UNIVERSIDAD AUTÓNOMA METROPOLITANA
UNIDAD IZTAPALAPA

División de Ciencias Básicas e Ingeniería
Departamento de Ingeniería Eléctrica
Posgrado en Ciencias y Tecnologías de la Información

TRANSMISIÓN ADAPTATIVA DE VÍDEO EN REDES DE COMPUTADORAS CON PROPIEDADES DINÁMICAS

Idónea Comunicación de Resultados:

Erik Miguel Díaz Salazar

Para obtener el grado de

Maestro en Ciencias y Tecnologías de la Información

Asesores:

Dr. Víctor Manuel Ramos Ramos
Departamento de Ingeniería Eléctrica, UAM - Iztapalapa

M. en C. Carlos Ernesto Carrillo Arellano
Departamento de Electrónica, UAM - Azcapotzalco

Sustentada el 19 de Noviembre de 2019 a las 12:30 hrs. ante el jurado:

Dr. Enrique Stevens Navarro, UASLP, Presidente
Dra. Graciela Roman Alonso, UAM - Iztapalapa, Secretaria
Dr. Víctor Manuel Ramos Ramos, UAM - Iztapalapa, Vocal

UNIVERSIDAD AUTÓNOMA METROPOLITANA
UNIDAD IZTAPALAPA
División de Ciencias Básicas e Ingeniería
Departamento de Ingeniería Eléctrica
Posgrado en Ciencias y Tecnologías de la Información

TRANSMISIÓN ADAPTATIVA DE VÍDEO EN REDES DE COMPUTADORAS CON PROPIEDADES DINÁMICAS

Idónea Comunicación de Resultados:

Erik Miguel Díaz Salazar

Para obtener el grado de

Maestro en Ciencias y Tecnologías de la Información

Asesores:

Dr. Víctor Manuel Ramos Ramos
Departamento de Ingeniería Eléctrica, UAM - Iztapalapa

M. en C. Carlos Ernesto Carrillo Arellano
Departamento de Electrónica, UAM - Azcapotzalco

Sustentada el 19 de Noviembre de 2019 a las 12:30 hrs. ante el jurado:

Dr. Enrique Stevens Navarro, UASLP, Presidente
Dra. Graciela Roman Alonso, UAM - Iztapalapa, Secretaria
Dr. Víctor Manuel Ramos Ramos, UAM - Iztapalapa, Vocal

Resumen

El *streaming* de vídeo es uno de los servicios más utilizados hoy en día. Su gran atractivo es la posibilidad de poder acceder a él a través de múltiples dispositivos como teléfonos inteligentes, smart TV's, tablets, consolas de videojuegos, entre otros, sin la necesidad de depender de una plataforma en específico como en sus inicios era requerido en estos servicios. Para el funcionamiento de un sistema de *streaming* se requiere de dos componentes principales: Al menos un servidor web y un cliente capaz de reproducir audio y vídeo. Cada uno de estos elementos han impuesto distintas líneas de investigación que buscan mejorar el desempeño de la tecnología en general. En el lado del servidor se encuentra almacenado el contenido multimedia, el cual está compuesto por distintas versiones del mismo, diferenciándose entre sí por su tasa de bits, conocidas en la literatura como *representaciones* de vídeo; así mismo, cada una de estas se encuentran segmentadas en pedazos fijos de entre cuatro y diez segundos de duración. Por otro lado, se tiene el archivo manifiesto o MPD, el cual utilizando una estructura jerárquica basado en XML describe cómo están organizadas las *representaciones* en el servidor. Finalmente, también se encuentran los algoritmos de Tasa de bits Adaptativa o ABR, los cuales son ejecutados en el lado del cliente; su función principal es medir las condiciones de la red, tomando como parámetros de medición el ancho de banda, throughput, niveles de buffer, entre otros. Lo anterior, con el propósito de que el cliente seleccione a través del MPD la mejor *representación* durante la sesión de *streaming*.

En su conjunto, a estos mecanismos se les conoce como *Streaming* Dinámico Adaptativo sobre HTTP o DASH, el cual es el estándar de mayor aceptación actualmente para la transmisión de vídeo. Entre sus principales características ofrece la posibilidad de una reproducción continua, reduciendo en gran medida el número de interrupciones, o evitando la oscilación entre las representaciones debido a sub o sobreestimaciones en las condiciones de la red, entre otros.

A pesar de los beneficios que ofrece DASH, diversos trabajos en la literatura han identificado diferentes problemas que pueden afectar seriamente su desempeño. Es por ello por lo que en este trabajo de investigación se propone establecer un mecanismo que utilice estimadores mixtos usando como parámetros la estimación del *throughput* y los niveles del buffer. Por otro lado, se toman en consideración la resolución del dispositivo y el tipo de contenido que se está reproduciendo. A través de este se puede obtener un peso de calidad de percepción, muy útil como valor de ponderación en los estimadores, y mediante un tipo de plan de servicio se puede acotar un máximo de tasas de bits que el cliente DASH puede solicitar durante el proceso de reproducción. Además, para la toma de decisiones entre los estimadores implementados, se utiliza un esquema basado en la teoría de juegos denominado Game Theory Approach o GTA que busca encontrar un consenso entre dos o más clientes DASH en la elección de una tasa de bits ante un cuello de botella en la red. Finalmente se propone un algoritmo que busca el consenso entre dos o más estimadores con el propósito de capturar con una mayor precisión el comportamiento y la variabilidad de las condiciones de la red y elegir la tasa de bits idónea.

Agradecimientos

Al Dr. Víctor Manuel Ramos Ramos por aceptarme para la realización de este proyecto de investigación. Gracias por la confianza en mi trabajo, los consejos, las charlas y anécdotas compartidas, de lo contrario todo esto difícilmente sería posible.

Al M. en C. Carlos Ernesto Carrillo Arellano por ser co-asesor de este proyecto y sobre todo por proponer la línea de investigación.

A la Dra. Graciela Román Alonso y al Dr. Enrique Stevens Navarro por aceptar ser parte del jurado evaluador de esta Idónea Comunicación de Resultados y por tomarse el tiempo para brindarme sus observaciones.

A los profesores y compañeros del posgrado en Ciencias y Tecnologías de la Información de la UAM Unidad Iztapalapa por darme la oportunidad de crecer académica y personalmente.

Al Consejo Nacional de Ciencia y Tecnología, CONACYT y a la Universidad Autónoma Metropolitana por otorgarme cada uno el apoyo económico para cursar y concluir el posgrado cumpliendo así, con otra meta más en mi vida.

Finalmente a mi familia y principalmente a mis padres por su apoyo, paciencia, compresión, amor y cariño. Son mis ejemplos a seguir, gracias por todo.

Contenido

Lista de Figuras

Lista de Tablas

Lista de Bloques de Código

Lista de Acrónimos

Hypertext Transfer Protocol (HTTP)
Moving Picture Experts Group (MPEG)
Dynamic Adaptive Streaming over HTTP (DASH)
International Organization for Standardization (ISO)
HTTP Adaptive Streaming (HAS)
Adaptive Bitrate (ABR)
Extensible Markup Language (XML)
Media Presentation Description (MPD)
Quality of Experience (QoE)
Quality of Service (QoS)
Acknowledgement (ACK)
HyperText Markup Language (HTML)
Frames Per Second (FPS)
Transmission Control Protocol (TCP)
Real-Time Transport Protocol (RTP)
Real-Time Transport Control Protocol (RTCP)
Real Time Streaming Protocol (RTSP)
Content Delivery Network (CND)
Group of Pictures (GOP)
Stream Access Points (SAP)
Instantaneous Decoder Refresh (IDR)
Request For Comments (RFC)
Random Access Points (RAP)
Media Base File Format (MBFF)
Server And Network Assisted DASH (SAND)
Segment Fetch Time (SFT)
Digital Rights Management (DMR)
User Datagram Protocol (UDP)

1 Introducción

En los últimos años, el *streaming* de vídeo se ha vuelto una de las aplicaciones más utilizadas en Internet y va en crecimiento. De acuerdo con *Visual Networking Index: Forecast and Methodology 2017-2022*, se estima que para el año 2022 el tráfico originado por la transmisión de vídeo representará más del 82% del tráfico global de datos, como se muestra en la Figura 1.1 [1].

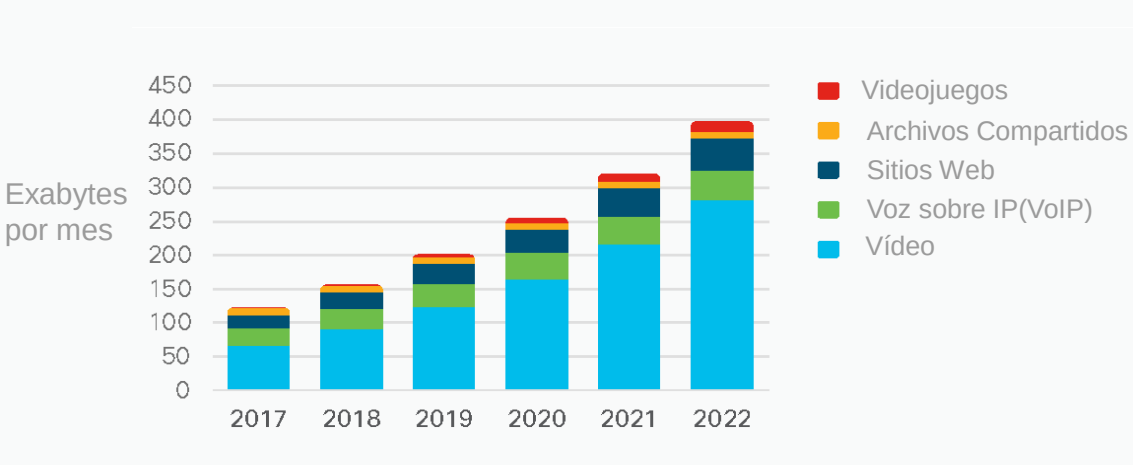

Figura 1.1: Cisco VNI Global IP Traffic Forecast, 2017-2022.

Dentro del propio tráfico de vídeo en Internet como puede verse en la Figura 1.2, en particular, el vídeo on demand y el live *streaming* tienen el gran potencial para atraer una gran cantidad de tráfico ya que comienzan a sustituir a las emisiones tradicionales. El live *streaming* ya representa más del 10% del tráfico de vídeo en Internet y se incrementará hasta alcanzar el 17% en el año 2022. Mientras que el tráfico originado por el video on demand se ha mantenido por encima del 61%. Cabe destacar que el crecimiento del tráfico de videovigilancia (*dropcams*), es de una naturaleza muy diferente a las aplicaciones de *streaming* antes mencionadas, toda vez que representa un flujo ascendente de tráfico desde una cámara de vídeo, ya que su carga se da de forma continua desde los hogares y las pequeñas empresas a la nube.

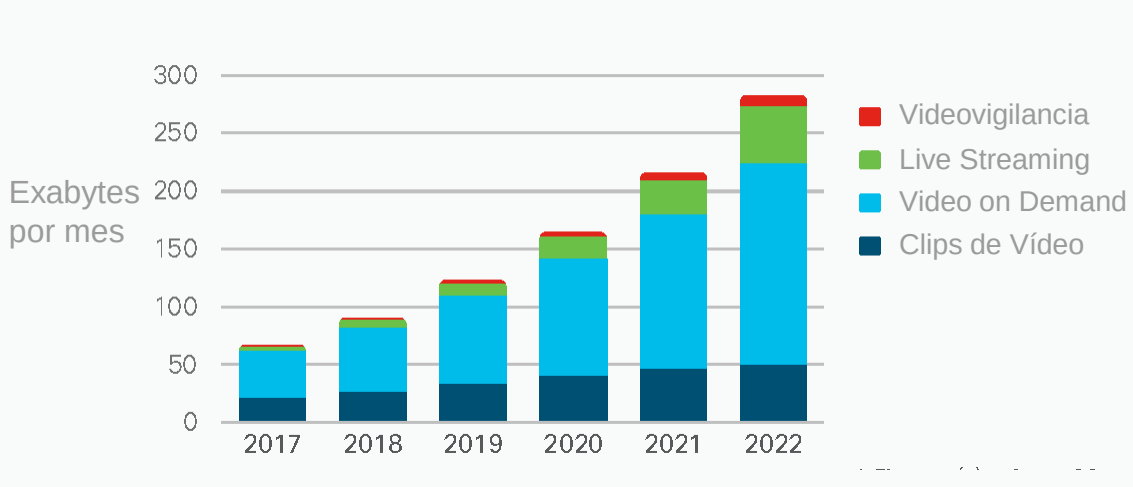

Figura 1.2: Tráfico global entre aplicaciones de vídeo, 2017-2022.

Por otro lado, el número de dispositivos que permiten visualizar contenidos en *streaming* también ha aumentado de forma que ahora es posible reproducir contenido multimedia en smartphones, smart TV's, tablets, consolas de videojuegos tanto de sobremesa como portátiles, entre otros. Ahora es posible ver en el autobús o en la comodidad de la sala, el capítulo de alguna serie o una película. Anteriormente, esto no era posible debido a la existencia de diversos protocolos de *streaming*, contenidos de vídeo en diferentes formatos que no eran siempre compatibles con la mayoría de los dispositivos. Además, la velocidad de los enlaces en el núcleo de Internet ha continuado exponenciándose vertiginosamente desde el año 2007.

Para resolver estos problemas, en años recientes se ha estandarizado el *Streaming* Dinámico Adaptativo sobre HTTP o DASH (Dynamic Adaptive Streaming over HTTP), donde en el lado del cliente se miden las condiciones de la red y se solicita un segmento de vídeo, que se encuentra almacenado en un servidor HTTP [2, 3]. Cada uno de estos segmentos sólo difieren en su tasa de bits, y están fragmentados de manera uniforme en términos temporales. En la Figura 1.3, se muestra el funcionamiento de esta técnica.

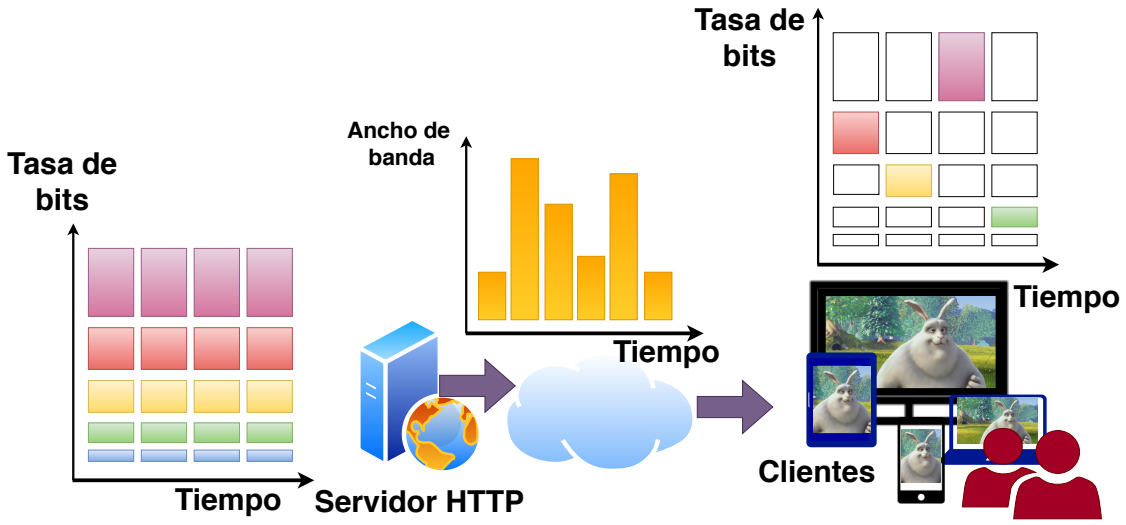

Figura 1.3: Streaming de vídeo utilizando DASH.

1.1 Planteamiento del Problema

El problema que es abordado en este trabajo de investigación es el de desarrollar una estrategia que permita al cliente seleccionar la *representación* de vídeo adecuada a través de una metodología de toma de decisiones. Dicha metodología debe utilizar más de un estimador, considerando diferentes parámetros de la red. Esto es debido a que las condiciones de la misma pueden ser altamente dinámicas provocando que la Calidad de la Experiencia o QoE se vea afectada durante la reproducción del contenido multimedia, lo cual resulta un reto. Sin una adecuada estrategia de adaptación, el cliente puede sufrir de interrupciones frecuentes además de una degradación significativa de la calidad visual, debido a constantes oscilaciones. Por ejemplo, una *representación* de vídeo con una tasa de bits más alta que el ancho de banda disponible causaría congestión en la red y, por otro lado, cuando es menor que el ancho de banda disponible, la calidad visual no sería aprovechada al máximo permitido. Habiendo establecido esto, surgen las siguientes interrogantes que serán resueltas posteriormente con el fin de dar una posible solución que sea factible. A partir de ello, se plantean las siguientes preguntas:

- ¿Cómo se integran varios parámetros para tomar una decisión mucho más inteligente?
- ¿Cómo hacer para que la toma de decisiones en cada parámetro no sólo se centre en momentos instantáneos sino que además se base en un historial de sucesos ocurridos en la red y decisiones pasadas?

1.2 Justificación

Muchos de los algoritmos de estimación del ancho de banda que han sido propuestos en la literatura, en la mayoría de los casos solamente se centran en ese parámetro y en cómo va evolucionando el mismo durante la reproducción. Algunos de estos algoritmos sólo hacen promedios mediante ventanas de tiempo para tomar la decisión de descargar el siguiente segmento. Sin embargo, existen varios problemas con esas estimaciones, ya que cuando se inicia una conexión TCP, en este caso una descarga HTTP por cada uno de los segmentos, se utiliza el mecanismo basado en una ventana de congestión para acotar la cantidad de datos que pueden ser enviados. Debido al mecanismo *congestion avoidance* de TCP, la ventana de congestión se va ampliando hasta que se detectan pérdidas o expiran los contadores de recepción de los acuses de recibo (ACK). Este mecanismo en forma de diente de sierra, provoca que se complique la estimación del ancho de banda, llevando a una incorrecta toma de decisiones. Sólo basarse en el ancho de banda puede ser perjudicial, así como en otros parámetros que no estén correlacionados.

