UNIVERSITAT POLITÈCNICA DE VALÈNCIA

_ TELECOM ESCUELA TÉCNICA VLC SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

DESARROLLO DE UN SISTEMA DE EVALUACIÓN DEL SERVICIO DE STREAMING DASH DE BAJA LATENCIA

Antonio Diyanov Nikolov

Tutor: Juan Carlos Guerri

Cotutor: Pau Arce

Trabajo Fin de Grado presentado en la Escuela Técnica Superior de Ingenieros de Telecomunicación de la Universitat Politècnica de València, para la obtención del Título de Graduado en Ingeniería de Tecnologías y Servicios de Telecomunicación

Curso 2019-20

Valencia, 5 de julio de 2020

Escuela Técnica Superior de Ingeniería de Telecomunicación
Universitat Politècnica de València
Edificio 4D. Camino de Vera, s/n, 46022 Valencia
Tel. +34 96 387 71 90, ext. 77190
www.etsit.upv.es

VLC/ CAMPUS
VALENCIA, INTERNATIONAL CAMPUS OF EXCELLENCE

EMAS
Gestión ambiental verificada
REG.NO. ES-CV- 000030

Resumen

En este proyecto se ha desarrollado un sistema prototipo para la evaluación del servicio de streaming utilizando el protocolo DASH y configurando la generación de contenidos para por un lado reducir la latencia y por otro mejorar la calidad de experiencia del usuario (QoE). Se han utilizado las herramientas del navegador Chrome de manera automatizada para modificar el ancho de banda disponible y evaluar la respuesta del sistema. Los diferentes escenarios de cambios de ancho de banda permiten evaluar el comportamiento del sistema en diferentes entornos. Para llevar a cabo estas simulaciones se ha utilizado Puppeteer, la biblioteca *Node.js* desarrollada por Google que proporciona una API de alto nivel para automatizar acciones en Chrome a través del protocolo Devtools, como pueden ser iniciar la reproducción de los videos, cambiar el ancho de banda disponible o guardar los resultados, como por ejemplo, de los cambios de calidad, tamaño del buffer y paradas. El sistema ha servido para analizar el efecto del tamaño del segmento en los sistemas de streaming DASH.

Palabras clave: QoE; DASH; Latencia; codificación de vídeo; emulación de red

Resum

En aquest projecte s'ha desenvolupat un sistema prototip per a l'avaluació de l'servei de transmissió utilitzant el protocol DASH i configurant la generació de continguts per una banda reduir la latència i per l'altra millorar la qualitat de l'experiència de l'usuari (QoE). S'han utilitzat les eines de el navegador Chrome de manera automatitzada per modificar l'ample de banda disponible i avaluar la resposta de el sistema. Els diferents escenaris de canvis d'ample de banda permeten avaluar el comportament de el sistema en diferents entorns. Per dur a terme aquestes simulacions s'ha utilitzat Puppeteer, la biblioteca *Node.js* desenvolupada per Google que proporciona una API d'alt nivell per automatitzar accions a Chrome a través del protocol Devtools Library, com poden ser iniciar la reproducció dels vídeos, canviar l'ample de banda disponible o guardar els resultats, com per exemple, dels canvis de qualitat, mida del la memòria intermèdia i parades. El sistema ha servit per analitzar l'efecte de la mida de l'segment en els sistemes de transmissió DASH.

Paraules clau: QoE; DASH; Latència; codificació de vídeo; emulació de xarxa.

Abstract

In this project, a prototype system has been developed for the evaluation of the transmission service using the DASH protocol and configuring content generation to reduce latency on the one hand and improve the quality of the user experience (QoE) on the other. The Chrome browser tools have been used in an automated way to modify the available bandwidth and evaluate the system response. Different scenarios of bandwidth changes allow evaluating the behavior of the system in different environments. Puppeteer, the *Node.js* library developed by Google that provides a high-level API to automate actions in Chrome through the Devtools protocol, such as starting the playback of videos, changing the width, has been used to carry out these simulations. available bandwidth or save the results, such as quality changes, buffer size and stops. The system has served to analyze the effect of segment size on DASH transmission systems.

Key words: QoE; DASH; Latency; video encoding; network emulation.

Índice

Capítulo 1. Introducción y objetivos

Durante los últimos años y en especial durante esta época de confinamiento que estamos viviendo, el tráfico de vídeo en internet ha aumentado considerablemente.

Debido a que el video es la fuente de datos que más impacto tiene en la congestión de la red a nivel global algunas redes de distribución de contenidos o CDN, como por ejemplo Netflix han tenido que reducir la calidad de sus contenidos para de esta manera generar un menor impacto en la red.

En este escenario de creciente tráfico en internet conviene destacar de qué tipos puede ser este tráfico:

- Streaming de video en directo: Utilizado mayoritariamente en transmisiones de eventos en directo, TV por internet, radio por internet, etc…
- Streaming de video bajo demanda: Utilizado en difusión de contenidos almacenados como en por ejemplo en Youtube, Netflix, HBO, Amazon Prime Video, etc…
- Aplicaciones interactivas: Utilizado en comunicaciones peer-to-peer (Videoconferencia WebRTC, Skype, llamadas de Whatsapp, TEAMS, Zoom, etc…).

En este trabajo nos centraremos en HTTP Streaming, escenario que puede ser utilizado para la difusión de contenidos o bien bajo demanda o bien en directo pero con un buffer elevado. Estos sistemas son también llamados *Pull-Based* debido a que es el usuario quien solicita información para descargar los contenidos.

En HTTP Streaming se utiliza un sistema adaptativo en el que es el cliente quien estima cuál es la congestión de la red. Para ello, se examinan parámetros como pueden ser la tasa de pérdidas, el retardo entre los extremos y la ocupación del buffer. A partir de esta información y sin ningún tipo de realimentación explícita de la red el *player* se adapta utilizando diferentes calidades en función del estado de la red.

HTTP Streaming es una solución que no soporta multicast debido a que HTTP utiliza TCP y TCP, por naturaleza soporta conexiones unicast. Por tanto, en aquellos escenarios en los que se desea transmitir el mismo contenido a un número elevado de usuarios mediante un flujo multicast, HTTP Streaming no es la solución adecuada. HTTP Streaming es adecuado cuando el contenido compartido es unicast.

En el entorno de HTTP Streaming nos centraremos en el protocolo MPEG-DASH, que es un protocolo de streaming adaptativo en el que el usuario demanda segmentos de video de diferentes calidades en función del ancho de banda disponible y del tamaño del buffer del player.

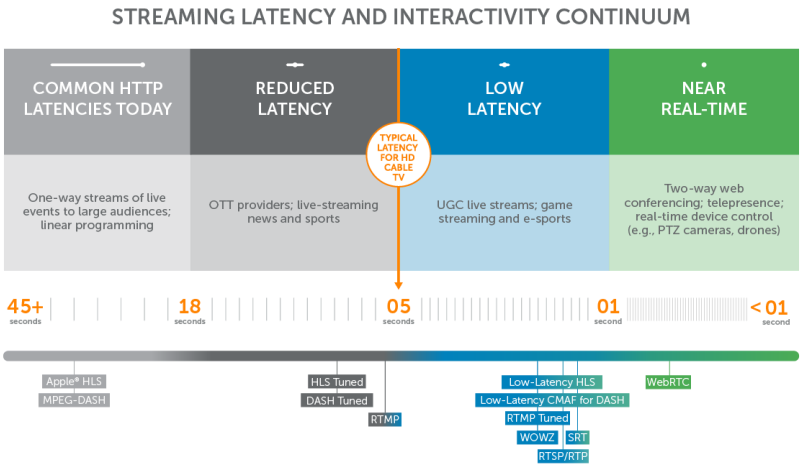


Figura 1.1: Protocolos de transmisiones en directo utilizados según la latencia.

En la Fig. 1.1 podemos observar que con la tecnología DASH la latencia o tiempo que pasa desde que se genera el contenido hasta que comienza su reproducción en el cliente se encuentra entre 18 y 45 segundos. Esta latencia es elevada con lo que no suele ser adecuado utilizar DASH para reproducir eventos en directo. Este es el motivo por el que DASH suele ser ideal para reproducir contenidos como en el caso de Youtube o Netflix debido a que los videos están ya almacenados y codificados con diferentes calidades, y el usuario elige la calidad adecuada en función de las condiciones de la red.

Si quisiéramos utilizar DASH para reproducir eventos en directo también sería posible mediante la tecnología de DASH de baja latencia (*Low-Latency Dash*) en la que configurando la generación de contenidos se puede llegar a reducir esta entorno a los 5 segundos.

Debido a que DASH trocea el video original en segmentos en el presente trabajo estudiaremos cuál es el tamaño de segmento adecuado para por un lado reducir la latencia y por otro mejorar la calidad de experiencia del usuario (QoE).

Para ello se plantearán diferentes escenarios de cambios de ancho de banda (periódicos, escalonados, etc…) en los que se comprobará la respuesta del sistema.

Los objetivos del presente trabajo por tanto son:

- Automatizar el proceso de realización de pruebas.
- Disponer de un sistema flexible y sencillo que abarca desde la codificación de contenidos hasta la extracción de parámetros de interés.
- Análisis de las prestaciones de DASH con diferentes tamaños de segmentos.

Capítulo 2. Conceptos de DASH

2.1 Concepto de Streaming Adaptativo

Las tecnología de streaming adaptativo garantiza una mejora en la calidad de experiencia del usuario ya que permite adaptar la calidad del video monitorizando la conexión. Por otro lado, tiene la ventaja de poder ser utilizada en una gran variedad de dispositivos.

Las tecnologías de streaming adaptativo comparten:

- Utilización de múltiples archivos con diferentes calidades a partir de una misma fuente (video original).
- Cambios de los segmentos descargados a lo largo del tiempo en función del estado de la red.
- Operaciones transparentes al usuario.

2.2 Estándar DASH

Dynamic Adaptive Streaming over HTTP (DASH), también conocido como MPEG-DASH [2] es un estándar ISO para la transmisión de contenido en vivo y bajo demanda que permite reproducir contenidos multimedia utilizando servidores web HTTP convencionales.

Se trata de un sistema de streaming adaptativo en el que los videos son almacenados en servidores y el usuario accede a estos para descargarlos y reproducirlos. En el servidor, existen varias copias del mismo video codificado en diferentes calidades o bitrates.

Los datos son enviados al cliente como un flujo continuo de trozos o segmentos de tamaño variable (2, 5, 10 segundos, etc…). Es fácil garantizar el *trick-play* o cambios de instantes de reproducción en el video puesto que esto implica demandar los segmentos adecuados de dicho instante de reproducción.

DASH permite al cliente seleccionar una calidad u otra en función del estado de congestión de la red. Utilizar segmentos de diferentes calidades permite que el buffer del cliente sea pequeño y optimizar la calidad de experiencia cuando la red está congestionada.

Antes de comenzar la reproducción el cliente descarga el MANIFEST o MPD (*Media Presentation Description*) que es un archivo que contiene la información necesaria para poder descargar y reproducir el video correctamente. El MPD contiene las URL de los diferentes segmentos y calidades del video, audio, tamaño de los segmentos, etc...

Una de las ventajas de utilizar DASH es su fácil adaptación en entornos con firewalls debido a que utiliza un servidor web convencional y los puertos 80 utilizados para la navegación web están siempre disponibles.

