UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica

Universitat Politècnica de València

Implementación de una solución de streaming bajo demanda usando DASH

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Alejandro Piqueres Chiva

Tutor: José Salvador Oliver Gil

Curso académico 2018/2019

Resumen

Aunque la TV tradicional ha tenido su evolución en las últimas décadas, internet también ha ido avanzando y se ha interpuesto inevitablemente en la trayectoria de la TV tradicional.

A día de hoy Internet es quien está al volante. El servicio de streaming que en los últimos años internet ha sido capaz de ofrecer a los usuarios hace que a día de hoy la popularidad de la TV tradicional decrezca, frente a un servicio de streaming que te ofrece el mismo contenido sin importar el momento ni el lugar, sin restricciones de tiempo o espacio, accesible con un dispositivo tan común como un teléfono móvil. Y es que el avance en los dispositivos tecnológicos se ha encargado de crear un atajo para que Internet sea el coche ganador en la carrera del entretenimiento. Pero este atajo tiene sus baches. Debido a este avance en los dispositivos, la cantidad y heterogeneidad de ellos capaces de visualizar contenido de streaming es masiva, y supone una gran complejidad a la hora de establecer un protocolo o un formato que sea compatible con todos los dispositivos. Es por eso que en este TFG se estudia un estándar que permite el consumo de contenido multimedia de forma adaptativa, ajustando principalmente la calidad de los contenidos a visualizar según el ancho de banda disponible por el usuario, desde cualquier dispositivo y aplanando el atajo a la meta. Ese estándar es MPEG-DASH.

Durante este proyecto se ha hecho un recorrido por la historia del streaming en internet, viendo así como ha evolucionado y ha dado lugar a lo que hoy en día es. Se ha creado una implementación local del estándar MPEG-DASH para poder estudiarlo simulando un entorno que muestre las características del estándar.

Índice general

1. Introducción.....5
1.1. Motivación.....5
1.2. Objetivos.....5

2. Streaming de video.....6
2.1. Definición de streaming.....6
2.2. Evolución.....6
2.3. Video bajo demanda vs transmisión en vivo.....7

2.4. Protocolos.....7
2.4.1. User Datagram Protocol (UDP).....7
2.4.2. Real-Time Transport Protocol (RTP).....8
2.4.3. Real-Time Streaming Protocol (RTSP).....8
2.4.4. Real-Time Transport Protocol (RTCP).....9
2.4.5. Transmission Control Protocol (TCP).....10
2.4.6. Hypertext Transfer Protocol (HTTP).....10
2.4.7. HTTP vs RTP.....11

2.5. Tipos de streaming basados en HTTP.....13
2.5.1. Descarga progresiva.....13
2.5.2. HTTP pseudo-streaming.....13
2.5.3. HTTP Adaptative Streaming (HAS).....13

2.6. Soluciones HAS propietarias.....15
2.6.1. Microsoft Smooth Streaming (MSS).....15
2.6.2. HTTP Live Streaming (HLS).....17
2.6.3. HTTP Dynamic Streaming (HDS).....22

2.7. Dynamic Adaptative Streaming over HTTP (MPEG-DASH).....25
2.7.1. Media Presentation Description (MPD).....26
2.7.2. Period.....26
2.7.3. Adaptation Set.....28
2.7.4. Representation.....29
2.7.5. Segments.....30
2.7.6. Tipos de segmentos.....31
2.7.7. Formato de los segmentos.....33
2.7.7.1. MPEG-2 Transport Stream.....33
2.7.7.2. ISO Base Media File Format ISO-BMFF.....33
2.7.7.2.1. Segment Index.....34
2.7.7.2.2. Initialization Segment.....34
2.7.7.2.3. Media Segments.....34
2.7.7.2.4. Profiles.....34

3. Implementación……..…….36

3.1. Escenario……..…..…….36
3.1.1. Servidor web……..……37
3.1.1.1. Internet Information Services (IIS)……..…….37
3.1.2. Cliente…….……38
3.1.3. FFmpeg…….……..…….41
3.1.4. MP4box……..…….42
3.1.5. NetLimiter……43

3.2. Proceso de implementación……..44

4. Pruebas……...…….48

4.1. Primera simulación……49
4.1.1. Segmentos de 2 segundos……..……49
4.1.2. Segmentos de 4 segundos……..……49
4.1.3. Segmentos de 8 segundos……..……50
4.1.4. Segmentos de 16 segundos……50
4.1.5. Conclusión primera simulación……50

4.2. Segunda simulación……...51
4.2.1. Segmentos de 2 segundos……..…..51
4.2.2. Segmentos de 4 segundos……...52
4.2.3. Segmentos de 8 segundos……...52
4.2.4. Conclusión segunda simulación……..53

5. Conclusiones……...54

6. Índice de figuras……..55

7. Índice de tablas……...56

8. Bibliografía……...57

9. Bibliografía adicional……..58

Capítulo 1

Introducción

1.1 Motivación

A día de hoy, el consumo de internet está por las nubes y solo va *en crescendo*. Cada vez es mayor el número de dispositivos conectados a internet y el tiempo invertido en ellos consumiendo los distintos servicios que internet ofrece. Se ha convertido en una necesidad y con ello se crea la necesidad de afrontar eficazmente todos los problemas que puedan ir surgiendo con esta tecnología. Aunque internet ofrece muchos servicios, en este tfg nos centramos en el servicio de streaming, donde la inmensa cantidad de dispositivos puede suponer un PROBLEMA para la compatibilidad, y se requiere un buen protocolo y estándar, una solución eficaz. Una solución eficaz a la necesidad de ofrecer contenido con la mayor calidad posible y sin discriminaciones, accesible a cualquiera, donde cada usuario representa un dispositivo diferente con características propias y limitaciones, las cuales no deben ser un impedimento para la solución eficaz que cumple con estas necesidades. Y eso es lo que es MPEG-DASH, un estándar que nos ofrece una alternativa a las implementaciones del servicio de streaming orientadas a dar soporte solo a sus propios dispositivos. Es poder ofrecer contenido con la máxima calidad posible a todos y cada uno de los usuarios sin excepciones.

1.2 Objetivo

El propósito de este proyecto es realizar una implementación de una solución de streaming bajo demanda basada en el protocolo DASH. Para ello, en este proyecto se desarrolla un escenario DASH básico que nos permite simular la entrega de contenido multimedia bajo demanda a través de los mecanismos del estándar MPEG-DASH. MPEG-DASH es una estándar que retransmite contenido de forma adaptativa. Esto implica que para simular un entorno DASH el servidor debe de disponer de diferentes alternativas del mismo contenido. Alternativas con distintas resoluciones y tasas de bits que ofrecer al cliente según el ancho de banda, entre otras cosas. Para ello se tienen que simular variaciones en el ancho de banda y así sacarle el máximo partido a este estándar, pudiendo así apreciar su punto fuerte en su totalidad que es su adaptabilidad.

Capítulo 2

Streaming de vídeo

El capítulo siguiente servirá para introducir el concepto de streaming y su evolución mediante datos históricos y los mismos protocolos, implicados en este servicio.

2.1 Definición de streaming

El streaming o retransmisión es la distribución continua de contenido multimedia a través de la red de manera que el usuario va consumiendo el contenido a medida que se descarga. Lo que principalmente se consigue es evitar tener que descargar un archivo por completo para poder consumirlo, ahorrando espacio de almacenamiento y tiempo.

2.2 Evolución

El origen del streaming, tuvo lugar en la época de los 90, cuando empezaron a aparecer los primeros ordenadores capaces de reproducir archivos multimedia. Ya que, aunque en la década anterior la informática ya había llegado "al mundo de a pie", la gran mayoría de ordenadores que se tenían aún no eran lo suficientemente potentes para procesar y recibir una señal en streaming. Por ejemplo, en 1995, un ordenador Dell tenía un procesador de 66MHz, 1Gb de espacio en disco duro, 8Mb de memoria RAM y valía alrededor de los 4000 dólares. Nada económico. Y, por si fuera poco, aspectos como el ancho de banda tampoco estaban a la altura.

Una vez se hubieran solventado estos problemas, sería necesario una cosa más, una red virtual capaz de transmitir contenido en multicast y unos portales a los que los navegantes pudiesen acceder para disfrutar del contenido retransmitido. Y ahí es donde entra Mbone, la red virtual que hizo posible la primera retransmisión en directo del grupo musical Severe Tire Damage, en 1993, el cual hizo historia al transmitir su concierto en vivo por todo el mundo. Al año siguiente, los Rolling Stones.

Más tarde, llegó Real Player, el primer sistema de reproducción de video en streaming, lanzado en 1997 por RealNetworks, compañía que en 1995 fue la primera en retransmitir un deporte, el partido de béisbol entre Los New York Yankees y los Seattle Mariners. Además, ese mismo año se empezaron a transmitir bandas sonoras.

Después de esto, a finales de la década de 2000, se popularizó la retransmisión de videos, ya que la contratación del suficiente ancho de banda para utilizar estos servicios se hizo lo suficientemente barato. Y esto fue gracias al avance tecnológico porque,

antes, si se quería retransmitir un evento en vivo, era necesario un equipo técnico, con varias personas que se ocupasen de los equipos, de las grabaciones, la edición del contenido y, sobre todo, de alquilar un satélite (DSNG). Sin embargo, a día de hoy, esto es mucho más simple y económico gracias a equipos que no necesitan ser transportados por más de una persona ni ser transmitidos a través de un satélite gracias a tecnologías de comunicaciones móviles como 3G, 4G con tarjetas SIM integradas en las cámaras.

Entonces, en 2005, llegó YouTube. La plataforma que revolucionó el consumo de contenido audiovisual en los hogares y, al poco tiempo, en 2008 surgió Spotify, plataforma para la reproducción de música vía streaming. Estas 2 plataformas llevaron al servicio streaming a la cumbre, ya que, en su práctica totalidad, son música y video lo que se consume a través de estos servicios.

Actualmente, una gran cantidad de plataformas se han subido al carro y ofrecen contenido audiovisual vía streaming. Netflix, HBO, Amazon Video, son solo algunas de ellas que ofrecen películas, series, documentales, de forma legal y otras como PopCorn que lo hacen de manera ilegal. Independientemente de su legalidad, todas estas plataformas tienen un éxito rotundo.

2.3 Video bajo demanda VS transmisión en vivo

La transmisión de video puede tomar la forma de contenido tanto en vivo como grabado. Con la transmisión en vivo, el contenido se reproduce a medida que se captura. Ejemplos de esto van desde chats de video y juegos interactivos hasta cámaras de endoscopia y drones de transmisión. El video bajo demanda, por otro lado, describe el contenido pregrabado que los usuarios conectados a Internet pueden solicitar. Algunas de las plataformas más populares que lo utilizan incluyen Netflix, Amazon Prime, HBO y YouTube. Stranger Things de Netflix y Game of Thrones de HBO son ejemplos de contenido bajo demanda.

2.4 Protocolos

2.4.1 User Datagram Protocol UDP

Un gran número de servicios de transmisión de video han empleado UDP como protocolo de transporte. De hecho, UDP es un simple protocolo de transmisión sin conexión sin diálogo de intercambio y no ofrece la garantía de entrega, pedido o retransmisión. Estas características simplifican la transmisión de datos y reducen el retardo de extremo a extremo lo que hace a UDP adecuado para aplicaciones en tiempo real, como la transmisión de video en vivo, que son sensibles al retraso. En consecuencia, para transferir flujos de datos en tiempo real, UDP se utilizó ampliamente

como un protocolo de transporte junto con protocolos de la capa de aplicación específicos, los protocolos de la familia RTP (Protocolo de transporte en tiempo real), dando lugar al Streaming tradicional.

2.4.2 Real-Time Transport Protocol RTP

RTP[3] es un protocolo de la capa de aplicación que proporciona funciones para la transmisión de información de extremo a extremo adecuadas para aplicaciones que transmiten datos en tiempo real a través de servicios de red de multicast o unicast. No aborda la reserva de recursos y no garantiza la calidad de servicio para los servicios en tiempo real. Además, otro protocolo de capa de aplicación, llamado Real Time Streaming Protocol (RTSP), se ha utilizado generalmente con RTP.

2.4.3 Real Time Streaming Protocol RTSP

RTSP es similar a HTTP en cuanto a sintaxis y operación, lo que dota al protocolo de cierta seguridad de la que carece RTP, ya que las formas de autentificación HTTP son directamente aplicables. Generalmente el cliente envía al servidor las peticiones RTSP, basadas en peticiones HTTP, aunque este protocolo permite que tanto el servidor como el cliente puedan enviar peticiones.

Las principales peticiones RTSP son:

DESCRIBE: Describe el objeto de la URL RTSP. Contiene datos necesarios para la reproducción. Se realiza al inicio.

SETUP: Indica la manera en la que se debe transportar un flujo de datos. Contiene la URL del stream multimedia y un especificador de transporte. El servidor resconde con la confirmación de los parámetros seleccionados y añade las partes no seleccionadas. Cada stream debe configurarse con SETUP como paso previo de enviar una petición de PLAY.

PLAY: Petición para que el servidor comience a enviar los datos de los streams seleccionados.

PAUSE: Detiene la reproducción de uno o más streams de forma temporal. Reanudable con la petición PLAY.

TEARDOWN: Finaliza la sesión. Los streams se detienen y se liberan los recursos asociados.

En la figura 2.1 se muestra un esquema del uso de estas peticiones entre un servidor RTSP y un cliente

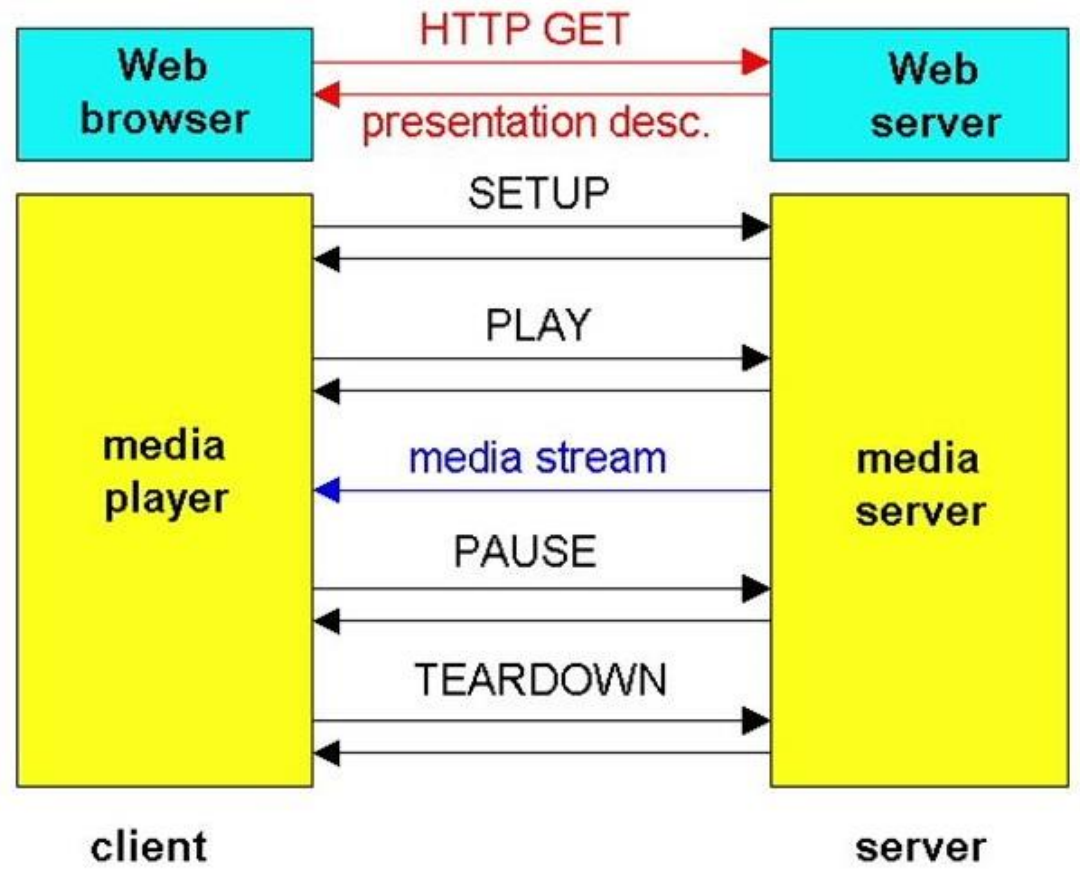

Figura 2.1: Esquema de comunicación entre un cliente (Web Browser) y el servidor RTSP (Web Server)

Sin embargo, el hecho de utilizar TCP para realizar estas peticiones no implica que los datos sean enviados por este mismo protocolo, pues RTSP es independiente del protocolo de transporte y puede enviar los datos usando tanto UDP, como TCP, aunque este último protocolo no es común en la transmisión de datos multimedia, ya que TCP puede provocar demoras y retrasos en el audio/vídeo debido a sistemas de control de errores y pérdidas.

2.4.4 Real-Time Transport Control Protocol RTCP

El transporte de datos se incrementa mediante un protocolo de control RTCP que proporciona un control mínimo y una funcionalidad de identificación de flujo para permitir la gestión de la entrega de medios para redes de multidifusión. Su función principal es informar acerca de la calidad del servicio.

Durante una sesión, este protocolo transmite paquetes de control de forma periódica a todos para controlar el estado de la conexión constantemente..

RTCP define varios tipos de paquetes:

Sender Report (SR): El emisor lo envía periódicamente informando sobre transmisión y recepción de los paquetes RTP enviados en un determinado intervalo de tiempo.

Receiver Report (RR): Sirve para el envío de estadísticas sobre la recepción por parte de los participantes.

Source Description (SDES): Usado para hacerles llegar a los participantes el CNAME3. Se puede utilizar tambien para el envío de otra info como número de teléfono, correo electrónico, nombre,…

End of participation(BYE): Indica la finalización de la participación en una sesión.

Application-specific message (APP): Diseña extensiones específicas de la aplicación para el protocolo RTCP.

