptive HTTP streaming,” inProceedings of the IEEE International Conference on Consumer Electronics, pp. 160–161, 2012.

[33] Tasksel Ubuntu tool: https://help.ubuntu.com/community/Tasksel

Apéndices

A: Configuración del entorno de trabajo

Para conseguir el entorno de trabajo deseado deberemos configurar la tabla de rutas de todos los sistemas con el fin de forzar que el flujo del streaming pase a través de WANem. Primero tendremos que configurar las máquinas desde Virtualbox, añadiendo a cada una de ellas los adaptadores de red correspondientes.

En WANem utilizaremos una red Solo-anfitrión para conectarlo al Host y dos redes internas, una para cada servidor. Cada servidor dispondrá de dos adaptadores de red, el primero será la red interna que usaremos para conectarlos a WANem y la segunda será un NAT. A continuación se explica la diferencia entre los tres adaptadores.

- Sólo-Anfitrión: Se crea una tarjeta de red virtual en el Host permitiendo conectarse con las máquinas virtuales sin necesitar que el Host utilice una tarjeta de red.
- NAT: Este modo permitirá a los servidores conectarse a Internet a través del Host sin necesidad de configurar la red en cuestión.
- Red Interna: Red virtual sólo visible entre máquinas virtuales conectadas a ella.

Una vez configurados los adaptadores de red en Virtualbox, instalaremos *VirtualBox Guest Additions* para mejorar el rendimiento y añadir otras funciones como la posibilidad de pantalla completa y la compartición del portapapeles y medios extraíbles.

Seguidamente haremos uso del administrador de redes de los servidores y de WANem para definir manualmente las IP asociadas a las interfaces. En el caso de los servidores únicamente definiremos la IP asociada a la eth1, la que conecta con WANem, tal y como se puede observar en la Figura 28. La eth0, que conecta con el Host se configurará automáticamente mediante DHCP.

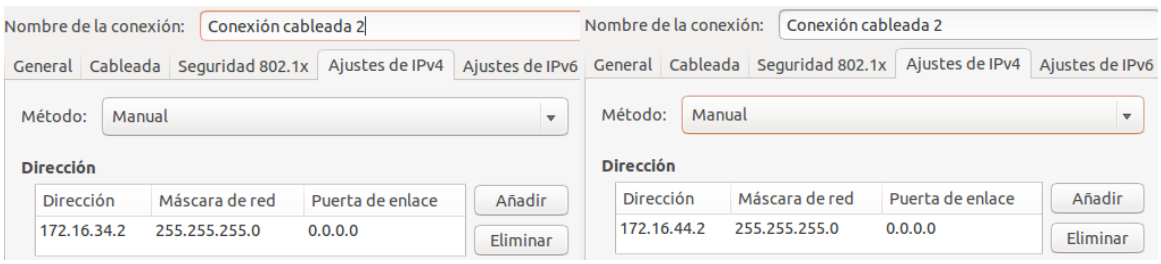

Figura 30: Configuración de la eth1 de ambos servidores

En el caso de WANem definiremos manualmente las IPs asociadas a la eth1 y eth2, las que conectan con los servidores, tal y como se puede observar en la Figura 29. La eth0, que conecta con el Host, se configurará nuevamente de forma automática mediante DHCP, obteniendo la IP `192.168.56.101`.