2.3 Funcionamiento DASH

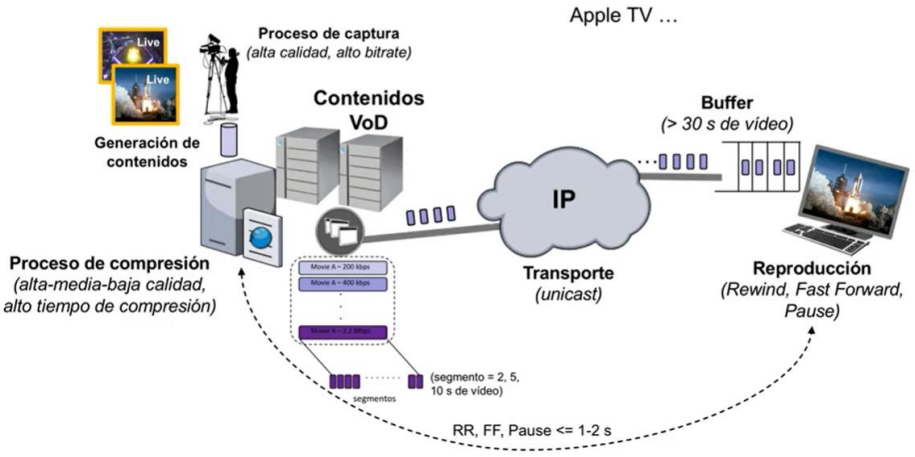


Figura 2.1: Escenario de utilización MPEG-DASH.

En la Fig. 2.1 podemos observar en el lado del servidor como el contenido está comprimido en diferentes calidades [3] (Por ejemplo: alta, media, baja). La codificación de los contenidos será muy eficiente en un escenario de video bajo demanda (VoD) debido a que se dispone de tiempo ilimitado para realizar dicha tarea. Una vez codificados a diferentes calidades o bitrates los contenidos a su vez serán segmentados con tamaños de segmento que suelen tener una duración del orden de varios segundos. El cliente solicita segmentos al servidor y estos son transmitidos mediante una conexión unicast. A medida que recibe segmentos los almacena en un buffer. El tamaño del buffer de almacenamiento depende de la configuración del player aunque suele ser del orden de 30 segundos. Al comenzar la reproducción el cliente podrá pausar el video, cambiar de instante de reproducción, etc… Durante la reproducción del video, de manera transparente al usuario, el player DASH solicita segmentos de la calidad que pueda garantizar la reproducción del video sin interrupciones.

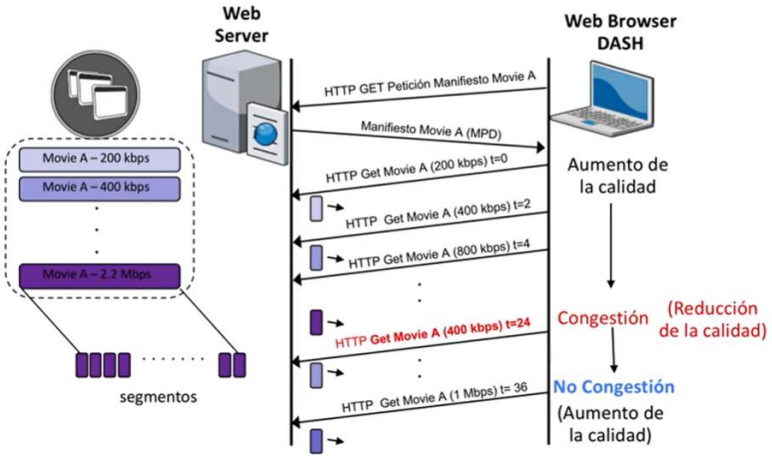


Figura 2.2: Codificación multi-bitrate y adaptación.

En la Fig. 2.2 se puede observar en el lado del servidor cómo el mismo video (Movie A) está codificado en diferentes calidades y cada una de estas calidades a su vez está troceada en segmentos que todos son de la misma duración.

El cliente es un navegador con un player que soporta DASH. Las peticiones que realiza el cliente son las peticiones GET típicas utilizadas en el protocolo HTTP. En estas peticiones el cliente demanda al servidor un recurso utilizando una URL concreta. El primer GET solicita el MPD, archivo que proporciona toda la información necesaria al cliente para acceder de forma correcta al contenido en el servidor web y llevar a cabo la adaptación de calidad. Una vez recibido el MPD, comienza a solicitar segmentos de calidad cada vez más alta en el caso de que no haya congestión. Si se produce la detección de congestión debido a que ha disminuido el ancho de banda disponible los segmentos que se piden a continuación son de menor calidad hasta que se pueda garantizar que en el cliente no haya interrupciones.

2.4 Estructura MPD

El MANIFEST contiene información para obtener los segmentos y realizar a cabo la adaptación de calidad.

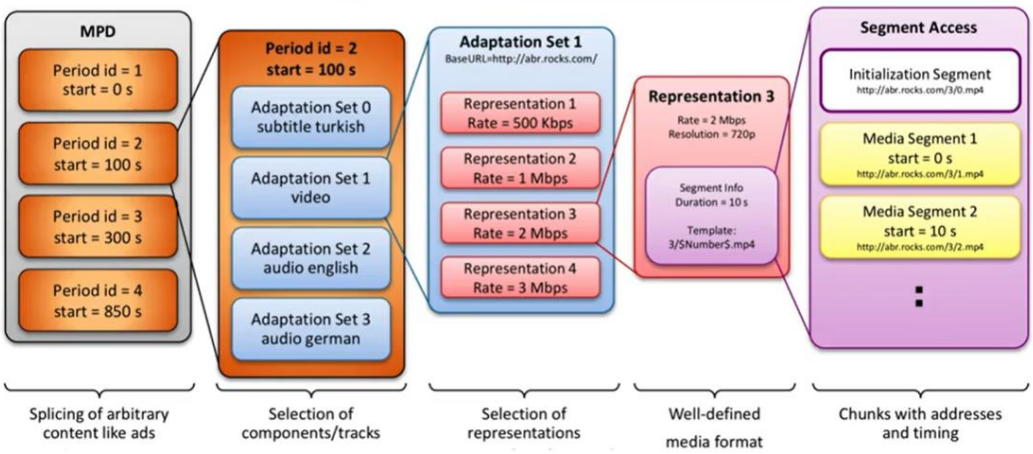


Figura 2.3: Estructura mpd.

En la Fig. 2.3 podemos observar la estructura del MPD. En primer lugar podemos dividir la estructura del contenido multimedia en diferentes periodos. Cada periodo contiene diferentes componentes o tracks, en este caso un track de subtítulos, uno de video y dos de audio. Centrándonos en el video podemos observar que existen diferentes representaciones o calidades. Es decir, el mismo video está codificado a 4 calidades diferentes. Cada una de las representaciones tiene la información del bitrate de codificación y del tamaño del segmento elegido. Todas las representaciones tienen la misma duración de segmento debido a que al cambiar de calidad la reproducción debe continuar al cambiar de un segmento a otro. Por último para acceder a cada segmento el MPD nos proporciona la URL y el tiempo de inicio y fin de cada uno de ellos.

```
<MPD xmlns="urn:mpeg:dash:schema:mpd:2011" minBufferTime="PT1.500S" type="static" mediaPresentationDuration="PT0H3M0.000S" maxSegmentDuration="PT0H0M10.008S"
  <ProgramInformation moreInformationURL="http://gpac.io">
    <Title>video.mpd generated by GPAC</Title>
  </ProgramInformation>
  <Period duration="PT0H3M0.000S">
    <AdaptationSet segmentAlignment="true" bitstreamSwitching="true" maxWidth="2592" maxHeight="1080" maxFrameRate="24" par="2592:1080" lang="eng">
      <SegmentList>
        <Initialization sourceURL="video_set1_init.mp4"/>
      </SegmentList>
      <Representation id="1" mimeType="video/mp4" codecs="avc3.640032" width="2592" height="1080" frameRate="24" sar="1:1" bandwidth="1729473">
        <BaseURL>video-1080_dash.mp4</BaseURL>
        <SegmentList timescale="12288" duration="122880">
          ...
        </SegmentList>
      </Representation>
      <Representation id="2" mimeType="video/mp4" codecs="avc3.640020" width="1728" height="720" frameRate="24" sar="1:1" bandwidth="863160">
        ...
      </Representation>
      <Representation id="3" mimeType="video/mp4" codecs="avc3.64001E" width="864" height="360" frameRate="24" sar="1:1" bandwidth="444795">
        ...
      </Representation>
    </AdaptationSet>
    <AdaptationSet segmentAlignment="true" lang="eng">
      ...
    </AdaptationSet>
  </Period>
</MPD>
```

Figura 2.4: Ejemplo de mpd.

En la Fig. 2.4 podemos observar el contenido de un fichero MPD. Algunos campos relevantes son:

- `maxSegmentDuration`: Indica el tamaño de los segmentos que en este caso es 10 segundos.
- `Width` y `Height`: Indica la resolución. Por ejemplo en la calidad alta es 2592x1080.
- `frameRate`: Indica número de imágenes por segundo: 24 en este caso.
- `bandwidth`: Indica el bitrate de la calidad seleccionada. La calidad más alta tiene un bitrate de 1729 kbps mientras que la más baja tiene un bitrate de 444 kbps.

```
<SegmentList timescale="12288" duration="122880">
  <SegmentURL mediaRange="912-379005" indexRange="912-955"/>
  <SegmentURL mediaRange="379006-2691912" indexRange="379006-379049"/>
  <SegmentURL mediaRange="2691913-4950119" indexRange="2691913-2691956"/>
  <SegmentURL mediaRange="4950120-7288799" indexRange="4950120-4950163"/>
  <SegmentURL mediaRange="7288800-9694163" indexRange="7288800-7288843"/>
  <SegmentURL mediaRange="9694164-12005463" indexRange="9694164-9694207"/>
  <SegmentURL mediaRange="12005464-13292564" indexRange="12005464-12005507"/>
  <SegmentURL mediaRange="13292565-15654890" indexRange="13292565-13292608"/>
  <SegmentURL mediaRange="15654891-18023832" indexRange="15654891-15654934"/>
  <SegmentURL mediaRange="18023833-20296944" indexRange="18023833-18023876"/>
  <SegmentURL mediaRange="20296945-22719247" indexRange="20296945-20296988"/>
  <SegmentURL mediaRange="22719248-25121673" indexRange="22719248-22719291"/>
  <SegmentURL mediaRange="25121674-27544677" indexRange="25121674-25121717"/>
  <SegmentURL mediaRange="27544678-29743644" indexRange="27544678-27544721"/>
  <SegmentURL mediaRange="29743645-32062408" indexRange="29743645-29743688"/>
  <SegmentURL mediaRange="32062409-34316421" indexRange="32062409-32062452"/>
  <SegmentURL mediaRange="34316422-36692869" indexRange="34316422-34316465"/>
  <SegmentURL mediaRange="36692870-38975559" indexRange="36692870-36692913"/>
</SegmentList>
```

Figura 2.5: Segmentlist de mpd.

En la Fig. 2.5 se observa el contenido del campo `SegmentList` del mpd. Este campo contiene la información necesaria para acceder a los segmentos y su correcta reproducción.

Capítulo 3. Descripción del sistema de pruebas

3.1 Sistema de pruebas

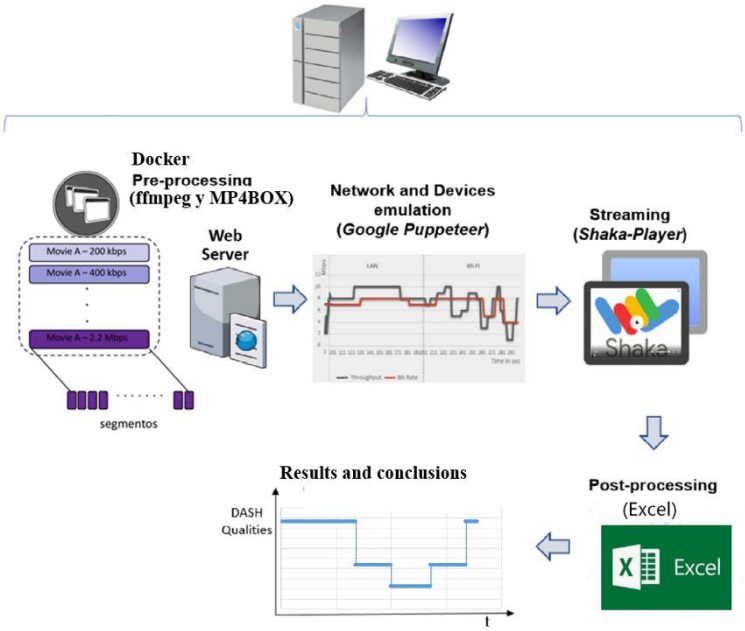


Figura 3.1: Sistema de pruebas.