2.4.5 Transmission Control Protocol TCP

TCP es el protocolo de la capa de transporte usado por HTTP. A diferencia de UDP, protocolo que se emplea en RTP, TCP es un protocolo orientado a la conexión, pues el protocolo TCP intercambia información de control(negociación) previamente al intercambio de información, en modo full-dúplex. TCP también ofrece un servicio de transferencia fiable, ya que se asegura de que los mensajes llegan en el orden correcto y sin errores. Además, trata de ofrecer una mejor transferencia controlando la congestión de la red transitada y los recursos del emisor y receptor.

2.4.6 Hypertext Transfer Protocol HTTP

HTTP es el protocolo utilizado para intercambiar o transferir hipertexto y el más usado en la actualidad para la transferencia de elementos. Funciona como un protocolo de solicitud-respuesta basado en un modelo computacional cliente-servidor. Está situado en la capa de aplicación de la pila OSI.

Creado en 1989 por Tim Berners-Lee, “padre de la WWW (World Wide Web)” y creador también de HTML (HyperText Markup Language), como sistema de comunicación en el CERN(Organización Europea para la Investigación Nuclear), donde el creador trabajaba como científico en aquel entonces, antes de fundar el W3C (World Wide Web Consortium). HTTP fue estandarizado por el IETF(Internet Engineering Task Force) y el W3C.[1]

Su funcionamiento comienza con el inicio de una conexión TCP con el servidor. Acto seguido, la petición se envía a través de su interfaz de socket, mediante el cual, accede a los recursos del servidor. Por defecto, el servidor no cierra la conexión hasta que el

cliente no acabe. Es lo que se conoce por conexión persistente. En caso de usar conexiones no persistentes, el cliente establece una nueva conexión para cada solicitud, que es lo que más sobrecarga genera. Por lo tanto, con las conexiones persistentes se evita tener que generar tantas conexiones, evitando también la estrategia de inicio lento propio del control de congestión, siendo mayor el rendimiento con el tiempo y reduciendo el uso de memoria, cpu, latencia, etc.

METODOS DE PETICIÓN:GET,HEAD,POST,PUT…

GET: Solicita una representación de un recurso específico. Las peticiones que usan el método GET sólo deben recuperar datos.

HEAD: Pide una respuesta idéntica a la de una petición GET, pero sin el cuerpo de la respuesta.

POST: Se utiliza para enviar una entidad a un recurso en específico, causando a menudo un cambio en el estado o efectos secundarios en el servidor.

PUT: Reemplaza todas las representaciones actuales del recurso de destino con la carga útil de la petición.

DELETE: Borra un recurso en específico.

CONNECT: Establece un túnel hacia el servidor identificado por el recurso.

OPTIONS: Es utilizado para describir las opciones de comunicación para el recurso de destino.

TRACE: Realiza una prueba de bucle de retorno de mensaje a lo largo de la ruta al recurso de destino.

PATCH: Es utilizado para aplicar modificaciones parciales a un recurso.

2.4.7 HTTP vs RTP

La entrega de contenido de video a través de Internet comenzó en la década de 1990, siendo el principal desafío la entrega oportuna y el consumo de grandes cantidades de datos. En una época en la que solo habían redes IP gestionadas RTP resultó ser un protocolo eficiente. Pero a día de hoy, las redes administradas han sido reemplazadas por redes de distribución de contenido (CDN), muchas de las cuales no son compatibles con la transmisión RTP.

Además, los paquetes RTP a menudo tienen problemas con firewalls dado que es el servidor quien controla el flujo de transmisión de video. Esto significa que es el servidor quien envía paquetes al cliente sin parar y el firewall no tiene forma de saber si el envío se basa en una solicitud de un cliente o si se trata de un cierto ataque, como un ataque de denegación de servicio, por lo tanto, sus paquetes a menudo son interceptados por firewalls y / o Traducciones de Direcciones de Red (NAT).Otro inconveniente es que RTP hace uso de servidores especiales para transmitir y además requiere que el

servidor administre una sesión de transmisión por separado para cada cliente, haciendo un seguimiento del estado de cada dispositivo, lo que hace que los despliegues a gran escala requieran muchos recursos.

Gracias al aumento del ancho de banda de Internet y el tremendo crecimiento de Internet, el contenido multimedia ahora se puede entregar de manera eficiente en segmentos más grandes usando HTTP, además, la transmisión basada en HTTP o HTTP streaming, tiene varios beneficios que dan solución a estos inconvenientes: Primero, la infraestructura de Internet ha evolucionado para ser compatible con HTTP. Por ejemplo, los CDN proporcionan cachés de borde localizados, consiguiendo que el tráfico de larga distancia disminuya.

En segundo lugar, en HTTP es el cliente quien controla la transmisión. Esto se basa en el hecho de que el cliente, en la petición que envía, indica la cantidad de información que espera recibir en la respuesta, de modo que el firewall puede comprobar que el servidor está enviando esa cantidad, por lo que el bloqueo de la conexión debido a problemas de seguridad rara vez ocurre.

Finalmente, otro beneficio de HTTP streaming es el uso de servidores web, además, el cliente es quien administra la transmisión, lo que significa que el servidor no necesita mantener la información de estado de la sesión del cliente. Esto se traduce en una muy buena escalabilidad.

Pero el “resurgir” de HTTP como protocolo de streaming no se le puede achacar solo al aumento de ancho de banda y el crecimiento de Internet. HTTP no tuvo lugar después de RTP, sino al contrario, pero éste no ha sido siempre lo que es a día de hoy. En su día, versiones anteriores de HTTP resultaron ser menos convenientes que el protocolo RTP. Pero con el tiempo, versión tras versión, HTTP ha ido mejorando haciendo al protocolo atractivo para el servicio de streaming.

Estas son las versiones por las que ha pasado HTTP:

0.9 (lanzada en 1991)
Una única método de petición posible, GET. No soporta cabeceras. Como esta versión no soporta POST, el cliente no puede enviarle mucha información al servidor.

HTTP/1.0 (mayo de 1996)
Esta es la primera revisión del protocolo que especifica su versión en las comunicaciones. Permite los métodos de petición GET, HEAD y POST.

HTTP/1.1 (junio de 1999)
Versión más usada actualmente; Las conexiones persistentes están activadas por defecto y funcionan bien con los proxies. También permite al cliente enviar múltiples peticiones a la vez por la misma conexión (pipelining) lo que hace posible eliminar el tiempo de RTT (Round-Trip delay) por cada petición.

HTTP/1.2 (febrero de 2000)

Fue solo experimental

HTTP/2 (mayo de 2015)
Mejoras en el empaquetamiento los datos y en el transporte. Aumento en el uso de una única conexión (menos conexiones), eliminación de información duplicada, compresión de cabeceras, descarte de información mediante el algoritmo HPACK, servicio 'server push' (predicciones). HTTP pasa a ser un protocolo binario, de interpretación mucho más sencilla, más compacto en el transporte y menos propenso a errores. Antes estaba basado en texto.

Estos motivos han conllevado al desarrollo de investigaciones sobre HTTP Streaming por parte de grandes empresas, las cuales, algunas han desarrollado su propia plataforma como por ejemplo HTTP Live Streaming de Apple, Smooth Streaming de Microsoft, y HTTP Dynamic Streaming de Adobe.

2.5 Tipos de streaming basados en HTTP

Para la transmisión de video basada en HTTP, existen esencialmente tres categorías que se muestran de forma resumida en la tabla 2.1. Estas categorías son:

2.5.1 Descarga Progresiva

Esta técnica se basa en la descarga de ficheros de datos de un servidor HTTP. En este método la reproducción de video puede comenzar tan pronto como la cantidad necesaria de datos, que puede ser de unos 2 a 10 segundos de información de video, se descargue y luego se almacene en el búfer en su memoria.

Como está almacenado en su memoria, puede decidir cuándo pausarlo o puede avanzar y retroceder para ver las partes del video. Pero no podrá avanzar más allá de lo que haya descargado.

No necesita la aplicación de protocolos especiales, pero necesita que se defina un formato que será procesado basado en el contenido parcial que se tiene. La descarga se realiza usando el máximo ancho de banda del que disponen cliente y servidor, y no hay ningún control para evitar cortes en la reproducción, así que, si no se cuenta con el ancho de banda suficiente, esto afectará gravemente a la visualización.

2.5.2 HTTP Pseudo Streaming

Es como la descarga progresiva, pero agregando la posibilidad de adelantar o retroceder la reproducción sin la necesidad de descargar las partes que no se hayan descargado en el salto. Como en muchos casos no se visualiza el contenido entero, de esta manera se reduce el ancho de banda empleado. Pseudo-streaming requiere adaptaciones tanto para el lado del cliente como para el lado del servidor.

2.5.3 HTTP Adaptative Stream HAS

En éste método las características del video a reproducir se escogen de forma dinámica en base generalmente al ancho de banda y la capacidad de CPU del dispositivo del cliente.

Para conseguir esto, el video a transmitir debe ser codificado y divido en segmentos de pocos segundos. Como resultado se tiene el video original segmentado a diferentes tasas de bits y resoluciones de video como se muestra en la Figura 2.1.

En una sesión de transmisión de HAS típica, el cliente comienza realizando una solicitud HTTP al servidor para obtener información de las diferentes representaciones de audio y video disponibles, que están contenidas en un archivo específico, el manifiesto. Este archivo le informara de la disponibilidad de las representaciones y como solicitarlas.

Por lo general, el cliente envía una solicitud HTTP GET desde el reproductor al servidor HAS por cada segmento, el cual es escogido por el cliente según la disponibilidad de éste, el ancho de banda y la ocupación del búfer de reproducción.

El objetivo de HAS es ofrecer la mejor calidad de video y la mejor experiencia de visualización posible, sin importar la conexión, el software o el dispositivo. Esto implica, más situaciones donde el cliente consume contenido desde un dispositivo dinámico como puede ser un móvil o una Tablet, donde su ancho de banda estará determinado según el operador contratado, la intensidad de la señal, que no es constante (interferencias en conexiones inalámbricas) y el hardware del móvil, ya que no todos los móviles suportan 4G. Entonces esto supone que hay que ser estricto con el control del ancho de banda. Es por ello que en HAS, el cliente verifica la condición de la red de forma periódica, según la duración de los segmentos. De esta forma si se desea un tipo diferente de velocidad de datos o un tipo diferente de calidad de video, en el siguiente fragmento se puede cambiar.

De esta forma se soluciona la inanición del buffer o buffering que se da cuando el reproductor consume todo el contenido del buffer y éste se queda vacío congelando la visualización. Básicamente se trata de mantener el nivel del buffer en un nivel estable. Si por ejemplo se perdieran paquetes o las condiciones de la red se están degradando, el buffer empezaría a vaciarse conforme se consume. Al detectar esto, el cliente solicitará segmentos de menor calidad de video de forma gradual hasta conseguir la estabilidad de antes.

En general, al principio de una sesión HAS, el reproductor trata de llenar el buffer de reproducción rápidamente seleccionando el nivel de calidad de video más bajo. Una vez lleno, el reproductor empieza a decodificar y reproducir el contenido del buffer pero no solicita más segmentos hasta que cierta cantidad del buffer se haya consumido. Esto crea periodos de inactividad en los que el reproductor no descarga, solo espera a que parte del buffer sea consumido se puede ver muy bien en la plataforma de YouTube con la opción estadísticas para nerf. Éstas técnicas permiten a HAS soportar la transmisión en vivo, y no solo bajo demanda como sus antecesores. Además, también trata de hacer coincidir las velocidades con las que el buffer se “llena y vacía”, es decir, que el servidor transmita a la tasa de bits de codificación que coincida con la tasa de consumo.

Entonces, la velocidad de bits de codificación que se establece se ajusta a la velocidad de visualización del video y permite reproducir el contenido con la mayor calidad posible dentro de un marco de seguridad que garantice estabilidad.

Como se ha mencionado antes, las principales empresas han implementado su propia solución como HTTP Live Streaming (HDS), de Apple para iOS, HTTP Dynamic Streaming (HDS) de Adobe y Microsoft Smooth Streaming (MSS). Las tres soluciones tienen un funcionamiento muy similar, pero han sido creadas por empresas privadas y por lo tanto adaptadas a sus propias tecnologías por lo que hay incompatibilidades con dispositivos. También hay diferencias en la duración y el formato de los segmentos y el fichero manifiesto. Esto impulsó a crear un estándar que fuera capaz de aunar estas tecnologías y eliminar todo tipo de restricción. Este estándar es MPEG-DASH.

<table>
<tr><td></td><td colspan="5">Streaming</td></tr>
<tr><td rowspan="2"></td><td colspan="2" rowspan="2">Streaming Tradicional</td><td colspan="3">Streaming Alternativo</td></tr>
<tr><td>Progressive Download</td><td>HTTP Pseudostreaming</td><td>Dynamic Adaptive Streaming over HTTP</td></tr>
<tr><td>Protocolo de Aplicación</td><td colspan="2">RTP-RTSP-RTCP</td><td colspan="3">HTTP</td></tr>
<tr><td>Protocolos de Transporte</td><td>UDP</td><td>TCP</td><td colspan="3">TCP</td></tr>
<tr><td>Protocolo de Red</td><td colspan="5">IP</td></tr>
<tr><td>Soporte Multicast</td><td>SI</td><td colspan="4">No</td></tr>
<tr><td>Soporte Unicast</td><td colspan="5">Si</td></tr>
<tr><td>Ante una perdidda de un paquete</td><td>El nivel de aplicación determina que hara con el paquete perdido</td><td colspan="4">Retransmisión del mismo</td></tr>
<tr><td>Capacidad de Multiplexación de varios Stream</td><td colspan="2">SI</td><td colspan="3">NO</td></tr>
<tr><td>Problemas de NATEO y FIREWALL</td><td>Si</td><td>NO</td><td colspan="3">NO</td></tr>
<tr><td>Quality Service</td><td colspan="2">SI</td><td colspan="3">NO</td></tr>
<tr><td>Permancia en cache</td><td colspan="2">No</td><td>Si</td><td>SI</td><td>Solo en fragmentos de tamño reducido</td></tr>
<tr><td rowspan="3">Soportado por Browsers</td><td colspan="2">No</td><td colspan="3">SI</td></tr>
<tr><td colspan="5">Necesario para la Reproduccción</td></tr>
<tr><td colspan="2">Reproductor</td><td colspan="3">Browser+Plugin / Reproductor</td></tr>
<tr><td>Control sobre la transmisión</td><td colspan="2">Si</td><td>No</td><td>SI</td><td>SI</td></tr>
<tr><td>Necesidad de Servidores</td><td colspan="2">Servidores de Streaming</td><td colspan="3">Servidores Web</td></tr>
<tr><td rowspan="2">Inmediates</td><td colspan="5">Se reproduce conforme llega</td></tr>
<tr><td colspan="2">Solo usa el ancho de banda que necesita</td><td colspan="3">Utilizan máximo ancho de banda para descarga</td></tr>
<tr><td>Capacidad de Adaptación al ancho de banda</td><td colspan="2">Si</td><td>NO</td><td>No</td><td>SI</td></tr>
<tr><td>Opción de modificar algortimo de compresión en curso</td><td colspan="2">Si</td><td colspan="3">NO</td></tr>
</table>

Tabla 2.1: Tabla Resumen de las diferentes tecnologías de streaming

A continuación, se explica brevemente en qué consisten las soluciones privadas y más extensamente el estándar MPEGDASH.

2.6 SOLUCIONES HAS PROPIETARIAS

2.6.1 Microsoft Smooth Streaming (MSS)

En la solución de Microsoft el servidor mantiene un archivo MP4 ISO Base Media File Format por cada resolución y tasa de bits a la que se ha codificado la fuente audio/video. Estos archivos están organizados internamente en fragmentos de 2 segundos a los que se puede acceder por rangos de bytes, pero realmente el archivo MP4 contiene el contenido al completo a esa determinada tasa de bits. Cuando el cliente hace una solicitud de un intervalo de tiempo específico, el servidor la traduce en rangos de bytes en el archivo MP4 y extrae los fragmentos correspondientes, de forma que esta división del contenido es tan solo virtual. Esto ofrece enormes beneficios de gestión de archivos.

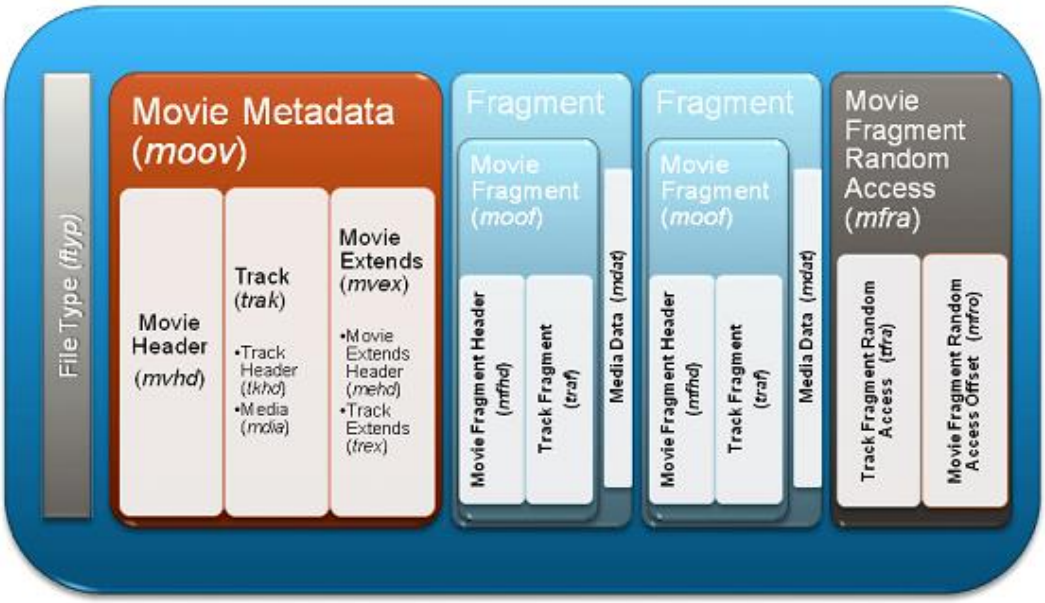

Figura 2.2: Formato de un fragmento de Microsoft Smooth Streaming.[6]

Como se puede apreciar en la figura 2.2, cada archivo MP4 contiene metadatos relacionados con el archivo multimedia completo (moov) y con cada fragmento (moof) además de datos multimedia de cada fragmento (mdat). El archivo se cierra con índice (mfra) que permite al servidor localizar fácilmente los ficheros.