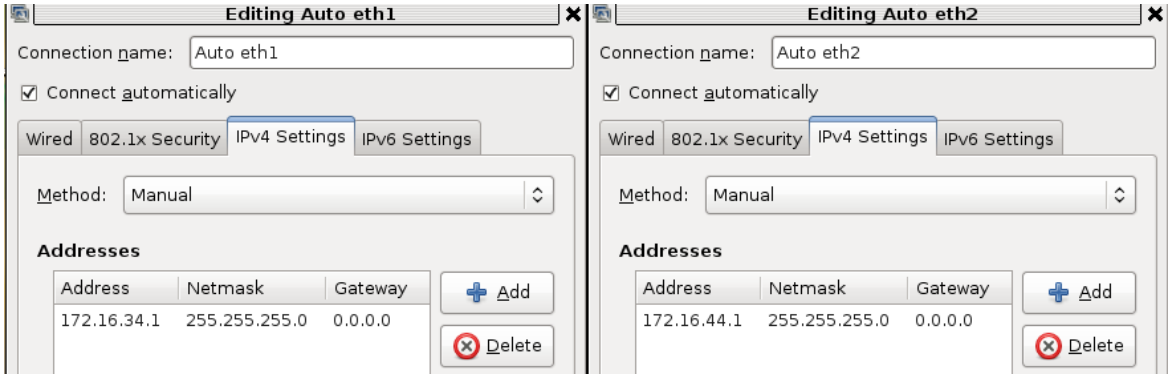

Figura 31: Configuración de la eth1 y la eth2 de WANem

Finalmente ya podremos configurar las tablas de rutas.

A.1: Host

Para definir la nueva tabla de rutas en el Host utilizamos un script *sh* de modo que sólo tengamos que ejecutarlo una vez estén iniciados los servidores y WANem. El script es el siguiente:

```
#!/bin/bash
# -*- ENCODING: UTF-8 -*-
sudo route add -net 172.16.34.0/24 gw 192.168.56.101
sudo route add -net 172.16.44.0/24 gw 192.168.56.101
exit
```

Quedando la tabla de rutas tal y como se muestra en la Tabla 7:

Destino	Pasarela	Genmask	Indic	Métric	Ref	Uso	Interfaz
0.0.0.0	192.168.1.1	0.0.0.0	UG	0	0	0	wlan0
172.16.34.0	192.168.56.101	255.255.255.0	UG	0	0	0	vboxnet0
172.16.44.0	192.168.56.101	255.255.255.0	UG	0	0	0	vboxnet0
192.168.1.0	0.0.0.0	255.255.255.0	U	9	0	0	wlan0
192.168.56.0	0.0.0.0	255.255.255.0	U	0	0	0	vboxnet0

Tabla 7: Tabla de rutas del Host

A.2: Servidores

En los servidores, para lograr ejecutar los comandos que manipulan la tabla de encaminamiento IP, modificamos el script rc.local. Cualquier comando que coloquemos en el fichero /etc/rc.local será ejecutado al final del arranque del sistema permitiéndonos así no tener que volver a configurar la máquina cada vez que iniciemos el sistema.

Para el servidor 1 rc.local queda de la siguiente manera:

```
#!/bin/sh -e
#
# rc.local server 1
#
# This script is executed at the end of each multiuser runlevel.
# Make sure that the script will "exit 0" on success or any other
# value on error.
#
# In order to enable or disable this script just change the execution
# bits.
#
# By default this script does nothing.
route add -net 192.168.56.0/24 gw 172.16.34.1
exit 0
```

Para el servidor 2:

```
#!/bin/sh -e
#
# rc.local server 2
#
# This script is executed at the end of each multiuser runlevel.
# Make sure that the script will "exit 0" on success or any other
# value on error.
#
# In order to enable or disable this script just change the execution
# bits.
#
# By default this script does nothing.
route add -net 192.168.56.0/24 gw 172.16.44.1

exit 0
```

Quedando las tablas de rutas de ambos servidores tal y como podemos observar en la Tabla 8 y la Tabla 9 respectivamente:

Destino	Pasarela	Genmask	Indic	Métric	Ref	Uso	Interfaz
0.0.0.0	10.0.2.2	0.0.0.0	UG	0	0	0	eth0
10.0.2.0	0.0.0.0	255.255.255.0	U	1	0	0	eth0
172.16.34.0	0.0.0.0	255.255.255.0	U	1	0	0	eth1
192.168.56.0	172.16.34.1	255.255.255.0	UG	0	0	0	eth1