En la Fig. 3.1 se puede observar el sistema de pruebas utilizado en el proyecto. En primer lugar mediante el uso del framework *ffmpeg* se extrae el audio del video original y se codifica el video en 3 calidades diferentes (alta, media y baja).

A continuación mediante el uso de la herramienta MP4Box se generan los videos para streaming junto con el MPD.

Una vez preparados los videos se instala en un servidor web de Apache el reproductor Shaka Player de manera que accediendo a la URL del Shaka Player y configurando la dirección del MPD el video es reproducido utilizando el estándar DASH.

Para realizar las simulaciones de cambios de anchos de banda se utiliza JavaScript con la librería de Puppeteer, ya que proporciona métodos para controlar Chrome automáticamente y recolectar información de los cambios de anchos de banda, interrupciones del video, tamaño del buffer en cada instante, etc…

Por último, una vez obtenidos los datos con diferentes escenarios de cambios de anchos de banda, estos datos son procesados con Excel y obtenidas conclusiones. Como entorno de programación en JavaScript se ha optado por utilizar VisualStudio.

Las herramientas de *ffmpeg* , MP4Box y el servidor web son alojados en contenedores Docker para de esta manera facilitar el despliegue del proyecto independientemente del sistema operativo utilizado.

En los siguientes puntos se profundizará en cada una de las herramientas mencionadas anteriormente.

3.2 Despliegue Contenedor Docker

Docker es un proyecto de código abierto que automatiza el despliegue de aplicaciones dentro de contenedores de software, proporcionando una capa adicional de abstracción y automatización de virtualización de aplicaciones en múltiples sistemas operativos. [5]

En este proyecto se han utilizado dos contenedores docker: por un lado un contenedor Docker que contiene las herramientas *ffmpeg* y MP4Box y por otro lado un contenedor Docker que aloja un servidor web Apache.

El despliegue del proyecto se ha realizado utilizando los contenedores Docker empleados en las prácticas de la asignatura Comunicaciones Multimedia, ETSIT.

Los pasos a seguir para realizar el despliegue del proyecto en Windows se describen a continuación.

3.2.1 Iniciación del directorio de trabajo

- Crear una carpeta de nombre TFG_DASH. Dicha carpeta contiene el *player* DASH y el vídeo utilizado.
- Descomprimir el fichero que contiene el *player* shaka-player.

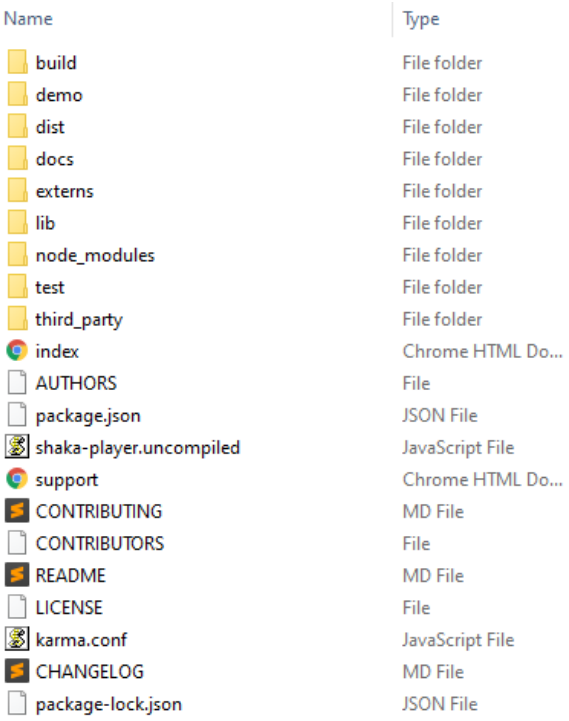


Figura 3.2: Contenido carpeta shakaplayer.

En la Fig. 3.2 se observa el contenido de la carpeta shakaplayer

3.2.2 Iniciación de los Dockers

- Abrimos la consola mediante la utilidad cmd y nos dirigimos a la carpeta raíz TFG_DASH:.

  C:\> `cd C:\Desktop\TFG_DASH`

- Realizamos la autenticación para acceder al repositorio de Gitlab donde están los contenedores Docker.
  >> `docker login registry.gitlab.com`

  Nombre: `gitlab+deploy-token-37225`

  Contraseña: `Hai5BFrAz3ZjTMzTvJ9k`

- Arrancamos el contendor Docker que contiene las herramientas *ffmpeg* y MP4Box.

  C:\> `docker run --name=client --rm -ti -v %cd%:/workdir registry.gitlab.com/comm/etsit-comunicaciones-multimedia-video-perdidas:v2 bash`

- Abrimos otra consola cmd y accedemos al directorio de trabajo.

  C:\> `cd C:\temp\TFG_DASH`

- Arrancamos el contenedor Docker que contiene el servidor web Apache.

  C:\> `docker run -dit --name my-apache-app --rm -p 8081:80 -v %cd%:/usr/local/apache2/htdocs/ httpd:2.4`

De esta manera el proyecto queda listo para la utilización de las herramientas de *ffmpeg*, MP4Box y el servidor web que aloja el Shaka Player.

3.3 Software para la codificación (ffmpeg)

3.3.1 Definición FFMPEG

Ffmpeg [6] es una colección de software libre que puede grabar, convertir (codificar) y hacer streaming de audio y vídeo. Incluye *libavcodec*, una biblioteca de códecs. *Ffmpeg* está desarrollado en GNU/Linux, pero puede ser compilado en la mayoría de los sistemas operativos, incluyendo Windows.

Incluye 3 herramientas:

- `ffmpeg` es la herramienta principal y más utilizada de todas. Incluye las herramientas de conversión y tratamiento del video.
- `ffplay` es un reproductor de multimedia básico.
- `ffprobe` es un analizador de contenido multimedia que muestra información técnica util.

Ffmpeg es una pieza clave en este proyecto puesto que permite convertir el video original y codificarlo en diferentes calidades.

3.3.2 Comandos utilizados

Extraemos el audio y codificamos el vídeo original en 3 calidades [7]. Para la extracción del audio utilizamos el siguiente comando:

```
# ffmpeg -i tos-1080p.mp4 -c:a copy -vn video-audio.mp4
```

- `-i` indica la input utilizado, en nuestro caso el video tos-1080p.mp4
- `-c:a copy` indica que el códec de audio utilizado es `copy` con lo que el audio se copiará sin que sean realizadas operaciones de codificación, filtrado o decodificación.
- `-vn` deshabilita la grabación de video, es decir, la selección o mapeo automático de cualquier transmisión de video.

Para codificar el video a diferentes calidades utilizamos los siguientes comandos:

- Calidad 1**:**

```
# ffmpeg -i tos-1080p.mp4 -an -c:v libx264 -x264opts 'keyint=24:min-keyint=24:no-scenecut' -b:v 2000k -maxrate 2000k -bufsize 1000k -vf 'scale=-1:1080' video-1080.mp4
```

- Calidad 2**:**

```
# ffmpeg -i tos-1080p.mp4 -an -c:v libx264 -x264opts 'keyint=24:min-keyint=24:no-scenecut' -b:v 1000k -maxrate 1000k -bufsize 500k -vf 'scale=-1:720' video-720.mp4
```

- Calidad 3**:**

```
# ffmpeg -i tos-1080p.mp4 -an -c:v libx264 -x264opts 'keyint=24:min-keyint=24:no-scenecut' -b:v 512k -maxrate 512k -bufsize 256k -vf 'scale=-1:360' video-360.mp4
```

- `-i` indica la input utilizado, en nuestro caso el video tos-1080p.mp4
- `-an` desactiva la grabación del audio
- `-c:v libx264` indica que el códec utilizado es H264.
- `-x264opts` permite modificar opciones de la codificación del video
- `keyint` especifica la longitud máxima del GoP (Group of Pictures), es decir, la cantidad de imágenes entre tramas I (intras). En nuestro caso es 24 frames.
- `min-keyint=24` especifica la cantidad mínima del GoP.
- `no-scenecut` bloquea la introducción automática de intras.
- `-b:v [value]` indica el bitrate del video resultante.
- `-maxrate` indica el bitrate máximo del video resultante.
- `bufsize [value]` especifica el tamaño del buffer del decodificador, que determina la variabilidad de la tasa de bits de salida.
- `-vf filtergraph` crea un filtro que reduce la resolución a la indicada.

3.4 Software para la segmentación DASH (MP4Box)

MP4Box [8] es un conversor MPEG-4 utilizado para segmentar el video original y crear el MPD.

Generamos los videos para streaming y el MPD mediante el siguiente comando:

```
# MP4Box -dash 10000 -out video.mpd video-1080.mp4 video-720.mp4
video-360.mp4 video-audio.mp4
```

- `-dash [value]` crea los archivos DASH a partir de los videos en diferentes calidades, el audio y el tamaño de los segmento especificado. Genera también el MPD [9].

En este proyecto se han utilizado tamaños de segmento de 10, 5, 2, 1 y 0.5 segundos con el objetivo de analizar la respuesta del player.

3.5 Servidor Web (Apache)

Apache HTTP Server es un software de servidor web gratuito y de código abierto para plataformas Unix. En nuestro caso es utilizado para alojar el Shaka Player. Para acceder al servidor nos dirigimos desde el navegador a la dirección localhost:8081 .

3.6 Software para la realización de pruebas (Puppeteer)

3.6.1 Introducción Puppeteer

Puppeteer [8] se utilizará como herramienta para la automatización de las pruebas de cambio del ancho de banda de la red. La librería "Puppeteer" ha sido lanzada por Google. Ofrece una interfaz basada en *node.js* que permite ejecutar y controlar Chrome (o Chromium) en modo Headless a través del protocolo DevTools ejecutando un script desde la línea de comandos. El modo Headless significa que no se observa la pestaña del navegador mientras se realizan las operaciones demandadas. En el presente proyecto el modo Headless está desactivado.

La mayoría de las tareas que se pueden realizar en el navegador pueden ser automatizadas utilizando Puppeteer. Algunos ejemplos de tareas que puede realizar Puppeteer son:

- Hacer capturas de pantalla y generar archivos PDF de páginas.
- Automatizar el envío de formularios, manejo de interfaces de usuario web, manejo de pulsaciones de teclado, etc...
- Crear un entorno de pruebas actualizado y automatizado.
- Ejecutar pruebas directamente sobre la última versión de Chrome utilizando funciones de JavaScript.
- Capturar parámetros de interés del funcionamiento en tiempo real de páginas web.

En la Fig. 3.3 muestra una captura de lo que el usuario vería al ejecutar el Script que arranca el Chrome con Puppeteer. En la esquina superior izquierda de la pantalla se muestra un mensaje que indica que un software de prueba está controlando Chrome.