Lo primero que el cliente solicita al servidor es el manifiesto. El archivo manifiesto de MSS está en formato XML.

Existen 2 archivos manifiesto en MSS:

El archivo manifiesto del cliente (.ismc). Describe los códecs empleados, las tasas de codificación y los códigos de tiempo de los fragmentos disponibles.

El archivo manifiesto del servidor (.ism). Describe la relación entre los contenidos multimedia, las tasas de bits y los archivos MP4.

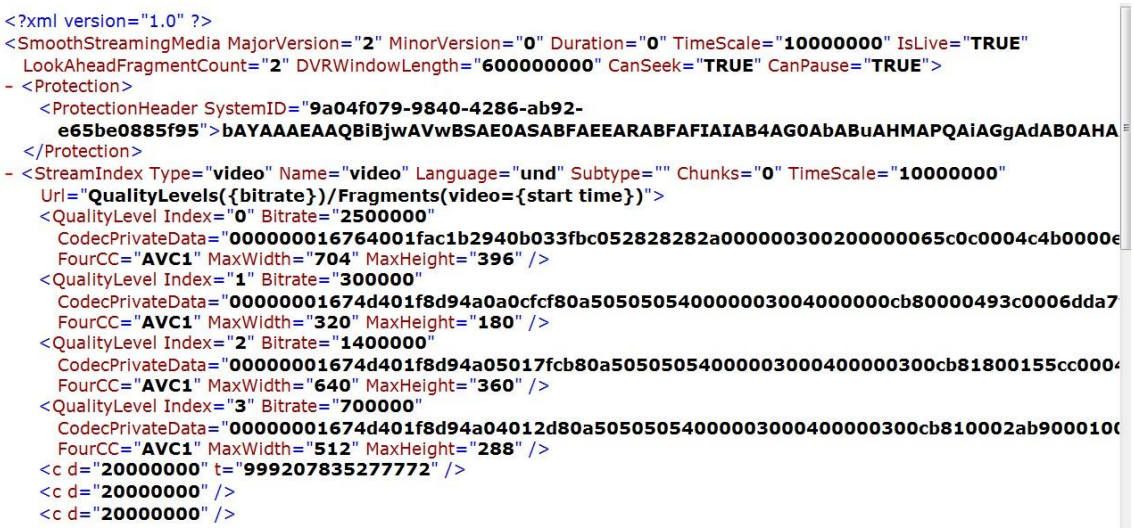

Figura 2.3: Archivo manifiesto de Microsoft Smooth Streaming.

Una vez el cliente obtiene su manifiesto sabe cómo solicitar fragmentos en forma de una URL, como por ejemplo:

“http://film.pax.com/MSI.ism/QualityLevels(320000)/Fragments(video=500000000)”

donde (320000) representa la tasa de bits codificada y (500000000) el desplazamiento de inicio del fragmento expresado en 100ns la unidad. Estos valores están representados en el manifiesto del cliente.

Con estos datos se realiza una búsqueda en el manifiesto del servidor (.ism) para la tasa de bits solicitada y lo mapea con el archivo de video (.ismv) y/o audio (.isma) asociado/s. El servidor, a partir de (tfra), selecciona del archivo MP4 que coincide con la tasa de bits, el cuadro de fragmento (moof + mdat) correspondiente que debe enviar. Una vez elegido es extraído y enviado al cliente. El fragmento queda almacenado en caché para que no lo tenga que pedir otro cliente al servidor origen.

Los códecs que soporta la solución de Microsoft son H.264 para video y AVC para audio

En cuanto a la compatibilidad, Microsoft ha enfocado su solución principalmente sus propios (XBOX, Windows Phone, etc). Los dispositivos iOS no pueden utilizar esta solución.

2.6.2 HTTP Live Streaming (HLS)

La solución de Apple está basada en 3 elementos principales. Un servidor que se encarga de codificar y segmentar los archivos fuente video/audio que le van llegando, una CDN donde se alojan los servidores que almacenan estos segmentos, y el propio cliente, que los solicita.

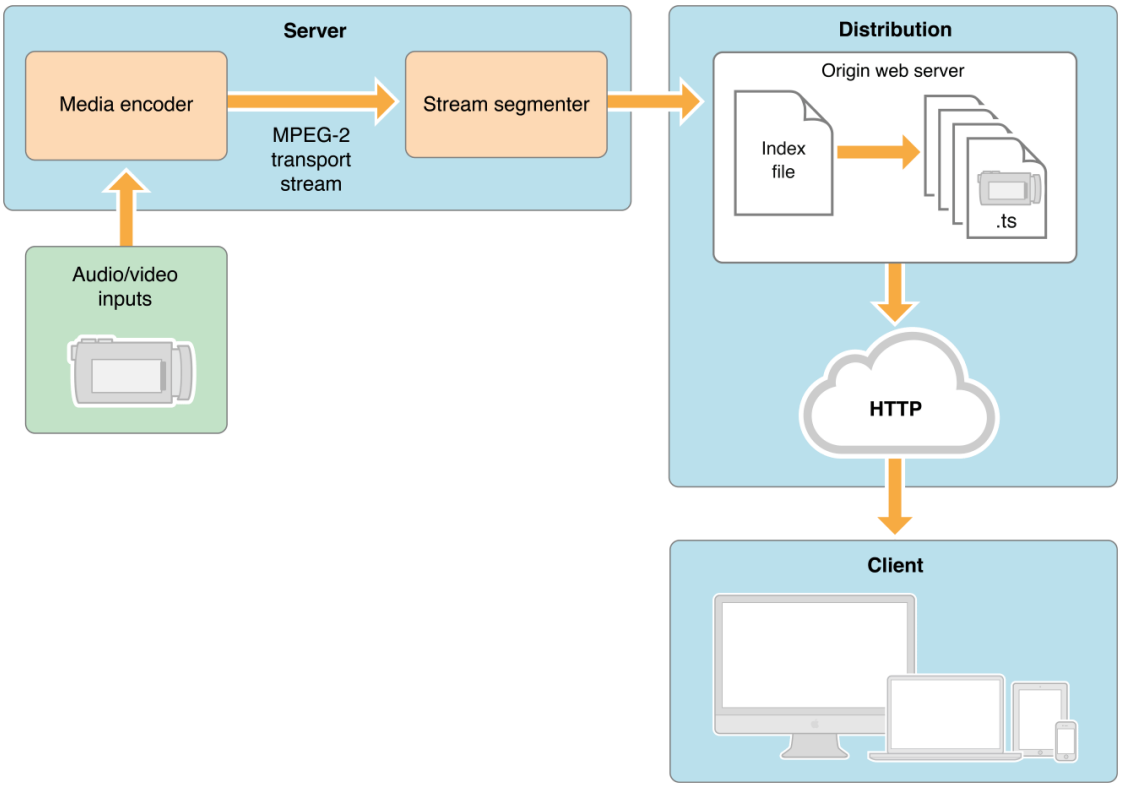

Figura 2.4: Representación del funcionamiento y la arquitectura de Apple HTTP Live Streaming.[7]

Una vez el servidor codifica el flujo video/audio, lo encapsula en formato MPEG 2TS para transportarlo al segmentador donde es dividido en fragmentos de 10 segundos. Durante la segmentación, un número que se incrementa progresivamente le es asignado a cada fragmento a medida que se van creando. Una vez fragmentadas las distintas versiones del video original el segmentador crea un archivo índice (Index File) o manifiesto (.m3u8). Este archivo indica las calidades en las que se ha codificado el archivo fuente y por lo tanto están disponibles.

Los segmentos son enviados como archivos (.ts) junto con el archivo manifiesto al conjunto de servidores que forman la CDN. Estos servidores se encargan de aceptar las

solicitudes de los clientes y de distribuir el contenido. Dentro de la CDN hay servidores que mantienen contenido en la caché llamados servidores intermedios o mid-tier. Cuando un servidor frontera (Edge-Server) no tiene el contenido que le solicita el cliente, puede acudir a estos servidores en lugar de acudir al servidor de origen, para así reducir el número de consultas que reciben.

Por último, el cliente elige la calidad del contenido a descargar, la descarga y reproduce sin cortes.

El manifiesto de la solución de Apple tiene la peculiaridad de que se actualiza de forma periódica, y se debe volver a pedir para que el cliente tenga las referencias a los fragmentos que van llegando. Esta actualización es necesaria por la estructura del manifiesto.

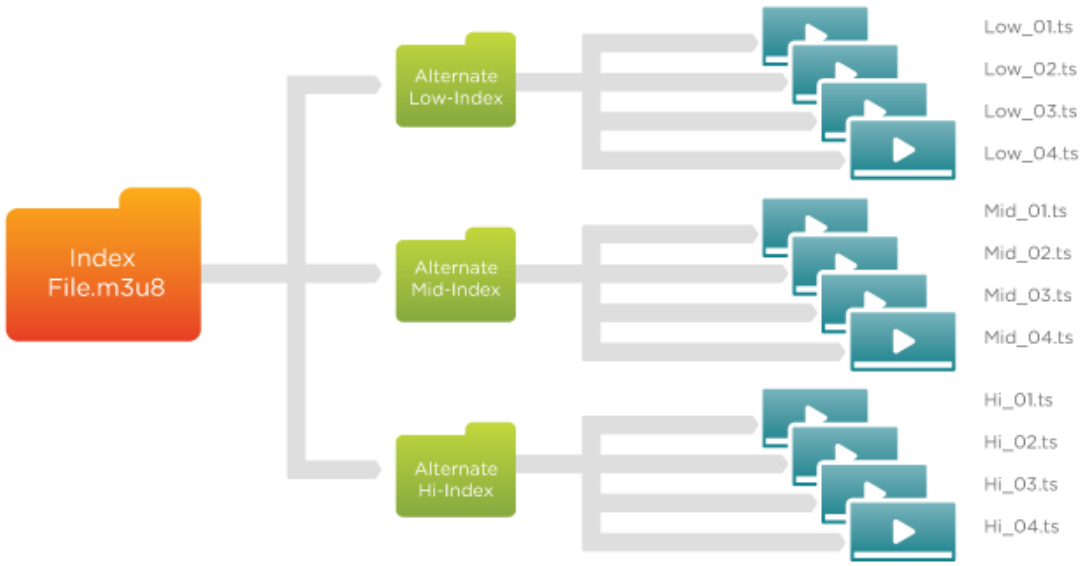

Figura 2.5: Representación de la estructura de archivos de Apple HTTP Live Streaming.[8]

```
#EXTM3U
#EXT-X-STREAM-INF:PROGRAM-ID=1,BANDWIDTH=800000,RESOLUTION=720x404
http://nubeoxlivegeo-live.hls.adaptive.level3.net/hls-live/axnhd/_definst_/liveevent/livestream1.m3u8
#EXT-X-STREAM-INF:PROGRAM-ID=1,BANDWIDTH=96000,CODECS="mp4a.40.2"
http://nubeoxlivegeo-live.hls.adaptive.level3.net/hls-live/audio-only-aac/axnhd/_definst_/liveevent/livestream1.m3u8
#EXT-X-STREAM-INF:PROGRAM-ID=1,BANDWIDTH=1200000,RESOLUTION=720x404
http://nubeoxlivegeo-live.hls.adaptive.level3.net/hls-live/axnhd/_definst_/liveevent/livestream2.m3u8
#EXT-X-STREAM-INF:PROGRAM-ID=1,BANDWIDTH=1600000,RESOLUTION=1280x720
http://nubeoxlivegeo-live.hls.adaptive.level3.net/hls-live/axnhd/_definst_/liveevent/livestream3.m3u8
#EXT-X-STREAM-INF:PROGRAM-ID=1,BANDWIDTH=2300000,RESOLUTION=1280x720
http://nubeoxlivegeo-live.hls.adaptive.level3.net/hls-live/axnhd/_definst_/liveevent/livestream4.m3u8
#EXT-X-FAXS-CM:URI="/hls-live/axnhd/_definst_/liveevent/livestream1.drmmeta"
```

Figura 2.6: Manifiesto Apple HTTP Live Streaming

El manifiesto contiene referencias que apuntan a otros archivos .m3u8, llamados sub-manifiestos. Cada sub-manifiesto corresponde a una de las calidades a las que ha sido codificado el archivo original y contiene la secuencia de fragmentos correspondiente a esa versión de codificación.

```
#EXTM3U
#EXT-X-MEDIA-SEQUENCE:380284
#EXT-X-ALLOW-CACHE:NO
#EXT-X-VERSION:2
#EXT-X-FAXS-CM:URI="livestream1.drmmeta"
#EXT-X-KEY:METHOD=AES-128,URI="https://keyserver1.adobeaccess.vualto.com/faxsks/vualto/key",IV=0x3d32059bb9268f76946de252b7701
#EXT-X-TARGETDURATION:8
#EXTINF:8,
../../../../hls-live/streams/axnhd/events/_definst_/liveevent/livestream1Num380284.ts
#EXTINF:8,
../../../../hls-live/streams/axnhd/events/_definst_/liveevent/livestream1Num380285.ts
#EXTINF:8,
../../../../hls-live/streams/axnhd/events/_definst_/liveevent/livestream1Num380286.ts
#EXTINF:8,
../../../../hls-live/streams/axnhd/events/_definst_/liveevent/livestream1Num380287.ts
#EXTINF:8,
../../../../hls-live/streams/axnhd/events/_definst_/liveevent/livestream1Num380288.ts
#EXTINF:8,
../../../../hls-live/streams/axnhd/events/_definst_/liveevent/livestream1Num380289.ts
```

Figura 2.7: Submanifiesto de Apple HTTP Live Streaming

Como se ve en la figura 2.7, los fragmentos que componen el submanifiesto son distintos a los del submanifiesto de la figura 2.8, que tienen un número que los identifica distinto. Si se vuelve a pedir el sub-manifiesto más tarde, la secuencia de números de los fragmentos es distinta, ha progresado.

```
#EXTM3U
#EXT-X-MEDIA-SEQUENCE:380294
#EXT-X-ALLOW-CACHE:NO
#EXT-X-VERSION:2
#EXT-X-FAXS-CM:URI="livestream1.drmmeta"
#EXT-X-KEY:METHOD=AES-128,URI="https://keyserver1.adobeaccess.vualto.com/faxsks/vualto/key",IV=0x3d32059bb9268f76946de252b770
#EXT-X-TARGETDURATION:8
#EXTINF:8,
../../../../hls-live/streams/axnhd/events/_definst_/liveevent/livestream1Num380294.ts
#EXTINF:8,
../../../../hls-live/streams/axnhd/events/_definst_/liveevent/livestream1Num380295.ts
#EXTINF:8,
../../../../hls-live/streams/axnhd/events/_definst_/liveevent/livestream1Num380296.ts
#EXTINF:8,
../../../../hls-live/streams/axnhd/events/_definst_/liveevent/livestream1Num380297.ts
#EXTINF:8,
../../../../hls-live/streams/axnhd/events/_definst_/liveevent/livestream1Num380298.ts
#EXTINF:8,
../../../../hls-live/streams/axnhd/events/_definst_/liveevent/livestream1Num380299.ts
```

Figura 2.8: Submanifiesto de Apple HTTP Live Streaming

Eso ocurre porque los archivos ".ts" se van actualizando de forma secuencial, lo que significa que, si se quiere poder acceder a los fragmentos que vayan llegando, es preciso ir pidiendo el archivo manifiesto de forma periódica.

Es por esto que la solución de HLS decida dividir el contenido en fragmentos tan largos (relativamente mucho más largos que las soluciones de Microsoft (2 segundos) y de Adobe (4 segundos)). Cuanto más largos sean los fragmentos, menor será el número de fragmentos en los que se divida el contenido, lo que se traduce en menos actualizaciones del manifiesto.

Los códecs que HLS soporta son:

- Vídeo:
    - o H.264 Baseline Level 3.0, Baseline Level 3.1, Main Level 3.1, and High Profile Level 4.1 [6] [7].
- Audio:
    - o HE-AAC
    - o AAC-LC hasta 48 kHz, audio estéreo.
    - o MP3 (MPEG-1 Audio Layer 3) 8 kHz a 48 kHz, audio estéreo.
    - o AC-3 (para Apple TV, solo en modo “pass-through”).

Respecto a la compatibilidad de dispositivos, con esta solución se tiene que, al igual que Microsoft ha hecho con su solución, Apple ha desarrollado esta solución enfocada en sus propios productos como son: iPhone, iPod, iPad, entre otros. Sin embargo, más tarde, HLS incluyó el formato MPEG4 en su solución permitiendo así la compatibilidad con un rango más amplio de dispositivos. De hecho, HLS aún es utilizado en muchas de las plataformas de streaming más importantes como Twitch, tal y como se muestra en la figura 2.9.