Tabla 8: Tabla de rutas del servidor 1

Destino	Pasarela	Genmask	Indic	Métric	Ref	Uso	Interfaz
0.0.0.0	10.0.2.2	0.0.0.0	UG	0	0	0	eth0
10.0.2.0	0.0.0.0	255.255.255.0	U	1	0	0	eth0
172.16.44.0	0.0.0.0	255.255.255.0	U	1	0	0	eth1
192.168.56.0	172.16.44.1	255.255.255.0	UG	0	0	0	eth1

Tabla 9: Tabla de rutas del servidor 2

A.3: WANem

En WANem no hará falta configurar la tabla de rutas, que quedará tal y como se muestra en la Tabla 10.

Destino	Pasarela	Genmask	Indic	Métric	Ref	Uso	Interfaz
169.254.0.0	0.0.0.0	255.255.0.0	U	1	0	0	eth3
172.16.34.0	0.0.0.0	255.255.255.0	U	1	0	0	eth1
172.16.44.0	0.0.0.0	255.255.255.0	U	1	0	0	eth2
192.168.56.0	0.0.0.0	255.255.255.0	U	1	0	0	eth0

Tabla 10: Tabla de rutas de WANem

Una vez todo está configurado podemos comprobar el correcto funcionamiento mediante pings. Además, será necesario instalar el servidor HTTP Apache en las máquinas que harán de servidor para poder acceder al contenido DASH desde el Host. Se instalará de manera sencilla haciendo uso de Tasksel [33].