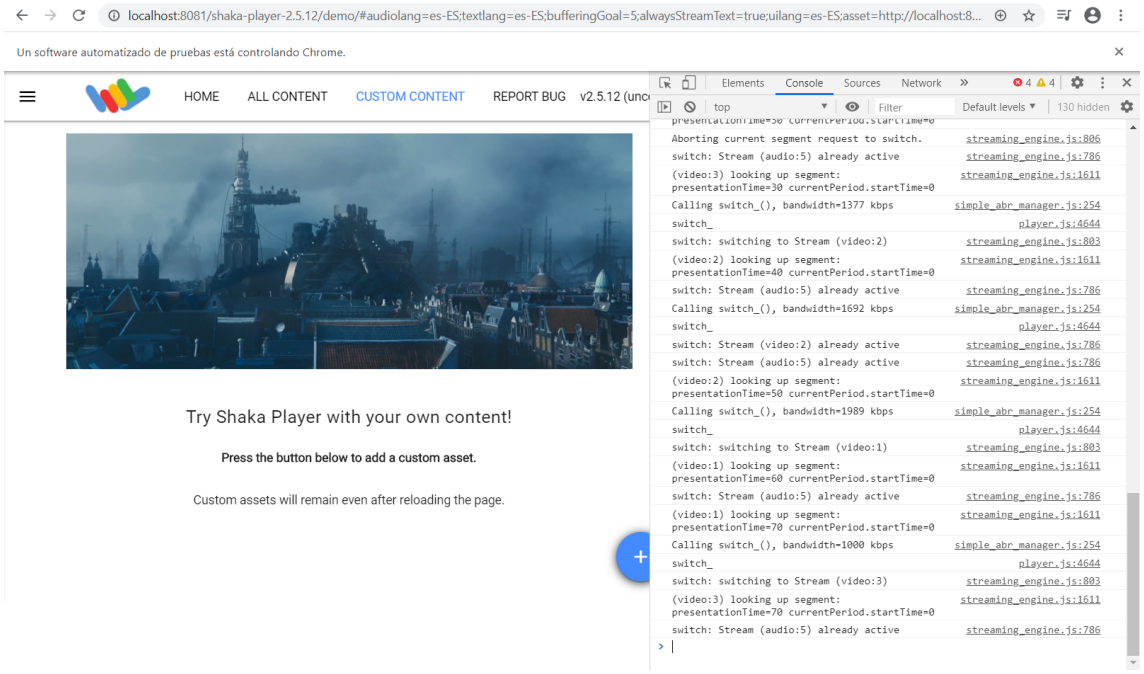


Figura 3.3. Chrome controlado por Puppeteer, modo headless desactivado

3.6.2 Diagrama de proceso del programa

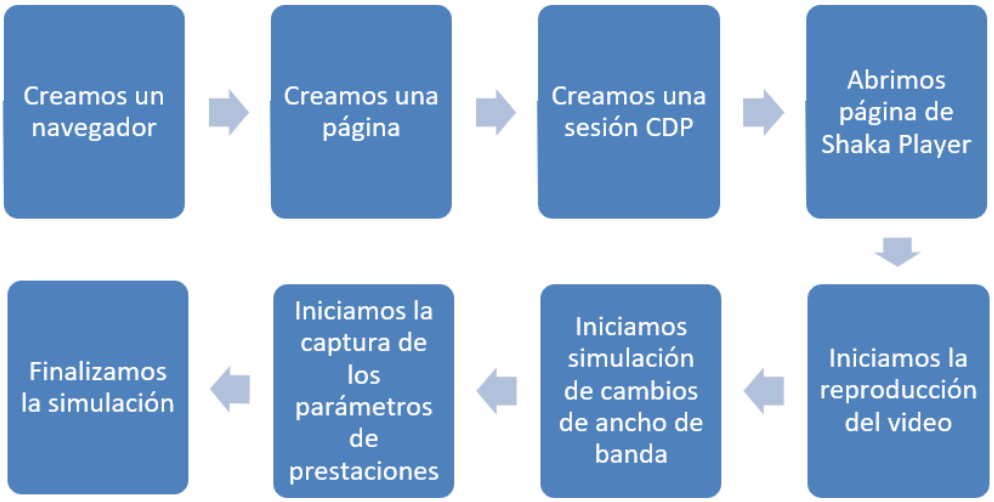


Figura 3.4: Diagrama de proceso del programa.

En la Fig 3.4 podemos observar un diagrama de proceso del programa implementado. Cada cuadrado representa la acción llevada a cabo de manera automatizada sobre el navegador Chrome. En el siguiente punto analizaremos con más detalle las acciones llevadas a cabo en cada proceso.

3.6.3 Implementación de Puppeteer

Para utilizar Puppeteer en primer lugar debemos instalar la librería correspondiente:

```
# npm i puppeteer
```

Una vez instalado, el primer paso es crear un archivo de JavaScript e importar la librería Puppeteer:

```
var puppeteer = require('puppeteer');
```

Definimos una función asíncrona que incluye todo el código que será ejecutado:

```
async function run(){
      //Código a ejecutar
}
run();
```

Dentro de la función ejecutamos en primer lugar:

```
var options = {headless : false, executablePath: process.argv[2], devtools: true};
var browser = await puppeteer.launch(options);
```

De esta manera creamos una variable de tipo browser que contiene las opciones del Puppeteer:

- `headless : false` Indica que el modo headless está desactivado con lo que se mostrará el navegador al usuario. Si estuviese activado el usuario no vería nada mientras se ejecuta el programa.
- `executablePath: process.argv[2]` Indica el Path o directorio del navegador Chrome. En este caso indicamos `process.argv[2]` puesto que dicho directorio será pasado como parámetro al ejecutar el archivo JavaScript.
- `devtools: true` Implica que al arrancar el Chrome las herramientas de desarrollador se abren automáticamente.

La palabra reservada `await` indica que la ejecución de la función asíncrona es detenida hasta que termine de ejecutar la función actual.

El siguiente paso consiste en crear una página:

```
var page = await browser.newPage();
```

Creamos una página y la guardamos en la variable page.

```
const client = await page.target().createCDPSession();
```

Se establece una sesión CDP (Protocolo de Chrome Devtools) con el servidor web donde se aloja la implementación de Shaka Player.

```
await page.goto('http://localhost:8081/shaka-player-2.5.12/demo/#audiolang=es-ES;textlang=es-ES;' + shaka_parameters + 'alwaysStreamText=true;uilang=es-ES;asset='+ process.argv[3] + ';panel=CUSTOM%20CONTENT;build=uncompiled;v');
```

Abrimos la página del Shaka Player en el navegador Chrome.

Las variables `shaka_parameters` contienen los parámetros relativos al buffer utilizado que se explicarán con detalle en el siguiente punto mientras que `process.argv[3]` contiene el Path o directorio del MANIFEST (MPD).

```
await page.waitFor(2000);
await page.click('[aria-label="Reproducir"]');
```

Esperamos 2 segundos para que la página cargue por completo y comenzamos la reproducción del video.

```
async function getBuffer() {
      //Código que obtiene la ocupación del buffer
}
var bufferInterval = setInterval(getBuffer, 1000);
```

Mediante la función `getBuffer()` se obtiene la ocupación del buffer en segundos y utilizando `setInterval(getBuffer, 1000)` se ejecuta dicho método cada 1 segundo de manera que obtenemos un Log de la ocupación del buffer cada segundo. En el siguiente punto se explicará el funcionamiento de Shaka Player y se verá con detalle el funcionamiento de este método.

```
while (await page.evaluate(() => window.isPlaying)) {
      for ( var j = 0; j <data.length; j++){
            await client.send('Network.emulateNetworkConditions', {
            'offline': false,
            'downloadThroughput': parseInt(data[j]['bw'],10) * 1024/8,
            'uploadThroughput': parseInt(data[j]['bw'],10) * 1024/ 8,
            'latency': 5
            })
            if (!(await page.evaluate(() => window.isPlaying))) break;
                  await page.waitFor(parseInt(data[j]['time'], 10));
            }
}
```

Mediante este método se realiza la simulación de cambios de anchos de banda. En primer lugar se crea un bucle *while* que comprueba que el video no ha llegado al final. Mientras el video no haya llegado al final se ejecuta un bucle *for* que recorre un archivo JSON que contiene los cambios de anchos de banda en cada escenario. Por ejemplo, para el escenario1 el archivo escenario1.json tiene el contenido observado en la Fig 3.5:


```
[
  {
    "time": 20000,
    "bw": 3300
  },
  {
    "time": 20000,
    "bw": 1000
  },
  {
    "time": 20000,
    "bw": 3300
  },
  {
    "time": 20000,
    "bw": 1000
  },
  {
    "time": 20000,
    "bw": 3300
```


Figura 3.5: Contenido archivo escenario1.json.

- `time`: Indica el tiempo en segundos que permanece seleccionado el mismo ancho de banda.
- `bw`: Indica el ancho de banda en kbps seleccionado durante el tiempo `time`.

Para poder hacer referencia al archivo de data en el código en primer lugar debemos importarlo:

```
let data = require(process.argv[4]);
```

`process.argv[4]` es el Path o directorio donde se encuentra el archivo. Este directorio se pasa como parámetro al arrancar el programa.

Para aplicar el ancho de banda deseado se utiliza el método

```
client.send('Network.emulateNetworkConditions',config)
```
[11]

Donde config es un JSON que contiene los siguientes atributos

- `offline`: Conectividad con el servidor.
- `downloadThroughput`: Velocidad de bajada simulada en bytes/s
- `uploadThroughput`: Velocidad de subida simulada en bytes/s
- `Latency`: Latencia simulada en ms

Una vez aplicado el ancho de banda se espera la cantidad de ms mediante el método

```
await page.waitFor();
```

Al terminar el video finaliza la simulación y los resultados de cambios de calidad son obtenidos a partir del Log que proporciona Shaka Player mientras que los resultados de la ocupación del buffer se obtienen del Log generado en el Script.

En la Fig. 3.6 se muestran los tres escenarios de evolución del ancho de banda considerados en nuestra simulación.

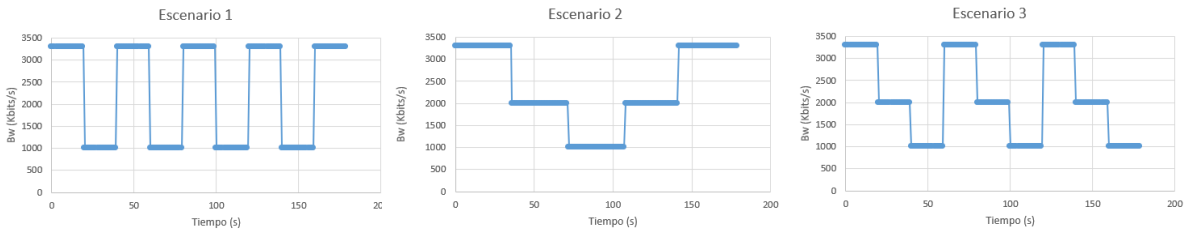


Figura 3.6: Escenarios simulados de cambios de ancho de banda.

En el anexo 1 y 2 se puede ver el código completo tanto del Script como de los JSON de los diferentes escenarios y el archivo de configuración de los parámetros del buffer.

3.7 Software para la reproducción del video (Shaka Player)

3.7.1 Introducción Shaka Player

Shaka Player [12] es una biblioteca JavaScript de código abierto que permite la reproducción de contenido multimedia en formatos DASH y HLS en un navegador estándar, sin necesidad de utilizar plugins o Flash. También admite almacenamiento y reproducción offline de contenidos usando IndexedDB. El contenido se puede almacenar en cualquier navegador.

El objetivo principal de Shaka Player es hacer lo más fácil posible el streaming adaptativo de video y audio utilizando navegadores modernos.

Antes de estudiar los parámetros de Shaka Player utilizados en el presente trabajo conviene ver qué opciones de configuración nos ofrece Shaka Player.

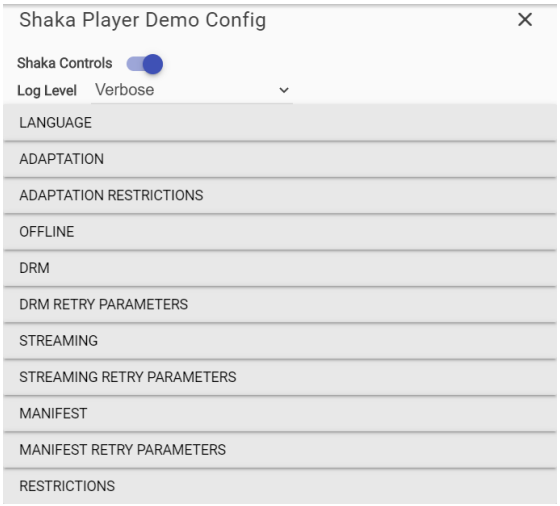


Figura 3.7. Opciones Shaka Player

En la Fig. 3.7 podemos observar las diferentes opciones de configuración que nos ofrece el reproductor Shaka Player.