En la Figura 1.4, se muestra cómo a través de una adecuada toma de decisiones será posible adaptarse a las condiciones de la red y determinar cuál es la siguiente representación que será descargada sin depender sólo de un parámetro. Se espera encontrar que, mediante el uso de más de un único parámetro, estos conjuguen y se tome una decisión más inteligente. El problema está en monitorear varias variables y su comportamiento temporal.

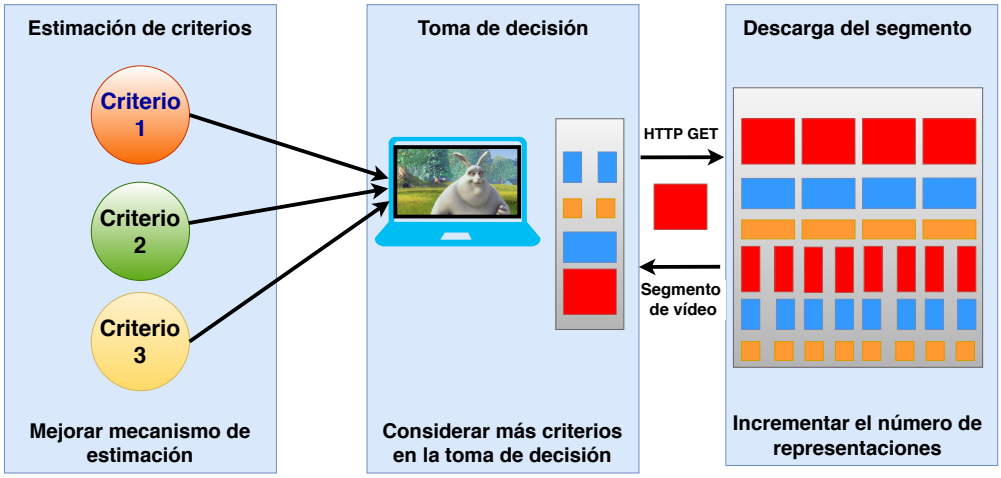

Figura 1.4: Modelo del proceso de adaptación de HAS.

1.3 Objetivo General

Proponer y evaluar nuevas estrategias para la estimación, creación y selección de segmentos para DASH de manera que se mejore el proceso de adaptación del *streaming* de vídeo sobre HTTP en redes altamente dinámicas.

1.4 Objetivos Particulares

- Realizar la implementación de un sistema de *streaming* de DASH, el cual permita desde un cliente acceder al contenido multimedia de un MPD almacenado en un servidor web.
- Incorporar un nuevo mecanismo de estimación de parámetros en un reproductor de vídeo compatible con DASH y evaluar su desempeño.
- Evaluar los beneficios potenciales que podrían obtenerse al incorporar más representaciones de vídeo.

1.5 Metodología y Fases

(i) Estudiar y comprender de forma clara los procesos que se realizan para el funcionamiento del *streaming* adaptativo de vídeo sobre HTTP, desde la preparación de los contenidos multimedia, hasta la descarga y reproducción de estos.

(ii) Estudiar las estrategias propuestas en la literatura para mejorar el proceso de adaptación del *streaming*, particularmente las estrategias de estimación del ancho de banda.

(iii) Proponer una nueva estrategia de estimación, así como seleccionar cuáles criterios podrían fortalecer el proceso de toma de decisiones.

1.6 Organización de la Idónea Comunicación de Resultados

Este trabajo de investigación se organiza de la siguiente forma: En la Sección 2, se describen brevemente los componentes que definen un formato de un archivo de vídeo, asimismo algunos protocolos y esquemas de *streaming*. Del mismo modo, en la Sección 3 se abordan las características más importantes del esquema de *streaming* DASH. En la Sección 4, se muestra una síntesis del estado del arte de los algoritmos de adaptación de tasas de bits o ABR. En la Sección 5, se describe el modelo del algoritmo ABR propuesto. La Sección 6, detalla las características de los componentes del banco de trabajo. En la Sección 7, se muestran los resultados obtenidos. En la Sección 8, se presentan las conclusiones y el trabajo futuro de este trabajo de investigación.

2 Antecedentes

En la práctica, el concepto de *streaming* significa que cuando un espectador hace clic en el botón de reproducción en un sitio web, el vídeo o audio comienza a reproducirse inmediatamente y continúa de forma más o menos fluida hasta el final. Para que esto suceda, la tasa de bits del archivo debe ser menor que la capacidad de ancho de banda del cliente, ya que, de lo contrario, el contenido multimedia sufriría de interrupciones con demasiada frecuencia. Una definición más técnica de *streaming* es el contenido multimedia entregado a través de un servidor de *streaming*, que es un programa de software que se encarga exclusivamente de la entrega de contenido multimedia. Esto contrasta con un servidor HTTP tradicional que procesa los sitios web y ofrece la entrega de todo contenido compatible, incluyendo HTML, imágenes JPEG y GIF, archivos PDF, entre otros. Esto más adelante demostrará ser una gran ventaja.

2.1 Formato de Vídeo

Existen diferentes formatos para archivos de vídeo ya que sirven para distintos propósitos y no son los mismos para todos los casos. Por ejemplo, ya sea que se requiera subir un vídeo a alguna plataforma de *streaming* como YouTube o se requiera proyectarlo en una pantalla grande, por lo que se querrá producir un vídeo en el mejor formato y calidad para su propósito. A continuación, se mencionan las principales propiedades que hay que tener en cuenta.

2.1.1 Tamaño del Fotograma (*Frame Size*)

El tamaño del fotograma está relacionado con la *resolución*, la cual es expresada por el número de pixeles en los que se divide el fotograma en horizontal y vertical. Cada pixel es un fragmento de datos que será almacenado, por lo tanto, cuanto mejor sea la resolución, mayor será el tamaño del fotograma y entonces necesitará mayor ancho de banda para su transmisión. Los más comunes hoy en día se muestran en la Tabla 2.1.

2.1.2 Relación de Aspecto (*Aspect Ratio*)

Es el término usado para describir las dimensiones de una imagen, comparando el ancho con el alto, expresándose en forma de relación ancho:alto (el ancho siempre en primer lugar). 16:9 es la relación de aspecto más común hoy en día. En la Tabla 2.2, se muestran las relaciones de aspecto más comunes.

Tabla 2.1: Resoluciones para un vídeo.

Resolución		Calidad
3840x2160	2160p	4K
2560x1440	1440p	2K
1920x1080	1080p	Máxima resolución para HD
1280x720	720p	Mínima resolución para HD
854x480	480p	Resolución estándar
640x360	360p	Resolución tradicional para un sitio web
426x240	240p	Resolución minima para un vídeo en Youtube

Tabla 2.2: Relaciones de aspecto más comunes.

Relación de aspecto		Uso	Dispositivo
4:3	1.33:1	Canales Estándar	Televisores Antiguos
16:9	1.77:1	Canales HD	Televisores HD
21:9	2.35:1	Cine en General	Salas de Cine
14:10	1.4:1	Cine en Formato IMAX	Salas Especializadas
19:10	1.9:1	IMAX Digital	Salas Especializadas

2.1.3 Velocidad del Fotograma (*Frame Rate*)

Es la velocidad a la que se capturan y envían los fotogramas, tiene un gran impacto en el estilo y la experiencia de visualización de un vídeo, se mide en Cuadros por Segundo o fps (*Frames per Second*). Las diferentes velocidades de fotogramas producen resultados diferentes, lo que a su vez determina el grado de realismo en el vídeo. Al elegir una frecuencia muy alta, los objetos empezarán a tener un aspecto antinatural sufriendo lo que se conoce como el *Efecto Telenovela* (*Soap Opera Effect*), esencialmente el vídeo muestra demasiados detalles [4]. Por otro lado, al elegir una frecuencia demasiado baja, el vídeo empezará a tener un aspecto entrecortado y proporcionará una mala calidad de la experiencia. La Figura 2.1, muestra la diferencia de una escena de un segundo al contar con 24 fps y 60 fps. Las opciones más comunes y cómo se utilizan se describen a

continuación:

- **24 fps**, es el estándar para películas y programas de televisión. Se determinó que era la velocidad mínima necesaria para capturar vídeo sin dejar de mantener un un movimiento realista. Incluso si el vídeo se graba a una velocidad superior de fotogramas, a menudo se produce y se muestra a 24 fps.
- **30 fps**, este ha sido el estándar utilizado para televisión desde sus primeros días y hoy en día se mantiene en uso, aunque algunas producciones utilizan 24 fps más cinematográfico. Los vídeos con demasiados movimientos como los deportes, suelen beneficiarse al utilizar fotogramas adicionales.
- **60 fps**, esta velocidad se reserva para grabar escenas de movimiento superiores a 30 fps, como por ejemplo como los videojuegos. Esto es debido a que ocurren muchos sucesos en pantalla en un mismo momento y el uso de más fotogramas ayuda a dar más detalles; para los contenidos que requieran mostrar movimientos en cámara lenta como es el caso de los eventos deportivos como atletismo, baloncesto, fútbol, entre otros. Estos al utilizar repeticiones tengan la posibilidad de ralentizar el momento sin dejar de mantener un vídeo nítido.

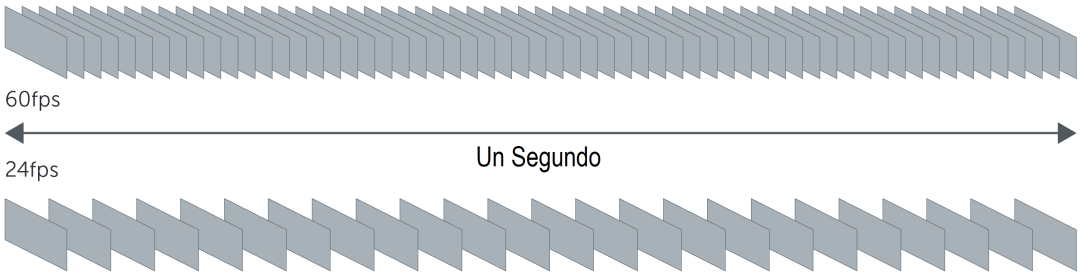

Figura 2.1: Comparación entre 24 fps y 60 fps.

2.1.4 Tasa de Bits (*Bitrate*)

Es la cantidad de bits transmitidos en un intervalo de tiempo determinado. Se considera en promedio una combinación de los flujos de tiempo y audio, y que está directamente relacionada con la calidad del vídeo (a mayor tasa de bits, mayor calidad). La tasa de datos de vídeo se da en bits por segundo (b/s), por lo que la especificación de la tasa de datos para un archivo de vídeo en reproducción se da en megabits por segundo (Mb/s).

La tasa de bits para un vídeo con calidad Blu-ray en el rango de 20 Mb/s, el DVD con calidad estándar en 6 Mb/s, el vídeo web HD a 2 Mb/s, y el vídeo para teléfonos se da en kilobits (kb/s). Por ejemplo, en un *streaming* usando el codec H.264, estos serían los datos obtenidos:

LD 240p 3G Mobile @H.264 baseline profile 350 kbps (3 MB/min)
LD 360p 4G Mobile H.264 main profile 700 kbps (6 MB/min)
SD 480p WiFi @H.264 main profile 1200 kbps (10 MB/min)
HD 720p @H.264 high profile 2500 kbps (20 MB/min)
HD 1080p @H.264 high profile 5000 kbps (35 MB/min)

en donde "LD" significa *Low Definition*, "'SD' *standard definition* y "'HD' *high definition*.

2.1.5 Codecs de Vídeo

Codec es la abreviatura de codificador-decodificador y describe el método mediante el cual los datos de vídeo se codifican en un archivo y se decodifican cuando se reproduce el archivo. La mayor parte del vídeo se comprime durante la codificación, por lo que los términos codec y compresor se utilizan a menudo de forma intercambiable. Las compresiones pueden ser sin pérdidas, lo que significa que no suprimen ningún dato, o con pérdidas, lo que significa que hay información perdida durante la codificación. Los codecs sin pérdida son de mayor calidad que los codecs con pérdida, pero producen tamaños de archivo más grandes.

En una sesión de *streaming* de vídeo se debe de evitar la conversión de un codec a otro (transcodificación) con pérdidas hasta que el proceso finaliza. Esto significa que su edición final puede exportarse a H.264 para la web, pero no debe utilizar el mismo archivo para edición adicional, como la creación de DVD's, entre otros. Los siguientes codecs de vídeo existen para satisfacer diferentes necesidades:

- H.264/AVC. Probablemente el más utilizado. Se caracteriza por ser uno de los codecs más eficientes, ya que permite tamaños de archivo más pequeños a la vez manteniendo una alta calidad; ofrece también opciones para la compresión sin pérdidas o con pérdidas. Es compatible con MP4 y se puede reproducir en la mayoría de los reproductores y diferentes servicios de *streaming*.
- Dentro del estándar de MPEG-4, la parte 10 describe un formato avanzado de codificación que se utiliza en los discos Blu-ray. Erróneamente se dice que H.264

y MPEG-4 se trata del mismo códec, aunque en MPEG-4 las partes 2, 12 o 14 también describen otros formatos de compresión que son MPEG-4, y el formato contenedor puede contener también MPEG en versiones anteriores de menor calidad como MPEG-1 o MPEG-2.

- DivX. Al igual que la versión de código abierto XviD, es uno de los codecs más antiguos que está diseñado para maximizar la calidad de vídeo a cambio de tener tamaños de archivo significativamente mayores. Se utiliza comúnmente en una variedad de entornos comerciales donde hay menos preocupación por el tamaño del archivo.

- MPEG-2. Como predecesor de MPEG-4, MPEG-2 es el codec estándar para su uso en DVD's y en los primeros discos Blu-ray. No se recomienda para el *streaming* de contenidos multimedia.

- HEVC (H.265). Es la codificación de vídeo de alta eficiencia o HEVC (High Efficiency Video Coding), mejor conocido como H.265. El objetivo de H.265 es duplicar la compresión manteniendo una calidad de vídeo similar o igual a la de su predecesor H.264. Está diseñado para aprovechar resoluciones muy altas como 4K y 8K

2.1.6 Contenedores de Vídeo

Los contenedores de vídeo agrupan y almacenan cada uno de los elementos de un vídeo en un solo paquete. Los elementos incluyen las secuencias de audio y vídeo, subtítulos, metadatos de vídeo, codecs, entre otros. Cada tipo de contenedor es compatible con determinados codecs de vídeo. Entre los contenedores más comunes se encuentran:

- MP4. Es el contenedor que probablemente es lo más cercano a un estándar. MP4 puede utilizar todas las versiones de MPEG-4 y H.264 y es compatible con una amplia variedad de reproductores. Los vídeos que utilizan MP4 pueden tener tamaños de archivo relativamente pequeños y conservar una alta calidad. Es utilizado por la mayoría de los servicios de *streaming*.

- AVI. Uno de los formatos de archivo de vídeo más antiguos y universalmente aceptados. Puede utilizar una enorme variedad de codecs, lo que permite una gran variedad de configuraciones. Mientras que los vídeos AVI pueden reproducirse en una amplia serie de reproductores, los tamaños de los archivos tienden a ser grandes, lo que los hace menos adecuados para el *streaming* o la descarga ante conexiones lentas. Sin embargo, es una excelente opción para los vídeos que se almacenan en un equipo.

- MOV. Apple desarrolló el contenedor MOV para utilizarlo con su reproductor Quicktime. Los vídeos que utilizan MOV generalmente tienen una calidad muy alta pero también tamaños de archivo bastante grandes. Los vídeos MOV no tienen mucha compatibilidad con los reproductores de terceros.

- FLV. Diseñado para el reproductor Flash de Adobe, los vídeos FLV fueron muy comunes durante varios años gracias a su tamaño de archivo muy pequeño y a una amplia gama de plugins de navegador y reproductores de vídeo Flash de terceros.

- WMV. Los vídeos de Windows Media suelen tener el tamaño de archivo más pequeño, lo que los convierte en una buena opción si se necesita enviar por medios como el correo electrónico u otros métodos con limitantes de tamaño de archivo. Sin embargo, esto viene con la desventaja de tener una caída significativa en la calidad.

2.2 Protocolos de *streaming*

Durante un live *streaming* o bajo demanda, se requiere de protocolos para la entrega de datos a través de Internet. La entrega de contenido multimedia utiliza tanto protocolos de transmisión como protocolos basados en HTTP. Los protocolos de *streaming* como RTMP, permiten una entrega rápida de vídeo utilizando servidores dedicados, mientras que los basados en HTTP se basan en el uso de servidores web tradicionales que optimizan la calidad de la experiencia o QoE.

2.2.1 Real-Time Transport Protocol (RTP)

El Protocolo de Transporte de Tiempo Real, es un protocolo que especifica la forma en que gestiona la entrega de datos multimedia en tiempo real en redes *unicast* o *multicast*. En comparación con TCP, favorece la integridad de los datos. RTP en conjunto con el protocolo RTCP favorecen una entrega rápida, utilizando mecanismos para compensar cualquier pérdida menor en la integridad de la información. Es capaz de codificar los flujos de datos multimedia como audio y vídeo, dividirlos en paquetes y transmitirlos a través de una red IP para posteriormente sincronizarlos en su destino. RTP, suele ejecutarse sobre UDP [5].

2.2.2 Real-Time Transport Control Protocol (RTCP)

El Protocolo de Control de RTP, se encarga de las comunicaciones y tareas de información para un correcto control de flujo de datos de RTP. Los paquetes de este protocolo no transportan contenido multimedia, sino que en conjunto con RTP se encargan del transporte

y empaquetado de los datos. Se usa para transmitir los parámetros de una sesión multimedia, teniendo como función principal la de informar sobre la calidad del servicio o QoS. Durante una sesión de *streaming*, RTCP transmite de forma periódica paquetes de control a los involucrados para controlar el estado de la conexión en todo momento [5].