B: Programa para automatizar la generación del contenido DASH

```
#include <vector>       /* std::vector */
#include <iostream>     /* std::cout */
#include <sstream>      /* stringstream */
#include <cstdlib>      /* std::atoi */
#include <stdio.h>      /* printf */
#include <stdlib.h>     /* system, NULL, EXIT_FAILURE */
#include <string>       /* std::string */
#include "AnyOption.h"

void create( int argc, char* argv[] );
void setHelp(AnyOption*);
void setOptions(AnyOption*);
void audio_encoding(std::string input, int flag, std::string input2,
std::string destinationk, std::string canales, std::string fm,
std::string inp_a, std::vector<std::string> bitrate_list_a, std::string
pista);
void segmentation(std::string dest, std::string input2,
std::vector<std::string> seg_list, int flag, int m,
std::vector<std::string> bitrate_list);
unsigned int gcd (unsigned int n1, unsigned int n2);
AnyOption *opt;
std::vector<std::string> list;
std::vector<std::string> list2;
std::string destinationk;

int main(int argc, char* argv[])
{
      create( argc, argv );
      return 0;
}

void create(int argc, char* argv[])
{
      AnyOption *opt = new AnyOption();
      opt->setVerbose();
      opt->autoUsagePrint(true);
      setHelp(opt);
      setOptions(opt);
      opt->processFile("./DASHGen.config");
      opt->processCommandArgs(argc, argv);

      std::string flagg = opt->getValue("flag");
      int flag = std::atoi(flagg.c_str());
      std::cout << "FLAG: " << flag << "\n";

      std::string input = opt->getValue("input_dir"); /*
/home/joma/Documentos/ */
      std::string dest = opt->getValue("dest_dir");  /*
/home/joma/Documentos/out/ */
      std::ostringstream in;
      in << input << opt->getValue("input_file");    /* sintel.y4m */
      std::string inp = in.str();                          /*
/home/joma/Documentos/sintel.y4m */
      std::string input2 =
inp.substr(inp.find_last_of("/")+1,inp.find_last_of(".")-
inp.find_last_of("/")-1); /* sintel */
      std::ostringstream in_a;
      in_a << input << opt->getValue("audio_file");  /* sintel.flac */
```

```
	std::string inp_a = in_a.str();		/*
/home/joma/Documentos/sintel.flac */
	std::cout << "INPUT: " << inp << "\n";
	std::string fps = opt->getValue("fps");		/* 30 */
	std::cout << "FPS: " << fps << "\n";

	std::string preset = "slow";

	std::stringstream seg_duration;
	seg_duration << opt->getValue("segment_dur");	/* 1,5,15 */
	std::vector<std::string> seg_list;
	while( seg_duration.good() ) {
		std::string substr;
		getline( seg_duration, substr, ',' );
		seg_list.push_back( substr );
	}
	int segment_dur[ seg_list.size() ];
	for ( unsigned int e=0; e<seg_list.size(); ++e )	{
		segment_dur[e] = std::atoi(seg_list.at(e).c_str());
	}

	std::stringstream brate;
	brate << opt->getValue("bitrate");		/*
50,100,150,200,250,300,400,500,600,700,900,1200,1500,2000,2500,3000,4000
,5000,6000,7000,8000 */
	std::vector<std::string> bitrate_list;
	while( brate.good() )	{
		std::string substr;
		getline( brate, substr, ',' );
		bitrate_list.push_back( substr );
	}
	int bitrate[ bitrate_list.size() ];
	for ( unsigned int e=0; e<bitrate_list.size(); ++e ) {
		bitrate[e] = std::atoi(bitrate_list.at(e).c_str());
	}

	std::stringstream wi;
	wi << opt->getValue("width");		/*
320,320,320,480,480,480,480,854,854,854,854,1280,1280,1280,1280,1920,192
0,1920,1920,1920,1920 */
	std::vector<std::string> w_list;
	while( wi.good() )	{
		std::string substr;
		getline( wi, substr, ',' );
		w_list.push_back( substr );
	}
	std::string w[ w_list.size() ];
	for ( unsigned int e=0; e<w_list.size(); ++e ) {
		w[e] = w_list.at(e);
	}

	std::stringstream he;
	he << opt->getValue("height");		/*
240,240,240,360,360,360,360,480,480,480,480,720,720,720,720,1080,1080,10
80,1080,1080,1080 */
	std::vector<std::string> h_list;
	while( he.good() )	{
		std::string substr;
		getline( he, substr, ',' );
```

```
            h_list.push_back( substr );
      }