3.7.2 Parámetros de configuración de red

En cuanto a la configuración de red existen tres menús de configuración de retransmisiones de mensajes en caso de fallo: retransmisiones de MANIFEST, segmentos y licencia. De esta manera las retransmisiones de segmentos pueden ser tratados de manera diferente a las retransmisiones de MANIFEST.

Para acceder a estos parámetros podemos o bien abrir la opción correspondiente en la interfaz de usuario o bien desde la línea de comandos en el apartado de Consola de herramientas de desarrolladores utilizar el comando de Shaka Player correspondiente.

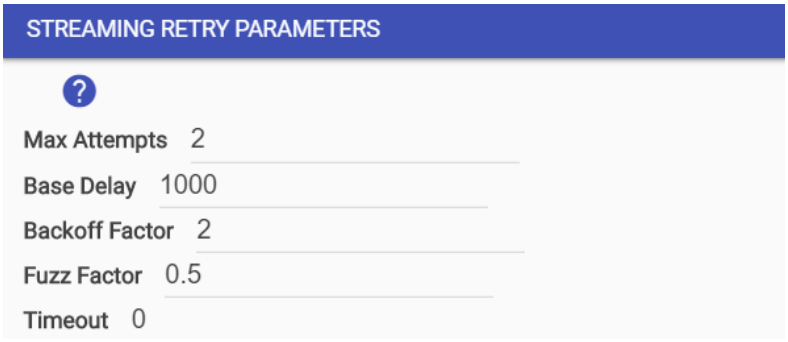


Figura 3.8. Opciones de retransmisión de segmentos a través del menú de Shaka Player

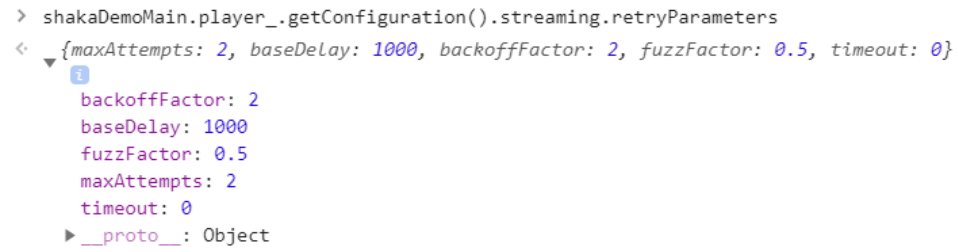


Figura 3.9. Opciones de retransmisión de segmentos a través de consola de herramientas del desarrollador

En las Fig 3.8 y 3.9 se pueden observar las opciones de retransmisión de segmentos desde el menú que ofrece Shaka Player y desde herramientas del desarrollador, respectivamente.

- `timeout`: Tiempo en ms a partir del cual abortamos la retransmisión. El valor de 0 indica nunca.
- `maxAttempts`: Máximo número de intentos de retransmisión.
- `baseDelay`: Tiempo de espera en ms entre retransmisiones.
- `backoffFactor`: Multiplicador entre retransmisiones
- `fuzzFactor`: Factor aplicado a cada tiempo de espera para evitar que varios clientes elijan el mismo tiempo de espera

Cada vez que reintentamos, `backoffFactor` se aplica al tiempo de espera entre reintentos. Por ejemplo, si el `baseDelay` es 1s, y el `backoffFactor` es 2 tendremos la siguiente secuencia de tiempos de espera entre retransmisiones:

1. Demanda de segmento en t = 0 segundos
2. Tiempo de espera de 1, Reintento en t = (0 + 1) = 1 segundo
3. Tiempo de espera de 2, Reintento en t = (1 + 2) = 3 segundos
4. Tiempo de espera de 4, Reintento en t = (3 + 4) = 7 segundos
5. Tiempo de espera de 8, Reintento en t = (7 + 8) = 15 segundos

y así sucesivamente. Para evitar que muchos clientes demanden segmentos a un servidor al mismo tiempo, también se aplica un factor adicional `fuzzFactor` . Un `fuzzFactor` de 0.5 significa que atenuamos el tiempo de espera 50% en cualquier dirección. Es decir, si el tiempo de espera ideal es 8, el tiempo de espera real se elegirá al azar entre 4 y 12. Así quedaría el ejemplo anterior tras aplicar el `fuzzFactor`:

1. Demanda de segmento en t = 0 segundos
2. Tiempo de espera de 1±50% (0.5 a 1.5 segundos para retransmitir)
3. Tiempo de espera de 2±50% (1 a 3 segundos para retransmitir)
4. Tiempo de espera de 4±50% (2 a 6 segundos para retransmitir)
5. Tiempo de espera de 8±50% (4 a 12 segundos para retransmitir)

En nuestra simulación se han tomado los valores por defecto vistos en las figuras 3.8 y 3.9 puesto que son los valores recomendados que ofrecen un correcto funcionamiento del sistema.

3.7.3 Parámetros de configuración del buffer

Respecto a la configuración del buffer existen tres parámetros clave: `bufferingGoal`, `rebufferingGoal` y `bufferBehind`. Todos ellos se expresan en segundos.

- `bufferingGoal`: Es la cantidad de contenido que intentamos almacenar en el buffer. Por ejemplo, si se establece el valor de 30, buscamos segmentos hasta que tengamos al menos 30 segundos almacenados en el buffer.
- `rebufferingGoal`: Es la cantidad de contenido que tenemos que almacenar antes de poder empezar a reproducir el video. Por ejemplo, si elegimos el valor de 15, permanecemos en un estado de almacenamiento de segmentos en el buffer hasta que tengamos al menos 15 segundos almacenados. Esto afecta tanto el almacenamiento en buffer al inicio como al cambio de instantes de reproducción posterior.
- `bufferBehind`: Es la cantidad de contenido que guardamos en el buffer detrás del instante de reproducción actual. Por ejemplo, si este valor es 30, mantenemos 30 segundos de contenido almacenado detrás instante actual del video. Cuando tenemos más de 30 segundos almacenados en el buffer, el contenido se eliminará desde el inicio del buffer para ahorrar memoria. Esto es un mínimo; si el tamaño del segmento es mayor que el `bufferBehind`, entonces se usará este en su lugar.

En el proyecto se han utilizado los valores por defecto en los campos `rebufferingGoal` y `bufferBehind` de 2 y 30 segundos respectivamente.

El campo de `bufferingGoal` ha sido modificado a 5, 10, 20 segundos para los 3 escenarios de estudio.

3.7.4 Configuración y lectura buffer Shaka Player

Una vez Puppeteer ha arrancado la página de Shaka Player para poder utilizar cualquier función de esta librería esta debe ser ejecutada en un bloque:

```
await page.evaluate(() => función()
```

puesto que dicha función debe ser ejecutada en la página del Shaka Player y no en el entorno de JavaScript.

Para poder modificar los parámetros relativos al buffer se ha creado un archivo JSON llamado shaka_config.json cuyo contenido se puede observar en la Fig.3.10

```
[
  {
    "rebufferingGoal": 2,
    "bufferingGoal": 5,
    "bufferBehind": 30
  }
]
```

Figura 3.10. Contenido archivo shaka_config

Para poder hacer referencia a este archivo en el código en primer lugar debemos importarlo:

```
let shaka_config = require(process.argv[5]);
```

`process.argv[5]` es el Path o directorio donde se encuentra el archivo. Este directorio se pasa como parámetro al arrancar el programa.

```
var shaka_parameters = '';

shaka_parameters= shaka_parameters +
'rebufferingGoal='+parseInt(shaka_config[0]['rebufferingGoal'], 10) + ';';
```

```
      shaka_parameters= shaka_parameters +
'bufferingGoal='+parseInt(shaka_config[0]['bufferingGoal'], 10) + ';';

      shaka_parameters= shaka_parameters +
'bufferBehind='+parseInt(shaka_config[0]['bufferBehind'], 10) + ';';
```

Guardamos en la variable `shaka_parameters` el contenido de la configuración del buffer. El valor de la variable quedaría de la siguiente manera:

rebufferingGoal=2;bufferingGoal=5;bufferBehind=30

Concatenamos este valor a la URL del Shaka Player antes de arrancar la página.

```
      await page.goto('http://localhost:8081/shaka-player-
      2.5.12/demo/#audiolang=es-ES;textlang=es-ES;' + shaka_parameters +
      'alwaysStreamText=true;uilang=es-ES;asset='+ process.argv[3] +
      ';panel=CUSTOM%20CONTENT;build=uncompiled;v');
```

De esta manera quedan establecidos los valores de configuración del buffer.

Por otro lado, para leer el buffer se ha utilizado la siguiente función:

```
async function getBuffer() {
            var timestamp = new Date().getTime();
             var buffer = await page.evaluate(() => {
                  var video = shakaDemoMain.video_;
                  var behind = 0;
                  var ahead = 0;
                  var currentTime = video.currentTime;
                  var buffered = video.buffered;
                   for (var i = 0; i < buffered.length; i++) {
                        if (i == (buffered.length-1)) {
                      ahead = buffered.end(i) - currentTime;
                      behind = currentTime - buffered.start(i);
                      break;
                  }
               }
            return  ahead.toFixed(0);
      });
      console.log(buffer*1000);
      }
      var bufferInterval = setInterval(getBuffer, 1000);
```

`shakaDemoMain.video_.buffered.end(i)`: Contiene la cantidad de segundos almacenados en el buffer desde el inicio de la reproducción.

`shakaDemoMain.video_.currentTime`: Contiene la cantidad de segundos transcurridos desde el inicio de la reproducción hasta el instante de video actual.

Si realizamos la resta de ambos valores obtenemos la cantidad de segundos que tiene el buffer en el instante actual.

Este es el valor que devuelve la función y que mostramos por pantalla para luego poder guardar y dibujar las gráficas pertinentes.

3.7.5 Obtención de cambios de calidad

La información relativa a los cambios de calidad se encuentra el Log que proporciona el Shaka Player. Dicho Log se puede encontrar en la consola de herramientas del desarrollador como se puede observar en la Fig 3.11

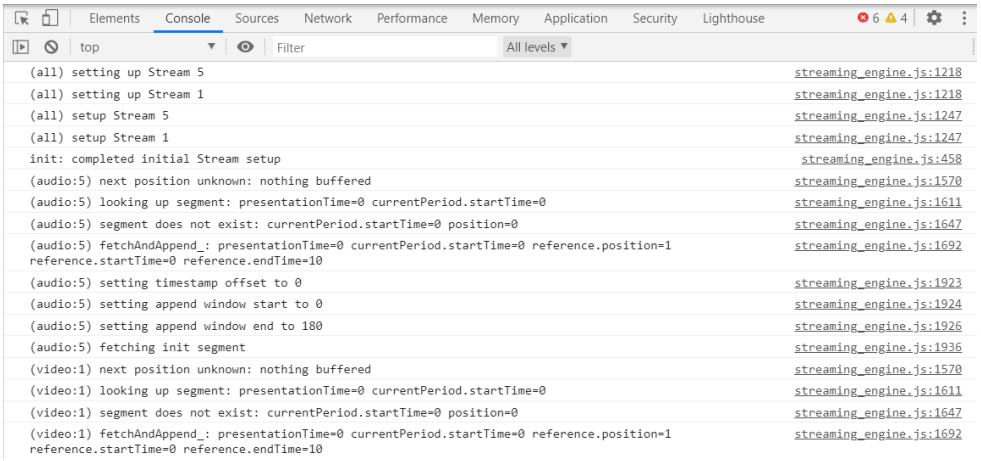


Figura 3.11. Log de cambios de calidad Shaka Player

A partir de la información obtenida los cambios de calidad se detectan utilizando los textos "`switching to Stream`" y el campo "`fetchAndAppend_: reference.startTime=`".

Tantos los datos de los cambios de calidad, como los de la ocupación del buffer son procesados con Excel para obtener las gráficas y resultados presentados en el siguiente punto.

Capítulo 4. Medidas y resultados

4.1 Escenario 1

Para seleccionar el valor de los anchos de banda de la simulación se ha tenido en cuenta que el *player* escoge de entre las calidades disponibles aquella que garantiza que no hay interrupciones. Para ello se tiene en cuenta la tasa de codificación del video, el tiempo que tarda en descargar el segmento y el tamaño del buffer.