2.2.3 Real Time Streaming Protocol (RTSP)

El Protocolo de Transmisión en Tiempo Real, proporciona la posibilidad de transmitir contenido multimedia, permitiendo el control de canales y mecanismos a través de múltiples sesiones para la entrega de datos sincronizados, ya sea contenido transmitido por live *streaming* o bajo demanda. Desde un punto de vista funcional, el protocolo RTSP cumple la función de control remoto de los sistemas de comunicación multimedia (redes y servidores). Las solicitudes de control mediante RTSP pueden enviarse a través de TCP o UDP. Sin embargo, la desventaja de RTSP es la ausencia de mecanismos de recuperación del sistema, por lo que una vez que el cliente pierde la información acerca del estado de una sesión no hay ningún método para enviar peticiones de control al servidor, en consecuencia, la implementación de RTSP requiere de métodos a prueba de fallos u opción de control de sesión [6].

2.2.4 Real Time Messaging Protocol (RTMP)

Es un protocolo desarrollado por Macromedia, está basado en TCP y permite mantener conexiones persistentes. Para que un stream se mantenga de forma fluida, permitiendo una transmisión con la mayor información posible, este se divide en segmentos y su tamaño se negocia de forma dinámica entre el cliente y el servidor, aunque en ocasiones se mantiene sin cambios. Por defecto, el tamaño determinado para los segmentos de audio es de 64 bytes y el de vídeo u otro tipo de datos, es de 128 bytes. Los fragmentos de diferentes streams se pueden intercalar y multiplexar en una sola conexión. Después de establecer una conexión TCP, se establece una conexión RTPM realizando inicialmente un proceso de acuerdo entre cliente y servidor. Posteriormente, se puede negociar una conexión intercambiando mensajes AMF (Action Message Format) codificados [7].

2.3 HTTP *Streaming*

Después de conocer algunos de los protocolos de *streaming* más utilizados se debe de tener en cuenta que presentan algunos inconvenientes. En Internet, hoy en día en existen las Redes de Distribución de Contenidos o CDN (Content Delivery Network), las cuales en su mayoría no soportan el protocolo RTP. Por otro lado, algunos firewalls no permiten los

paquetes que generan. Otro inconveniente es que con RTP o RTPM el servidor tiene que gestionar una sesión diferente por cada cliente. Por otro lado, algunos de estos protocolos como RTP requieren la asistencia de otro protocolo como RTCP y RTSP, y por lo tanto se incrementa la información que se necesita transmitir por la red.

Debido a los motivos antes mencionados surge HTTP *streaming*. Es una técnica de transferencia de datos de estilo push que permite a un servidor web enviar datos continuamente a un cliente a través de una única conexión HTTP que permanece abierta indefinidamente. Cabe aclarar que técnicamente, esto va en contra de los principios de HTTP, sin embargo HTTP Streaming es considerado un método eficiente para transferir todo tipo de datos dinámicos o de otro tipo entre un servidor y un cliente.

Con HTTP Streaming, el servidor está configurado para mantener una solicitud de un cliente y mantener abierta la respuesta para que pueda transferir los datos a través de ella. Cuando las actualizaciones correspondientes a la solicitud están disponibles en el lado del servidor, el servidor envía una respuesta a través del canal $request - response$, y sólo cierra la conexión cuando se le indica explícitamente que lo haga. De esta manera, un cliente puede conocer las actualizaciones del servidor y recibirlas instantáneamente sin la sobrecarga asociada a HTTP, eliminando también la necesidad de realizar sondeos. Para lograr una respuesta indefinida, el servidor debe responder a las peticiones de los clientes especificando mecanismos como *Chunked Transfer Encoding*, donde el flujo de datos se divide en una serie de segmentos que no se sobreponen. Estos segmentos se envían y reciben independientemente unos de otros [8, 9].

2.4 Streaming Adaptativo sobre HTTP

Como se puede ver el HTTP Streaming parece ser la solución idónea, pero se puede mejorar utilizando la técnica *Streaming* Adaptativo sobre HTTP o HAS (HTTP Adaptive Streaming). A través de esta técnica, se escoge la mejor calidad del contenido a reproducir en función de las condiciones de la red y de la capacidad de cómputo del cliente. Para ello, se deberá codificar el archivo en diferentes tasas de bits y posteriormente se deberán segmentar en fragmentos fijos de entre dos a quince segundos de duración. Al iniciar la reproducción, se estiman las condiciones de la red en el lado del cliente y se escoge la tasa de bits. A lo largo de la reproducción del contenido, el cliente enviará información al servidor de forma que pueda mejorar o empeorar la calidad de la reproducción.

3 *Streaming* Dinámico Adaptativo sobre HTTP

El *Streaming* Dinámico Adaptativo sobre HTTP o MPEG-DASH (Dynamic Adaptive *Streaming* over HTTP, ISO/IEC 23009-1 [10]), es un mecanismo de *streaming* desarrollado por Moving Picture Expert Group o MPEG, que se convirtió en estándar internacional en noviembre de 2011 y fue publicado en abril de 2012. En este mecanismo de streaming, el control recae en el lado del cliente, el cual solicita la información mediante HTTP a uno o varios servidores web estándar. En la Figura 3.1 se puede observar un escenario base de un *streaming* entre un servidor HTTP y un cliente DASH, donde el formato y las funcionalidades de las cajas rojas se definen más adelante. El resto se puede omitir. Asimismo, se puede observar que el contenido multimedia se encuentra almacenado en un servidor web y se entrega al cliente mediante este protocolo. La Figura 3.2 muestra el proceso que realiza el cliente para reproducir el contenido que se describe a continuación:

a) Se obtiene el archivo manifiesto o MPD mediante HTTP, correo electrónico u otro medio de comunicación.

b) Posteriormente, el manifiesto es analizado o parseado para obtener la información de cómo se encuentra organizada la información necesaria para la reproducción.

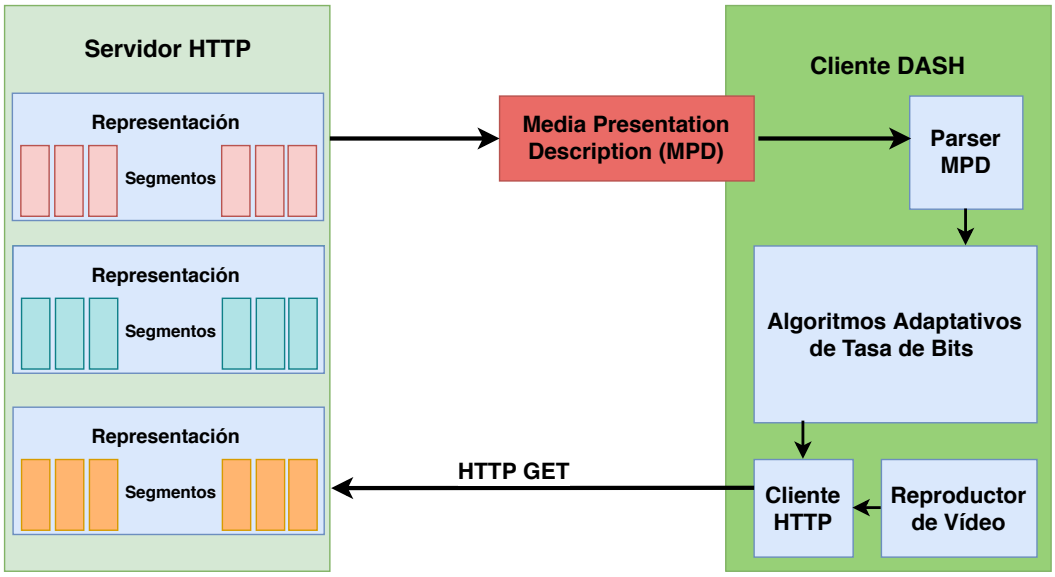

Figura 3.1: Escenario de un cliente DASH y un servidor HTTP.

c) Contando con la información requerida, son seleccionadas las características adecuadas y el cliente comienza a realizar las peticiones de los segmentos por medio de HTTP.

d) Durante la petición de segmentos, un estimador monitorea los cambios en el ancho de banda de la red local y decide si es necesario que el siguiente segmento sea de distinta tasa de bits.

Una de las principales características de DASH es que puede soportar *streaming* bajo demanda o en vivo. En este último caso, el archivo MPD se irá actualizando con el paso del tiempo, por lo que el cliente deberá solicitar las actualizaciones.

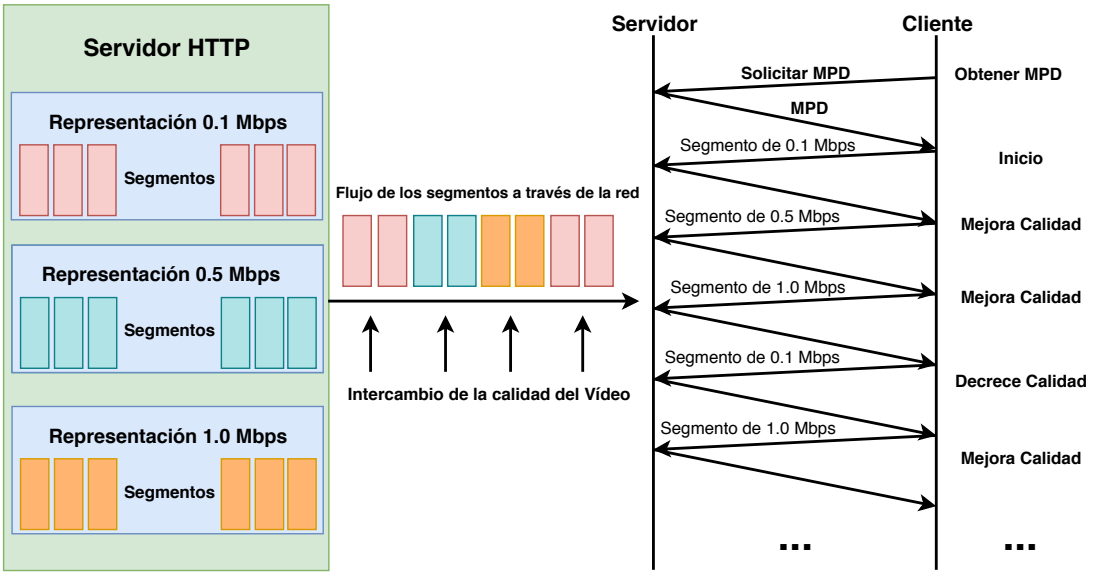

Figura 3.2: Escenario de un cliente DASH y un servidor HTTP.

3.1 Media Presentation Description (MPD)

El archivo Media Presentation Description o MPD es un documento XML, que se basa en un modelo de datos jerárquicos, ilustrado en la Figura 3.3. Cada MPD puede contener uno o más Períodos. Cada uno de esos Períodos contiene componentes de audio y vídeo, con diferentes códecs, o con diferentes tipos de información como subtítulos, entre otros. Estos componentes tienen ciertas características como tasa de bits, la frecuencia de imagen, los canales de audio, entre otros, que no cambian durante un Período. El cliente tiene la capacidad de poder adaptarse durante un Período en función de la tasa de bits, resoluciones, codecs, que están disponibles en un Período determinado. En cada

período se separa el contenido por ejemplo, para insertar anuncios publicitarios, o cambiar el ángulo de cámara en un evento deportivo, transmitido en tiempo real. En otro caso, si un anuncio sólo debe estar disponible en cierta resolución, mientras que el contenido principal está disponible desde la definición estándar hasta la más alta, el generador de contenido solamente introduciría un Período propio para el anuncio.

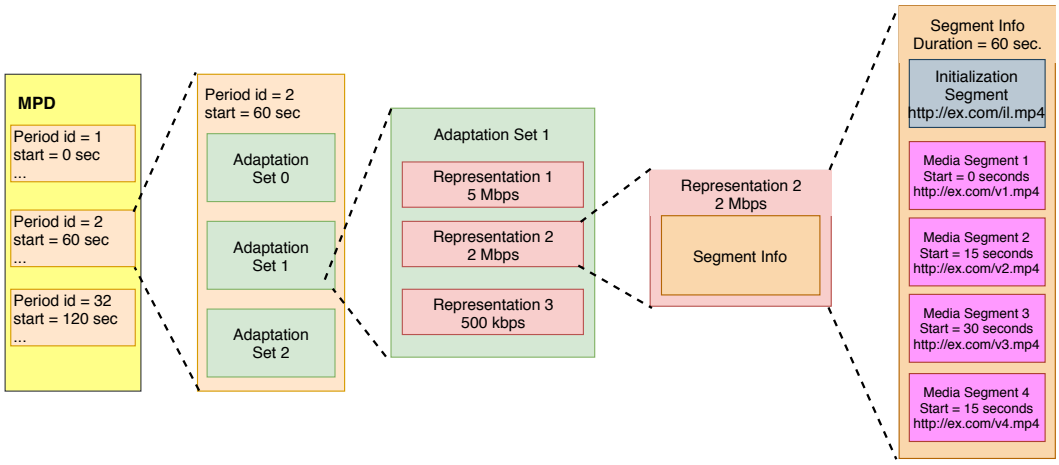

Figura 3.3: Modelo de datos de la Presentación de la Descripción Multimedia.

El contenido multimedia como el vídeo, el audio o los archivos de subtítulos, está organizado dentro de los conjuntos de adaptación o *AdaptationSets*. Por ejemplo, los componentes con el mismo códec, idioma, resolución, formato de canal de audio (5.1 o estéreo), entre otros, pueden estar dentro del mismo conjunto de adaptación. Este mecanismo permite al cliente evitar elegir una serie de contenidos que no cumplen las condiciones de la red en la que se aloja. Un período también podría contener un subconjunto que permita la restricción de combinaciones de conjuntos de adaptación y permitir sólo el contenido de alta definición con formato de canal de audio 5.1. En resumen, la composición jerárquica de un archivo MPD consiste en:

- Una secuencia de uno o más períodos.
- Cada período contiene uno o más conjuntos de adaptación. En los casos que un conjunto de adaptación contenga múltiples componentes multimedia, entonces cada componente se describe individualmente.
- Cada conjunto de adaptación contiene una o más representaciones.

- Los conjuntos de adaptación, representaciones y subrepresentaciones comparten atributos y elementos comunes.
- Cada Período puede contener uno o más Subconjuntos que restrinjan la combinación de conjuntos de adaptación en la reproducción del contenido multimedia.
- Cada representación consiste en uno o más Segmentos. Los Segmentos contienen metadatos para acceder, decodificar y presentar el contenido multimedia incluido. Las representaciones también pueden incluir Sub-representaciones para describir y extraer información parcial de una representación.
- Cada segmento consiste de uno o más subsegmentos.

Es importante señalar que existen restricciones en las dimensiones de algunos elementos de un archivo MPD, las cuales se señalan a continuación:

- El tamaño del MPD antes y después de la resolución xlink no debe exceder los 256 kb.
- El MPD tiene un máximo de 64 períodos antes y después de la resolución xlink.
- El MPD tiene un máximo de 16 conjuntos de adaptación por período.
- El MPD tiene un máximo de 16 representaciones por conjunto de adaptación.
- La duración del segmento será de al menos 1 segundo, excepto para el último segmento de un período que puede ser más corto.
- Los segmentos de subtítulos tendrán un tamaño máximo de 512 kb .
- Cuando se señalicen o no los subsegmentos, cada segmento de audio o vídeo tendrá una duración menor que 15 segundos.

3.1.1 Período

Un archivo MPD, puede contener uno o varios períodos. Se define un período como parte del contenido multimedia reproducido en el tiempo, el cual consta de un inicio y una duración. Además, un período puede contener varios conjuntos de adaptación. Su tiempo de inicio se describe a continuación:

- A través de su atributo @*start*.

- Si @*start* no existe, pero si el período anterior contiene el atributo @*duration*, entonces el tiempo de inicio del período actual, es la suma del tiempo de inicio del anterior más su duración.
- Si @*start* no existe y el período es el primer elemento del MPD, además el atributo @*type* tiene valor *'static'*, entonces el valor inicial del período es cero.
- Si @*start* no existe, y el período anterior no contiene el atributo @*duration* o es el primero en el MPD, y el atributo @*type* tiene valor *'dynamic'*, entonces, es un *Período de Disponibilidad Anticipada*, el cual es utilizado para anunciar contenido diferente al multimedia. Una vez que el contenido multimedia se encuentra disponible, se conoce el valor del atributo @*start* y se utiliza un *período Regular*.

```
<!-- Periodo -->
<Period duration="PT0H9M56.46S">
    <Representation>
      <BaseURL>bunny_3936261bps/BigBuckBunny_4snonSeg.mp4</BaseURL>
      <SegmentBase indexRangeExact="true" indexRange="891-2722">
        <Initialization range="0-890" />
      </SegmentBase>
    </Representation>
  </AdaptationSet>
</Period>
```

Bloque 3.1: Sintaxis XML de un período.

3.1.2 Conjuntos de Adaptación

Un conjunto de adaptación, identificado como un elemento *Adaptation-Set*, consiste en un conjunto de una o varias representaciones que contienen versiones intercambiables de sus respectivos contenidos, diferenciados por su tasa de bits. Una sola representación, únicamente puede ser reproducida en un momento, por lo que una podría ser suficiente para proporcionar contenido multimedia que puede ser reproducido. Sin embargo, múltiples representaciones proporcionan al cliente la posibilidad de adaptarse según las condiciones actuales de la red, garantizando una reproducción fluida.

3.1.3 Representaciones

Las representaciones se dividen en Segmentos para permitir el cambio entre representaciones individuales durante la reproducción. Los segmentos se describen por una URL y, en algunos casos, por un rango adicional de bytes si se almacenan en solo un archivo. Los segmentos de una representación suelen tener el mismo tamaño en términos de tiempo y están dispuestos de acuerdo con la línea de tiempo de la presentación de medios, que representa la línea de tiempo para la sincronización, lo que permite un cambio suave de las representaciones durante la reproducción. Los segmentos también podrían tener un tiempo de disponibilidad señalada, a partir de la cual son accesibles para escenarios de *streaming* en tiempo real. MPEG-DASH no restringe la longitud del segmento ni indica cuál es la longitud óptima. Esto puede ser elegido dependiendo del escenario, por ejemplo, los Segmentos más largos permiten una compresión más eficiente ya que el Grupo de Imágenes o GOP (Group of Pictures) podría tener una sobrecarga menor o mayor en la red, debido a que cada Segmento se solicitará a través de HTTP y con cada solicitud se introducirá una cierta cantidad de sobrecarga HTTP. Por el contrario, los Segmentos más cortos se utilizan tanto para escenarios en directo como para condiciones de ancho de banda muy variables, como redes móviles, ya que permiten una descarga más rápida y flexible entre tasas de bits individuales.