      std::string h[ h_list.size() ];
      for ( unsigned int e=0; e<h_list.size(); ++e ) {
            h[e] = h_list.at(e);
      }

      std::stringstream brate_a;
      brate_a << opt->getValue("bitrate_a");          /* 64,96,128,164 */
      std::vector<std::string> bitrate_list_a;
      while( brate_a.good() ) {
            std::string substr;
            getline( brate_a, substr, ',' );
            bitrate_list_a.push_back( substr );
      }

      std::string fm = opt->getValue("fm");                 /* 48000 */
      std::string canales = opt->getValue("canales");       /* 2 */
      std::string pista = opt->getValue("pista");           /* 0:0 */

      std::string aux;

      int fps2 = std::atoi(fps.c_str());
      int GOP = segment_dur[0]*fps2; //1*30
      if (seg_list.size()>1) {
            for (unsigned int m=1; m<seg_list.size(); ++m)
            {
                  GOP = gcd(GOP,segment_dur[m]*fps2);
            }
      }
      std::ostringstream min_keyint_2;
      min_keyint_2 << GOP;
      std::string min_keyint = min_keyint_2.str();
      std::string keyint = min_keyint;                  //Ej: keyint =
min_keyint = 30 fps * 2 seg = 60

      std::cout <<
"==========================================================\n";
      std::cout << "===============GOP DE " << min_keyint << "
FRAMES====================\n";
      std::cout <<
"==========================================================\n";
      std::cout << '\n';

      std::cout << "==========MAKING DIRECTORY===============\n";
      std::string folderk = "mkdir -p ";
      destinationk = dest;
      destinationk.append(input2);
      folderk.append(destinationk);
      std::cout << folderk << '\n' << '\n';
      if (flag==true)
            system(folderk.c_str());

      for (unsigned int n=0; n<sizeof(bitrate)/sizeof(bitrate[0]); ++n)
      {
            std::ostringstream bs;
            bs << bitrate[n];
            std::string br = bs.str();
            std::cout << "BITRATE = " << br << "\n";
```

```
            std::cout << "==========DASH VIDEO
ENCODING==================\n";
            std::string x264 = "x264 ";

            x264.append("--output ");
            aux = destinationk;
            aux.append("/");
            aux.append(input2);
            aux.append("_");
            aux.append(br);
            list.push_back(aux);
            x264.append(aux);
            x264.append("k.h264 ");

            x264.append("--fps ");
            x264.append(fps);
            x264.append(" ");

            x264.append("--preset ");
            x264.append(preset);
            x264.append(" ");

            x264.append("--bitrate ");
            x264.append(br);
            x264.append(" ");

            x264.append("--vbv-maxrate ");      //std::string vbv_maxrate
= bitrate;
            std::ostringstream vbv_m;
            vbv_m << bitrate[n];
            std::string vbv_mr = vbv_m.str();
            x264.append(vbv_mr);
            x264.append(" ");

            x264.append("--vbv-bufsize ");      //std::string vbv_bufsize
= 2*vbv_maxrate;
            std::ostringstream vbv_b;
            vbv_b << 2*bitrate[n];
            std::string vbv_bs = vbv_b.str();
            x264.append(vbv_bs);
            x264.append(" ");

            x264.append("--min-keyint ");
            x264.append(min_keyint);
            x264.append(" ");

            x264.append("--keyint ");
            x264.append(keyint);
            x264.append(" ");

            x264.append("--scenecut 0 --no-scenecut --pass 1 --video-
filter \"resize:width=");
            x264.append(w[n]);
            x264.append(",height=");
            x264.append(h[n]);
            x264.append("\" ");

            x264.append(inp);
```

```
            std::cout << x264 << '\n' << '\n';
            if (flag==true)
                  system(x264.c_str());

            std::cout << "==========H264 raw video to an MP4
container===============\n";