En el primer escenario (Fig 4.1) se aplican cambios bruscos de ancho de banda cada 20 segundos. Se empieza con el ancho de banda de 3300 kbps que se ha comprobado experimentalmente que garantiza seleccionar la calidad más alta. Tras 20 segundos el ancho de banda cambia a 1000 kbps seleccionando la calidad más baja, y así sucesivamente.

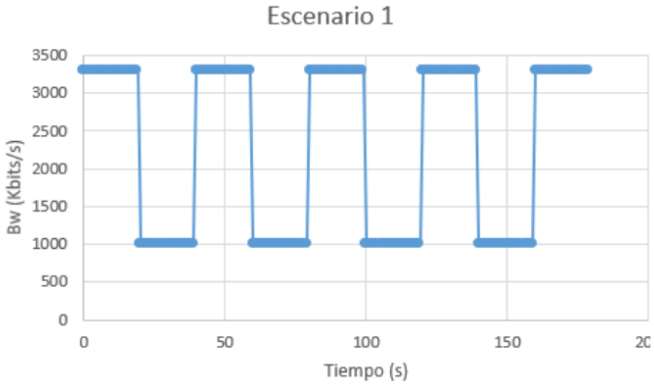


Figura 4.1. Perfil cambios de ancho de banda Escenario 1

4.1.1 Simulación con tamaño de buffer de 20 segundos

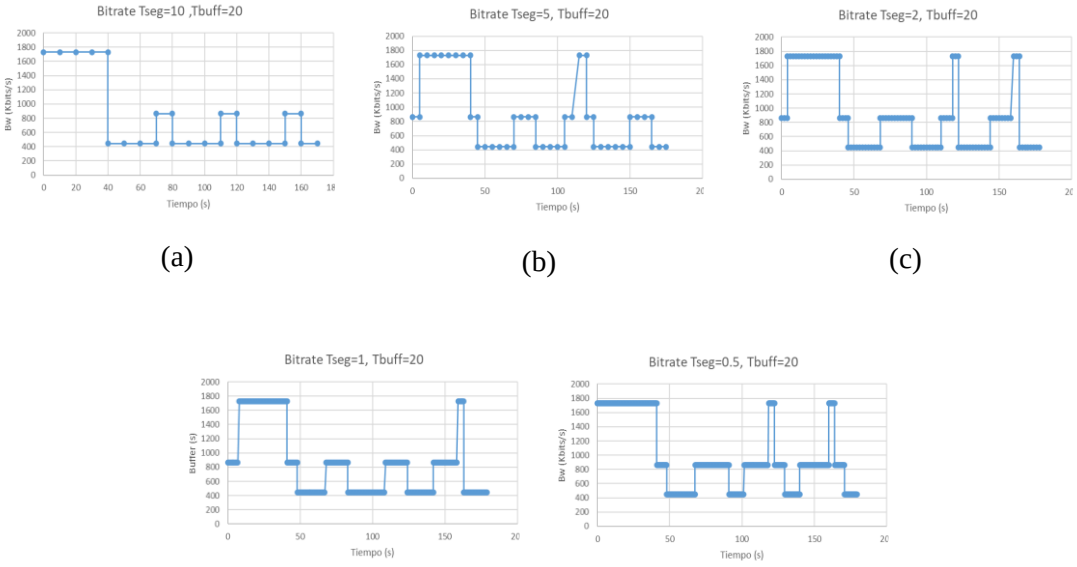


Figura 4.2. Cambios de calidad para Tamaño de segmento = 0.5, 1, 2, 5, 10 segundos, Tamaño buffer = 20 segundos

Cuando el tamaño de buffer o contenido que intentamos almacenar en él (`bufferingGoal`) es de 20 segundos los cambios de calidad obtenidos se pueden observar en la Fig 4.2.

Un primer dato importante es que a medida que disminuye la duración del segmento se obtienen más puntos al realizar la simulación. Esto es debido a que los puntos representan los segmentos demandados por el protocolo DASH. Cuanto menor es la duración del segmento, más segmentos es necesario pedir para reproducir el mismo video.

En segundo lugar se puede observar que dado el perfil de cambios de ancho de banda, el algoritmo de adaptación del Shaka Player tiende a elegir menos la calidad más alta cuanto mayor es la duración de segmento. Este efecto se observa claramente cuando la duración de segmento es de 10 segundos, subfigura (a) de Fig 4.1. Al cambiar el ancho de banda disponible a 3300 kbps el player comienza a incrementar la calidad pasando de la calidad baja a la calidad media. Debido a que el ancho de banda de 3300 kbps solo permanece disponible durante 20 segundos, el player no consigue llegar a incrementar la calidad a alta puesto que ya ha descargado 2 segmentos de 10 segundos y tiene el buffer lleno.

En la subfigura (c) se observa cómo con una duración de segmento de 2 segundos la selección de la calidad alta si que suele ser más usual aunque su duración sea corta.

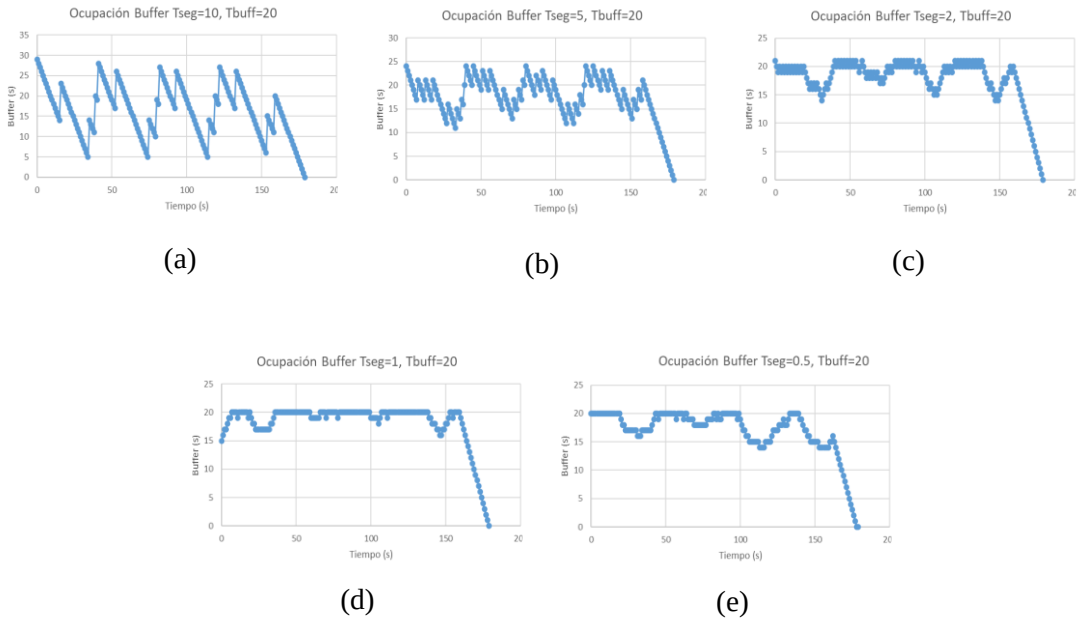


Figura 4.3. Ocupación del buffer para Tamaño de segmento = 0.5, 1, 2, 5, 10 segundos, Tamaño buffer = 20 segundos

En cuanto a la ocupación del buffer en el escenario 1 con un `bufferingGoal=20` podemos observar en la Fig 4.3 que en ningún momento llega a 0, con lo que nunca se produce la interrupción del video.

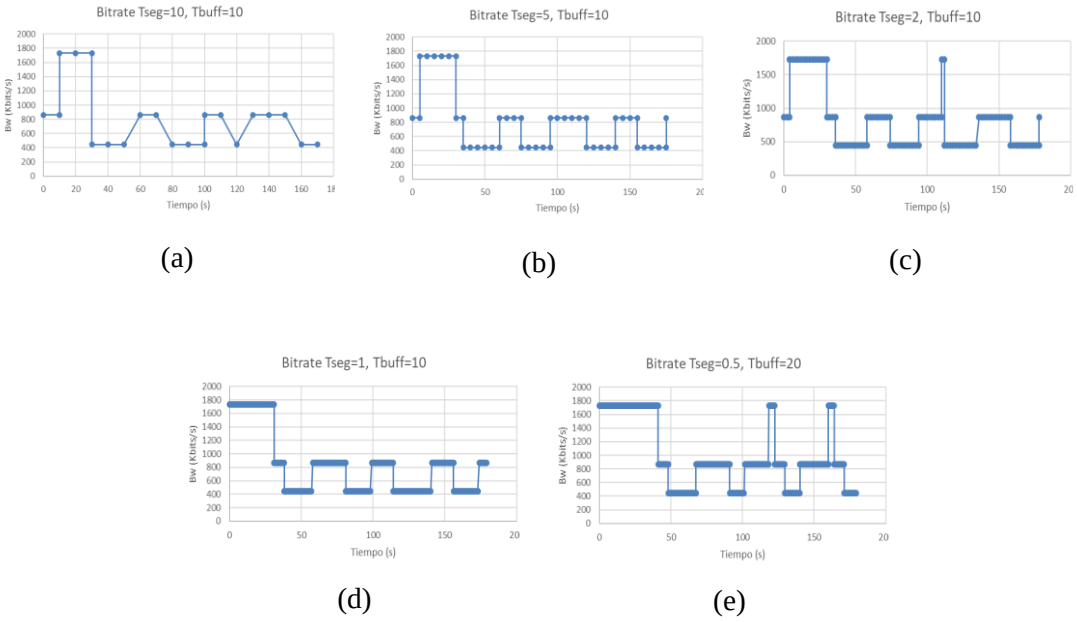
También observamos que cuanto más grande es la duración de segmento, más se vacía el buffer y por tanto incrementa la probabilidad de la interrupción del video.

La calidad media obtenida para cada duración de segmento es:

Duración de segmento	Ancho de banda medio
10 s	775,2957 kbps
5 s	869,8216 kbps
2 s	893,6689 kbps
1 s	864,7743 kbps
0.5 s	986,2723 kbps

Tabla 4.1. Calidad media obtenida escenario 1, tamaño de buffer = 20 segundos

4.1.2 Simulación con tamaño de buffer de 10 segundos



(a) (b) (c)

(d) (e)

Figura 4.4. Cambios de calidad para Tamaño de segmento = 0.5, 1, 2, 5, 10 segundos, Tamaño buffer = 10 segundos

En la Fig. 4.4 se pueden observar los cambios de calidad obtenidos cuando el tamaño de buffer o `bufferingGoal` es de 10 segundos. Los resultados de cambios de ancho de banda son casi idénticos a los obtenidos cuando el tamaño de buffer era de 20 segundos.