3.1.4 Segmentos

Los segmentos pueden subdividirse en subsegmentos más pequeños que representan un conjunto de unidades de menor tamaño en el segmento en cuestión. En este caso, se establece un índice del segmento disponible en el segmento que el cliente puede descargar con anterioridad. Durante la reproducción del contenido multimedia, el cambio entre las representaciones no es posible en ningún momento y se deben tener en cuenta ciertas restricciones. Por lo tanto, no se permite que los segmentos se superpongan, por ejemplo, tampoco se permiten las dependencias entre segmentos. Para permitir el cambio entre representaciones, DASH introduce Puntos de Acceso al Stream o SAP (Stream Access Points) en los que esto es posible. Por ejemplo, cada segmento comienza normalmente con una trama de Actualización Instantánea Decodificada o IDR (Instantaneous Decoder Refresh) en H.264/AVC para poder cambiar la representación después de transmitir un segmento previo.

Los segmentos son referenciados a través de URL's como son definidos en el RFC 3986 [11], usando HTTP o HTTPs restringidos por un rango de bytes. El rango de bytes puede ser

señalado a través de un rango de atributos y debe ser compatible con el RFC 2616 [12]. Los segmentos forman parte de una representación, mientras que elementos como *BaseURL*, *SegmentList*, *SegmentTemplate* y *SegmentList* pueden añadir información adicional, como ubicación, disponibilidad y otras propiedades. En específico, una representación puede contener sólo una de las siguientes opciones:

- Uno o más elementos *SegmentList*.
- Un elemento *SegmentTemplate*.
- Uno o varios elementos *BaseURL*, un solo elemento *SegmentBase* y no habrá elementos *SegmentTemplate* o *SegmentList*.

3.1.4.1 SegmentBase

A través de *SegmentBase* se pueden referenciar segmentos, ya que se utilizará cuando sólo exista un segmento por representación, que luego es referenciado por una URL en el elemento *BaseURL*. Si una representación contiene más segmentos, se debe utilizar *SegmentList* o *SegmentTemplate*. El Bloque 3.2 muestra una representación que utiliza *SegmentBase*.

```
  <!-- Segment Base -->
<Representation mimeType="video/mp4"
                    frameRate="24"
                    bandwidth="1558322"
                    codecs="avc1.4d401f" width="1277" height="544">
  <BaseURL>http://cdn.bitmovin.net/bbb/video-1500k.mp4</BaseURL>
  <SegmentBase indexRange="0-834"/>
</Representation>
```

Bloque 3.2: Sintaxis XML para SegmentBase.

Este ejemplo hace referencia a un solo segmento a través de la *BaseURL*, que tiene una calidad de vídeo de 1500 kbps del contenido correspondiente. El índice de la calidad se describe mediante el atributo *SegmentBase indexRange*. Esto quiere decir que la información sobre los Puntos de Acceso Aleatorio o RAP (Random Access Points) y otra información de inicialización se encuentran en los primeros 834 bytes.

3.1.4.2 SegmentList

SegmentList contiene una lista de elementos *SegmentURL* que el cliente debe reproducir en el orden en que aparecen en el MPD. Un elemento *SegmentURL* contiene una URL de un segmento y posiblemente un rango de bytes. Además, podría producirse un índice al principio de la lista de segmentos. El Bloque 3 muestra la sintaxis de la representación utilizando *SegmentList*.

```
  <!-- Segment List -->
<Representation mimeType="video/mp4"
                   frameRate="24"
                   bandwidth="1558322"
                   codecs="avc1.4d401f" width="1277" height="544">
                   <SegmentList duration="10">
<Initialization sourceURL="http://cdn.bitmovin.net/bbb/video-1500/init.mp4"
    />
  <SegmentURL media="http://cdn.bitmovin.net/bbb/video-1500/segment0.m4s"/>
  <SegmentURL media="http://cdn.bitmovin.net/bbb/video-1500/segment1.m4s"/>
  <SegmentURL media="http://cdn.bitmovin.net/bbb/video-1500/segment2.m4s"/>
  <SegmentURL media="http://cdn.bitmovin.net/bbb/video-1500/segment3.m4s"/>
  <SegmentURL media="http://cdn.bitmovin.net/bbb/video-1500/segment4.m4s"/>
  </SegmentList>
</Representation>
```

Bloque 3.3: Sintaxis XML de SegmentList.

3.1.4.3 SegmentTemplate

Este elemento proporciona un mecanismo para construir una lista de segmentos dada una plantilla, en donde los identificadores específicos serán sustituidos por valores dinámicos para crear dicha lista. Lo anterior implica algunas ventajas, por ejemplo un archivo MPD basado en *SegmentList* puede llegar a ser muy grande porque cada segmento necesita ser referenciado de forma individual. Comparado con *SegmentTemplate*, esta lista puede ser descrita por una cantidad pequeña de líneas que indican cómo construir una lista grande de segmentos.

En el Bloque 3.4 se muestra a *SegmentTemplate* basado en números, en lugar de tener múltiples referencias de segmentos individuales a través de *SegmentURL*. Como se muestra en el ejemplo de *SegmentList*, *SegmentTemplate* puede describir este caso en unas pocas líneas, lo que hace que el MPD sea más compacto. Esto es útil para contenido multimedia

como películas de larga duración con múltiples representaciones donde un MPD con *SegmentList* podría medir varios megabytes, lo que aumentaría considerablemente la latencia de inicio de un *streaming*, ya que el cliente tiene que recuperar el MPD antes de que pueda comenzar con el proceso de reproducción.

```
  <!-- Segment Template -->
 <Representation mimeType="video/mp4"
                   frameRate="24" bandwidth="1558322"
                   codecs="avc1.4d401f" width="1277" height="544">
  <SegmentTemplate media="http://cdn.bitmovin.net/bbb/video-1500/segment-$
      Number$.m4s"
                 initialization="http://cdn.bitmovin.net/bbb/video-1500/init
                     .mp4"
                 startNumber="0"
                 timescale="24"
                 duration="48"/>
</Representation>
```

Bloque 3.4: Sintaxis XML de SegmentTemplate.

SegmentTemplate también puede contener un identificador *$Time$*, que es sustituido por el valor del atributo "t" de *SegmentTimeline*. Este elemento proporciona una alternativa al atributo *duration* con las siguientes características adicionales:

- Especificar duraciones arbitrarias en el segmento.
- Especificar la duración precisa en los segmentos.
- Especificar interrupciones en la línea de tiempo del contenido.

SegmentTimeline también utiliza un proceso de compresión de longitud, que es altamente eficiente cuando hay una secuencia de segmentos con la misma duración. Al utilizarse *SegmentTimline* con *SegmentTemplate*, se deben aplicar las siguientes condiciones:

- Deberá de existir al menos un contenedor *SegmentIndexBox*.
- Todos los valores de *SegmentTimeline* describirán el tiempo exacto, igual a la información en *SegmentIndexBox* [13].

El Bloque 3.5 muestra un extracto de un archivo MPD con un *SegmentTemplate* que se basa en una *SegmentTimeline*:

```
<Representation mimeType="video/mp4"
                    frameRate="24"
                    bandwidth="1558322"
                    codecs="avc1.4d401f" width="1277" height="544">
  <SegmentTemplate media="http://cdn.bitmovin.net/bbb/video-1500/segment-$
      Time$.m4s"
                       initialization="http://cdn.bitmovin.net/bbb/video
                           -1500/init.mp4"
                       timescale="24">
    <SegmentTimeline>
      <S t="0" d="48" r="5"/>
    </SegmentTimeline>
  </SegmentTemplate>
</Representation>
```

Bloque 3.5: Sintaxis XML de SegmentTimeline.

3.2 Formato de los segmentos

El ISO Media Base File Format o MBFF [10], contiene la estructura de los segmentos y la información de los datos multimedia principalmente relacionada con el tiempo de presentación del audio, vídeo, entre otros. Los formatos de los segmentos especifican la sintaxis y la semántica de los recursos que están asociados con los HTTP-URL's identificados por el archivo MPD. Por ejemplo, una petición HTTP-GET hacia un recurso identificado en el archivo MPD es respondida con HTTP-Response que incluye el cuerpo de una entidad que se ajusta al formato de un segmento. La Figura 3.4, muestra el formato de un segmento en DASH que se centra en contenedores MPEG. A su vez, los segmentos pueden dividirse como se describe enseguida.

3.2.1 Tipos de Segmentos

- **Initialization Segment**, contiene la información de inicialización para acceder a la *representación* correspondiente. Este segmento es procesado por un gestor de medios para permitir la reproducción de los Segmentos de Medios.

- **Segment Index**, si existe este tipo de segmento, se utilizará un *sidxbox* definido en ISO/IEC 14496-12.

- **Media Segment**, este tipo de segmentos pueden subdividirse en: simples (*Media Segment*), indexados (*Indexed Media Segment*) y sub-indexados (*Sub-Indexed Media Segment*).

  - *Simple Media Segment*
    - Puede tener un *styp box*
    - Contiene uno o más conjuntos de fragmentos multimedia autocontenidos, donde cada uno de estos conjuntos consta de un *moof box* y un *mdat box*, los cuales contienen muestras multimedia y que no utilizan referencias de datos externos.
    - Contiene un *traf box*, que a su vez contiene un *tfdt box*.
    - Puede contener uno o más *sidx boxes*, el primero de los cuales aparece antes que algún *box moof*.
  - *Indexed Media Segment*
    - Utiliza el formato de *Simple Media Segment* y además en cada fragmento multimedia autocontenido *box moof* es inmediatamente seguido por *box mdat*.
    - *box sidx* es obligatorio.
  - *Sub-Indexed Media Segment*
    - Utiliza el formato de los segmentos *Indexed Media Segment*.
    - Debe aparecer *box ssix*, que continúa inmediatamente después de *box sidx*

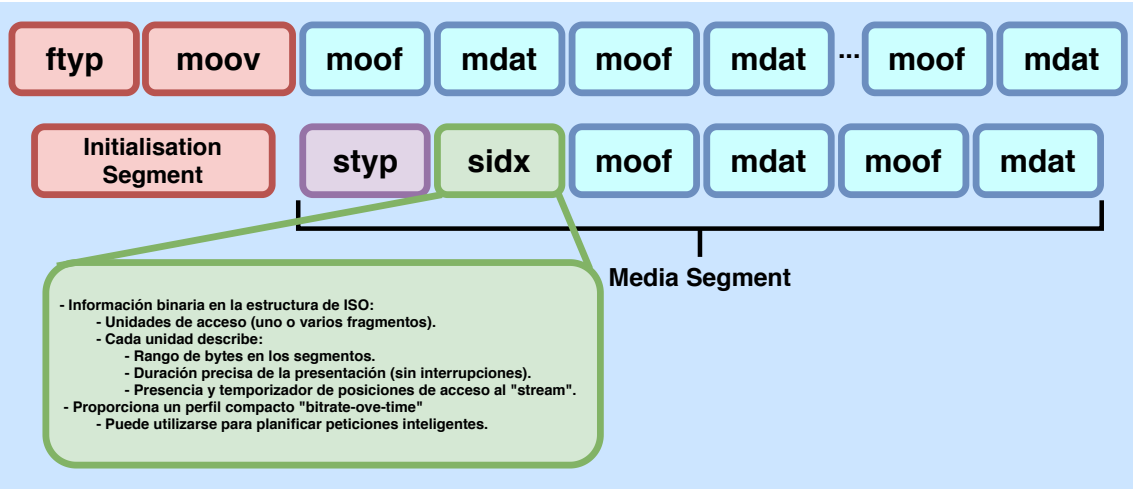

Figura 3.4: Formatos de los segmentos ISO Base Media File Format en DASH.

3.2.2 MPEG-2 Transport Stream

Formato del contenedor que encapsula Packetized Elementary Streams o PES, que define la manera de trasmitir los *streams* elementales con corrección de errores y utilidades de sincronización la integridad de la transmisión cuando la señal se degrada.

3.3 Perfiles de DASH

Un perfil (*profile*) impone una serie de restricciones específicas. Estas restricciones afectan a las características del archivo MPD y a los formatos de los segmentos, así como a su contenido como los tipos de contenido, los códecs y formatos de protección, o a las medidas cuantitativas, entre otros. Un perfil también puede definirse como un permiso para clientes DASH que sólo implementan las características requeridas por el perfil para procesar el archivo MPD. Sin embargo, como la operación de un cliente DASH no está especificada bajo alguna norma, tampoco se especifica cómo un cliente DASH se ajusta a un perfil particular. Por lo tanto, los perfiles sólo especifican restricciones en el MPD y segmentos en lugar del comportamiento del cliente DASH. En la Figura 3.5 se muestra una forma general de cómo se agrupan los perfiles.

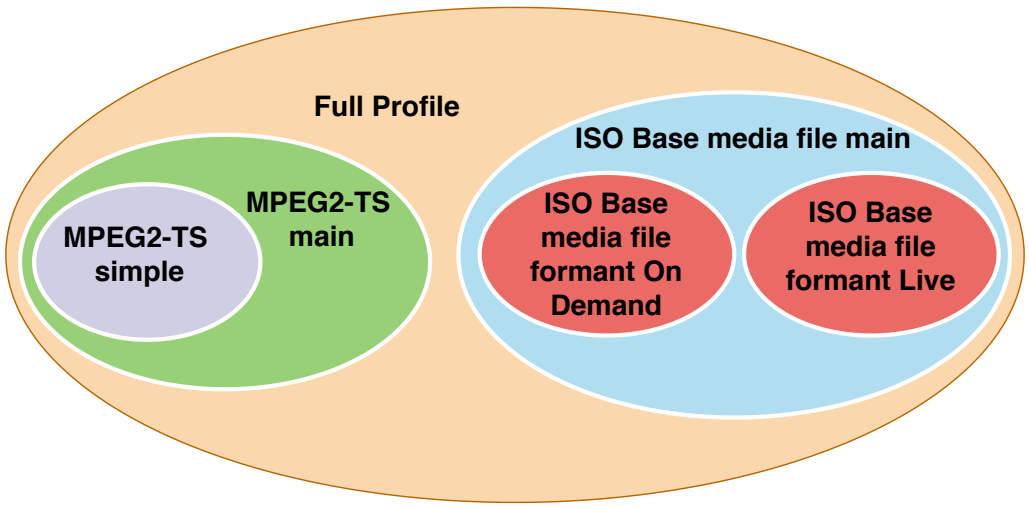

Figura 3.5: Descripción de los perfiles MPEG-DASH.

4 Algoritmos de Adaptación de Tasa de Bits (ABR)

En DASH, los algoritmos de adaptación de tasa de bits o ABR (Adaptive BitRate, por sus siglas en inglés) se ejecutan principalmente en cada cliente de forma distribuida. El objetivo de estos algoritmos es garantizar la calidad de la experiencia (QoE) para los usuarios bajo la presencia de fluctuaciones del ancho de banda debido a factores como la intensidad de la señal (en conexiones inalámbricas), congestión de la red, eventos de convergencia en la red, entre otros. Aunque estas fluctuaciones suelen ser comunes en Internet, también pueden ocurrir en redes domésticas o incluso en redes gestionadas donde a menudo se implementan controles de acceso y herramientas de calidad de servicio (QoS).

Los algoritmos ABR pueden tener en cuenta factores como las estimaciones del ancho de banda, la ocupación del buffer de reproducción, características del dispositivo, preferencias del espectador o características del contenido, aunque con diferentes pesos o calificaciones de calidad. Dado que la QoE del espectador debe determinarse en tiempo real durante la reproducción, generalmente se utilizan métricas objetivas que incluyen el número de paradas de buffer, la duración del retardo de inicio, la frecuencia y la cantidad de oscilaciones, y la inestabilidad del contenido multimedia. Debido a su diseño, los estándares como DASH no exigen ningún algoritmo ABR en particular, dejando a los desarrolladores la tarea de innovar e implementar su propio método. La Figura 4.1, muestra una clasificación de los algoritmos ABR, según la entidad del sistema donde es implementada la lógica, que a continuación se describen [14].

4.1 Adaptación basada en el servidor

Este tipo de esquemas utilizan un método de modelado de tasa de bits en el lado del servidor y no requieren ninguna cooperación por parte del cliente. Por lo tanto, el cambio de las tasas de bits es controlado de forma implícita por el modelador. El cliente aún toma sus propias decisiones, pero las decisiones son determinadas en el servidor. Sin embargo, requieren de una gran complejidad, especialmente cuando el número de clientes aumenta. Estos esquemas también necesitan modificaciones en el archivo MPD o en un software personalizado para implementar la lógica de adaptación de la tasa de bits. Esto puede ser percibido como una violación de los principios de diseño estándar de DASH, debido a que el servidor debe ser un servidor HTTP estándar y los algoritmos de adaptación deben ejecutarse en el lado del cliente.

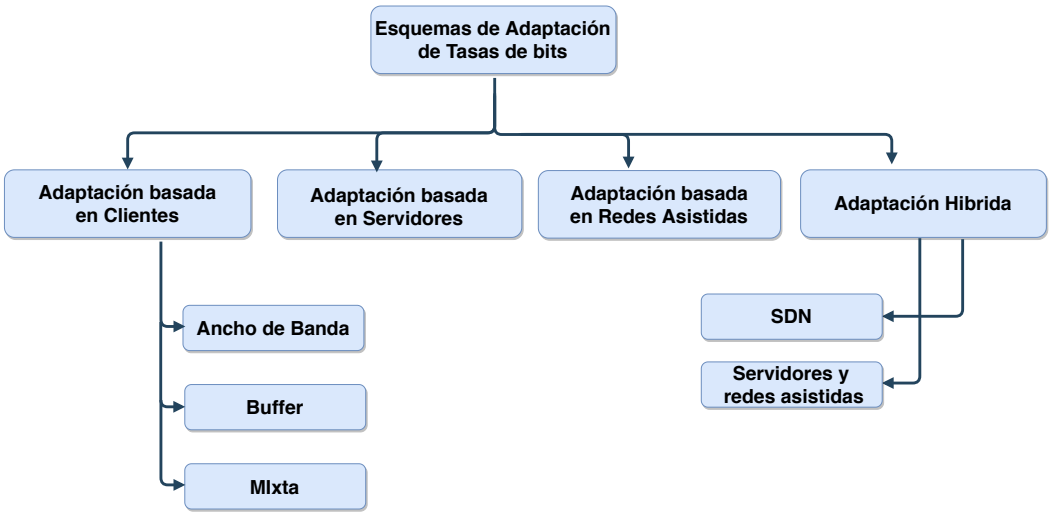

Figura 4.1: Clasificación de los esquemas de adaptación.