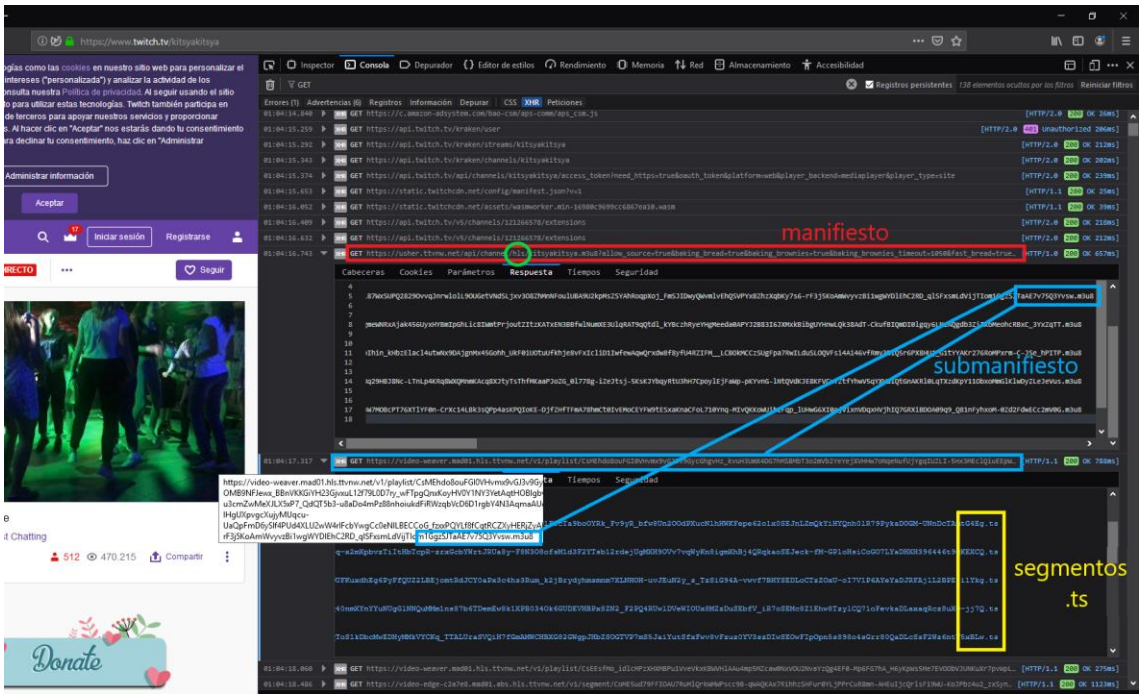

Figura 2.9: Captura del tráfico de contenido en vivo de twitch.tv

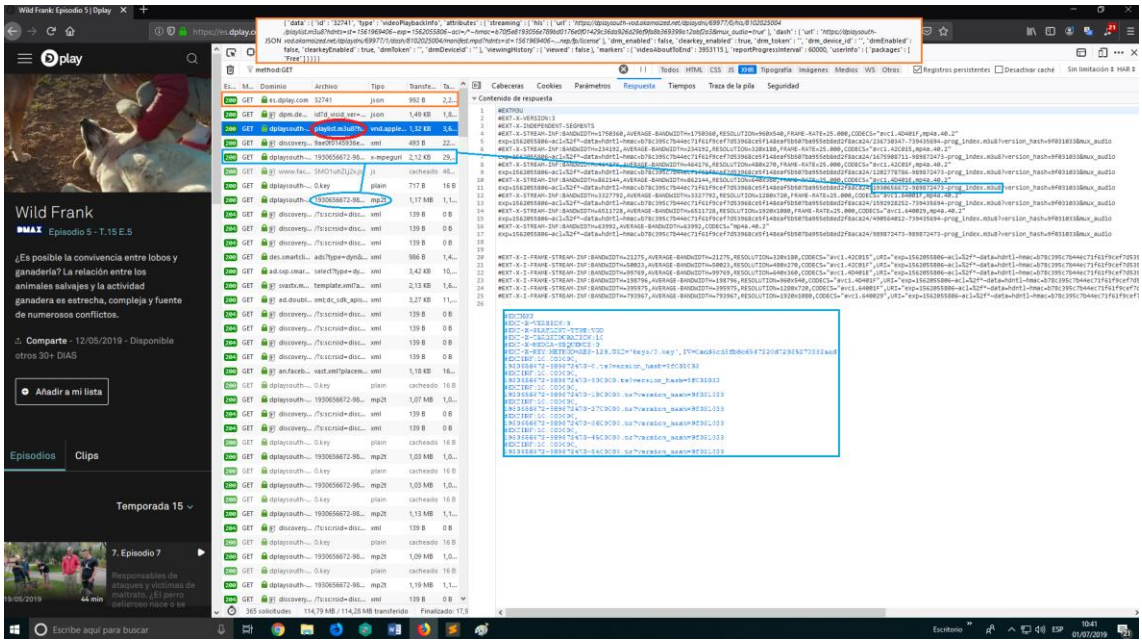

Figura 2.10: Captura del tráfico de contenido de dplay.com

2.6.3 HTTP Dynamic Streaming (HDS)

La solución de Adobe presenta una novedad frente a las otras 2 soluciones. Los fragmentos de HDS utilizan el formato F4V basado en MPEG4 como en MSS. Sin embargo, en HDS cuando se crean los fragmentos en los que se divide el flujo audio/video se les asigna un número que va incrementándose y asignándose conforme se crean, como en HLS. Este número sirve para poder identificar los fragmentos. Estos fragmentos (de intervalos de 4 segundos) son agrupados en segmentos. Cada segmento está compuesto por 1 o más fragmentos y compone un archivo (.f4f), que es la unidad de datos que se envía, junto con el archivo índice o manifiesto .f4m al servidor. El servidor que almacena los segmentos (.f4f) y el manifiesto (.f4m), que como se ve en la parte de distribución de la figura 2.11, debe tener instalado el módulo Origen HTTP.

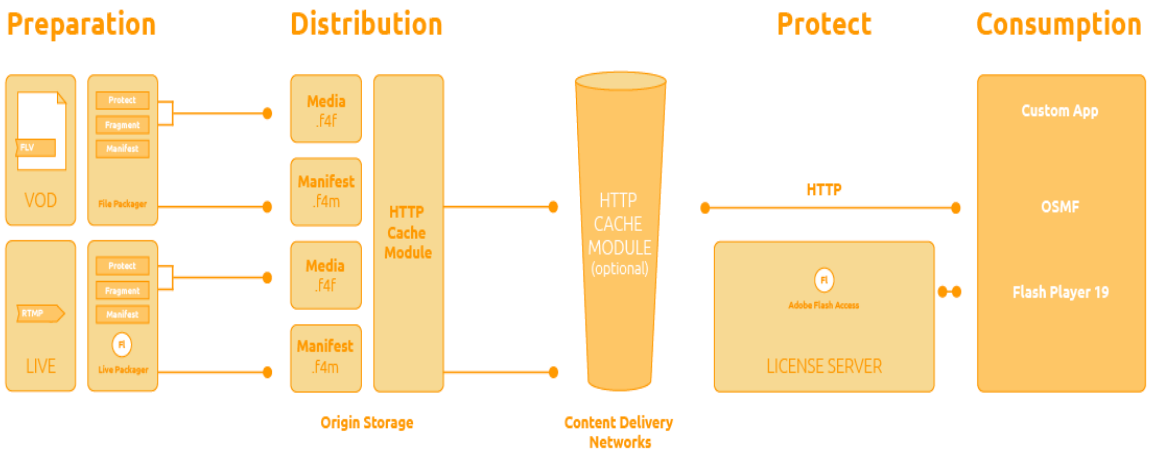

Figura 2.11: Representación del funcionamiento y la arquitectura de Adobe HTTP Dynamic Streaming[9]

La información relativa a la relación entre los segmentos y el número de fragmento, así como la correspondencia de los números de fragmento con los intervalos de tiempo la proporciona el archivo manifiesto. Éste documento es un XML en el que, al igual que en HLS, se referencian otros archivos (.f4m), submanifiestos, que representan las distintas calidades disponibles en las que se ha codificado y dividido el flujo original. En definitiva, versiones de un mismo audio o vídeo con diferente tasa de bits, como se aprecia en la figura 2.12.

```
<manifest xmlns="http://ns.adobe.com/f4m/2.0">
  <baseURL>http://nubeoxlivegeo-live.hds.adaptive.level3.net/hds-live/axnhd/_definst_/liveevent/</baseURL>
  <dvrInfo windowDuration="60"/>
  <media href="livestream1.f4m" bitrate="800"/>
  <media href="livestream2.f4m" bitrate="1200"/>
  <media href="livestream3.f4m" bitrate="1600"/>
  <media href="livestream4.f4m" bitrate="2300"/>
</manifest>
```

Figura 2.12: Manifiesto de Adobe HTTP Dynamic Streaming

Pero lo que contiene cada uno de estos submanifiestos no es una secuencia de fragmentos a una determinada tasa de bits como ocurre en HLS, sino que contiene la información de arranque o “bootstrap”. Esta información esta codificada en un archivo binario en el cual, se indican las asociaciones entre intervalos de tiempo, números de fragmento y segmentos. Básicamente, proporciona un mapa que indica donde encontrar un determinado fragmento.

Figura 2.13: Submanifiesto de Adobe HTTP Dynamic Streaming

Este mapa es el que necesita el cliente para poder solicitar posteriormente los fragmentos. Por lo tanto, como siempre, lo primero que hace el cliente es solicitar el manifiesto al servidor. Con el archivo manifiesto podrá pedir el archivo sub-manifiesto que se adapte a sus necesidades.

Si se trata de transmisión en vivo, el cliente descarga la secuencia de fragmentos más próximos al “momento actual” que le indica la información de “bootstrap”. El número de fragmentos descargados debe de ser suficiente como para llenar el buffer del cliente y poder iniciar la reproducción. Una vez iniciada la reproducción el cliente irá consumiendo los fragmentos que aparecen en el sub-manifiesto, por lo que deberá de ir refrescándose para que se muestren los nuevos fragmentos.

En caso de contenido bajo demanda, los fragmentos se van descargando y almacenando en el buffer.

Por último, el servidor decide el tiempo de vida que el archivo manifiesto permanece en caché según si el contenido es bajo demanda (permanente en cache), o en vivo (entre uno y dos segundos).

Los códecs que soporta la solución de Adobe son:

- Vídeo:
    - o GIF.
    - o PNG.
    - o JPEG.
    - o H.264 (Baseline, High, Main, Extended).
    - o VP6.
- Audio:
    - o MP3.

o AAC (main profile, low complexity, HE/SBR).

En cuanto a compatibilidad, debido a los problemas de seguridad que Adobe tuvo en el pasado y sus restricciones propietarias Apple decidió no dar soporte al reproductor Flash en sus dispositivos con iOS.

2.7 Dynamic Adaptative Streaming over HTTP (MPEG-DASH)

MPEG-DASH, es la principal solución de streaming adaptativo basado en HTTP. Desarrollado por MPEG en 2010 y publicado como estándar en 2012, ésta solución elimina las limitaciones de las soluciones anteriores y ofrece una solución compatible con cualquier tipo de dispositivo.
La idea se basa en otorgar al cliente la posibilidad de elegir lo que las otras soluciones imponían, como, por ejemplo, el códec que se va a aplicar para el video y para el audio, el sistema de encriptación, etc.

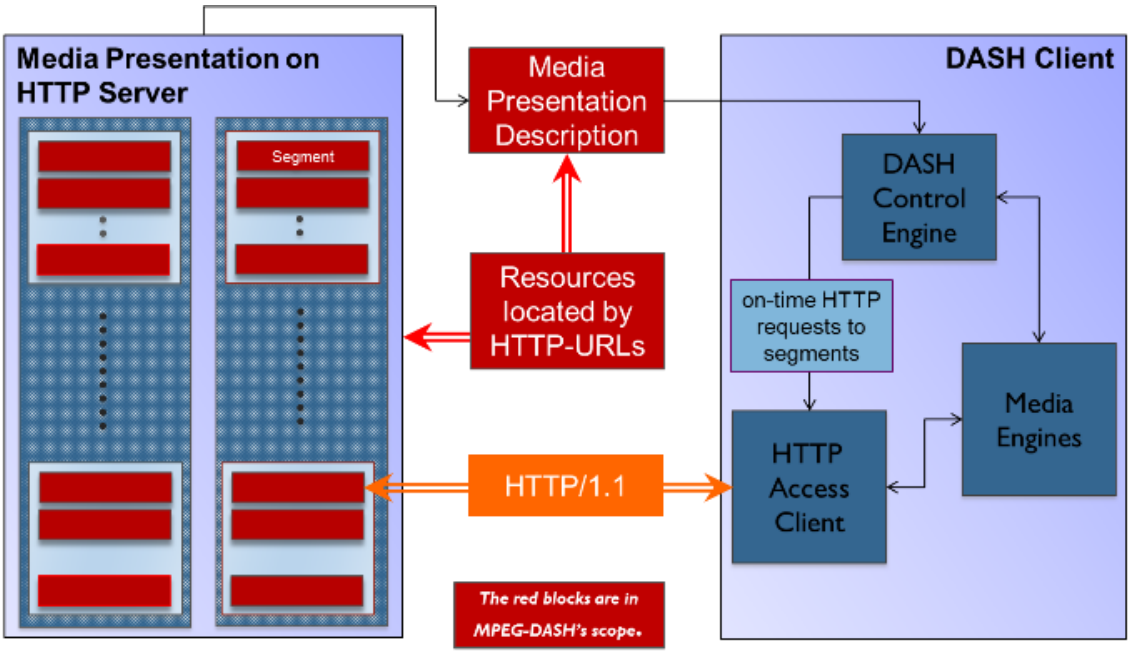

Figura 2.14: Representación del funcionamiento y la arquitectura del estándar MPEG-DASH.[10]

En ésta solución, el cliente solicita la información mediante el protocolo HTTP al servidor web, un servidor estándar, y al igual que en las soluciones anteriores, lo primero que tendrá que solicitar es el archivo con los metadatos necesarios para acceder al contenido, es decir, el manifiesto de DASH, el Media Presentation Description (MPD).

Una vez ha interpretado el MPD, selecciona las características y comienza a realizar el streaming obteniendo los segmentos mediante HTTP.
Como en las otras soluciones el ancho de banda es monitorizado y realizará cambios en las características si es necesario.

2.7.1 Media Presentation Description (MPD)

El manifiesto de MPEG-DASH es un documento XML que contiene toda la información sobre los segmentos, sus relaciones e información necesaria para elegir entre ellos y otros metadatos que los clientes pueden necesitar.
A continuación, se explica las partes más importantes del MPD, comenzando desde el nivel superior de la jerarquía (Periods), hasta el final (Segments).

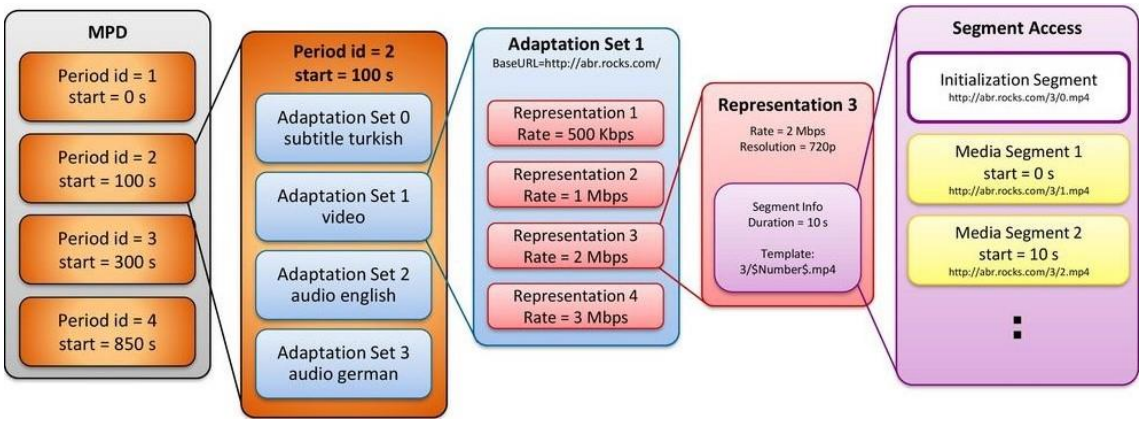

Figura 2.15: Representación de los elementos que forman el manifiesto de MPEG-DASH y su jerarquía[11]

2.7.2 Period

Un Period, contiene una parte del contenido multimedia con una hora de inicio y una duración. Un MPD puede contener varios Period, bien para dividir cada capítulo de una serie o para separar un anuncio del contenido normal, por ejemplo.
Veamos qué elementos y atributos contiene cada Period en la tabla 2.2:

Nombre del elemento o atributo	Uso	Descripción
Period		Especifica la información del Period
@xlink:href	OP	Especifica una referencia a un elemento Period externo.
@id	OP	Especifica un identificador para el Period, que debe de ser único dentro del MPD.
@start	OP	Especifica el PeriodStart del Period.
@duration	OP	Especifica la duración del Period.
@bitstreamSwitching	OPD default: false	Si su valor es 'true' es equivalente a que el atributo @bitstreamSwitching de cada AdaptationSet dentro del Period sea 'true'.
BaseURL	0..N	Contiene una Base URL que puede ser utilizada para resolver las URLs del Period.
SegmentBase	0..1	Especifica la información de Segment Base. Se sobrescribe por la información de AdaptationSet.SegmentBase y Representation.SegmentBase, en el caso de que existan estos elementos.
SegmentList	0..1	Especifica la información de Segment List. Se sobrescribe por la informaci´on de AdaptationSet.SegmentList y Representation.SegmentList, en el caso de que existan estos elementos.
SegmentTemplate	0..1	Especifica la información de Segment Template. Se sobrescribe por la información de AdaptationSet.SegmentTemplate y Representation.SegmentTemplate, en el caso de que existan estos elementos.
AdaptationSet	0..N	Especifica un AdaptationSet.
Subset	0..N	Especifica un Subset.
Leyenda Para los atributos: OB = obligatorio, OP: Opcional, OPD= opcional con valor por defecto, COB=condicionalmente obligatorio Para los elementos: <mínimo>...<máximo> (N = ilimitado) Los elementos aparecen en negrita y los atributos van precedidos de @		

Tabla 2.2: Semántica del elemento Period.

2.7.3 Adaptation Set

Los Adaptation Set representan los componentes multimedia y sus alternativas de codificación que contiene un Period. Un Period puede contener uno o más Adaptation Set dónde, por ejemplo, un Adaptation Set sea para el flujo de audio, y otro para el de video. También podrían estar los 2 flujos en un mismo Adaptation Set, pero por separado es más eficiente en cuanto a consumo de ancho de banda. Lo más habitual es encontrar los flujos de audio y video separados, incluso múltiples Adaptation Set solo para el audio (uno por idioma disponible). “Aquí podemos ver como el cliente ya puede echar mano y seleccionar por ejemplo el idioma”

Veamos qué elementos y atributos contiene cada Adaptation Set en la tabla 2.3:

Nombre del elemento o atributo	**Uso**	**Descripción**
AdaptationSet		Descripción del Adaptation Set.
@xlink:href	OP	Especifica una referencia a un elemento AdaptationSet externo.
@id	OP	Especifica un identificador para el AdaptationSet, que debe de ser único dentro del Period.
CommonAttributesElements		Especifica los atributos y los elementos comunes.
@contentType	OP	Especifica el tipo del componente del contenido multimedia para el Adaptation Set actual.
@bitstreamSwitching	OP	Si su valor es true indica que se puede cambiar de Representation dentro del mismo Adaptation Set durante la presentación
ContentComponent	0..N	Especifica las propiedades de un componente del contenido multimedia contenido en el Adaptation Set actual.
BaseURL	0..N	Contiene una Base URL que puede ser utilizada para resolver las URLs del Adaptation Set.
SegmentBase	0..1	Especifica la información de Segment Base. Se sobrescribe por la información de Representation.SegmentBase, en el caso de que exista este elemento.