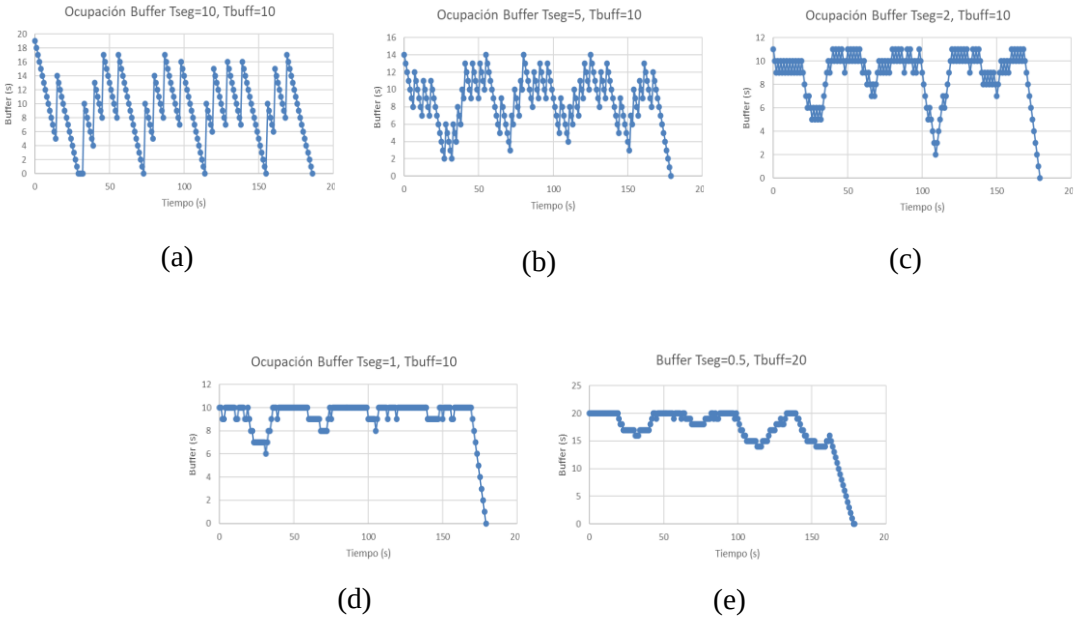


Figura 4.5. Ocupación del buffer para Tamaño de segmento = 0.5, 1, 2, 5, 10 segundos, Tamaño buffer = 10 segundos

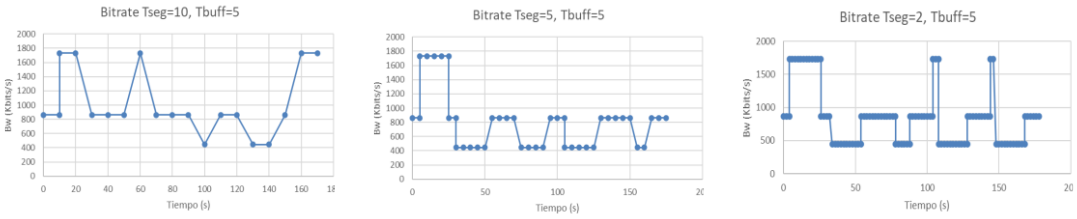
En cuanto a la ocupación del buffer en este caso observamos en la Fig 4.5 que para el caso de duración de segmento de 10 segundos este llega a 0 con lo que el video se detiene durante unos instantes. En los 4 casos restantes el video no se detiene aunque las fluctuaciones del buffer sean más altas que en la simulación anterior.

La calidad media obtenida para cada duración de segmento es:

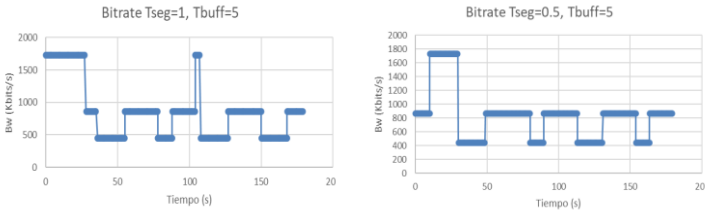
Duración de segmento	Ancho de banda medio
10 s	725,5237 kbps
5 s	761,9433 kbps
2 s	793,0921 kbps
1 s	823,9473 kbps
0.5 s	852,3342 kbps

Tabla 4.2. Calidad media obtenida escenario 1, tamaño de buffer = 10 segundos

4.1.3 Simulación con tamaño de buffer de 5 segundos



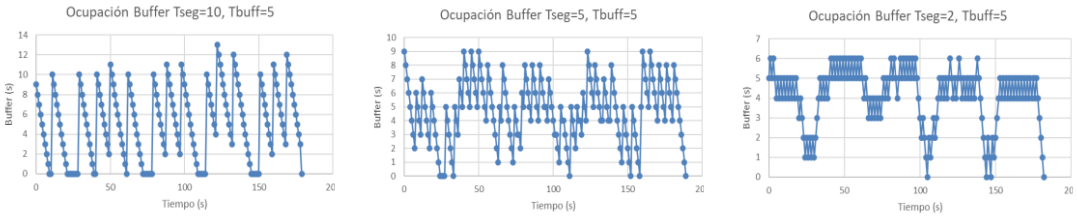
(a) (b) (c)



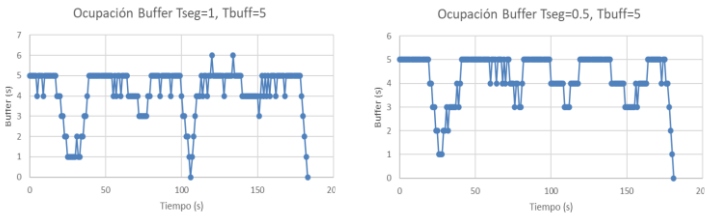
(d) (e)

Figura 4.6. Cambios de calidad para Tamaño de segmento = 0.5, 1, 2, 5, 10 segundos, Tamaño buffer = 5 segundos

En la Fig. 4.6 se pueden observar los cambios de calidad obtenidos cuando el tamaño de buffer o `bufferingGoal` es de 5 segundos. De nuevo los resultados son muy parecidos a la simulación anterior aunque se puede observar que cuando la duración de segmento es de 10 segundos los cambios de calidad son más dispersos y no siguen el perfil establecido.



(a) (b) (c)



(d) (e)

Figura 4.7. Ocupación del buffer para Tamaño de segmento = 0.5, 1, 2, 5, 10 segundos, Tamaño buffer = 5 segundos

En la Fig. 4.7 observamos la ocupación del buffer y en este caso a diferencia de la simulación anterior este llega a 0 cuando las duraciones de segmento son de 10, 5 y 2 y 1 segundos. Únicamente se consigue reproducir el video sin interrupciones cuando la duración de segmento es de 0.5 segundos.

La calidad media obtenida para cada duración de segmento es:

Duración de segmento	Ancho de banda medio
10 s	937,9932 kbps
5 s	772,7428 kbps
2 s	816,3346 kbps
1 s	844,7012 kbps
0.5 s	825,617 kbps

Tabla 4.3. Calidad media obtenida escenario 1, tamaño de buffer = 5 segundos

Aunque el bitrate obtenido en el caso de que la duración de segmento sea de 10 segundos es mayor que en el resto de casos, las interrupciones también son mayores con lo que la QoE sería peor.

4.2 Escenario 2

En el segundo escenario (Fig 4.8) se aplican cambios de ancho de banda de manera progresiva comenzando por 3300 kbps, bajando a 2000 kbps, 1000 kbps y luego subiendo a 2000 kbps y 3300 kbps para terminar. Cada cambio de ancho de banda tiene una duración de 36 segundos.

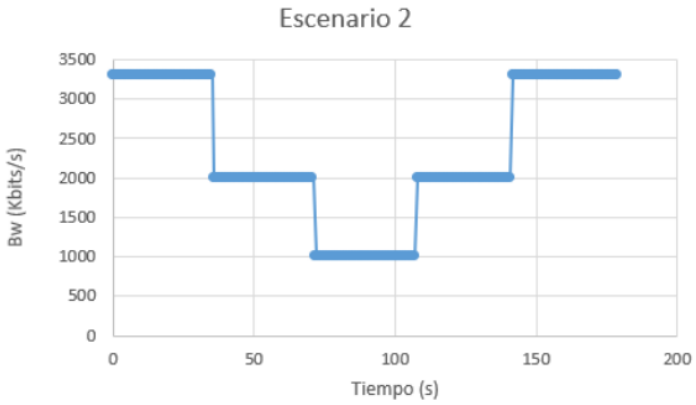


Figura 4.8. Perfil cambios de ancho de banda Escenario 2

4.2.1 Simulación con tamaño de buffer de 20 segundos

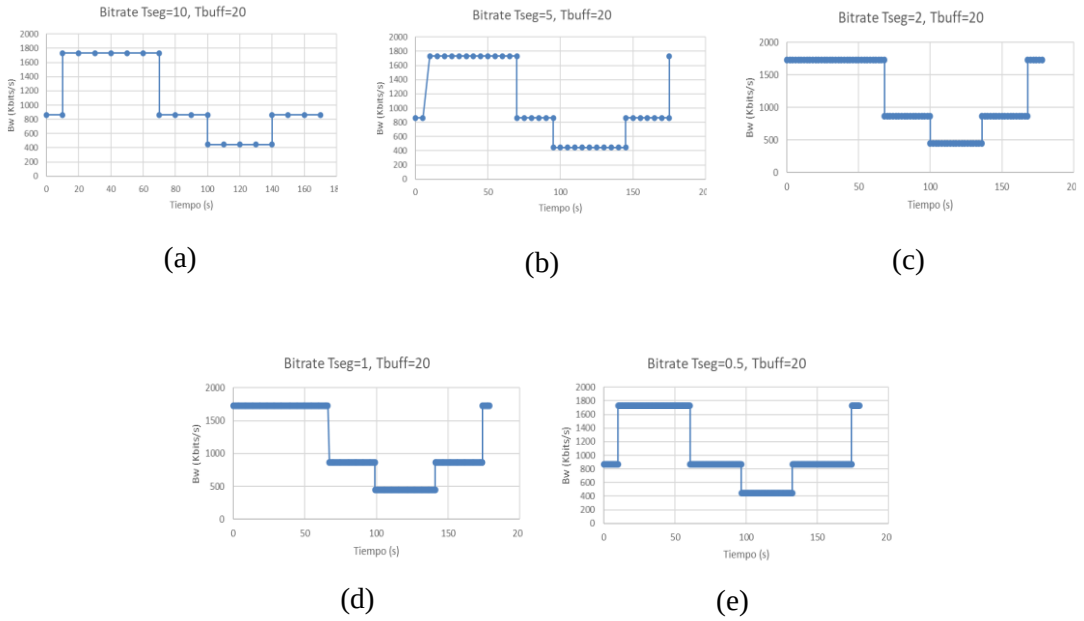


Figura 4.9. Cambios de calidad para Tamaño de segmento = 0.5, 1, 2, 5, 10 segundos, Tamaño buffer = 20 segundos

En la Fig. 4.9 se pueden observar los cambios de calidad obtenidos cuando el tamaño de buffer o `bufferingGoal` es de 20 segundos. Se puede observar que a diferencia del escenario 1 en el que los cambios eran más bruscos, en este caso el player tiene un seguimiento más fiel al perfil de tráfico establecido.

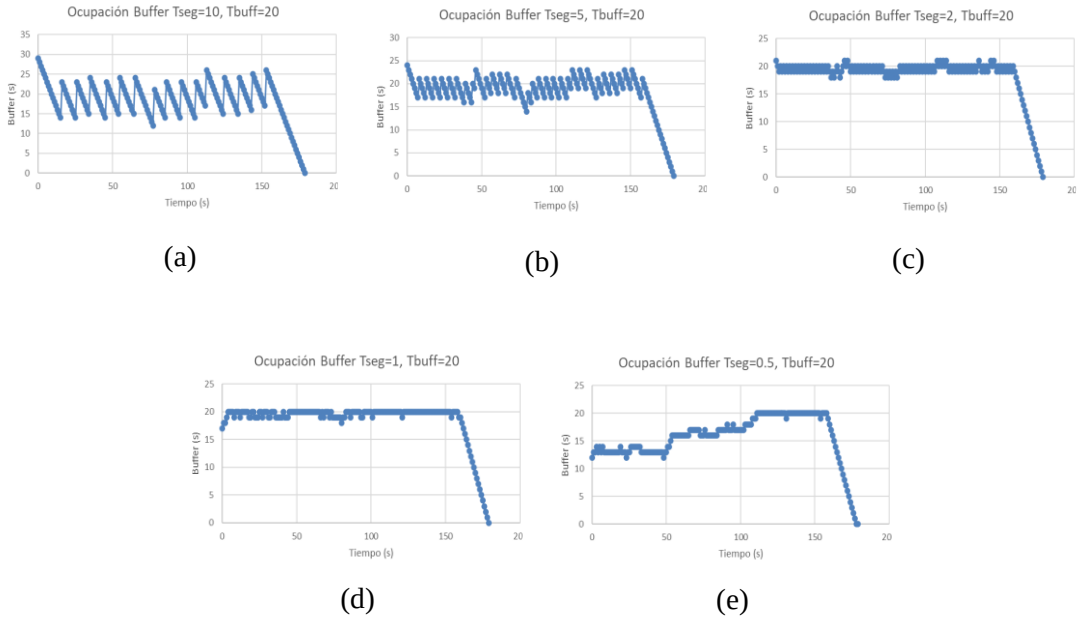


Figura 4.10. Ocupación del buffer para Tamaño de segmento = 0.5, 1, 2, 5, 10 segundos, Tamaño buffer = 20 segundos

En cuanto a la ocupación del buffer en la Fig. 4.10 observamos que no se produce en ningún momento la interrupción del video y de nuevo cuanto más grande es la duración del segmento más se vacía el buffer.