4.2 Adaptación basada en redes asistidas

Las redes asistidas permiten a clientes DASH tomar decisiones en la red durante el proceso de adaptación de la tasa de bits. Esto ocurre cuando se recopila información sobre las condiciones de la red y se informa a los clientes sobre las tasas de bits adecuadas que pueden seleccionarse. El proceso dentro de la red necesita componentes especiales (agente/proxy) desplegados en la red para monitorear el estado y las condiciones. También, ofrecen información a nivel de red que permite a los clientes DASH utilizar eficientemente los recursos.

4.3 Adaptación híbrida

Una adaptación de la tasa de bits híbrida, se compone de varias entidades de la red que colaboran juntas y recopilan información útil sobre las condiciones de la red y permiten a los clientes DASH seleccionar una tasa de bits. Este esquema de adaptación consiste en Redes Definidas por Software o SDN (Software Defined Networking), que permiten controlar los recursos de la red y las capacidades de monitoreo, simplificando programación y despliegue de recursos físicos. Por otro lado, se propone una arquitectura asistida por servidor y red para DASH o SAND (Server And Network Assisted DASH), la cual permite un intercambio bidireccional de mensajes entre clientes y elementos de la red que activa un mecanismo de control para priorizar flujos, reservas en el ancho de banda y la adaptación de tasas de bits en función de las condiciones de la red y de los clientes [15].

4.4 Adaptación basada en el cliente

La mayoría de los esquemas de adaptación de tasa de bits que se han propuesto en la literatura residen en el lado del cliente [16]. Estos esquemas intentan adaptarse a variaciones del ancho de banda cambiando a una tasa de bits adecuada en función de uno o más parámetros, como el ancho de banda o el *throughput* disponible, el tamaño del buffer de reproducción, entre otros. El cliente utiliza uno o más parámetros de entrada para su algoritmo ABR con el fin de elegir el nivel de tasa de bits adecuado para el siguiente segmento en ser descargado. Estos algoritmos se esfuerzan por evitar problemas comunes en el *streaming* como retardos en el arranque (especialmente en *live streaming*), inestabilidad del vídeo, oscilaciones en las tasas de bits, inanición en el buffer y por otro lado, mantener o mejorar la QoE del espectador. Como lo muestra la Figura 4.1, este esquema comprende tres subclases de adaptación: ancho de banda disponible, buffer de reproducción y adaptaciones mixtas, las cuales son descritas a continuación.

4.4.1 Adaptación basada en el ancho de banda disponible

El cliente toma sus estimaciones basándose en el ancho de banda de red disponible, que normalmente se calcula como el tamaño del segmento dividido por el tiempo de descarga. En [17], los autores proponen un algoritmo de adaptación que intenta detectar fluctuaciones del ancho de banda y la congestión utilizando un flujo de red suavizado basado en un tiempo de obtención de segmento o SFT (Segment Fetch Time) que mide el tiempo que inicia con el envío de un mensaje HTTP GET hasta recibir el último byte del segmento. Probe AND Adapt (PANDA) [18] estima con precisión el ancho de banda disponible y trata de eliminar el problema del estado estacionario ON-OFF, así como de reducir las oscilaciones de la tasa de bits cuando varios clientes están en un cuello de botella en el mismo enlace que comparten.

4.4.2 Adaptación basada en el buffer de reproducción

En esta clase de esquema, el cliente se basa en los niveles de ocupación del buffer como criterio para seleccionar la tasa de bits correspondiente a la siguiente representación que será descargada. La Figura 4.2, ilustra varias regiones de ocupación, cualquier curva $f(B)$ dentro de la región viable puede generar tasas de bits entre $R_{\min}$ y $R_{\max}$ dada la ocupación del buffer. La tasa de bits más baja ($R_{\min}$) se selecciona cuando el nivel del buffer se encuentra en el área de reserva B, sobrepasada esta zona, la tasa de bits se basa en función del algoritmo que define a $f(B)$. El área intermedia entre la reserva y el punto en el que $f(B)$ es igual a $R_{\max}$ es definida como amortiguador y pasado este punto se encuentra

la reserva superior. El amortiguador siempre debe de mantenerse por encima de B para que sea capaz de absorber las fluctuaciones causadas por variaciones de capacidad. Por lo tanto, $f(B)$ siempre descargará un segmento antes de que el buffer se contraiga al área de la reserva dentro del área segura, de lo contrario se encontrará en el área de riesgo.

Entre los casos más citados se tiene a BBA (Buffer-based approach, por sus siglas en inglés), cuyo objetivo es maximizar la calidad promedio del contenido multimedia, evitando eventos innecesarios de *rebuffering* [19]. Sin embargo, BBA cae en una degradación de la QoE por fluctuaciones del ancho de banda en contenidos de larga duración. El algoritmo BOLA (Buffer Occupancy based Lyapunov Algorithm, por sus siglas en inglés), trata la adaptación de tasa de bits como un problema de maximización de utilidad [20]. Esta utilidad está asociada a una tasa de bits promedio y al tiempo que tarda un evento de rebuffering, mientras que se adapta a cambios en la red. BOLA es el algoritmo que está implementado y disponible en el *framework* DASH JS [21].

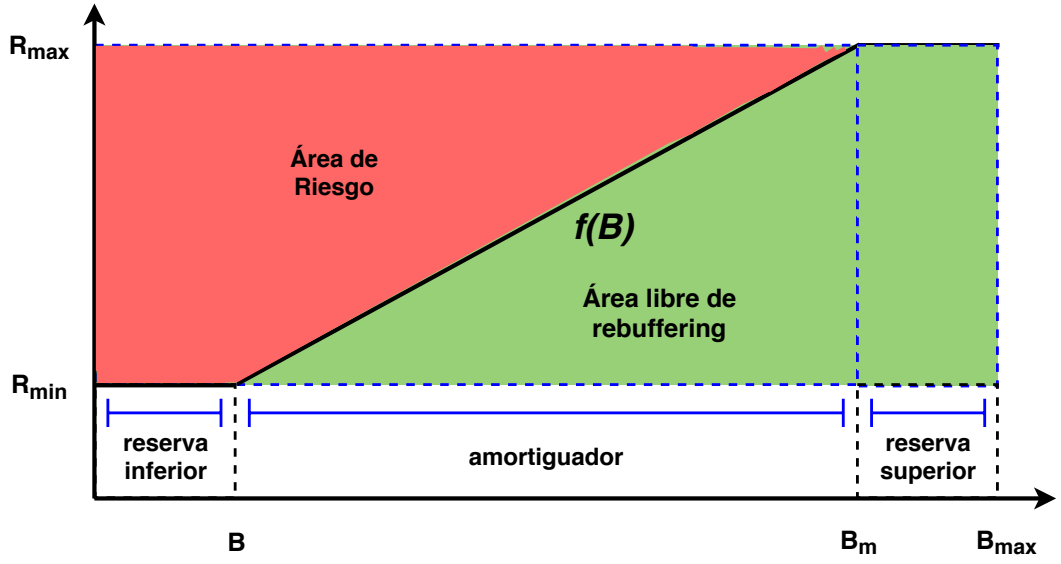

Figura 4.2: Adaptación basada en la ocupación del buffer.

4.4.3 Adaptación mixta

El cliente hace su selección basándose en una combinación de parámetros que incluyen el ancho de banda disponible, la ocupación del buffer, el tamaño y/o duración del segmento, entre otros. ELASTIC (fEedback Linearization AdaptIve STreamIng Controller), es un estimador basado en la teoría del control de retroalimentación, que genera un flujo TCP de larga duración, evitando un comportamiento de estado estacionario ON-OFF que conduce a sobreestimaciones de ancho de banda [22]. ELASTIC puede garantizar una equidad del ancho de banda entre clientes, pero sin tener en cuenta la QoE del espectador. SARA (Segment-Aware Rate Adaptation) se basa en variaciones del tamaño de cada segmento, estimación del ancho de banda disponible y la ocupación del buffer [23]. Como HTTP utiliza TCP, el rendimiento de un segmento depende del tamaño del archivo y, por lo tanto, los autores proponen personalizar el MPD (Media Presentation Description) para incluir este parámetro. Para cada nueva descarga de segmento, el estimador decide la representación del nuevo segmento basándose en el ancho de banda estimado y el estado actual del buffer. GTA (Game Theoretic Approach) es un esquema que utiliza la toma de decisiones de un conjunto de estimadores tomados de la literatura y aplica la teoría de juegos para lograr equidad entre conjuntos de clientes que compiten por el ancho de banda disponible [24].

5 Modelo del algoritmo ABR propuesto

De acuerdo con los objetivos presentados en la Sección 1 y a la revisión de las propuestas en la literatura de la Sección 4, se ha implementado un algoritmo de adaptación de tasas de bits que considera varios parámetros a través de un esquema mixto. Como se mencionó en la sección anterior, a través de una serie de modificaciones en el *parser* del reproductor DASH JS, es posible obtener información acerca del tamaño de los archivos de cada segmento, además de su respectivo peso de calidad. También, se ha implementado un conjunto de reglas, que permiten utilizar estimadores que consideran como parámetros el throughput y niveles del buffer. Además, se toma en cuenta la resolución del dispositivo DR={240p, 360p, 480p, 720p, 1080p}, tipo de contenido CT={animacion, deportes, cine, noticias, documental} y como en algunos proveedores de *streaming*, un tipo de plan de servicio SPT={platino, oro, plata, bronze, normal}. Asimismo, no existe una correlación entre las tasas de bits y los diferentes tipos de contenido, ya que estos poseen características únicas, por ejemplo, escenas con diferentes velocidades de movimientos, iluminación, entre otros, que dan como resultado diferentes estimaciones de calidad. En el contexto de DASH, incluso bajo un escenario en donde el ancho de banda se mantiene constante, la tasa de bits puede variar, debido a la complejidad de las escenas en un vídeo por su contenido. La Figura 5.1, muestra la relación no lineal entre las tasas de bits y peso de calidad de percepción.

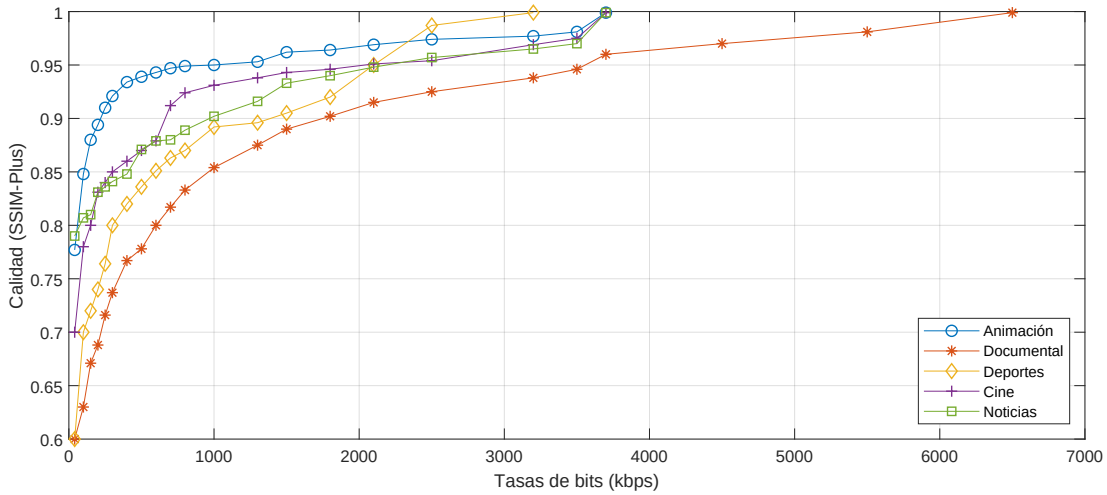

Figura 5.1: Calidad contra la tasa de bits.

5.1 Algoritmos de estimación

Para implementar el esquema propuesto basado en GTA (B-GTA), se toma como referencia la estructura que conforma el conjunto de reglas del reproductor DASH JS descrito en la Sección 5. En primer lugar, se establece una función que toma como parámetros la resolución del dispositivo, el tipo de contenido que está descrito en el archivo MPD y un plan de suscripción de servicio deliberadamente elegido, con el propósito de realizar un mapeo y determinar una tasa de bits máxima que el cliente podrá seleccionar. El Algoritmo 1 describe parte del procedimiento que ha sido implementado en la función de mapeo.

Algoritmo 1: Obtención de peso de calidad y tasa de bits máximo inicial.

```
mientras play hacer
    si DR == 1080p and CT == Animation entonces
        si SPT == normal entonces
            SSIMplusQT = 0.964;
            MaxBitrate = 2400;
        fin
    fin
    si DR == 720p and CT == Animation entonces
        si SPT == normal entonces
            SSIMplusQT = 0.949;
            MaxBitrate = 900;
        fin
    fin
    devolver C_1 = [SSIMplusQT, MaxBitrate]
fin
```

Los estimadores que se han implementado determinarán las condiciones de la red, basándose en sus propias medidas. En primer lugar, a través del Algoritmo 2, se obtiene una estimación del throughput si se alcanza un umbral determinado por los tiempos de descarga y latencia al dar inicio la sesión de *streaming*. En caso contrario, determinará sus estimaciones utilizando el tamaño del archivo del segmento (W_{i+1}), el peso de calidad de percepción que es proporcional a la tasa de bits. De acuerdo la Ecuación 1 definida en el algoritmo SARA [23], se tiene que H_n es el conjunto de segmentos y el tiempo de descarga del próximo segmento esta dado por $\frac{W_{i+1}}{H_n}$:

$$H_n = \frac{\sum_{i=1}^{n} w_i}{\sum_{i=1}^{n} \frac{w_i}{d_i}}, \tag{1}$$

donde d_i es el tiempo de descarga del $i - esimo$ segmento, mientras que w_i es el peso de calidad de percepción proporcional a la tasa de bits.

Para el segundo estimador, definido en el Algoritmo 3, se obtiene el throughput a partir de los tiempos de descarga, latencia y tamaño del archivo del segmento, obtenido del MPD. La implementación de este estimador se basa en el controlador definido en el algoritmo ELASTIC [22], a traves de la Ecuación 2 para estimar la tasa de bits:

$$BW_{est} = \frac{throughput}{d - k_p * Queue - k_i * qI}, \tag{2}$$

con d que un índice de drenado del buffer en la sesión de *streaming*, teniendo como valor asignado 1 si la reproducción esta en curso y 0 en caso contrario. $Queue$ esta definido por el nivel del buffer y la duración del segmento. k_p y k_i son constantes que se obtienen a partir de una función de utilidad de las tasas de bits determinada en BOLA [20]. qI es un estado adicional determinado por la técnica de linealización por retroalimentación usada en ELASTIC que toma en cuenta el tiempo de descarga del segmento anterior y el estado del buffer.

Algoritmo 2: Estimación de una tasa de bits por ancho de banda.

Resultado: bitrate
mientras *play* **hacer**
  si *Request* **entonces**
    $latency_{ms}$ = Tresponse-Trequest;
    $download_{ms}$ = Tfinish-Tresponse;
    $bytes = W_{i+1}$;
    throughput = (bytes*8)/(downloadMS/1000);
    si *thresholdLatency* **entonces**
      $BW_{est} = throughput/1000$;
      $latency = latency_{ms}/1000$;
    en otro caso
      $BW_{est} = \frac{W_{i+1}}{H_n}$;
      $latency = AVGlatency()$;
    fin
    $BW_{max} = BW_{est}$;
    si $BW_{max} > C_1[1]$ **entonces**
      $BW_{max} = C_1[1]$;
      $MaxBitrate = C_1[0]$;
    fin
    $Quality_{max} = mapBitrate(BW_{est})$;
  fin
fin

Algoritmo 3: Estimación de una tasa de bits por ancho de banda y buffer.

Resultado: bitrate
mientras *play* **hacer**
  si *Request* **entonces**
    $latency_{ms}$ = Tresponse-Trequest;
    $download_{ms}$ = Tfinish-Tresponse;
    $bytes = W_{i+1}$;
    throughput = (bytes*8)/(downloadMS/1000);
    si *play* **entonces**
      $d = 1$;
    en otro caso
      $d = 0$;
    fin
    $buffer = buffer_{max}/segmentD$;
    $queueBuff = segmentD/currentBuffer$;
    $qI = qI + downloadT(Queue - buffer)$;
    $BW_{est} = throughput/(d - (k_p * queueBuff) - (k_i * qI))$;
    $BW_{max} = BW_{est}$;
    si $BW_{est} > C_1[2]$ **entonces**
      $BW_{max} = C_1[1]$;
    fin
    $Quality_{max} = mapBitrate(BW_{est})$;
  fin
fin

5.2 Teoría de juegos y la toma de decisiones

En la teoría de juegos, los juegos no cooperativos son una rama muy importante. Este tipo de juegos se centran en el estudio y el análisis en la toma de decisiones competitivas en las que participan varios actores con intereses parciales o que están totalmente en conflicto sobre el resultado de un proceso de decisión que está afectado por sus acciones. Por ejemplo, en las redes de comunicaciones (alámbricas e inalámbricas) diferentes nodos participan en numerosos escenarios no cooperativos, como la asignación de recursos, la elección de frecuencias o potencia de transmisión, el reenvío de paquetes o la gestión de interferencias, entre otros [25, 26]. Un juego no cooperativo refleja la situación competitiva en la que cada jugador necesita tomar sus propias decisiones independientemente de los demás, dadas las posibles opciones de los otros jugadores y su efecto en las utilidades del jugador. Este esquema no significa siempre que los jugadores no cooperen, sin embargo, implica que la cooperación que surja se dé sin que exista comunicación o la coordinación de estrategias entre ellos [27].