            std::string mp4box1 = "MP4Box ";
            mp4box1.append("-add ");
            mp4box1.append(aux);
            mp4box1.append("k.h264 ");

            mp4box1.append("-fps ");
            mp4box1.append(fps);
            mp4box1.append(" ");

            mp4box1.append(aux);
            mp4box1.append("k.mp4 ");

            std::cout << mp4box1 << '\n' << '\n';
            if (flag==true)
                  system(mp4box1.c_str());
      }

      audio_encoding(input, flag, input2, destinationk, canales, fm,
inp_a, bitrate_list_a, pista);

      for (unsigned int m=0;
m<sizeof(segment_dur)/sizeof(segment_dur[0]); ++m)
      {
            std::cout <<
"=========================================================\n";
            std::cout << "===============SEGMENTOS DE " << seg_list[m]
<< " SEGUNDOS======================\n";
            std::cout <<
"=========================================================\n";
            std::cout << '\n';
            segmentation(dest, input2, seg_list, flag, m, bitrate_list);
      }
}

void audio_encoding(std::string input, int flag, std::string input2,
std::string destinationk, std::string canales, std::string fm,
std::string inp_a, std::vector<std::string> bitrate_list_a, std::string
pista)
{
      int bitrate_a[ bitrate_list_a.size() ];
      for ( unsigned int e=0; e<bitrate_list_a.size(); ++e )      {
            bitrate_a[e] = std::atoi(bitrate_list_a.at(e).c_str());
      }

      std::cout << "==========DASH AUDIO ENCODING==================\n";
      for (unsigned int n=0; n<sizeof(bitrate_a)/sizeof(bitrate_a[0]);
++n)
      {
            std::ostringstream bs_a;
            bs_a << bitrate_a[n];
            std::string br_a = bs_a.str();

            std::string ffmpeg = "ffmpeg ";
```

```
            ffmpeg.append("-i ");
            ffmpeg.append(inp_a);
            ffmpeg.append(" ");

            ffmpeg.append("-map ");
            ffmpeg.append(pista);
            ffmpeg.append(" ");

            ffmpeg.append("-acodec libfaac ");

            ffmpeg.append("-ac ");
            ffmpeg.append(canales);
            ffmpeg.append(" ");

            ffmpeg.append("-ar ");
            ffmpeg.append(fm);
            ffmpeg.append(" ");

            ffmpeg.append("-b:a ");
            ffmpeg.append(br_a);
            ffmpeg.append("k ");

            std::string aux2;
            aux2 = destinationk;
            aux2.append("/");
            aux2.append(input2);
            aux2.append("_audio_");
            aux2.append(br_a);
            aux2.append("k.mp4");
            list2.push_back(aux2);
            ffmpeg.append(aux2);

            std::cout << ffmpeg << '\n' << '\n';
                  if (flag==true)
                        system(ffmpeg.c_str());
      }
}

void segmentation(std::string dest, std::string input2,
std::vector<std::string> seg_list, int flag, int m,
std::vector<std::string> bitrate_list)
{
      std::cout << "==========DASH SEGMENTATION===============\n";
      std::cout << "==========FINAL LIVE DASHING===============\n";
      std::string fo = "mkdir -p ";
      for (unsigned int q=0; q<bitrate_list.size()+list2.size(); ++q)
      {
            fo.append(dest);
            fo.append(input2);
            fo.append("/");
            fo.append(seg_list[m]);
            fo.append("sec/live/");
            std::ostringstream qq;
            qq << q+1;
            std::string qqq = qq.str();
            fo.append(qqq);
            fo.append(" ");
      }
```

```
	std::cout << fo << '\n' << '\n';
	if (flag==true)
		system(fo.c_str());

	std::string mp4box3 = "MP4Box ";
	mp4box3.append("-dash ");
	mp4box3.append(seg_list[m]);
	mp4box3.append("000 ");

	mp4box3.append("-frag ");
	mp4box3.append(seg_list[m]);
	mp4box3.append("000 -rap -frag-rap -segment-name
'$RepresentationID$/");
	mp4box3.append(input2);
	mp4box3.append("_");
	mp4box3.append(seg_list[m]);
	mp4box3.append("s_seg' -profile live ");