SegmentList	0..1	Especifica la información de Segment List. Se sobrescribe por la información de Representation.SegmentList, en el caso de que exista este elemento.
SegmentTemplate	0..1	Especifica la información de Segment Template. Se sobrescribe por la información de Representation.SegmentTemplate, en el caso de que exista este elemento.
Representation	0..N	Especifica una Representation.
Leyenda Para los atributos: OB = obligatorio, OP: Opcional, OPD= opcional con valor por defecto, COB=condicionalmente obligatorio Para los elementos: <mínimo>...<máximo> (N = ilimitado) Los elementos aparecen en negrita y los atributos van precedidos de @. Las listas de elementos y atributos aparecen en negrita y cursiva.		

Tabla 2.3: Semántica del elemento Adaptation Set.

2.7.4 Representation

Un conjunto de Representation de un Adaptation Set representa un mismo conjunto de componentes de contenido codificado de diferentes maneras, siendo cada Representation una alternativa con diferentes características. Cada Representation se diferencia de las otras dentro del mismo Adaptative Set en la resolución, la tasa e bits, número de fotogramas, etc. Las Representation también se pueden codificar con diferentes códecs, lo que permite el soporte para clientes con diferentes códecs compatibles (como ocurre en los navegadores, con algunos MPEG-4 AVC / h.264 y algunos con VP8).

Normalmente, las Representation se eligen de forma automática, sin embargo, muchos reproductores permiten al usuario elegir unas características distintas. Por ejemplo, casi todos los reproductores permiten cambiar la opción de calidad del modo automático a diversas opciones. Si un usuario solo quiere escuchar la canción de un video mientras hace otras cosas, puede reducir la calidad del video al mínimo y ahorrar ancho de banda.

Veamos qué elementos y atributos contiene cada Representation en la tabla 2.4:

Nombre del elemento o atributo	Uso	Descripción
Representation		Descripción de la Representation.
@id	OB	Especifica un identificador para la Representation, que debe de ser u´nico dentro del Period, a no ser que la Representation sea funcionalmente id´entica a otra dentro del mismo Period.

@bandwidth	OB	Bandwidth mínimo necesario.
CommonAttributesElements		Especifica los atributos y los elementos comunes.
BaseURL	0..N	Contiene una Base URL que puede ser utilizada para resolver las URLs de la Representation.
SubRepresentation	0..N	Especifica la información sobre una SubRepresentation que se encuentra dentro de una Representation.
SegmentBase	0..1	Especifica la información de Segment Base.
SegmentList	0..1	Especifica la información de Segment List.
SegmentTemplate	0..1	Especifica la información de Segment Template.
Leyenda Para los atributos: OB = obligatorio, OP: Opcional, OPD= opcional con valor por defecto, COB=condicionalmente obligatorio Para los elementos: <mínimo>...<máximo> (N = ilimitado) Los elementos aparecen en negrita y los atributos van precedidos de @. Las listas de elementos y atributos aparecen en negrita y cursiva.		

Tabla 2.4: Semántica del elemento Representation.

2.7.5 Segments

Los Segment son referencias a segmentos de contenido multimedia en las que se ha dividido el contenido de la Representation a la que pertenece, asociados con una URL HTTP. También puede tener asociado un rango de bytes. Cada Segment tiene asociado un intervalo de disponibilidad descrito mediante una hora de inicio y una hora de finalización.

Cada Representation tiene asignado Segment Information, que indica la ubicación, disponibilidad y propiedades de todos los tipos de Segments del Representation.

Segment Base

El elemento SegmentBase se usa para cuando se trata de un solo Media Segment por representación y en BaseURL se encuentra la URL del Media Segmen. Si hay varios MediaSegment, se utilizará SegmentList o SegmentTemplate para describir la Segment Information.

Si hay más de un MediaSegment en la Representation, habrá un atributo @duration o el elemento SegmentTimeline.

Segment List

Segment List está definida por uno o más elementos de SegmentList. Cada elemento SegmentList contiene una lista de elementos Segment URL con las URLs de los

segmentos y posiblemente un rango de bytes. El elemento Segment URL también puede contener un Index Segment.

Segment Template

Segment Template está definido por el elemento SegmentTemplate, que hace uso de identificadores específicos que se sustituyen por valores dinámicos asignados a Segments, para crear una lista de segmentos.

Los identificadores de SegmentTemplate:

-RepresentationID: @id de la Representation en la que se encuentra.

-Number: número del Segment correspondiente.

-Bandwidth: @bandwidth de la Representation en la que se encuentra.

-Time: atributo @t del elemento SegmentTimeline para el Segment al que se está siendo accedido. El uso de Number o bien Time, pero no ambos al mismo tiempo.

Segment Information

Segment Information contiene información sobre los segmentos Initialization Segment, Index Segment y Bitstream Switching Segment, las URLS de los segmentos y su disponibilidad. Esto último para contenido en vivo (MPD@type='dynamic').

En el siguiente punto veremos estos diferentes tipos de segmentos y sus correspondientes Segment Information.

2.7.6 Tipos de segmentos

Initialization Segment

Proporciona la información requerida de inicialización para reproducir los segmentos de la Representation que corresponde.

Initialization Segment Information

Los elementos SegmentBase, SegmentList y SegmentTemplate pueden proporcionar el Initialization Segment Information mediante el elemento InitializationSegment o mediante el atributo @initialization del elemento SegmentTemplate. Si no se

proporciona esta información, entonces todos los Media Segments de la Representation se inicializan de forma automática.

Media Segment

Contiene y encapsula el contenido multimedia descrito en él. También contiene una cierta cantidad de unidades de acceso completas, un Stream Access Point(SAP) como mínimo e información para reproducir el contenido multimedia sin problemas.

Este tipo de segmento puede ser dividido en Subsegments por el Segment Index. Este tipo de segmento debe aparecer en el Index Segment, a no ser que el formato multimedia empleado sea ISO Base media file format. En tal caso puede estar incluido en el mismo Media Segment

Media Segment information

Las características de cada Media Segment (URL con/sin rango de bytes, número de segmento, tiempo inicio, duración) que hay en una Representation son especificadas mediante los elementos SegmentList y SegmenTemplate.

Los elementos SegmentList y SegmenTemplate especifican estas características. Si estos elementos no están presentes, significa que la Representation solo tiene un Media Segment, y su URL será especificada mediante el elemento BaseURL.

Index Segment

Contiene información de indexación los Media Segments.

Index Segment information

El Index Segment puede proporcionar a cada Segment su información sobre indexación Mediante el elemento RepresentationIndex, el atributo @index y el atributo @indexRange del elemento SegmentList.SegmentURL o por el atributo @index del elemento SegmentTemplate.

Bitstream Switching Segment

Contiene los datos necesarios para moverse a la Representation que tiene asignada.

Bitstream Switching Segment information

Cada Representation de un Adaptative Set cuyo valor del atributo @bitstreamSwitching sea ‘true’ puede tener asignado un Bitstream Switching Segment, representado por el elemento BitstreamSwitching o el atributo @bitstreamSwitching del elemento SegmentTemplate.

2.7.7 Formato de los segmentos

El formato de un segmento especifica la semántica y la sintaxis de los medios relacionados con HTTP-URLs en el MPD.

El formato de los segmentos empleados en DASH está basado en formatos de contenedores MPEG.

2.7.7.1 MPEG-2 Transport Stream

Formato de contenedor con corrección de errores y sincronización de los streams para asegurar la transmisión cuando la señal se disminuye. En ISO/IEC 13818-1 se pueden encontrar más especificaciones.

No vamos a profundizar en este formato porque se suele utilizar en DASH para contenido en vivo y en este proyecto la implementación se realiza sobre contenido bajo demanda que emplea como formato de encapsulamiento ISO Base Media File Format. Además, el contenido con formato TS MPEG-2 puede encapsularse dentro de un ISO Base Media File Format. Esto hace que el formato de medios empleado por DASH sea en su mayoría sea ISO Base Media File Format.

2.7.7.2 ISO base media file format ISO-BMFF

Especificado en ISO/IEC 14496-12 (MPEG-4 Part 12) define una estructura general para ficheros multimedia.

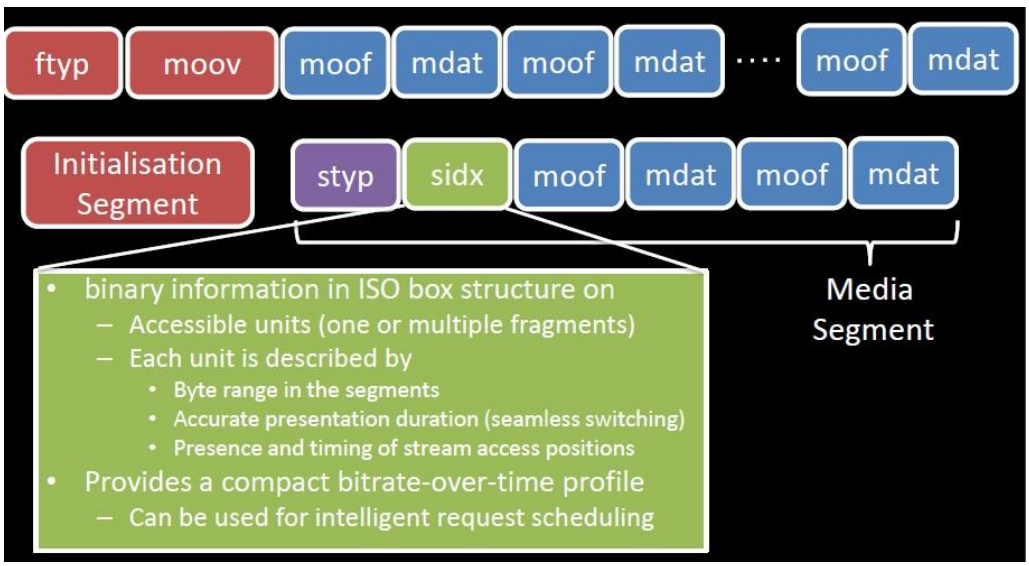

Figura 2.16: Formato de un segmento ISO base media file format en DASH.[4]

2.7.7.2.1 Segment Index

Casilla 'sidx'. almacena metadatos de las ubicaciones precisas de rango de bytes de los segmentos que forman las casillas (moov) y (ftyp).

2.7.7.2.2 Initialization Segment

Contiene las casillas (moov) y (ftyp). Dentro de la casilla (moov) se encuentra la casilla (mvex), que indica que se debe esperar fragmentos de película.

2.7.7.2.3 Media Segments

Tipos de Media Segments:

- Simple Media Segments
  - Puede contener una casilla Segment Type(styp).
  - Contiene un conjunto o más de fragmentos de película auto-contenidos. Un conjunto contiene una casilla (moof) y (mdat), con las muestras multimedia que no utilizan referencias de datos externos.
  - Contiene una casilla Track Fragment Box(traf), con una casilla Track Fragment Decode Time Box (tfdt)
  - Puede que contenga una o varias casillas (sidx). La primera aparece antes que cualquier casilla (moof).
- Indexed Media Segments
  - Igual que el formato Simple Media Segment y fragmentos auto-contenidos, la casilla (moof) junto con su (mdat).
  - La casilla (sidx) es obligatoria.
- Sub-Indexed Media Segments
  - Mismo formato que Indexed Media Segments
  - Aparece la casilla Subsegment Indexed (ssix), que sigue a la casilla (sidx).

2.7.7.2.4 Profiles

Los profiles crean restricciones basadas en en el formato de los segmentos y características del MPD.

Se utiliza el atributo "@profiles" para indicar que un MPD cumple un determinado Profile del MPD. Se puede asignar varios Profiles a un MPD.

En el estándar DASH, como se muestra en la figura 2.17, hay 6 Profiles, 3 de ISO Base media file format, 2 de MPEG2-TS y uno que contiene a todos.

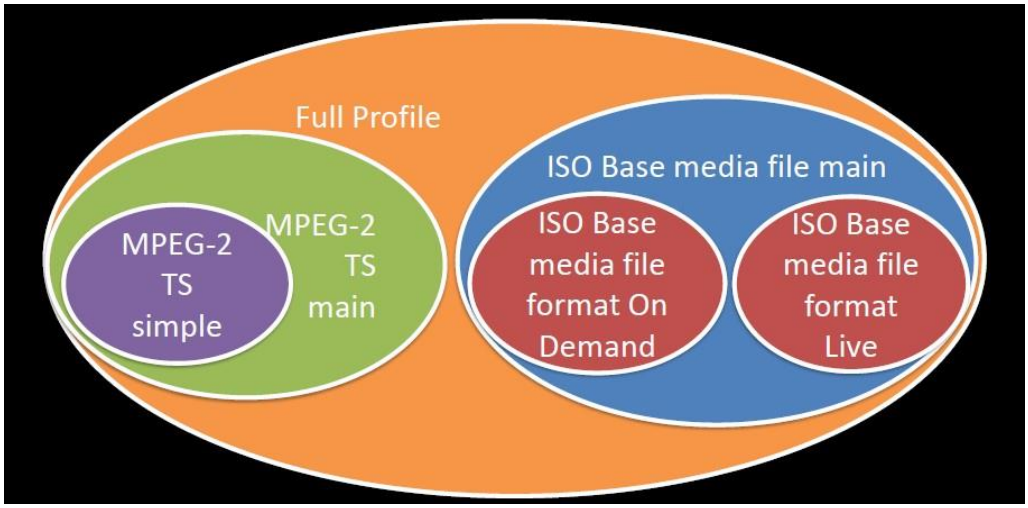

Figura 2.17: Profiles de MPEG-DASH.[4]

Estos son los seis profiles definidos en MPEG-DASH:

Full profile - “urn:mpeg:dash:profile:full:2011”.

- ISO Base Media File Format on Demand - “urn:mpeg:dash:profile:isoff-on-demand:2011”.

ISO Base Media File Format Live - “urn:mpeg:dash:profile:isoff-live-2011”.

ISO Base Media File Format main - “urn:mpeg:dash:profile:isoff-main:2011”.

MPEG2-TS main - “urn:mpeg:dash:profile:mp2t-main:2011”.

MPEG2-TS simple - “urn:mpeg:dash:profile:mp2t-simple:2011”.

En nuestra implementación utilizaremos el segundo perfil de este listado, destinado al acceso bajo demanda, que es el escenario objeto de estudio en nuestro proyecto.

Capítulo 3

Implementación

En esta parte del proyecto se hace una implementación de un escenario DASH básico, mostrado en la figura 3.1, donde se simula localmente el consumo de un archivo multimedia bajo demanda mediante la técnica MPEG-DASH. Aunque la simulación sea en un entorno local, el programa implicado en la restricción de ancho de banda NetLimiter se puede programar para que genere limitaciones de ancho de banda de forma totalmente controlada y autónoma simulando un escenario real en su totalidad.

3.1 Escenario

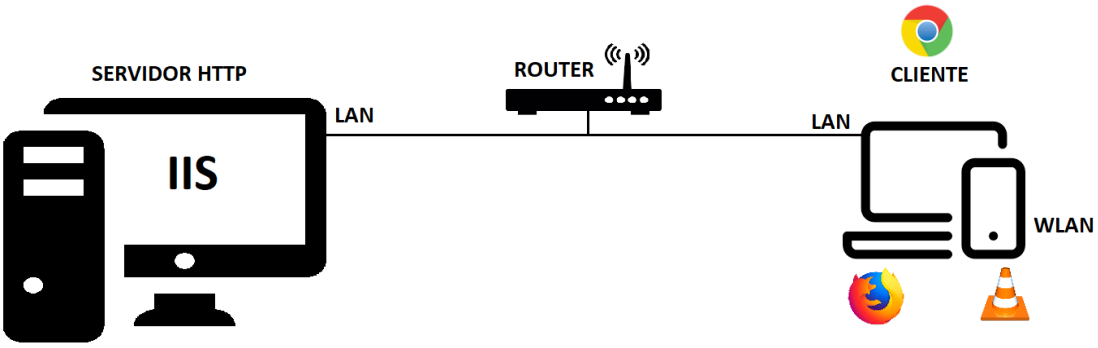

Figura 3.1: Escenario de la implementación de MPEG-DASH

Para crear un escenario DASH hacen falta:

Un servidor web

Un cliente

Software codificador de audio y video

Software para compatibilizar el contenido con DASH

3.1.1 Servidor Web

Si la idea es transmitir el contenido desde nuestro ordenador necesitaremos un servidor web. En este caso el sistema operativo del ordenador que hace de servidor es Windows 10 Pro.

3.1.1.1 Internet Information Services (IIS)

Desde la versión de Windows XP Professional, Microsoft incluye en sus sistemas operativos el servicio IIS. IIS es un servidor web junto con una serie de servicios que están integrados en todos los sistemas operativos de Microsoft desde esta versión. Éste va a ser nuestro servidor donde se va a almacenar el contenido a transmitir.

Este servicio no viene instalado por defecto, por lo tanto, lo primero que hay que hacer es habilitar este servicio. Para activarlo hay que acceder a Panel de control\Programas y características, opción Activar o desactivar características de Windows. Entre la lista de servicios que aparece se encontrará Internet Information Services desmarcada. Tan solo hay que marcarla para activar los servicios. En la figura 3.2 se destaca la información para el proceso de instalación.