Adicionalmente, se dice que un juego es estático si los jugadores realizan sus acciones una sola vez, independientemente unos de otros. Por el contrario, un juego dinámico es aquel en el que los jugadores tienen cierta información sobre las elecciones de los demás y pueden actuar más de una vez. Para describir un juego estático o dinámico no cooperativo se tiene una serie de componentes básicos: el conjunto de jugadores, sus acciones o estrategias y los beneficios o utilidades. Un juego no cooperativo se puede definir como una tupla $\mathcal{G} = (N, (S_i)i_{\in\ N}, (U_i)_{i\in\ N})$, donde:

N es un conjunto finito de jugadores.

S_i es el conjunto de estrategias disponibles para el i-esimo jugador.

U_i: $S \to \mathbb{R}$, es una función de utilidad, dadas sus acciones o estrategias.

En un juego dinámico, las opciones de cada jugador dependen de la información disponible. También se debe de distinguir entre la noción de lo que es una acción y una de estrategia, donde esta última puede ser vista como el mapeo de la información con la que cuenta un jugador y dependiendo de las opciones con las se cuenten, se podrá decidir qué acciones dentro de un conjunto se tomarán.

Dada la descripción previa, el siguiente paso es determinar cuál de las tasas de bits obtenidas a partir de ambos estimadores es la mejor opción. En comparación con el conjunto de reglas implementadas en DASH JS, en donde se selecciona la tasa de menor valor, se propone implementar las funciones de toma de decisión que *Game Theory Approach* (GTA) proporciona. Entonces, se parte de un conjunto de dos o más jugadores (clientes DASH), utilizando un problema no cooperativo para que se alcance un consenso entre estos, eligiendo la mejor estimación.

El algoritmo basado en GTA formula una serie de funciones para la toma de decisiones de un algoritmo de adaptación de tasa de bits como un problema basado en un juego no cooperativo, a través de un proceso de negociación, con el objetivo de llegar a un consenso en la elección de una tasa de bits adecuada para la descarga del próximo segmento. El problema se define como un juego $\mathcal{G}$(E,m,A,U,S), donde la tupla G consiste de un conjunto de estimadores (E), un servidor DASH (m), un conjunto de acciones (A), un conjunto de utilidades (U) y un conjunto de estrategias (S). Estos elementos en GTA pueden formar un proceso de negociación, o un acuerdo, entre ellos que puede mejorar sus decisiones y maximizar sus utilidades (QoE). El objetivo final, es llegar a un consenso seleccionando sólo las decisiones ABR óptimas (tasa de bits) con sus correspondientes util-

idades máximas, resultado de la negociación, teniendo en cuenta varios escenarios durante una sesión de *streaming*. Los pasos para determinar una tasa de bits son:

a) Los estimadores realizan de forma independiente los cálculos sobre las condiciones de la red para obtener una tasa de bits máxima, dependiendo de los valores almacenados en C_1 dado el Algoritmo 1.

b) Cada estimador genera sus utilidades basándose en las tasas de bits y los respectivos pesos de calidad, según el valor máximo de C_1. Por otro lado, se van obteniendo conjuntos que calculando promedios sobre los cambios en las tasas de bits ($AVGSw$), duraciones de las interrupciones ($AVGStall$), tiempo que tarda en iniciar la reproducción del contenido ($TStart$) y la duración total ($TDuration$), generando la siguiente ecuación de utilidad:

$$U_{est} = cte*SSIMPlus_{val}–(cte*AVGSwi–cte*AVGStall–(cte*\frac{TStart}{TDuration})) \quad (3)$$

donde *cte* toma su valor del número de elemento que conforman CT, DR y SPT.

c) Se generan conjuntos de estrategias, a partir de las acciones determinadas por cada tasa de bits y su respectivo peso de calidad, integrando las utilidades previamente obtenidas.

d) Como paso de confirmación, se genera un segundo conjunto de acciones y utilidades, considerando la información de la descarga anterior, que se guardó previamente.

e) Se calculan los resultados del proceso de negociación, de acuerdo con los datos de las acciones, y los conjuntos de utilidades, a través de las siguientes ecuaciones:

$$OutcomeResultBitrate_i = (BitrateM–actionO_i)^{alphavalue} \quad (4)$$

$$OutcomeResultQuality_i = (SSIMPlus_{val}–actionO_i)^{alphavalue} \quad (5)$$

$$OutcomeResultUtility_i = (UtilityM - UtilityO[i])^{alphavalue} \quad (6)$$

f) Definidas las acciones, las utilidades, los resultados del proceso de negociación de cada estimador, se obtienen los valores de toma de decisiones, los cuales son resultado de un nuevo procedimiento similar a los pasos previamente descritos a partir de una nueva estimación. Entonces, se buscan diferencias entre los nuevos datos con los que anteriormente ya se contaba, si éstas entran dentro de un rango similar a los mismos

valores dados por C_1, se consideran como valores óptimos y se determina la tasa de bits. En caso contrario, se elegirá el de valor mínimo.

g) Los valores que se consideren óptimos serán almacenados y utilizados en la siguiente estimación.

h) Posteriormente, se somete la tasa de bits elegida a una serie de restricciones, donde por otro lado las acciones óptimas relacionadas definen un umbral máximo y uno mínimo, que el buffer de reproducción puede alcanzar. Además, la tasa de bits no debe sobrepasar una nueva estimación del throughput.

i) Finalmente, para evitar oscilaciones, se implementa un método similar al utilizado en BOLA en el que se compara la nueva estimación con la anterior y el cálculo del throughput.

6 Experimentación

6.1 Reproductor DASH JS

Dado que la entrega de contenidos de contenido en la Web se ha vuelto omnipresente hoy en día, los portales de vídeo en línea como Netflix, YouTube, Amazon, entre otros, cuentan con millones de usuarios que ven su contenido todos los días. La mayoría de estas plataformas adoptan soluciones que están migrando a DASH y HTML5 desde formatos y plugins propietarios. En consecuencia, los proveedores de contenido tienen que hacer la transición a reproductores basados en HTML5 para poder continuar la reproducción en los navegadores Web, surgiendo una gran variedad de proyectos de código abierto disponibles en torno a MPEG-DASH o que lo soportan. Una de estas herramientas es *dash js* [21], un reproductor de código abierto HTML5/JavaScript DASH. Aprovecha las interfaces MSE (Media Source Extensions) y EME (Encrypted Media Extensions) del W3C que incluye una amplia gama de funciones, incluyendo DRM (Digital Rights Management), subtitulado y *streaming* adaptativo. La Figura 6.1, muestra la arquitectura del reproductor DASH JS que consta de una serie de componentes, que son:

- El **Gestor de eventos** actúa como interfaz procesando los eventos emitidos para el Media Source API [28], cuando este emite el evento *webkitsourceopen*, el gestor activará la descarga del archivo MPD y, a continuación, se cargará al "MPD Parser".

- El programa de evento llama al **Buffer** que recuperará y almacenará los segmentos de una representación.

- El **Parser MPD** analizará el archivo, esto generará un objeto que contiene toda la información relevante sobre el MPD, como las representaciones definidas con sus segmentos, la tasa de bits de cada representación, la resolución de los fotogramas de vídeo. Si los segmentos están alineados en todas las representaciones, esta información es utilizada por el módulo Lógica de Adaptación para determinar qué representaciones están disponibles.

- La solicitud del archivo MPD se realiza con el módulo de Solicitud de Segmentos, que utiliza *xmlHttpRequest* para generar peticiones HTTP-GET. Además, proporciona un método asíncrono y otro síncrono para solicitar y recibir peticiones y respuestas HTTP. El Estimador de Ancho de Banda es utilizado por el módulo de Solicitud de Segmentos, para medir y estimar el throughput, con base en la Ecuación 7.

$$b_n = \frac{w_1 b_{\mathrm{n-i}} + w_2 b_{\mathrm{m}}}{w_1 w_2}, \tag{7}$$

donde b_{n-1} es el *throughput* calculado en el segmento $n-1$, b_{m} indica el *throughput* estimado cuando se está descargando el segmento $n-1$, mientras que w_1 y w_2 son factores de peso. En lugar de utilizar todos los valores de ancho de banda que se han calculado, se utiliza la estimación del *throughput* del segmento $n-1$. Los pesos permiten ajustar la influencia del segmento previamente descargado. En el caso del primer segmento, b_0, se toma en cuenta el ancho de banda estimado al descargar el archivo MPD. Esta estimación se realiza con cada segmento que se solicita y se descarga.

- El modulo **Buffer base** proporciona la implementación para el buffer. Ofrece la posibilidad de registrar gestores de eventos para acontecimientos específicos, por ejemplo, $criticalFillLevel$. Además, se proporcionan dos implementaciones de tipos de buffer. Primero, un buffer que opera en bytes, que puede ser usado para almacenar segmentos y consultar rangos de bytes de los segmentos almacenados y, segundo, un buffer que opera con el tamaño (en segundos) de un segmento.

  Debido a que Media Source API no permite acceder al buffer del elemento de vídeo en HTML5, se implementó el módulo **Overlay Buffer**. Este buffer hereda las funciones de Base Buffer e imita al buffer del elemento de vídeo. Con cada suceso emitido por Media Source API se llamará al método $bufferFillStateListener()$. Este método realiza un seguimiento del progreso del contenido multimedia que se está reproduciendo restando del buffer la cantidad de tiempo que ha pasado entre la última llamada de este método y la marca de tiempo actual. Además, activará la descarga de otros segmentos, que luego se transfieren al elemento de vídeo y el nivel del buffer se incrementa en función del tamaño de cada segmento. Este buffer no almacena bytes, sólo lleva un registro de cuántos segmentos se han introducido en el elemento de vídeo en segundos y el tiempo de reproducción actual del vídeo.

  La ocupación del buffer y la estimación del ancho de banda son parámetros con los que se puede determinar durante el *stream* qué tasa de bits o representación debe seleccionarse. Para permitir la implementación de algoritmos de adaptación de tasa de bits personalizados se proporciona la clase **Lógica de adaptación**. Cada nueva lógica o regla implementada debe heredar de esta clase para poder utilizarla

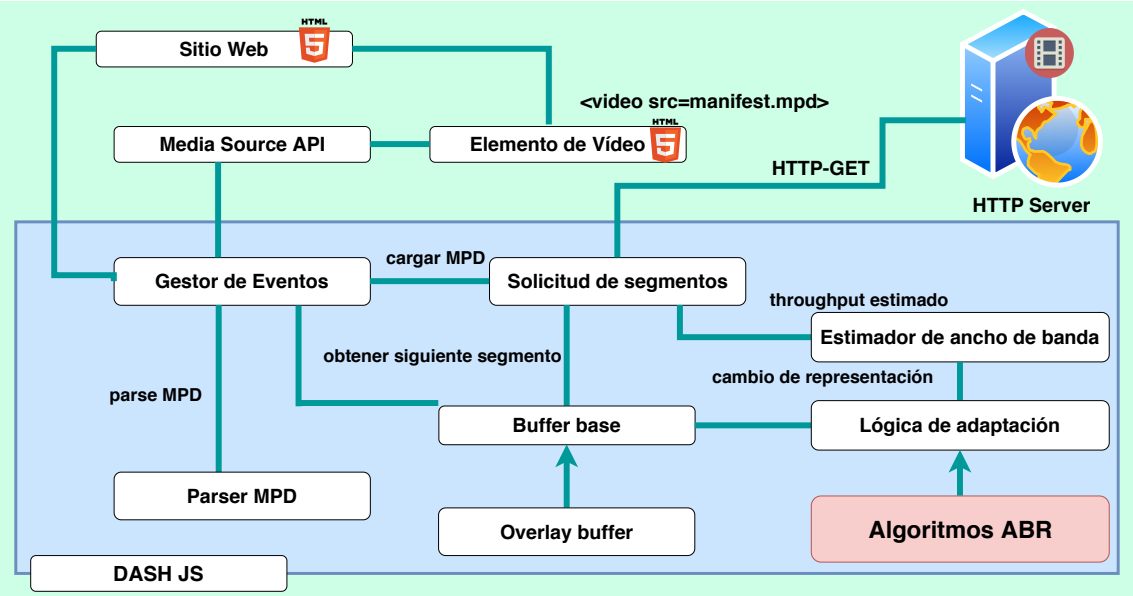

Figura 6.1: Arquitectura del reproductor DASH JS.

y la regla de adaptación es llamada después de cada segmento descargado.

6.1.1 Lógica ABR

En DASH-JS a través del módulo **Algoritmos ABR**, se encuentran implementadas un conjunto de reglas de adaptación de tasa de bits, divididas en reglas primarias y secundarias. De forma predeterminada, cada regla devuelve un índice máximo de la lista de tasa de bits disponible y la decisión final se toma del valor mínimo del conjunto.

6.1.1.1 Reglas primarias

Las reglas principales que rigen en ABR son ThroughputRule o BolaRule (BOLA, descrita en la Sección 4). Una instancia de reproductor puede ser forzada a elegir cualquiera de las dos reglas usando el método $MediaPlayer.setABRStrategy(value)$. Los algoritmos válidos son abrDynamic, abrBola y abrThroughput. La estrategia (algoritmo) ABR se puede cambiar durante una sesión de *streaming* y su valor por defecto es abrDynamic. La Figura 6.2, muestra el diagrama de flujo que describe la operación de las reglas ABR [29].

- BOLARule (abrBola) elige la tasa de bits basada en el nivel de buffer actual, con tasas de bits más altas para niveles de buffer más altos. Para evitar oscilaciones, BOLA no cambia la tasa de bits si la estimación del throughput no soporta una tasa

de bits más alta y a su vez realiza un seguimiento del porqué el nivel de buffer baja, manteniendo una tasa de bits más alta cuando el nivel de buffer baja por razones no relacionadas con limitaciones de ancho de banda de la red.

- ThroughputRule (abrThroughput) utiliza el historial de throughput reciente para realizar una estimación del throughput próxima. Una vez que tiene una estimación, ThroughputRule lo reduce en un factor de 90% de seguridad y entonces selecciona la tasa de bits más alta. Si ThroughputRule elige la tasa de bits superior y el nivel de buffer se eleva por encima de algún umbral, entonces no elegirá una tasa de bits inferior hasta que el nivel de buffer descienda por debajo del umbral, incluso si el nivel de buffer disminuye.

- DynamicRule ("abrDynamic") cambia entre BOLA y ThroughputRule en tiempo real, aprovechando los puntos fuertes de ambos.

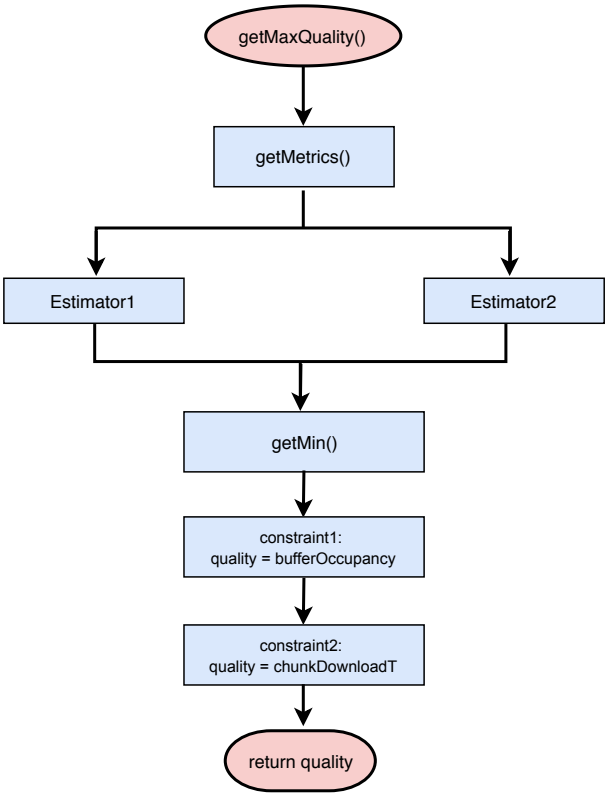

Figura 6.2: Diagrama simplificado que describe la operación de las reglas ABR.

6.1.1.2 Reglas secundarias

Mientras que las reglas primarias son un factor determinante, existe una serie de reglas secundarias utilizadas en algunos casos especiales.

- InsuficienteBufferRule evita el rebuffering, cuando el nivel de buffer es bajo, esta regla limita la tasa de bits a una estimación discreta del throughput. Si se produce un rebuffering, se fuerza la tasa de bits más baja para acelerar la reanudación de la reproducción.
- SwitchHistoryRule evita oscilaciones bruscas de la tasa de bits. Si alguna de las otras reglas ABR cambia de una tasa de bits, entonces SwitchHistoryRule no permitá el uso de esa tasa de bits ni de cualquier otra más alta.
- DroppedFramesRule evita las tasas de bits que no se pueden renderizar correctamente debido a las limitaciones del CPU.

6.1.1.3 Reglas personalizadas

Uno de los aspectos más importantes de DASH JS es la de proporcionar una serie de mecanismos para extender y/o modificar su algoritmo ABR a través de la definición de reglas personalizadas.

6.2 *Dummynet*

Es un emulador de redes desarrollado bajo el sistema operativo FreeBSD, posteriormente fue exportado a otros sistemas operativos, principalmente a Mac OS, Linux y Windows. Es capaz de interceptar el tráfico de la red y manipularlo, emulando el comportamiento de uno o más enlaces con características programables. *Dummynet* está dividido en tres partes: el emulador, el clasificador de paquetes *ipfw* y la interfaz de usuario */sbin/ipfw* [30]. La Figura 6.3, muestra el funcionamiento de *dummynet*, donde la capa superior está conectada a uno de los interlocutores que se comunicarán, y la capa inferior se conecta a la red. Para simular la presencia de una red entre ambos, es necesario determinar elementos en el flujo de datos como un tamaño (limitado) de cola y enlaces de comunicación (*pipe*) con un ancho de banda y un retardo determinados.