	for (unsigned int i=0; i<list.size(); ++i) {
		mp4box3.append(list.at(i));
		mp4box3.append("k.mp4 ");
	}

	for (unsigned int i=0; i<list2.size(); ++i) {
		mp4box3.append(list2.at(i));
		mp4box3.append(" ");
	}

	mp4box3.append("-out ");
	std::string live = dest;
	live.append(input2);
	live.append("/");
	live.append(seg_list[m]);
	live.append("sec/live/");
	live.append(input2);
	live.append("_live_dash");
	mp4box3.append(live);
	mp4box3.append(".mpd");

	std::cout << mp4box3 << '\n' << '\n' << '\n';
	if (flag==true)
		system(mp4box3.c_str());

	std::string replace = "sed -i
's_media=\"_media=\"\$RepresentationID\$/_g' ";
	replace.append(live);
	replace.append(".mpd");
	std::cout << '\n' << replace << '\n' << '\n' << '\n';
	if (flag==true)
		system(replace.c_str());

	std::cout << "==========FINAL VOD DASHING===============\n";
	std::string fo2 = "mkdir -p ";
	fo2.append(dest);
	fo2.append(input2);
	fo2.append("/");
	fo2.append(seg_list[m]);
	fo2.append("sec/ondemand/");
	std::cout << fo2 << '\n' << '\n';
```

```
        if (flag==true)
                system(fo2.c_str());

        std::string mp4box4 = "MP4Box ";
        mp4box4.append("-dash ");
        mp4box4.append(seg_list[m]);
        mp4box4.append("000 ");

        mp4box4.append("-frag ");
        mp4box4.append(seg_list[m]);
        mp4box4.append("000 -rap -frag-rap -profile onDemand ");

        for (unsigned int i=0; i<list.size(); ++i) {
                mp4box4.append(list.at(i));
                mp4box4.append("k.mp4 ");
        }

        for (unsigned int i=0; i<list2.size(); ++i) {
                mp4box4.append(list2.at(i));
                mp4box4.append(" ");
        }

        mp4box4.append("-out ");
        mp4box4.append(dest);
        mp4box4.append(input2);
        mp4box4.append("/");
        mp4box4.append(seg_list[m]);
        mp4box4.append("sec/ondemand/");
        mp4box4.append(input2);
        mp4box4.append("_onDemand_dash.mpd");

        std::cout << mp4box4 << '\n' << '\n' << '\n';
        if (flag==true)
                system(mp4box4.c_str());
}

unsigned int gcd (unsigned int n1, unsigned int n2) {
    return (n2 == 0) ? n1 : gcd (n2, n1 % n2);
}

void setOptions(AnyOption* opt)
{
    opt->setFlag("help", 'H');
    opt->setOption("input_file", 'i');
    opt->setOption("audio_file", 'a');
    opt->setOption("input_dir", 'd');
    opt->setOption("dest_dir", 'o');
    opt->setOption("fps", 'f');
    opt->setOption("canales", 'c');
    opt->setOption("fm", 'm');
    opt->setOption("segment_dur", 's');
    opt->setOption("pista", 'p');
    opt->setOption("bitrate", 'b');
    opt->setOption("bitrate_a", 'B');
    opt->setOption("width", 'w');
    opt->setOption("height", 'h');
    opt->setOption("flag", 'F');
}
```

```
void setHelp(AnyOption* opt)
{
    opt->addUsage("");
    opt->addUsage("Usage: ");
    opt->addUsage("");
    opt->addUsage(" -H  --help              Print help ");
    opt->addUsage(" -i  --input_file        Input video file");
    opt->addUsage(" -a  --audio_file        Input audio file");
    opt->addUsage(" -d  --input_dir         Input directory");
    opt->addUsage(" -o  --dest_dir          Destination directory");
    opt->addUsage(" -f  --fps               Input video FPS");
    opt->addUsage(" -c  --canales           Input audio chanels");
    opt->addUsage(" -p  --pista             Audio track");
    opt->addUsage(" -m  --fm                Input audio sampling rate");
    opt->addUsage(" -s  --segment_dur       Segment Duration for MPD");
    opt->addUsage(" -b  --bitrate           Video bitrates (exmaple: see
config file)");
    opt->addUsage(" -a  --bitrate_a         Audio bitrates (exmaple: see
config file)");
    opt->addUsage(" -w  --width             Width related with the
bitrate");
    opt->addUsage(" -h  --height            Height related with the
bitrate");
    opt->addUsage(" -F  --flag              Flag, 0 to print the code
but don't run it");
    opt->addUsage("");

}
```