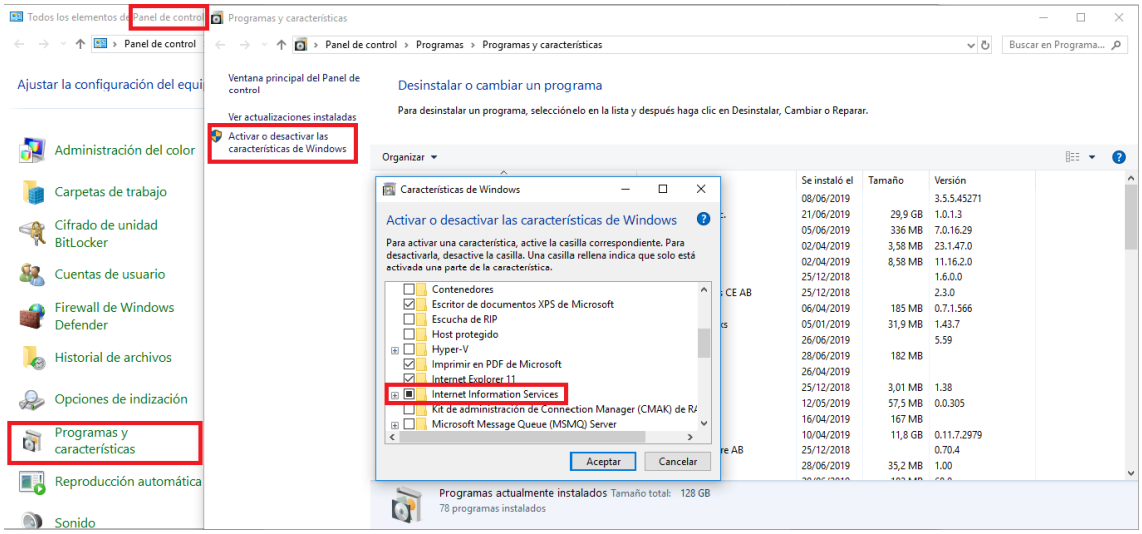

Figura 3.2: Activación del servicio Internet Information Services para Windows

Automáticamente se habrá creado la carpeta "C:\inetpub\ " donde "C:\inetpub\wwwroot\" será la carpeta de nuestro servidor a la que podremos acceder desde el navegador de internet escribiendo la IP local (127.0.0.1) o, directamente “localhost” en la barra de direcciones.

Una vez creado el servidor hay que configurarlo para que pueda leer el archivo MPD que generaremos más tarde sino no reconocerá la extensión .mpd. Lo que hay que hacer es acceder a la configuración del sitio web que hemos creado y añadir el tipo

MIME al servidor asignándole a los archivos con la extensión “.mpd”, el tipo MIME “application / dash + xml como se indica en la figura 3.3.

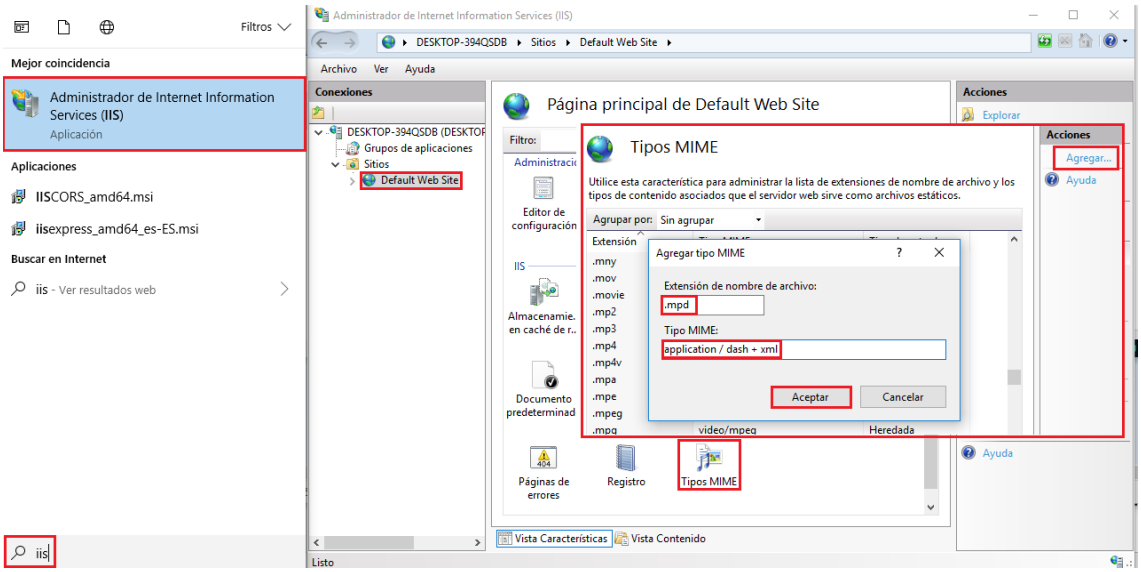

Figura 3.3: Configuración del servidor

El servidor ya estaría preparado para poder entregar contenido mediante DASH.

3.1.2 Cliente

Como cliente DASH podemos utilizar un navegador de internet como puede ser Firefox de Mozilla o Chrome e Google, o un reproductor que sea compatible con DASH. Si se elige la primera opción, nuestro navegador que haga de cliente debe estar adaptado para que reconozca un archivo .mpd y sepa qué hacer con él. Las diferentes opciones son:

- Instalar un Plug-in que, al introducir la dirección de un archivo con la extensión .mpd en la barra de direcciones, nos redirija a un reproductor DASH que consumirá el contenido al cual referencia. La extensión utilizada se llama Native MPEG-Dash + HLS Playback

- Usar un reproductor DASH de los que ofrecen por ejemplo en “http://reference.dashif.org/dash.js/”:

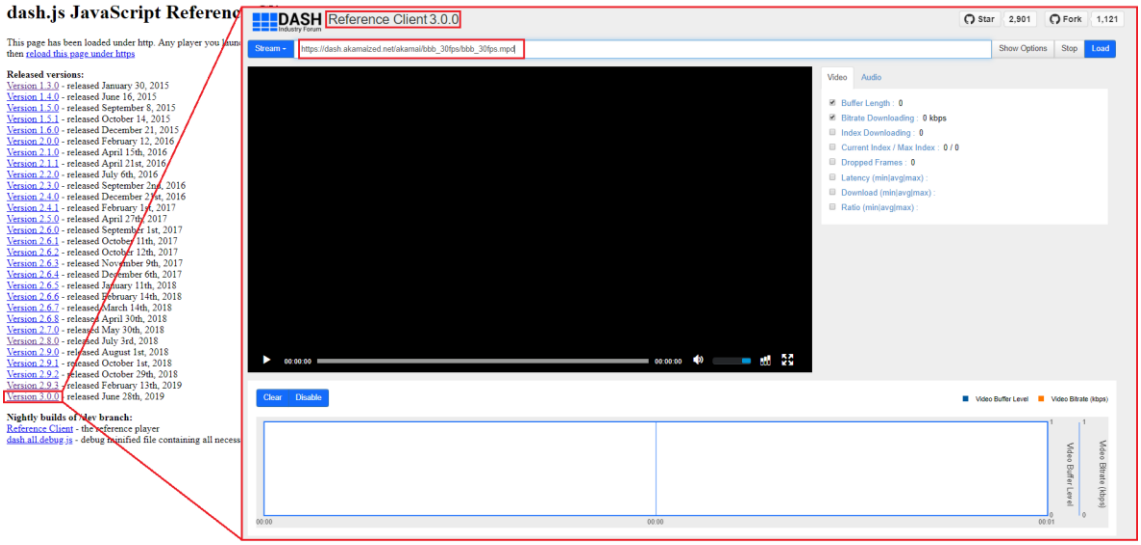

Figura 3.4: Reproductor multimedia DASH[12][13]

- Añadir un reproductor DASH min.js a nuestro servidor local y llamarlo desde el archivo HTML del servidor. Esto supondrá añadir unas pocas líneas al archivo HTML:

```
<center>
        <script src="dash.all.min.js"></script>
        <video width="700" height="500" data-dashjs-player src="manifest.mpd"
        controls="controls"> </video>
</center>
```

La segunda opción es encontrar un reproductor que pueda abrir archivos que estén en la red y sea compatible con DASH como VLC.

El reproductor VLC admite una gran cantidad de códecs y además tiene la opción de abrir archivos a partir de una dirección URL, como se ve en la figura 3.5, siendo muy sencillo su uso.

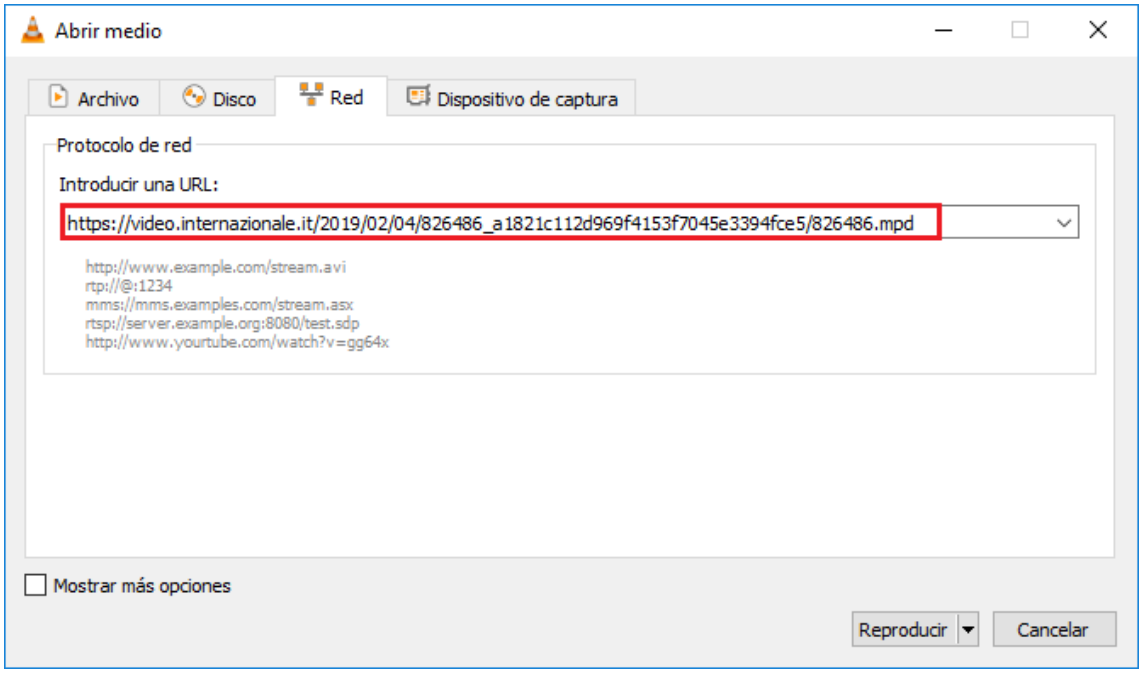

Figura 3.5: Reproducción del contenido mediante el reproductor multimedia VLC

Aunque usar el reproductor VLC es la opción más sencilla y rápida, usar un navegador tiene ventajas como poder usar las herramientas de desarrollador que el navegador tiene instaladas para capturar el tráfico y analizarlo a la vez que lo estas reproduciendo. En la figura 3.6 se pueden ver las herramientas de Chrome y Firefox. Aunque esto también se puede hacer usando la herramienta wireshark, estas herramientas de los navegadores facilitan la información que necesitamos de manera sencilla y optima, capturando tan solo el tráfico de esa página en concreto que queremos. De esta forma eliminamos una gran cantidad de datos que son irrelevantes. Además, también presenta la ventaja de que se puede integrar la reproducción directamente en un entorno de navegación web sin necesitar un reproductor aparte.

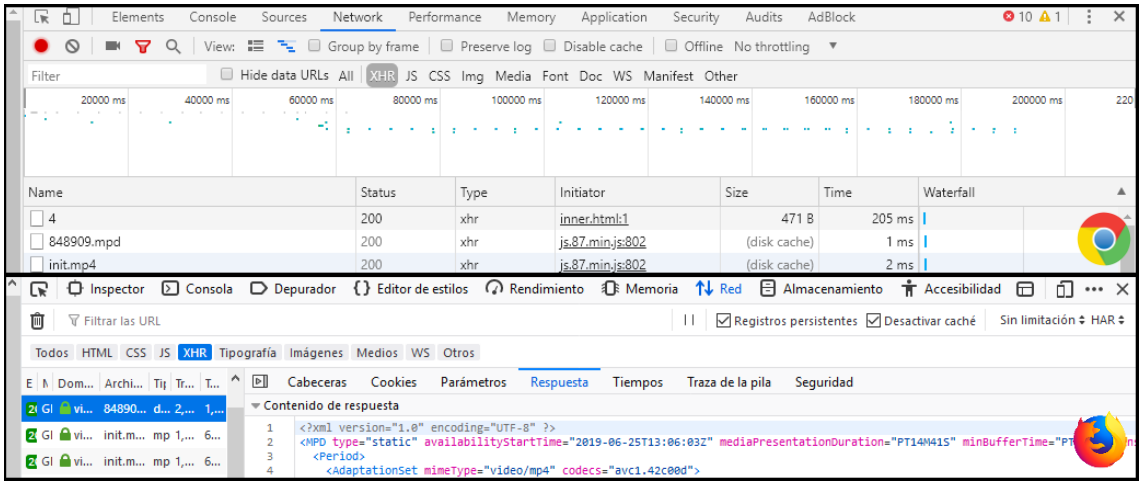

Figura 3.6: Herramienta incorporada en Mozilla y Chrome para capturar tráfico

3.1.3 FFmpeg

Este es el programa que vamos a utilizar para codificar el video y el audio del stream. También es capaz de convertir, grabar y hacer streaming. Fue desarrollado en un principio para Linux, pero está adaptado para ser compilado por casi todos los sistemas operativos y orientado tanto para usuarios avanzados como principiantes por su sencillez. Contiene una biblioteca llamada libavcodec para la codificación y decodificación de video y audio. Ademas, tiene la capacidad de elegir el códec que usará según la extensión del archivo sin que el usuario se lo indique.

Al descárgalo tendremos un archivo .zip que deberemos de descomprimir. Este programa se ejecuta por línea de comandos, por lo tanto, para poder utilizar el programa deberemos de añadir a las Variables de Entorno de nuestro ordenador, la ruta en la que hemos descomprimido el archivo y se encuentra nuestro programa

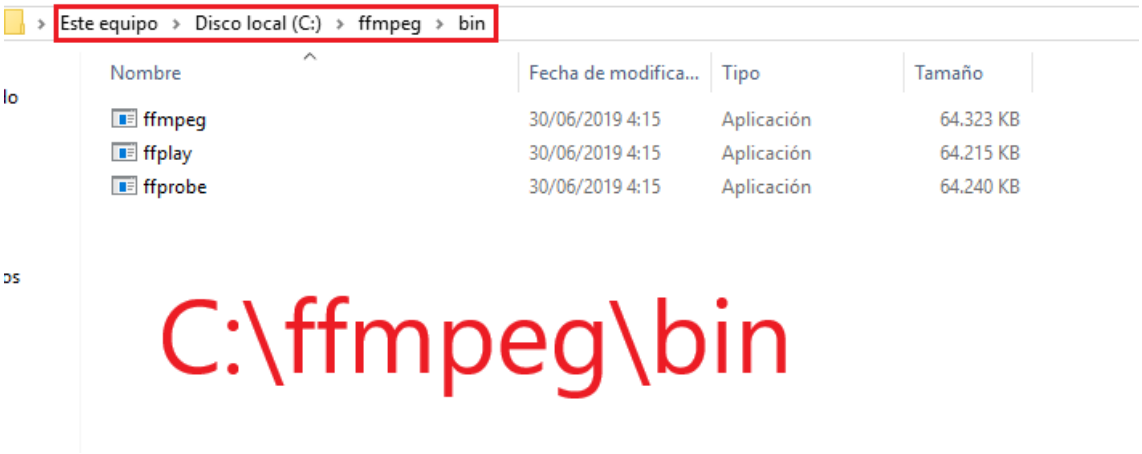

Figura 3.7: Programa FFmpeg para la transcodificación de los archivos

Para hacer esto hay que acceder a “Configuracion avanzada del sistema”-“Variables de entorno”. Una vez allí habrá que añadir la ruta de FFmpeg a la variable que coincida con el nombre de “Path” en “Variables del sistema”. En la figura 3.8 se destaca la información importante para seguir el proceso.

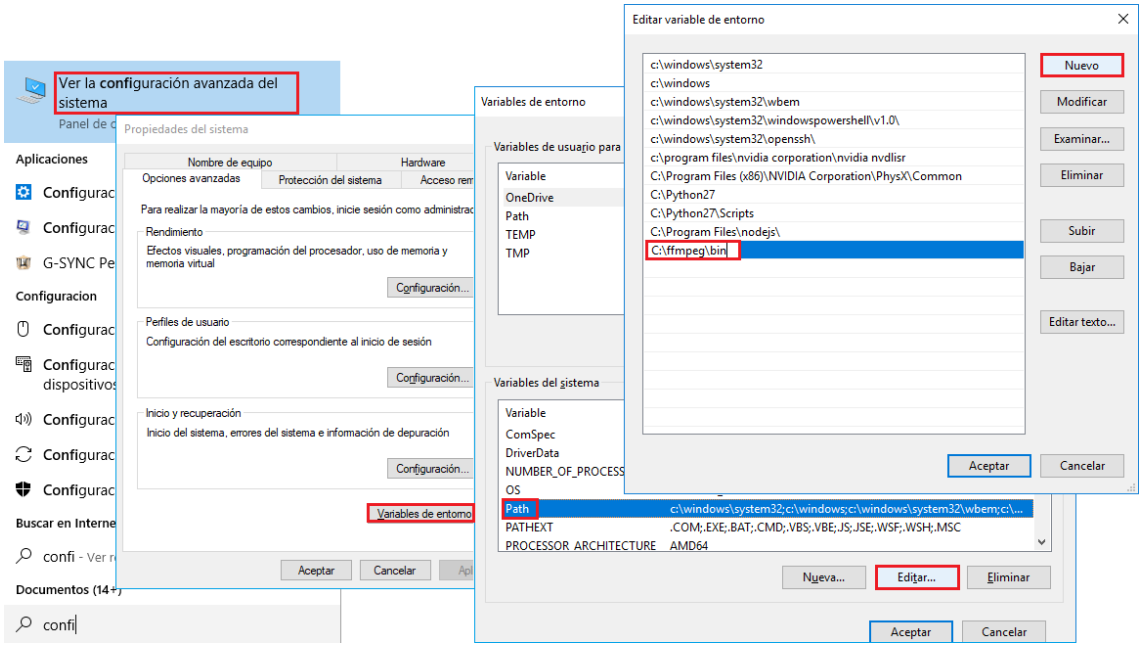

Figura 3.8: Preparación del programa FFmpeg

3.1.4 MP4box

Esta herramienta nos va a servir para convertir los archivos codificados en un formato compatible con DASH y generar los archivos necesarios para que el reproductor del cliente pueda reproducirlos. Para poder usarlo tan solo hay que instalar el framework GPAC que contiene a MP4box[5] entre sus componentes. Durante el proceso de instalación de GPAC permite elegir los componentes que se desean incluir en la instalación que, por defecto, estarán todos incluidos como se puede observar en la figura 3.9.