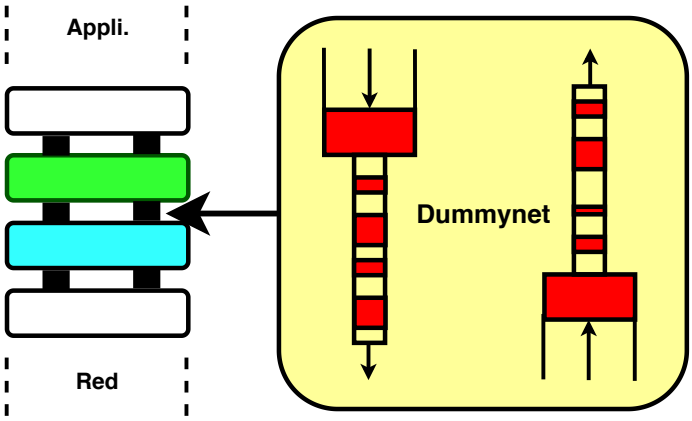

Figura 6.3: Descripción gráfica del emulador Dummynet.

- ***El emulador*** puede crear múltiples instancias de un objeto llamado *pipe*, el cual básicamente modela un enlace de red con ancho de banda programable, retardos y tamaño de la cola. *Pipe* permite configurar otras opciones ya existentes como especificar diferentes políticas de administración de colas, modelar algunos efectos de capa MAC tales como tiempos de transmisión variables y sobrecargas a nivel de enlace, también simular patrones de pérdida de paquetes.

- ***Clasificador de paquetes.*** *Dummynet* trabaja de la mano con un clasificador de paquetes *ipfw*, el cual intercepta los paquetes en varios puntos de la pila de protocolos decidiendo su destino. El clasificador se programa mediante un conjunto de reglas numeradas, cada una de las cuales contiene entre cero o más opciones que son utilizadas para hacer coincidir los paquetes con una acción, especificando qué hacer con aquellos paquetes que coinciden. Estas opciones incluyen direcciones, puertos, protocolos, banderas de protocolo y varios metadatos de paquetes. Un paquete es probado en cada una de las reglas en orden numérico, y la primera regla termina la búsqueda y causa la ejecución de la acción asociada. El objetivo es enviar el paquete a un *pipe*, el cual a su vez retrasará o desechará el paquete según corresponda, emulando el comportamiento del enlace adjunto. Después de la emulación, los paquetes no eliminados se vuelven a enviar a la pila de red para su procesamiento regular.

- ***Interfaz de Ususario.*** El usuario decide qué segmento del tráfico debe ser interceptado y por cuáles *pipes* debería ir. La configuración de los *pipes* (en términos del ancho de banda o retardos) puede ajustarse o modificarse en tiempo de ejecución según sea necesario. A través del ejemplo mostrado en el Bloque 6.1, puede ilustrarse la configuración de dos *pipes* con diferentes características y el tráfico que pasará a través de ellas hacia dos diferentes subredes:

```
#Configuracion de dummynet
#Establecimiento del ancho de banda y retardos de los enlaces
    simulados
ipfw pipe 5 config bw 4Mbit/s delay 7ms
ipfw pipe 8 config bw 1Mbit/s delay 10ms
#Configurando ipfw:
#Seleccion del trafico que pasara por el emulador
ipfw add 120 pipe 5 out dst-ip 10.2.0.0/24
ipfw add 130 pipe 8 out dst-ip 10.1.1.0/24
```

Bloque 6.1: Establecimiento del ancho de banda y retardos de los enlaces simulados.

La configuración de los *pipes* y colas puede cambiarse en tiempo de ejecución sin interrumpir el funcionamiento de la red. Del mismo modo, las reglas de *ipfw* pueden añadirse o eliminarse en tiempo de ejecución para modificar la configuración del sistema, y sin interrumpir el tráfico no afectado por dichas reglas. Mediante otro ejemplo que se muestra en el Bloque 6.2, es posible limitar el tráfico entrante a un *pipe* que utiliza TCP a 2 Mbps, mientras que a otro con UDP a 300 Kbps:

```
#Configuracion de dummynet
#Establecimiento de los protocolos a los enlaces
ipfw add pipe 2 in proto tcp
ipfw add pipe 3 in proto udp
#Limitacion del trafico entrante
ipfw pipe 2 config bw 2Mbit/s
ipfw pipe 3 config bw 300Kbit/s
```

Bloque 6.2: Limite del tráfico entrante a un *pipe.*

6.3 Banco de trabajo

La Figura 6.4, muestra la configuración del banco de trabajo, el cual está conformado por tres componentes: clientes DASH, emulador y servidor HTTP:

El reproductor DASH solicita y descarga el archivo MPD, que contiene la estructura de organización de los segmentos e información personalizada basado en las modificaciones propuestas en el algoritmo SARA [23], que se detallará más adelante. Para la arquitectura del reproductor se utiliza DASH JS [21], dadas sus facilidades para implementar algoritmos ABR.

También se tiene un equipo intermedio que tiene las funciones de *router* redirigiendo el tráfico entre los clientes y el servidor. Por otro lado, este ejecuta el emulador *Dummynet*, que mediante pequeños scripts manipula de forma controlada el tráfico utilizando variaciones del ancho de banda y retardos.

Además, se tiene el servidor DASH, el cual es un servidor HTTP convencional que almacena dos conjuntos de segmentos de cuatro y diez segundos de duración, además de un archivo MPD (uno por cada conjunto).

Finalmente, también se tiene el parser o analizador de los archivos MPD en el reproductor DASH JS, el cual ha sido modificado con el propósito de integrar datos como el tamaño de los segmentos o el peso de calidad de percepción determinado por SSIMPLUS a cada segmento, como se muestra en el Bloque 6.3.

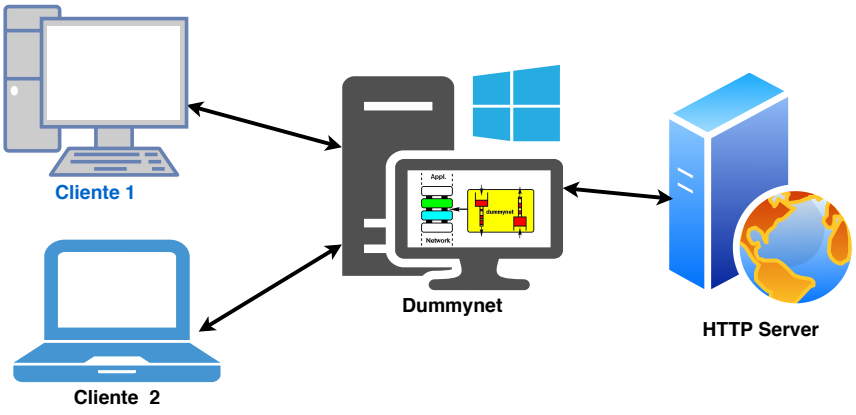

Figura 6.4: Organización del banco de trabajo para pruebas.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- MPD file-->
  <Period duration>
    <AdaptationSet>
      <Representation id="320x240 45.0kbps" mimeType="video/mp4" width=
          height= frameRate=   bandwidth=>
          <SegmentTemplate timescale="96" media= />
          <SegmentSize id="BigBuckBunny_4s1.m4s" size="168.0" qt="0.777"/>
            ...
            <SegmentSize id="BigBuckBunny_4s150.m4s" size="136.0" qt="0.777
                "/>
          </Representation>
</AdaptationSet>
  </Period>
</MPD>
```

Bloque 6.3: Estructura de un archivo MPD personalizado.

6.4 Codificación de las representaciones

6.4.1 Herramientas

FFmpeg

Es un conjunto de herramientas de software libre, multiplataforma para decodificar, codificar, transcodificar, transmitir, filtrar y reproducir casi cualquier formato de contenido multimedia. Este se encuentra compuesto por FFmpeg, FFserver, FFTplay, entre otros.

FFmpeg, es un conversor de audio y vídeo muy rápido que también puede obtener contenido multimedia de un *streaming* en vivo. Su ejecución consiste en la lectura de un número arbitrario de archivos de entrada, seguido por un conjunto de opciones y un número arbitrario de salidas [31]. El Bloque 6.4, muestra cómo codificar un archivo de vídeo de 200 kbps de tasa de bits con resolución 320x180p.

```
ffmpeg -i input.avi -s 320x180 -c:v libx264 -b:v 200k -g 90 -an
output.mp4
```

Bloque 6.4: Codificación de un archivo de vídeo.

Mp4box

Es un conversor MPEG-4, el cual puede importar vídeo MPEG-4 y flujos de audio en un contenedor MP4. A través de esta herramienta, es posible fragmentar cada archivo de vídeo para que DASH pueda interpretarlos, además también genera el archivo MPD [32]. El Bloque 6.5, muestra cómo preparar las representaciones de vídeo para que cada una esté fragmentada por segmentos de 4 segundos de duración, el audio estará fragmentado en segmentos de 2 segundos y se generará el archivo con extensión *mpd* nombrado manifest.

```
mp4box -dash 4000 -rap -profile
dashavc264:onDemand -mpd-title BBB
-out manifest.mpd -frag 2000
input_audio_128k.mp4
input_video_160x90_250k.mp4
input_video_320x180_500k.mp4
input_video_640x360_750k.mp4
input_video_640x360_1000k.mp4
input_video_1280x720_1500k.mp4
```

Bloque 6.5: Obtención de las representaciones de vídeo.

6.5 Contenido Multimedia

El corto animado *Big Buck Bunny* [33] consiste de 4315 fotogramas de extensión *png*, con relación de aspecto 16:9 y resolución de 1920x1080. Se encuentra disponible en 24 fps y 60 fps. El audio se mantiene a una calidad de 128 kb/s, La Tabla 6.1, describe las diferentes tasas de bits codificadas de las representaciones que se almacenaron en el servidor con sus respectivas resoluciones.

6.6 Duración de los segmentos

La segmentación del contenido multimedia, es una parte muy importante en el proceso de preparación de un servicio de *streaming*, ya que el procedimiento de adaptación permite cambiar entre las diferentes representaciones durante la sesión de *streaming*. Sin embargo, la duración óptima del segmento puede depender de un entorno fijo o móvil, un tipo de servicio, entre otros. Por ejemplo, los segmentos cortos son ideales para adaptarse rápidamente a los cambios de ancho de banda y evitar interrupciones, pero los segmentos más largos pueden tener una mejor eficiencia y calidad de codificación. Sin embargo, el cliente no es capaz de ajustar de forma tan flexible y rápida como sería posible con

segmentos más cortos y, por lo tanto, las tasas de bits se deterioran para tamaños de segmento más grandes.

Por lo tanto, es recomendable el uso de segmentos con duración de entre 2 a 4 segundos, lo que supondría una buena eficiencia de codificación y mejor flexibilidad para la adaptación del stream en oscilaciones del ancho de banda [34].

Tabla 6.1: Tasas de bits y sus correspondientes resoluciones.

Indice	Tasa de bits	Resolución
0	45 kbps	320x240
1	88 kbps	320x240
2	128 kbps	320x240
3	177 kbps	480x360
4	217 kbps	480x360
5	255 kbps	480x360
6	323 kbps	480x360
7	378 kbps	480x360
8	509 kbps	854x480
9	577 kbps	854x480
10	782 kbps	1280x720
11	1008 kbps	1280x720
12	1207 kbps	1280x720
13	1473 kbps	1280x720
14	2087 kbps	1920x1080
15	2409 kbps	1920x1080
16	2944 kbps	1920x1080
17	3340 kbps	1920x1080
18	3613 kbps	1920x1080
19	3936 kbps	1920x1080

7 Resultados

Para una evaluación del esquema implementado basado en GTA (B-GTA) a través de DASH JS, se utilizan segmentos de cuatro segundos de duración aplicando variaciones del ancho de banda y retardos determinados en la Tabla 7.1. Aplicando el script del Bloque 7.1 en el emulador Dummynet, se inyectan variaciones en el ancho de banda cada 45 segundos en la red del cliente. La Figura 7.1, muestra una serie de pruebas, en donde se mantiene constante el ancho de banda a una tasa de 4 Mb/s, indicando que la tasa de bits fue constante durante toda la reproducción del contenido multimedia. Posteriormente, aplicando variaciones de entre 1.5 a 3.5 Mb/s cada 45 segundos las tasas de bits estuvieron dentro del rango indicado. De igual manera, al aplicar variaciones de a 1 a 2 Mb/s y 0.5 a 1.5 Mb/s las tasas de bits se mantienen, aunque ya con menores valores debido a esas restricciones.

Tabla 7.1: Parámetros de los escenarios.

Duración del Segmento	Oscilación Ancho de banda	Retardos
4 y 15 segundos	4 Mbit/s	0-50 ms
	3.5-1.5 Mbit/s	100-200 ms
	2.2-1.0 Mbit/s	500-550 ms
	1.5-0.55 Mbit/s	700-750 ms

```
@echo on
@set CYGWIN=nodosfilewarning
@ipfw -q flush
@ipfw -q pipe flush
TIMEOUT 45

ipfw add pipe 4 src-ip 215.32.168.0/24 in
ipfw pipe 4 config bw1.51mbits/s
TIMEOUT 45

ipfw add pipe 4 src-ip 215.32.168.0/24 in
ipfw pipe 4 config bw 3.5mbits/s
```

Bloque 7.1: Limitación del ancho de banda en la red del cliente.

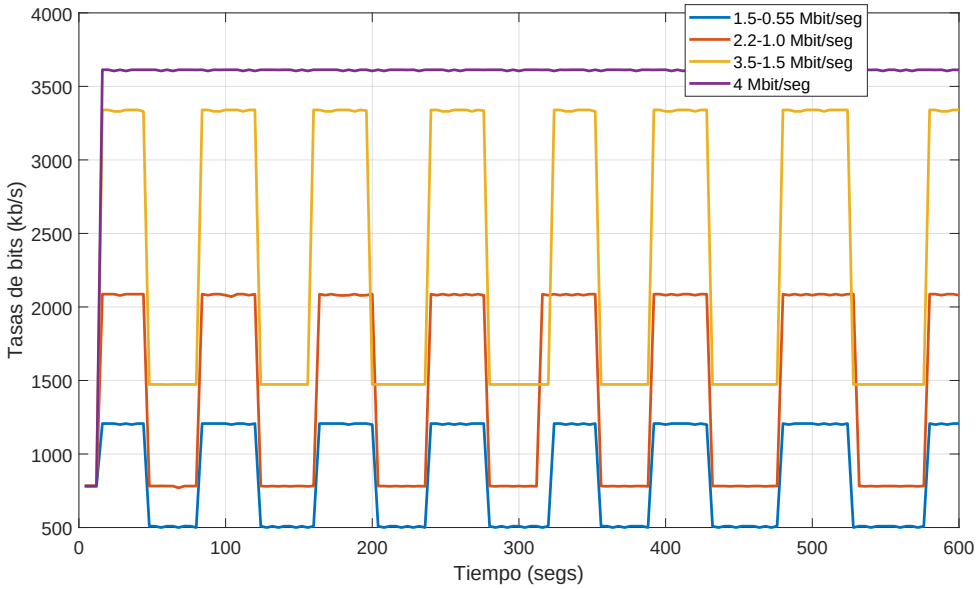

Figura 7.1: Oscilaciones de ancho de banda en segmentos de 4 segundos.

En otra serie de pruebas a la implementación B-GTA, se aplican diferentes retardos cada 45 segundos, ejecutando scripts como el del Bloque 7.2. En la Figura 7.2, se pueden observar cuatro pruebas. En primer lugar, se aplicaron retardos de entre 0 a 20 milisegundos, aquí la variación de tasa de bits es nula, manteniendo los valores máximos que los estimadores se lo permiten sin alterar la QoE del espectador. En otro caso, aplicando retardos de entre 100 a 200 milisegundos y 500 a 550 milisegundos existen variaciones (constantes) que ya limitan más los valores de tasas de bits que los estimadores pueden seleccionar. Por último ya aplicando retardos más prolongados de entre 700 a 750 milisegundos las tasas de bits alcanzan valores mínimos, incluso llegando a presentarse interrupciones en la reproducción.