C: Instalación y utilización de los clientes DASH

TAPAS para Ubuntu 14.04 LTS

Obtención de dependencias:

```
& sudo add-apt-repository ppa:mc3man/trusty-media
& sudo apt-get update
& sudo apt-get install python-twisted python-twisted-bin python-
twisted-core python-twisted-web gstreamer0.10-plugins-*
gstreamer0.10-ffmpeg gstreamer0.10-tools python-gst0.1
libgstreamer0.10-dev python-scipy python-psutil
```

Utilización:

```
$ python play.py -a elegimos el controlador (max o conventional)
$ python play.py -u especificamos la URL
$ python play.py -m elegimos el media engine (nodec, gst, fake)
$ DEBUG=2 python play.py activa el modo depuración
```

GPAC para Ubuntu 14.04 LTS, también instalará MP4Box

Obtención de dependencias e instalación:

```
$ sudo apt-get install git
$ git clone https://github.com/gpac/gpac.git
$ sudo apt-get install subversion make pkg-config g++ zlib1g-dev
libfreetype6-dev libjpeg62-dev libpng12-dev libopenjpeg-dev libmad0-
dev libfaad-dev libogg-dev libvorbis-dev libtheora-dev liba52-0.7.4-
dev libavcodec-dev libavformat-dev libavutil-dev libswscale-dev
libavresample-dev libxv-dev x11proto-video-dev libgl1-mesa-dev
x11proto-gl-dev linux-sound-base libxvidcore-dev libssl-dev libjack-
dev libasound2-dev libpulse-dev libsdl1.2-dev dvb-apps libavcodec-
extra libavdevice-dev libmozjs185-dev
$ cd gpac
$ ./configure
$ make
$ sudo make install
```

Utilización:

```
$ MP4Client –logs especificamos las herramientas de log y los
niveles, herramientas (dash, media, rti…), niveles (debug, info,
error…)
$ MP$Client –rti especifica el nombre del archivo que contendrá la
información en run-time de la memoria, CPU y FPS del reproductor.
```

D: Scripts para las pruebas

A fin de que las pruebas en los distintos clientes se realizaran de la misma forma, se crearon unos scripts para automatizarlas, aunque debido a que no se encontró una manera de relacionar los scripts ejecutados en el Host con los intercambios en las condiciones de red definidas, las condiciones de red se cambiarán manualmente por lo que puede haber ciertas diferencias entre pruebas.

D.1: TAPAS

Script para la realización de la primeras dos pruebas. En la primera cambiaremos la configuración del ancho de banda de la 1ra a la 2da pasados unos 40 segundos, para cada una de las duraciones de segmento. En la segunda prueba utilizaremos la 3ra de las configuraciones de red, nuevamente para cada de las duraciones de segmento.

```
#!/bin/bash
# -*- ENCODING: UTF-8 -*-
cd Descargas/tapas-master
python play.py -u
http://172.16.34.2/bbb_sunflower/Xsec/ondemand/bbb_sunflower_onDemand
_dash.mpd
exit
```

Script para la realización de la tercera prueba:

```
#!/bin/bash
# -*- ENCODING: UTF-8 -*-
cd Descargas/tapas-master
python play.py -u
http://172.16.34.2/bbb_sunflower/Xsec/ondemand/bbb_sunflower_onDemand
_dash.mpd
sleep 70
python play.py -u
http://172.16.34.2/bbb_sunflower/Xsec/ondemand/bbb_sunflower_onDemand
_dash.mpd
exit
```

D.1: MP4Client

Script para la realización de la primera prueba, pasados 60 segundos se cambiará la calidad de la configuración de red:

```
#!/bin/bash
# -*- ENCODING: UTF-8 -*-
MP4Client -logs dash@debug
http://172.16.34.2/bbb_sunflower/Xsec/live/bbb_sunflower_live_dash.mp
d
exit
```

Script para la realización de la tercera prueba:

```
#!/bin/bash
# -*- ENCODING: UTF-8 -*-
MP4Client -logs dash@debug
http://172.16.34.2/bbb_sunflower/Xsec/live/bbb_sunflower_live_dash.mp
d
sleep 100
MP4Client -logs dash@debug
http://172.16.34.2/bbb_sunflower/Xsec/live/bbb_sunflower_live_dash.mp
d
exit
```

Glosario

AAC	Advanced Audio Coding
ABR	Adaptive Bitrate
AVC	Advanced Video Coding
CDN	Content Delivery Network
DASH	Dynamic Adaptive Streaming over HTTP
DHCP	Dynamic Host Configuration Protocol
DRM	Digital rights management
EWMA	Exponentially Weighted Moving Average
GOP	Group of Pictures
HTTP	Hypertext Transfer Protocol
MPD	Media Presentation Descriptor
MPEG	Moving Picture Experts Group
NAT	Network Address Translation
QoE	Quality of Experience
RTT	Round-Trip Time
SVC	Scalable Video Coding
VoD	Video on Demand
WAN	Wide Area Network
XML	Extensible Markup Language