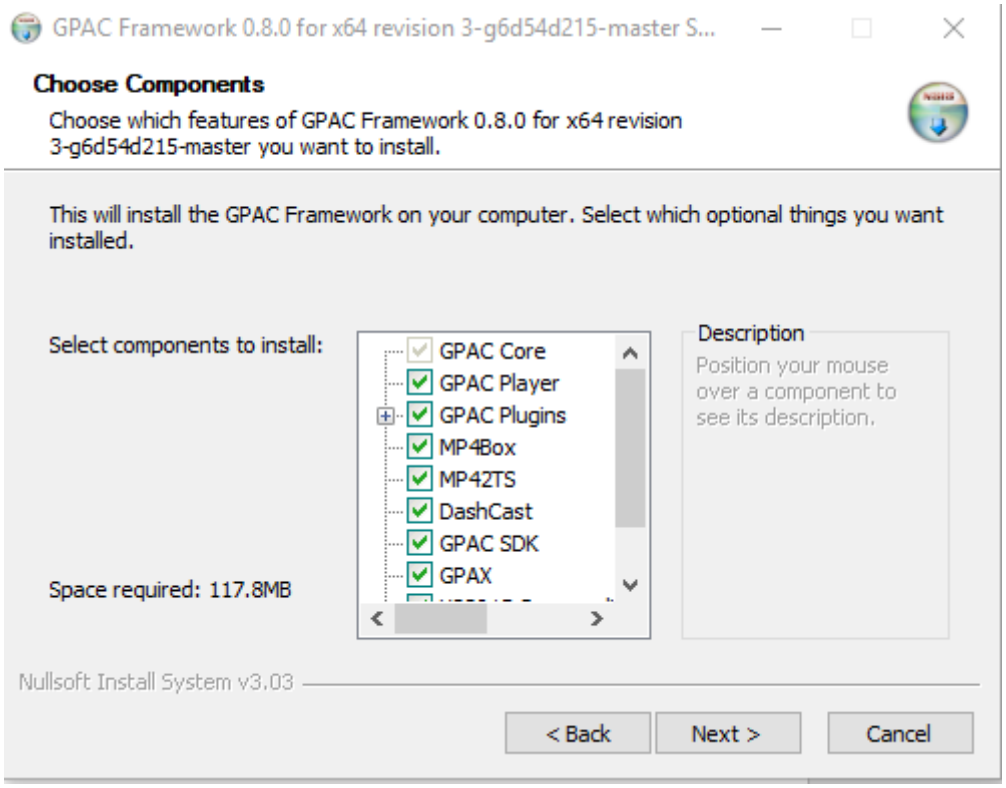

Figura 3.9: Instalación del programa GPAC

3.1.5 NetLimiter

Aunque ya estaría todo lo necesario para simular el escenario DASH, también se hará uso de la herramienta NetLimiter que nos permitirá efectuar cambios en nuestra red para poder simular una conexión inestable y poder experimentar un mayor número de cambios. Como un Smart Phone que va perdiendo la señal al estar en un ascensor, pasar por un túnel, etc.

Más específicamente nos permitirá limitar la tasa de entrada y salida de datos para cada aplicación de nuestro sistema. En nuestro caso, por ejemplo, nos bastaría con aplicar reglas de límite al navegador que vayamos a usar como cliente o al reproductor que vayamos a utilizar.

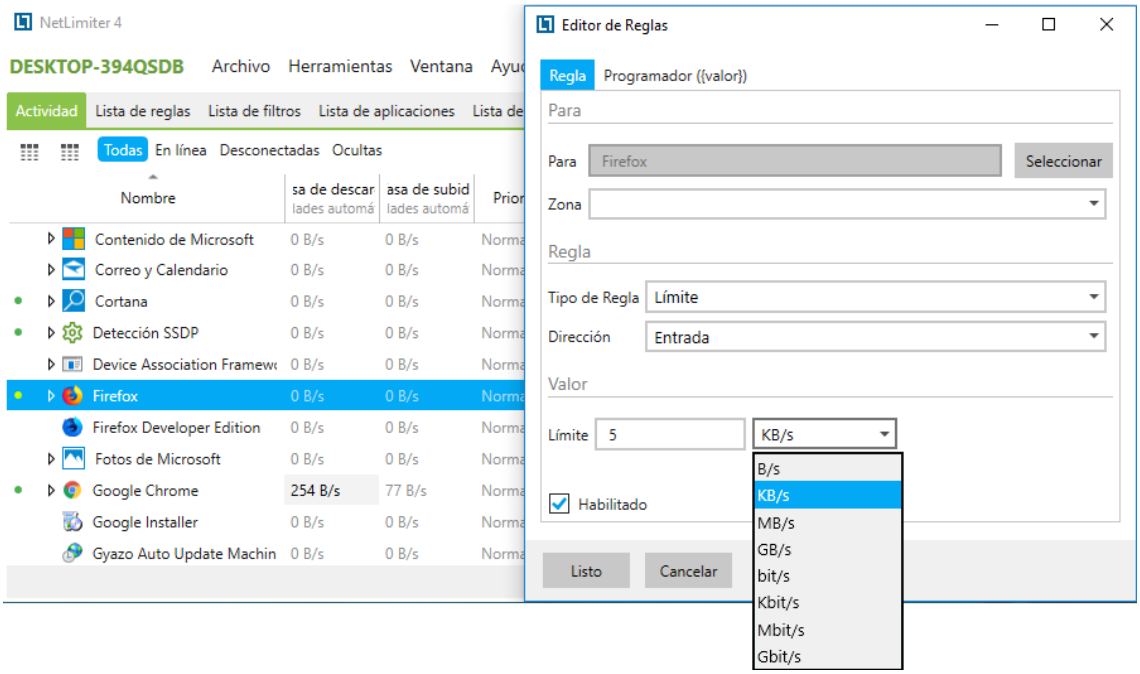

Figura 3.10: Uso de la herramienta NetLimiter

3.2 Proceso de implementación

Una vez tenemos el servidor y el cliente configurado como se ha indicado anteriormente y los programas necesarios instalados adecuadamente se puede proceder a poner en marcha la solución.

Lo primero que hay que hacer es escoger el archivo multimedia que queremos ver desde el cliente DASH. Se puede utilizar tanto un archivo que solo contenga audio, como una canción, solo video, o ambos. Para empezar, utilizar un único tipo de flujo (audio o video) nos permitirá hacer el proceso que viene a continuación de forma muy rápida y el análisis de la red será sumamente descriptivo.

Sin embargo, el contenido que vamos a utilizar va a ser un archivo que contenga tanto audio como video y una resolución base lo suficientemente buena como para poder apreciar los cambios de calidad de forma sensorial sin ocupar un espacio enorme en nuestro servidor.

Para empezar, lo mejor es crear una carpeta donde tener únicamente el archivo con el que vamos a trabajar, ya que se va a dividir en fragmentos y es mejor tener el directorio despejado. Además, es muy recomendable que el nombre del archivo sea corto y sencillo porque se va a escribir algunas veces.

Todo el proceso de “DASHificar” el archivo multimedia se va a llevar a cabo por comandos desde la consola de Windows. Por lo tanto, una vez en la consola tenemos que ir al directorio donde está el archivo y este será nuestro lugar de trabajo.

Una vez en el lugar de trabajo, primero, vamos a codificar el flujo de video a diferentes resoluciones y tasas de bits y fragmentarlo. Luego haremos lo mismo con el audio. El orden es indiferente.

El comando para codificar el video es

\espacioDASH> “ffmpeg -i original.avi -s 256x144 -c:v libx264 -b:v 200k -g 90 -an origen_video_256x144_200k.mp4”

donde:

“**-i original.avi**” es el archivo de entrada

“**-s**” es la resolución a la que se va a codificar el archivo

“**-c: v**” indica el códec de video que va a usar

“**-b: v**” es la tasa de bits a la que queremos codificar el video

“**-g 90**” le dice a ffmpeg que queremos un intervalo de fotogramas clave (longitud de GOP) de 90

“**-an**” para no codificar el audio

Por último, se escribe el nombre del archivo de salida

Conviene emplear el archivo de salida con un nombre representativo (original_video_256x144_200k). Esto ayudará a monitorizar la red de forma muy clara con los nombres de los paquetes que se vayan enviando en lugar de guiarse por el tamaño de estos.

El resultado de este código es una versión del archivo original sin audio, en formato fMP4 (MP4 fragmentado), a una resolución de 256x144, codificado en H264 y con una tasa de bits de 200kb. La idea es generar varios archivos a diferentes tasas de bits y resoluciones para poder ver de forma notoria el comportamiento de DASH. Por lo tanto, simplemente repetimos este proceso cambiando los valores de resolución y tasa de bits. Cuantos más archivos creemos más opciones le damos a DASH y más cambios se realizarán, pero también menos los apreciaremos visualmente.

El comando para audio es

\espacioDASH> “ffmpeg -i original.avi -c: a aac -b: a 128k -vn original_audio_128k.mp4”

donde

"**-i original.avi**" es el archivo de entrada

"**-c: a**" es el codec de audio

"**-b: a**" la tasa de bits del audio

"**-vn**" no codificar video

Por último, se escribe el nombre del archivo de salida.

El resultado del ejecutar el código anterior será un archivo de audio del archivo original, con una tasa de bits de 124kb codificado en AAC.

Los archivos de audio son muy pequeños y no representan una carga significativa en la red por lo que no suele variar mucho. Además, es relativamente complicado detectar un cambio de calidad de audio de forma sensorial, por lo que solo habrá un flujo de audio disponible en esta implementación. Se podría repetir el proceso con una tasa de bits diferente si se desean más.

Nuestro directorio de trabajo en el que se encontraba únicamente el archivo original debe de contener varios archivos multimedia con nombres representativos de sus características que le hemos asignado antes. Pero estos archivos aún no se pueden utilizar con DASH. Simplemente son versiones de un mismo archivo que ocupan tamaños diferentes. El cliente DASH necesita conocer donde están estas versiones y como solicitarlas según sus necesidades. De esto se encarga el archivo MPD, extensión .mpd. Este archivo contiene toda la información que el cliente DASH necesita para reproducir el contenido moviéndose entre las distintas versiones según las necesidades dando la imagen de ser un archivo único.

Para poder convertir estos archivos en compatibles para un cliente DASH, vamos a hacer uso de la herramienta mp4box descrita antes. Esta herramienta generará archivos de inicialización MPEG-4 de cada versión del video que el reproductor DASH consumirá en el momento de la carga y el archivo MPD ya comentado.

Un ejemplo del uso de esta herramienta mediante la consola de windows:

espacioDASH> "mp4box -dash 5000 -rap -profile dashavc264:onDemand -out manifiesto.mpd -frag 2000 original_audio_128k.mp4 original_video_320x180_500k.mp4 original_video_640x360_1000k.mp4 original_video_1280x720_1500k.mp4

donde

"**-dash 5000**" divide los archivos de entrada en segmentos de 5 segundos

"**-rap**" obliga a los segmentos a comenzar con puntos de acceso aleatorios. En otras palabras, permite la búsqueda del video.

"**-profile dashavc264: onDemand**" usa el perfil bajo demanda visto en el apartado 2.7.7.2.4.

"**-out manifiesto.mpd**" el nombre del archivo de salida

"**-frag 2000**" establece la longitud del fragmento en 2 segundos. Siempre menor que el valor especificado en "–dash"

Como resultado del comando ejecutado anterior se obtiene un archivo .mpd junto con 3 flujos de video con diferentes calidades y uno flujo de audio, divididos en segmentos de 5 segundos.

Ahora solo hay que pegar en la carpeta del servidor "C: \ inetpub \ wwwroot \" todos los archivos de la carpeta de trabajo salvo el original y ya tendríamos nuestro primer contenido reproducible mediante DASH.

Capítulo 4

PRUEBAS

En esta parte del proyecto, probamos que nuestra implementación del estándar funciona correctamente.

Para ello, a partir de un video de alta resolución[14], lo codificamos a varios bitrates y resoluciones y adaptamos al estándar MPEG-DASH. El proceso de adaptación se ha repetido múltiples veces para distintos tamaños de segmento.

Estas pruebas han sido realizadas para segmentos de 2, 4, 8 y 16 segundos y se han llevado a cabo programando 2 variaciones en el ancho de banda en la parte del cliente.

Entorno de trabajo:

Las pruebas se realizan en un ordenador conectado de forma cableada al mismo router en el que se encuentra conectado el ordenador servidor que contiene los archivos. Se utiliza la versión 3.0.0 del reproductor dash.js ofrecido por http://reference.dashif.org/dash.js/ en el navegador Google Chrome. Esta versión añade un gráfico a la interfaz del reproductor que aporta información interesante para el análisis. Además, se utilizará la herramienta del navegador Chrome para analizar el tráfico, ya que aporta la información que se necesita de forma directa y clara para saber que todo funciona correctamente. Por último, con el programa NetLimiter se limitará el ancho de banda de forma precisa y totalmente controlada.

Para el flujo de video emplearemos las resoluciones y tasas de bits indicadas en la Tabla 4.1.

Índex	Resolución	Tasa de bits
1	160x90	250kbits/s
2	320x180	500kbits/s
3	640x360	750kbits/s
4	640x360	1000kbits/s
5	1280x720	1500kbits/s

Tabla 4.1: Archivo de video codificado a distintas tasas de bits y resolución

4.1 Primera simulación

La reproducción empieza sin ningún límite en el ancho de banda hasta el segundo 30 donde se reduce a 900Kbps. Los resultados en función del tamaño de segmento establecido varían considerablemente.

4.1.1 Segmentos de 2 segundos

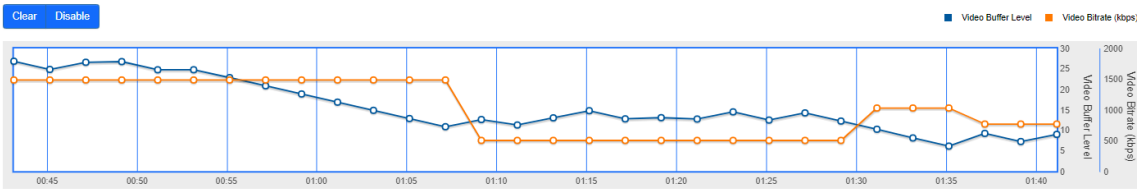

Figura 4.1: Buffer y bitrate de la prueba con segmentos de 2 segundos en la primera simulación

Como se puede observar en la figura 4.1, en esta prueba se solicita un archivo de mayor o de menor calidad a la mínima que el nivel de buffer del cliente aumenta o disminuye. Esto supone un mayor número de cambios de calidad para mantener el buffer lo más estable posible.

4.1.2 Segmentos de 4 segundos

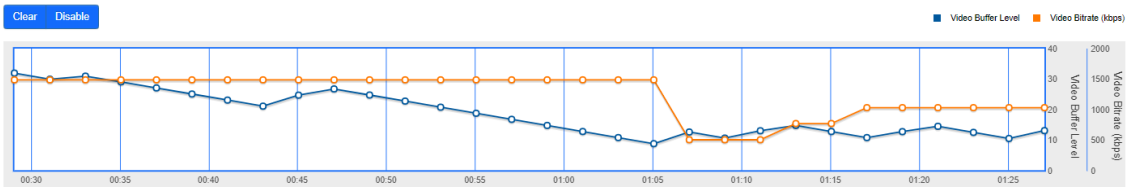

Figura 4.2: Buffer y bitrate de la prueba con segmentos de 4 segundos en la primera simulación

Muy similar a la anterior, pero dejando un poco más de espacio a que el buffer fluctúe, manteniendo la calidad durante un poco más de tiempo que la prueba anterior y consiguiendo en algunos casos, menos cambios de calidad como se puede apreciar en la figura 4.2.

4.1.3 Segmentos de 8 segundos

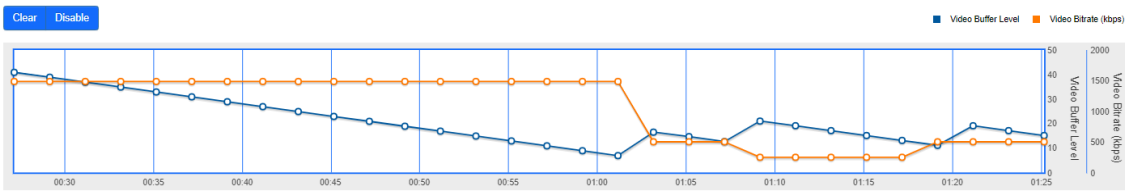

Figura 4.3: Buffer y bitrate de la prueba con segmentos de 8 segundos en la primera simulación

En esta tercera prueba la calidad se mantiene notablemente más tiempo que en las 2 pruebas anteriores, permitiendo que el buffer disminuya su nivel bastante, como se aprecia en el intervalo de tiempo 1:05-1:20 de la figura 4.3, aunque cambiando a una calidad mayor antes de que la imagen pueda quedarse congelada. Esto permite mantener una misma calidad más tiempo y disminuir el número de cambios.