```
@set CYGWIN=nodosfilewarning
@ipfw -q flush
@ipfw -q pipe flush
TIMEOUT 45
ipfw add pipe 4 src-ip 215.32.168.0/24 in
ipfw pipe 4 config delay 500ms
TIMEOUT 45
ipfw add pipe 4 src-ip 215.32.168.0/24 in
ipfw pipe 4 config delay 550ms
```

Bloque 7.2: Aplicación de retardos en la red del cliente.

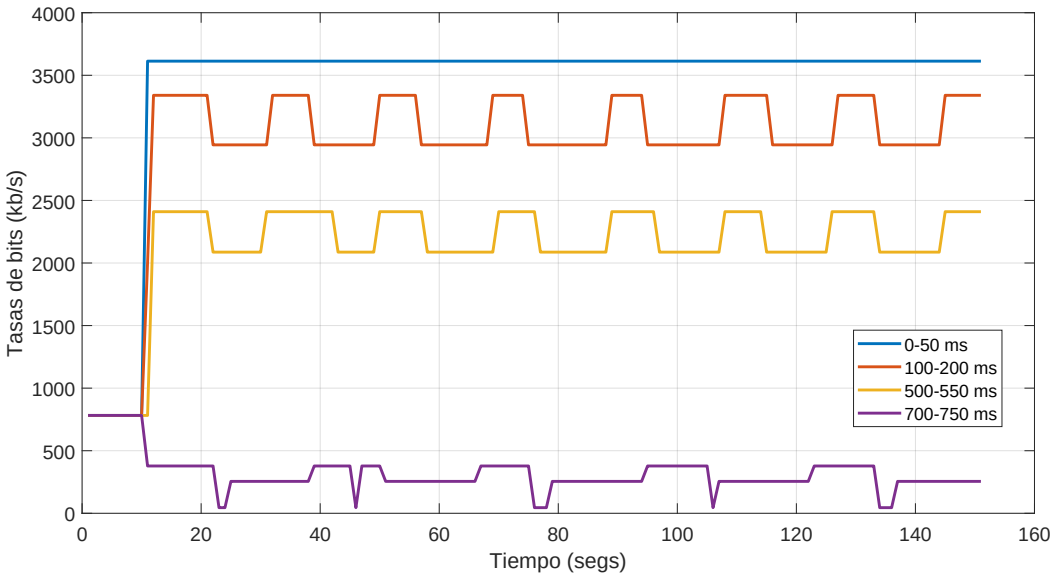

Figura 7.2: Retardos en segmentos de 4 segundos.

La Figura 7.3, muestra una comparación entre los algoritmos de adaptación de tasa bits BBA, ELASTIC, GTA y la implementación basada en este último B-GTA. A cada uno se le aplicaron oscilaciones en el ancho de banda de entre 3 y 1 Mb/s cada 45 segundos, similar al script del Bloque 7.1 durante una serie de sesiones de *streaming*. Se puede observar que tanto BBA como ELASTIC presentan comportamientos irregulares, que se traducen en una subestimación de los recursos, eligiendo tasas de bits inferiores. En comparación, GTA presenta una mejor adaptación a los cambios en el ancho de banda permitiéndole elegir tasas de bits justo por debajo de los límites establecidos que le evita desbordar el buffer de reproducción provocando interrupciones o degradar la QoE al elegir tasas de bits muy inferiores en este caso. Por otro lado, le sigue B-GTA que bajo el esquema que se ha propuesto no logra superar a GTA, sin embargo, mantiene una adaptación aceptable evitando la sub o sobre estimaciones en la elección de las tasas de bits.

Retomando el análisis acerca de la importancia de la duración de los segmentos de la Sección 6, la Figura 7.4, muestra los resultados entre los algoritmos BBA, ELASTIC, GTA y B-GTA, utilizando segmentos de 15 segundos de duración. En comparación con los resultados utilizando segmentos de 4 segundos, ninguno de los algoritmos pudo responder de forma óptima ante oscilaciones del ancho de banda pasando de 3 a 1 Mb/s cada 45 segundos. Los Algoritmos BBA y ELASTIC, reaccionaron con oscilaciones de diferentes magnitudes, mientras que GTA y B-GTA las contuvieron de mejor manera. Sin embargo,

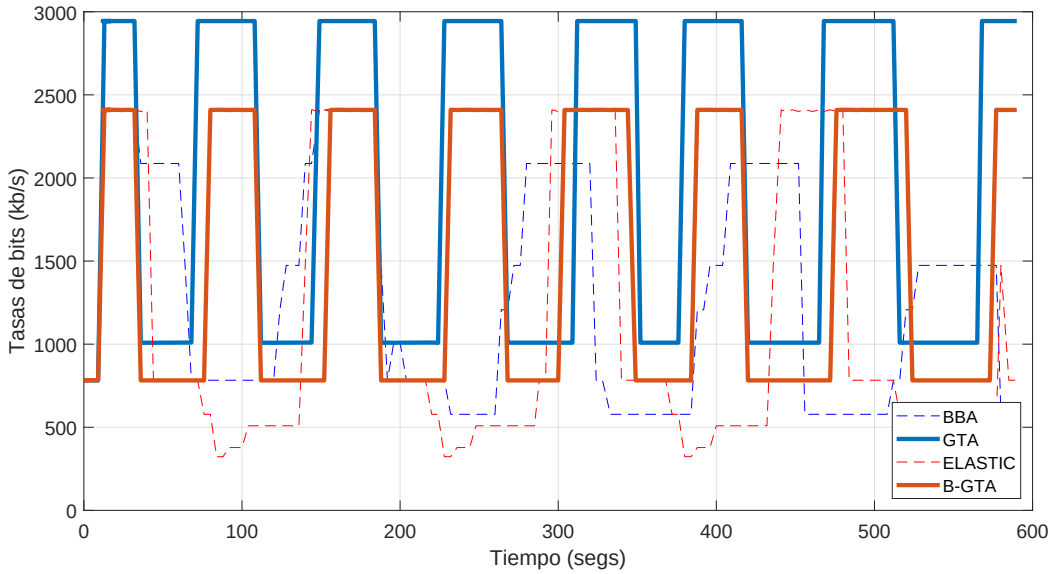

Figura 7.3: Algoritmos de adaptación de tasa de bits con segmentos de 4 segundos.

para el caso de B-GTA este llegó a estimar tasas de bits mínimas, incluso llegando a sufrir algunas interrupciones. Por lo anterior, estos resultados confirman que no es recomendable el uso de segmentos de larga duración, toda vez se puede traducir en una violación a los principios de DASH.

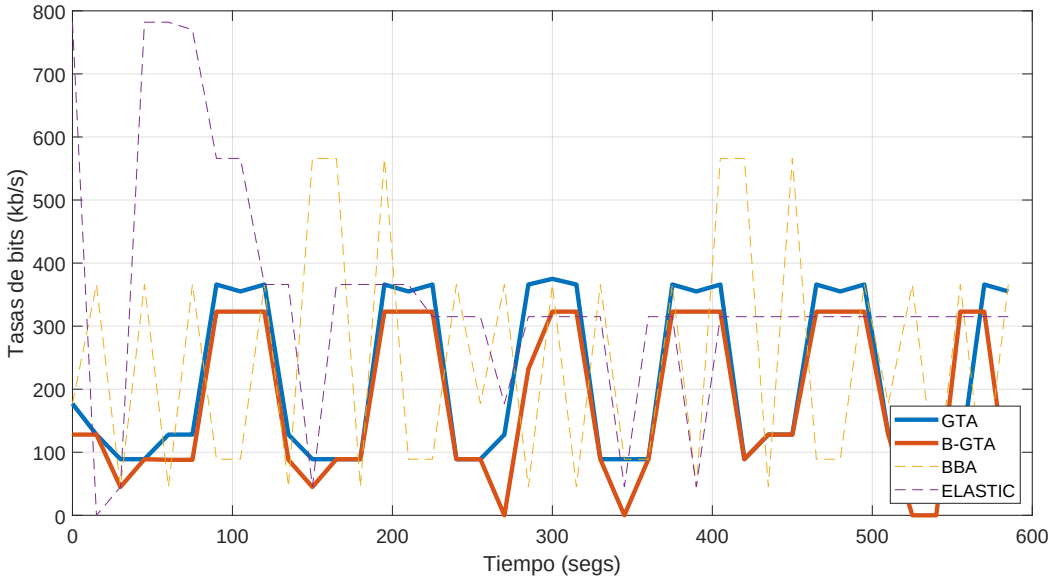

Figura 7.4: Algoritmos de adaptación de tasa de bits con segmentos de 15 segundos.

8 Conclusiones

En este trabajo de investigación se han estudiado los diferentes componentes que definen al *Streaming* Adaptativo sobre HTTP (HAS), centrándose en su estándar más popular, el *Streaming* Dinámico Adaptativo sobre HTTP (DASH). Esto, derivó en la implementación de un pequeño sistema de *streaming* que permite a un cliente acceder a un archivo MPD y solicitar a un servidor HTTP el contenido multimedia que tiene almacenado. Entonces, a través del reproductor DASH JS es posible visualizarlo en un dispositivo compatible. Mediante el uso de algoritmos de adaptación de tasas de bits, el cliente DASH es capaz de adaptarse a las condiciones de la red, toda vez que se han establecido mecanismos para la toma de decisiones con ayuda de un conjunto de parámetros centrados en diferentes estimaciones como el ancho de banda (determinado por el throughput) o monitorizando los niveles del buffer de reproducción. Asimismo, se han integrado una serie de métricas que permiten acotar las tasas de bits que un cliente puede seleccionar, tal es el caso del peso de calidad de percepción, tipo de contenido, resolución del dispositivo y tipo de servicio. Por otro lado, parte de esta información se puede obtener mediante una serie de modificaciones al archivo MPD y al parser o analizador del reproductor DASH JS.

Para hacer más inteligente la toma de decisiones, se han integrado las funciones que componen a Game Theory Approach o GTA, que se basa en un problema de teoría de juegos que resuelven la adaptación de tasa de bits entre una serie de conjuntos de múltiples clientes DASH, con el objetivo de que entre ellos alcancen un consenso a través de un proceso de negociación. Con ello, se desarrolló B-GTA, cuyo objetivo es que de dos estimadores se llegue a un consenso, a través de sus valores de estimación se caiga en un mismo subconjunto determinado por una de las funciones de mapeo.

Como resultado de una serie de pruebas aplicadas al sistema de *streaming* implementado, se realizó una serie de variaciones en el ancho de banda y se determinaron retardos de diferente duración. Con esto se demuestra que mientras se utilicen segmentos de corta duración, en este caso 4 segundos, el algoritmo de adaptación responderá de manera adecuada ante los cambios en la red y se podrá elegir debidamente la tasa de bits del siguiente segmento. Caso contrario al utilizar segmentos de 15 segundos de duración, con los que la mayoría de las propuestas no lograron responder adecuadamente.

Un punto importante que se debe mencionar es la utilización de conjuntos grandes de tasas de bits que permiten una mejor adaptación por parte de los algoritmos. Sin em-

bargo, esto representa un problema de costos en cuestiones de almacenamiento para los grandes proveedores de servicio de *streaming*. Esto, toda vez que sólo pueden almacenar pequeños conjuntos de tasas de bits, lo que puede llevar a una degradación de la calidad de la experiencia que es perceptible para los usuarios. Por lo anterior, surgen las siguientes preguntas que derivan en futuras líneas de investigación.

- ¿Cómo sería el rendimiento de un sistema de *streaming* codificando todas las representaciones usando H.265? La mayoría de las propuestas de la literatura utilizan conjuntos grandes de representaciones, las cuales están codificadas con H.264, con el propósito de suavizar el intercambio. Sin embargo, no se toman en cuenta los costos que esto representaría en almacenar todo este contenido en servidores. Entonces, H.265 vendría a solucionar este problema.
- ¿Qué beneficios traería almacenar o generar Segmentos con distinta duración? Live *Streaming* a diferencia de Video on Demand o VoD, permite la generación dinámica de segmentos con diferentes duraciones, mientras que en VoD su duración es fija.

Referencias

[1] "Cisco visual networking index: Forecast and trends, 2017–2022 white paper," Feb 2019.

[2] O. Oyman and S. Singh, "Quality of experience for HTTP adaptive streaming services," *IEEE Communications Magazine*, vol. 50, pp. 20–27, Apr. 2012.

[3] I. Sodagar, "The MPEG-DASH standard for multimedia streaming over the internet," *IEEE Multimedia*, vol. 18, pp. 62–67, Apr. 2011.

[4] A. Maxwell, "What is the soap opera effect (and how to make it go away)." [Online]. Available: `https://hometheaterreview.com/what-is-soap-opera-effect-and-how-to-make-it-go-away/`, Jan 2015.

[5] M. Rouse, "What is real-time transport protocol (rtp)?." [Online]. Available: `https://searchnetworking.techtarget.com/definition/Real-Time-Transport-Protocol/`.

[6] D. Schonfeld, "Video communication networks," in *Handbook of Image and Video Processing*, pp. 1031–1064, Elsevier, 2005.

[7] "Real-time messaging protocol (rtmp) specification." [Online]. Available: `https://www.adobe.com/devnet/rtmp.htm`.

[8] D. W. Connolly, "Content-transfer-encoding: packets for http." [Online]. Available: `http://1997.webhistory.org/www.lists/www-talk.1994q3/1147.html/`, Oct 1994.

[9] M. Seufert, S. Egger, M. Slanina, T. Zinner, T. Hobfeld, and P. Tran-Gia, "A survey on quality of experience of HTTP adaptive streaming," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 1, pp. 469–492, 2015.

[10] "Iso/iec 23009-1:2019." [Online]. Available: `https://www.iso.org/standard/75485.html/`, Aug 2019.

[11] "Uniform resource identifier (uri): Generic syntax." [Online]. Available: `https://tools.ietf.org/html/rfc3986/`.

[12] "Hypertext transfer protocol – http/1.1." [Online]. Available: `https://tools.ietf.org/html/rfc2616/`.

[13] Jean, Alexander, Alberto, J. Carlos, N. Bealey, Honey, Y. Cao, Artsi, Ahmed, Sumit, and et al., "Dash basics: Mpd and segments." [Online]. Available: `https://gpac.wp.imt.fr/2012/02/01/dash-support/`, Feb 2012.

[14] A. Bentaleb, B. Taani, A. C. Begen, C. Timmerer, and R. Zimmermann, "A survey on bitrate adaptation schemes for streaming media over HTTP," *IEEE Communications Surveys and Tutorials*, vol. 21, no. 1, pp. 562–585, 2019.

[15] "Mpeg dash requirements for a webpush protocol." [Online]. Available: `https://tools.ietf.org/html/draft-begen-webpush-dash-reqs-00`, Oct 2014.

[16] T. Stockhammer, "Dynamic adaptive streaming over (http)," in *Proceedings of the second annual (ACM) conference on Multimedia systems*, ACM Press, 2011.

[17] C. Liu, I. Bouazizi, and M. Gabbouj, "Rate adaptation for adaptive HTTP streaming," in *Proceedings of the second annual ACM conference on Multimedia systems*, ACM Press, 2011.

[18] Z. Li, X. Zhu, J. Gahm, R. Pan, H. Hu, A. C. Begen, and D. Oran, "Probe and adapt: Rate adaptation for HTTP video streaming at scale," *IEEE Journal on Selected Areas in Communications*, vol. 32, pp. 719–733, Apr. 2014.

[19] T.-Y. Huang, R. Johari, N. McKeown, M. Trunnell, and M. Watson, "A buffer-based approach to rate adaptation," in *Proceedings of the 2014 (ACM) conference on (SIGCOMM) - (SIGCOMM)*, ACM Press, 2014.

[20] K. Spiteri, R. Urgaonkar, and R. K. Sitaraman, "BOLA: Near-optimal bitrate adaptation for online videos," in *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*, IEEE, Apr. 2016.

[21] "Dash industry forum: Catalyzing the adoption of mpeg." [Online]. Available: `https://dashif.org/`.

[22] L. D. Cicco, V. Caldaralo, V. Palmisano, and S. Mascolo, "ELASTIC: A client-side controller for dynamic adaptive streaming over HTTP (DASH)," in *2013 20th International Packet Video Workshop*, IEEE, Dec. 2013.

[23] P. Juluri, V. Tamarapalli, and D. Medhi, "SARA: Segment aware rate adaptation algorithm for dynamic adaptive streaming over HTTP," in *2015 IEEE International Conference on Communication Workshop (ICCW)*, IEEE, June 2015.

[24] A. Bentaleb, A. C. Begen, S. Harous, and R. Zimmermann, "Want to play DASH?," in *Proceedings of the 9th ACM Multimedia Systems Conference on - (MMSys)*, ACM Press, 2018.

[25] R. Olfati-Saber, J. A. Fax, and R. M. Murray, "Consensus and cooperation in networked multi-agent systems," *Proceedings of the IEEE*, vol. 95, pp. 215–233, Jan. 2007.

[26] S. Lasaulce and H. Tembine, *Game Theory and Learning for Wireless Networks: Fundamentals and Applications*. Academic Press, sep 2011.

[27] Z. Han, D. Niyato, W. Saad, T. Başar, and A. Hjrungnes, *Game Theory in Wireless and Communication Networks: Theory, Models, and Applications*. New York, NY, USA: Cambridge University Press, 1st ed., 2012.

[28] "Media source api." [Online]. Available: `https://developer.mozilla.org/es/docs/Web/API/MediaSource`.

[29] K. S. Dash-Industry-Forum, "Abr logic." [Online]. Available: `https://github.com/Dash-Industry-Forum/dash.js/wiki/ABR-Logic/`, Nov 2018.

[30] M. Carbone and L. Rizzo, "Dummynet revisited," *SIGCOMM Comput. Commun. Rev.*, vol. 40, pp. 12–20, Apr. 2010.

[31] "ffmpeg documentation." [Online]. Available: `https://www.ffmpeg.org/ffmpeg.html#Description`.

[32] "Mp4box." [Online]. Available: `https://gpac.wp.imt.fr/mp4box/`.

[33] B. Institute, "Big buck bunny." [Online]. Available: `https://peach.blender.org/`, Aug 2008.

[34] "Mpeg-dash and hls segment length for adaptive streaming." [Online]. Available: `https://bitmovin.com/mpeg-dash-hls-segment-length/`, Apr 2015.

Casa abierta al tiempo
UNIVERSIDAD AUTÓNOMA METROPOLITANA

ACTA DE EXAMEN DE GRADO

No. 00081
Matrícula: 2173802216

Transmisión Adaptativa de Vídeo en Redes de Computadoras con Propiedades Dinámicas.

En la Ciudad de México, se presentaron a las 12:30 horas del día 19 del mes de noviembre del año 2019 en la Unidad Iztapalapa de la Universidad Autónoma Metropolitana, los suscritos miembros del jurado:

DR. ENRIQUE STEVENS NAVARRO
DR. VICTOR MANUEL RAMOS RAMOS
DRA. GRACIELA ROMAN ALONSO

Bajo la Presidencia del primero y con carácter de Secretaria la última, se reunieron para proceder al Examen de Grado cuya denominación aparece al margen, para la obtención del grado de:

MAESTRO EN CIENCIAS (CIENCIAS Y TECNOLOGIAS DE LA INFORMACION)

DE: ERIK MIGUEL DIAZ SALAZAR

y de acuerdo con el artículo 78 fracción III del Reglamento de Estudios Superiores de la Universidad Autónoma Metropolitana, los miembros del jurado resolvieron:

Aprobar

Acto continuo, el presidente del jurado comunicó al interesado el resultado de la evaluación y, en caso aprobatorio, le fue tomada la protesta.

ERIK MIGUEL DIAZ SALAZAR
ALUMNO

REVISÓ

MTRA. ROSALIA SERRANO DE LA PAZ
DIRECTORA DE SISTEMAS ESCOLARES

DIRECTOR DE LA DIVISIÓN DE CBI

DR. JESUS ALBERTO OCHOA TAPIA

PRESIDENTE

DR. ENRIQUE STEVENS NAVARRO

VOCAL

DR. VICTOR MANUEL RAMOS RAMOS

SECRETARIA

DRA. GRACIELA ROMAN ALONSO