4.1.4 Segmentos de 16 segundos

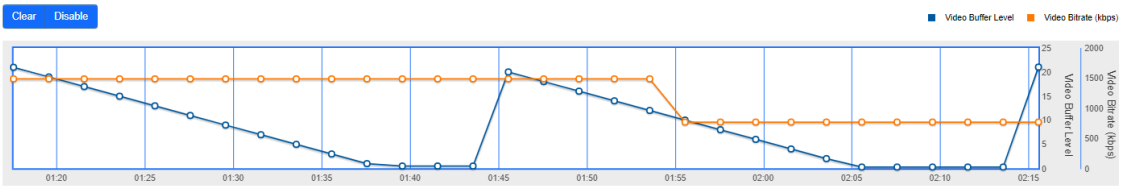

Figura 4.4: Buffer y bitrate de la prueba con segmentos de 16 segundos en la primera simulación

Esta última prueba con la primera limitación muestra como la calidad del video se mantiene aunque el buffer se vacíe del todo. En la figura 4.4 se puede observar como en los instantes 1:40 y 2:10 el buffer se ha vacíado por completo provocando la congelación del video. Por lo tanto, siendo una limitación no muy pronunciada y obteniendo este resultado con este tamaño de segmento, queda descartada para la siguiente limitación que se realiza debido a que el vídeo se detiene.

4.1.5 Conclusión primera simulación

Al mantener el buffer lo más estable posible, la calidad del video aumenta y disminuye un gran número de veces asegurando la estabilidad del buffer pero a cambio de muchos cambios de calidad que pueden ser percibidos por el usuario y por lo tanto se reduce su calidad de experiencia (QoE). Los segmentos de 8 segundos mantienen la calidad más

alta posible durante el máximo tiempo que puede sin llegar a vaciar el buffer. Esto supone que el usuario vea una calidad constante durante un largo periodo en lugar de fluctuaciones, pero tarde o temprano, el buffer se tiene que reestablecer y la calidad bajará de forma brusca y notoria. Como se ha visto, 16 segundos de segmento no es viable porque, aunque se mantenga el nivel de calidad más alto, el video se va a congelar varias veces hasta que cambie a una calidad inferior y esto es lo que se intenta evitar.

Concluyendo, como primera observación de adaptabilidad decir que la prueba de segmentos de 4 segundos es la que mejor se adapta a nuestro caso de estudio, ya que, aunque realiza más cambios que la prueba de segmentos de 8 segundos, ésta mantiene un equilibrio entre calidad y buffer. Esta opción nos ofrecerá varios cambios a calidades cercanas poco perceptibles y nos mantendrá el buffer estable.

4.2 Segunda simulación

La primera fluctuación realizada, se ha realizado con el objetivo de descartar los tamaños de segmento que no son capaces de adaptarse a una disminución del ancho de banda y mantener la reproducción constante sin detener el video. Esta limitación comenzará parecido a la anterior. En el segundo 5 se limita el ancho de banda a 900Kbps hasta llegar al minuto 1. En ese instante se vuelve a limitar, pero esta vez a 500Kbps durante 15 segundos para después comprobar que vuelve a la calidad más alta.

4.2.1 Segmentos de 2 segundos

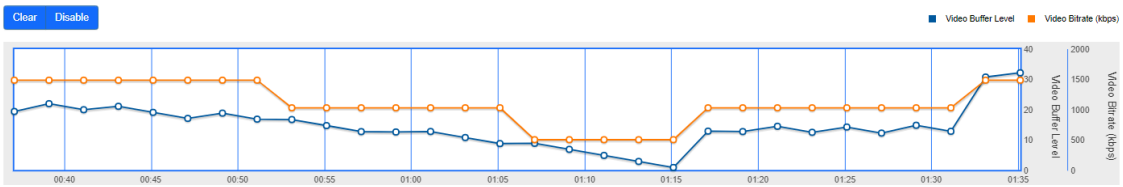

Figura 4.5: Buffer y bitrate de la prueba con segmentos de 2 segundos en la segunda simulación

Al igual que en la prueba anterior, en la figura 4.5 se observa como se realizan muchos cambios de calidad. Aunque estos cambios son a calidades cercanas, en esta implementación se han creado pocas versiones disponibles entre las que cambiar de calidad, por lo que estos cambios serán perceptibles por el usuario. La recuperación a la calidad más alta es muy rápida.

4.2.2 Segmentos de 4 segundos

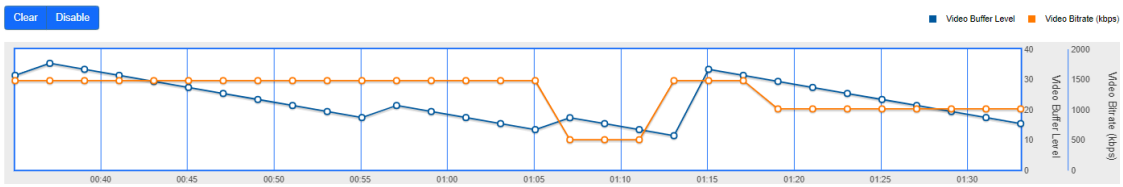

Figura 4.6: Buffer y bitrate de la prueba con segmentos de 4 segundos en la segunda simulación

Como se puede apreciar en la figura 4.6, para este tamaño de segmento se realizan menos cambios de calidad reteniendo un poco más cada una sin poner en riesgo el buffer. La recuperación a la calidad más alta es más lenta que en la prueba anterior.

4.2.3 Segmentos de 8 segundos

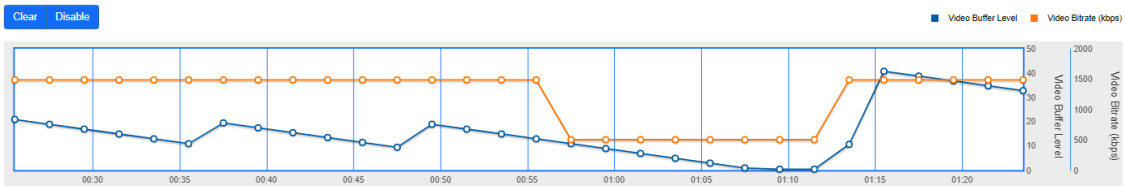

Figura 4.7: Buffer y bitrate de la prueba con segmentos de 8 segundos en la segunda simulación

A diferencia de la primera limitación, esta vez se hace una limitación más pronunciada y la retención de la calidad durante tanto tiempo en este caso, como se ve en la figura 4.7, ha conllevado a la detención del video un par de segundos alrededor del instante 1.10. La recuperación ha coincidido que la detención del video, lo que ha provocado que el buffer se llene muy rápido y con ello ha llegado inmediatamente a la calidad superior.

4.2.4 Conclusión segunda simulación

Esta vez, los segmentos de 8 segundos no han servido para ofrecer el servicio que este estándar debe proporcionar, al igual que los segmentos de 16 segundos de la primera limitación. En esta segunda simulación, al igual que en la primera, los segmentos de 4 segundos realizan menos variaciones que los segmentos de 2 segundos, y además el buffer se ha mantenido más estable. Por lo tanto, los segmentos de 4 segundos son la mejor opción para esta implementación.

Conclusiones

Después de haber estudiado diversos estándares lanzados por grandes compañías como Microsoft, Adobe y Apple cada una lanza sus soluciones en función a sus intereses, con restricciones en la codificación y en los formatos del contenido. Estas soluciones de transmisión adaptativa no proporcionan confianza al mercado porque cada una de ellas genera discriminación en ciertos dispositivos haciendo difícil la adopción de una de ellas como solución universal. Es por ello que surge la necesidad de crear una solución que ofrezca compatibilidad con todos los dispositivos, crear una implementación universal y estandarizarla. Y eso es MPEG-DASH.

En este proyecto se han estudiado estas soluciones y se ha hecho una sencilla implementación del estándar MPEG-DASH probando el funcionamiento de la tecnología adaptativa en la cual se basa. Para poner a prueba nuestra implementación y demostrar el uso de esta adaptabilidad se han realizado distintas pruebas generando limitaciones en el ancho de banda con el fin de simular distintos escenarios reales pero controlados. De esta manera se ha podido estudiar el comportamiento del estándar con diferentes tamaños de segmento. El funcionamiento de nuestra implementación ha sido el esperado, con una correcta adaptabilidad del flujo en los distintos escenarios simulados. Además, con esos resultados, se ha podido sacar información sobre qué tamaño de segmento es el más indicado para ofrecer la mejor solución para esta implementación. Han resultado ser, los segmentos de menor tamaño de entre los puestos a prueba, los que mejor resultado han dado. Más específicamente, los segmentos de 4 segundos, han demostrado un mayor equilibrio entre cambios de calidad y estabilidad del buffer, además de ser el que ha ofrecido mayor bitrate de media de entre los más indicados. Aun así, es conveniente añadir más versiones para que las variaciones de calidad no sean tan distintas y evitar que el usuario pueda percibir los cambios. Esto no significa que sea igual para todas las implementaciones. Según la forma de codificar y adaptar el contenido al estándar pueden surgir resultados diferentes. Además, se puede apreciar claramente en las pruebas, que cuanto mayor es la disminución del ancho de banda, es mejor utilizar segmentos de menor tamaño. Por lo que, para pequeñas bajadas de ancho de banda, un segmento de mayor tamaño sería más adecuado. Por lo tanto, disponer en el servidor de los archivos segmentados con diferentes tamaños puede resultar ser una solución más eficaz, aunque supone la necesidad más espacio de almacenamiento en el lado del servidor.

Índice de figuras

Figura 2.1: Esquema de comunicación entre un cliente (Web Browser) y el servidor RTSP (Web Server)……..…….9

Figura 2.2: Formato de un fragmento de Microsoft Smooth Streaming…….…..16

Figura 2.3: Archivo manifiesto de Microsoft Smooth Streaming…….…..17

Figura 2.4: Representación del funcionamiento y la arquitectura de Apple HTTP Live Streaming……...18

Figura 2.5: Representación de la estructura de archivos de Apple HTTP Live Streaming……...19

Figura 2.6: Manifiesto de Apple HTTP……..19

Figura 2.7: Manifiesto de Apple HTTP Live Streaming……...20

Figura 2.8: Submanifiesto de Apple HTTP Live Streaming……20

Figura 2.9: Captura del tráfico de contenido en vivo de twitch.tv……..21

Figura 2.10: Captura del tráfico de contenido de dplay.com……..22

Figura 2.11: Representación del funcionamiento y la arquitectura de Adobe HTTP Dynamic Streaming……..23

Figura 2.12: Manifiesto de Adobe HTTP Dynamic Streaming……...23

Figura 2.13: Submanifiesto de Adobe HTTP Dynamic Streaming……24

Figura 2.14: Representación del funcionamiento y la arquitectura de MPEG-DASH…25

Figura 2.15: Representación de los elementos que forman el manifiesto de MPEG-DASH y su jerarquía……26

Figura 2.16: Formato de un segmento ISO base media file format en DASH……33

Figura 2.17: Profiles de MPEG-DASH……...35

Figura 3.1: Escenario de la implementación de MPEG-DASH……..36

Figura 3.2: Activación del servicio Internet Information Services para Windows……...37

Figura 3.3: Configuración del servidor……...38

Figura 3.4: Reproductor multimedia DASH…….……39

Figura 3.5: Reproducción del contenido mediante el reproductor multimedia VLC…...40

Figura 3.6: Herramienta incorporada en Mozilla para capturar tráfico……40

Figura 3.7: Programa FFmpeg para la transcodificación de los archivos……41

Figura 3.8: Preparación del programa FFmpeg……...42

Figura 3.9: Instalación del programa GPAC……43

Figura 3.10: Uso de la herramienta NetLimiter……44

Figura 4.1: Buffer y bitrate de la prueba con segmentos de 2 segundos en la primera simulación……..49

Figura 4.2: Buffer y bitrate de la prueba con segmentos de 4 segundos en la primera simulación……..49

Figura 4.3: Buffer y bitrate de la prueba con segmentos de 8 segundos en la primera simulación……..50

Figura 4.4: Buffer y bitrate de la prueba con segmentos de 16 segundos en la primera simulación……..50

Figura 4.5: Buffer y bitrate de la prueba con segmentos de 2 segundos en la segunda simulación……..51

Figura 4.6: Buffer y bitrate de la prueba con segmentos de 4 segundos en la segunda simulación……..52

Figura 4.7: Buffer y bitrate de la prueba con segmentos de 8 segundos en la segunda simulación……..52

Índice de tablas

Tabla 2.1: Tabla Resumen de las diferentes tecnologías de streaming……15

Tabla 2.2: Semántica del elemento Period…….....27

Tabla 2.3: Semántica del elemento Adaptation Set……..29

Tabla 2.4: Semántica del elemento Representation……30

Tabla 4.1: Archivo de video codificado a distintas tasas de bits y resolución……....48

Bibliografía

[1] https://es.wikipedia.org/wiki/Protocolo_de_transferencia_de_hipertexto

[2] https://es.wikipedia.org/wiki/Streaming

[3] https://es.wikipedia.org/wiki/Protocolo_de_transmisi%C3%B3n_en_tiempo_real

[4] MPEG's Dynamic Adaptive Streaming over HTTP (DASH) –Enabling Formats for Video Streaming over the Open Internet. http://tech.ebu.ch/docs/events/ webinar043-mpeg-dash/presentations/ebu_mpeg-dash_webinar043.pdf. pág. 26, 31

[5] DASH Support in MP4Box https://gpac.wp.imt.fr/mp4box/dash/

[6] https://docs.microsoft.com/en-us/iis/media/smooth-streaming/smooth-streaming-primer

[7] Streaming Adaptativo en Internet http://www.javierrodriguez.com.es/blog/2011/10/24/streaming-adaptativo-en-internet-2/

[8] https://zencoder.com/es/hls-guide

[9] https://www.encoding.com/http-dynamic-streaming-hds/

[10] MPEG DASH: el streaming dinámico adaptativo sobre HTTP levanta vuelo https://www.panoramaaudiovisual.com/2012/02/29/mpeg-dash-el-streaming-dinamico-adapativo-sobre-http-levanta-vuelo/

[11] Multimedia Systems II: DASH Advanced Features http://sce2.umkc.edu/csee/lizhu/teaching/2017.spring.multimedia-communication/notes/lec16.pdf pág. 6

[12] http://reference.dashif.org/dash.js/

[13] http://reference.dashif.org/dash.js/v3.0.0/samples/dash-if-reference-player/index.html

[14] https://peach.blender.org/download/

Bibliografía adicional

También se ha sacado información general de estos textos para realizar el trabajo aunque no estan explicitamente indicados en éste.

http://www.javierrodriguez.com.es/blog/2011/10/24/streaming-adaptativo-en-internet-2/

https://letzgro.net/blog/the-variety-of-streaming-protocols/

https://ucema.edu.ar/publicaciones/download/documentos/660.pdf

http://42jaiio.sadio.org.ar/proceedings/simposios/Trabajos/EST/19.pdf

http://www.dit.upm.es/~posgrado/doc/TFM/TFMs2015-2016/TFM_Erika_del_Rocio_Intriago_Acuna_2016.pdf

https://www.encoding.com/mpeg-dash/

https://www.encoding.com/http-live-streaming-hls/

https://www.encoding.com/microsoft-smooth-streaming/

https://www.encoding.com/http-dynamic-streaming-hds/

https://www.encoding.com/packaging/

https://hal.archives-ouvertes.fr/tel-01249840/document

https://www.wowza.com/blog/streaming-protocols-latency

https://www.wowza.com/live-video-streaming/complete-guide-to-live-streaming

https://www.wired.com/story/adobe-finally-kills-flash-dead/

https://jfherrera.wordpress.com/2011/05/05/rtmp/

https://en.wikipedia.org/wiki/Real-Time_Messaging_Protocol

https://www.bogotobogo.com/VideoStreaming/images/mpeg_dash/DASH-IEEE-multimedia-preprint.pdf

https://mastermoviles.gitbook.io/graficos-y-multimedia/difusion-de-medios

https://riunet.upv.es/bitstream/handle/10251/109785/Valle%20-%20EVALUACI%C3%93N%20SUBJETIVA%20DE%20UN%20SISTEMA%20DE%20STREAMING%20DE%20V%C3%8DDEO%20BASADO%20EN%20DASH.pdf?sequence=1&isAllowed=y

https://e-archivo.uc3m.es/bitstream/handle/10016/22427/PFC_victor_mata_galiano_2014.pdf

https://es.wikipedia.org/wiki/Protocolo_de_transmisi%C3%B3n_en_tiempo_real

https://es.wikibooks.org/wiki/HTTP/Conexiones/Conexiones_Persistentes

https://code.tutsplus.com/es/tutorials/http-succinctly-http-connections--net-33707

https://github.com/arut/nginx-rtmp-module/wiki/Directives#mpeg-dash

https://es.wikiversity.org/wiki/Tecnolog%C3%ADas_multimedia_e_interacci%C3%B3n/Streaming

https://es.wikipedia.org/wiki/Flash_Video

https://bitmovin.com/status-mpeg-dash-today-youtube-netflix-use-html5-beyond/

https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=6761273

https://pdfs.semanticscholar.org/1d93/c81712a5a222f4db561fa6236d485e88ec61.pdf

https://es.wikipedia.org/wiki/Streaming#Protocolos_ligeros

https://www.wowza.com/blog/streaming-protocols

https://www.dacast.com/blog/streaming-protocols/

https://es.slideshare.net/christian.timmerer/mpegdash-overview-stateoftheart-and-future-roadmap

https://www.slideshare.net/agiladi/alex-giladi-dashtsr3

https://bitmovin.com/mp4box-dash-content-generation-x264/

http://forum.doom9.org/archive/index.php/t-172591.html

http://profesores.elo.utfsm.cl/~agv/elo323/2s10/projects/ApablazaBustamante/desc.html

https://docplayer.es/11389797-Dash-un-estandar-mpeg-para-streaming-sobre-http.html

https://zencoder.com/es/hls-guide

https://sites.google.com/a/iharrow.org.uk/compsci/1-1-data-representation/1-1-3-data-storage/3-midi-jpeg-mp3-mp4/4-mp4

https://www.bogotobogo.com/VideoStreaming/Files/iis8/RGB_Adaptive_HTTP_Streaming_Comparison_1211-01.pdf

https://www.bogotobogo.com/VideoStreaming/ffmpeg_AdaptiveLiveStreaming_SmoothStreaming.php

https://es.wikiversity.org/wiki/Tecnolog%C3%ADas_multimedia_e_interacci%C3%B3n/Streaming#HLS_(HTTP_Live_Stream)