La calidad media obtenida para cada duración de segmento es:

Duración de segmento	Ancho de banda medio
10 s	1011,008 kbps
5 s	1035,806 kbps
2 s	1145,299 kbps
1 s	1102,459 kbps
0.5 s	1046,609 kbps

Tabla 4.4. Calidad media obtenida escenario 2, tamaño de buffer = 20 segundos

4.2.2 Simulación con tamaño de buffer de 10 segundos

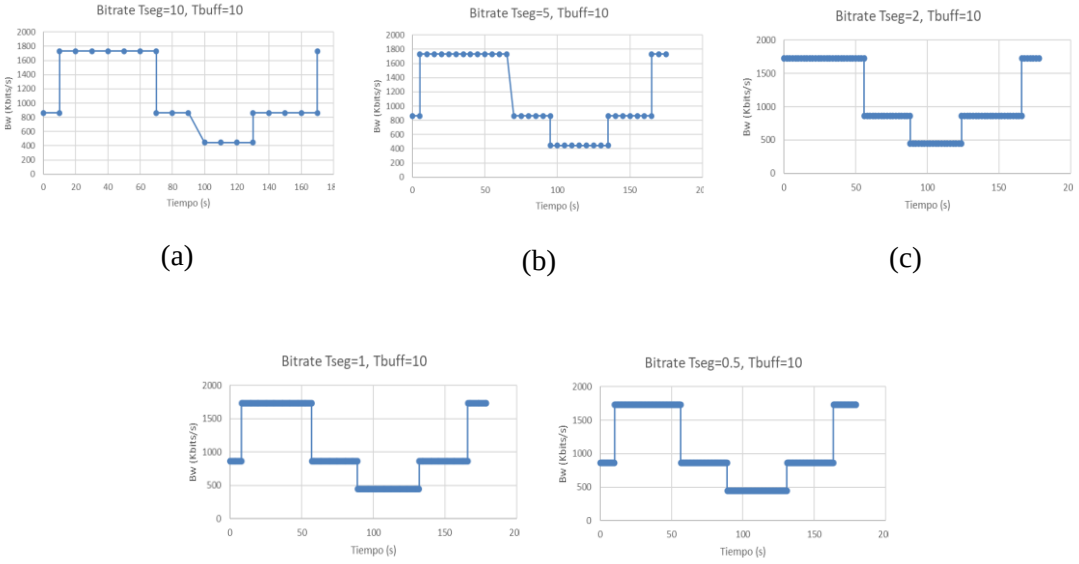


Figura 4.11. Cambios de calidad para Tamaño de segmento = 0.5, 1, 2, 5, 10 segundos, Tamaño buffer = 10 segundos

En la Fig. 4.11 se pueden observar los cambios de calidad obtenidos cuando el tamaño de buffer o `bufferingGoal` es de 10 segundos. En cuanto a los cambios de ancho de banda los resultados son casi idénticos a la simulación anterior.

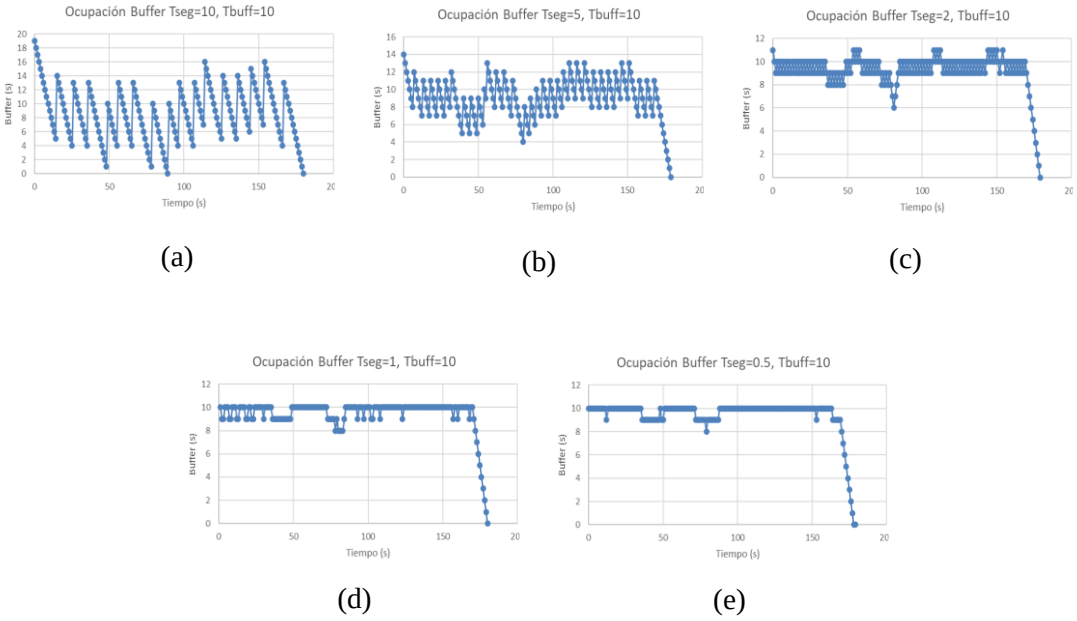


Figura 4.12. Ocupación del buffer para Tamaño de segmento = 0.5, 1, 2, 5, 10 segundos, Tamaño buffer = 10 segundos

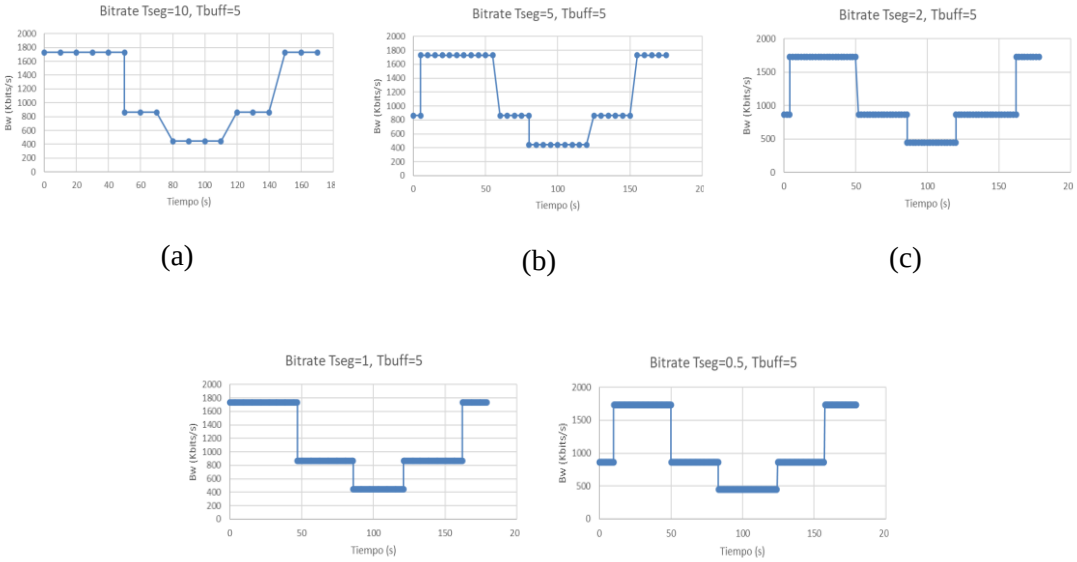
En cuanto a la ocupación del buffer en la Fig. 4.12 observamos que se produce la interrupción del video cuando la duración de segmento es de 10 segundos. En los 4 casos restantes el video se reproduce sin interrupciones.

La calidad media obtenida para cada duración de segmento es:

Duración de segmento	Ancho de banda medio
10 s	1011,008 kbps
5 s	1083,113 kbps
2 s	1097,17 kbps
1 s	1056,819 kbps
0.5 s	1063,947 kbps

Tabla 4.5. Calidad media obtenida escenario 2, tamaño de buffer = 10 segundos

4.2.3 Simulación con tamaño de buffer de 5 segundos

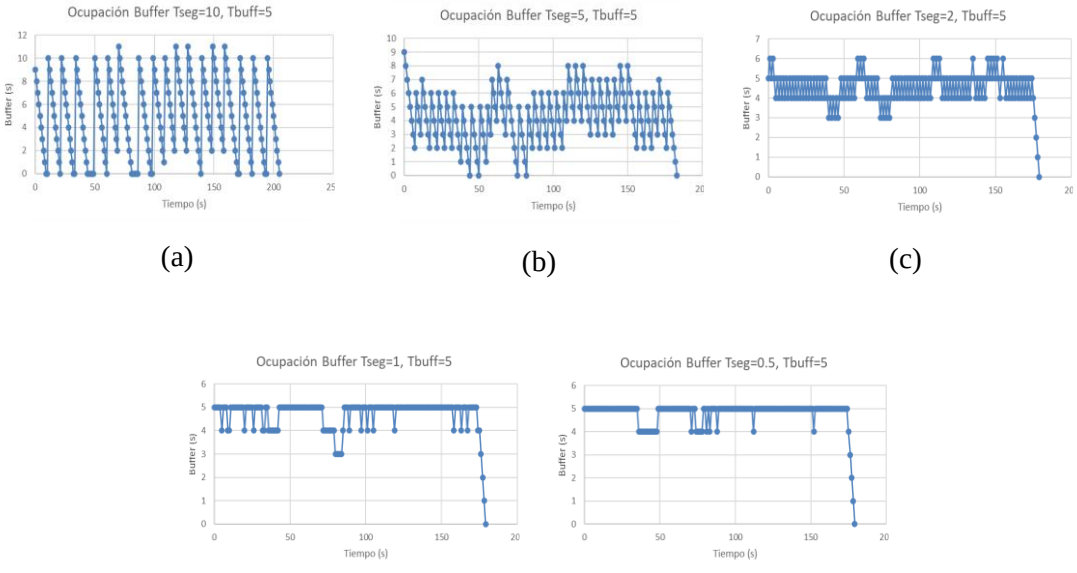


(a) (b) (c)

(d) (e)

Figura 4.13. Cambios de calidad para Tamaño de segmento = 0.5, 1, 2, 5, 10 segundos, Tamaño buffer = 5 segundos

En la Fig. 4.13 se pueden observar los cambios de calidad obtenidos cuando el tamaño de buffer o `bufferingGoal` es de 5 segundos. Se observan pequeñas desviaciones en el seguimiento del perfil de cambios de anchos de banda en el caso de duración de segmento de 10 segundos.



(a) (b) (c)

(d) (e)

Figura 4.14. Ocupación del buffer para Tamaño de segmento = 0.5, 1, 2, 5, 10 segundos, Tamaño buffer = 5 segundos

En cuanto a la ocupación del buffer se observa en la Fig. 4.14 que esta llega a 0 cuando la duración de segmento es de 10 y 5 segundos. En los 3 casos restantes el video se reproduce sin interrupciones.

La calidad media obtenida para cada duración de segmento es:

Duración de segmento	Ancho de banda medio
10 s	1107,265 kbps
5 s	1107,177 kbps
2 s	1072,942 kbps
1 s	1085,038 kbps
0.5 s	1066,271 kbps

Tabla 4.6. Calidad media obtenida escenario 2, tamaño de buffer = 5 segundos

4.3 Escenario 3

En el tercer escenario (Fig 4.15) se aplican cambios escalonados de ancho de banda empezando en 3300 kbps, bajando a 2000 kbps, 1000 kbps y luego volviendo a empezar. Los cambios de ancho de banda se aplican cada 20 segundos.

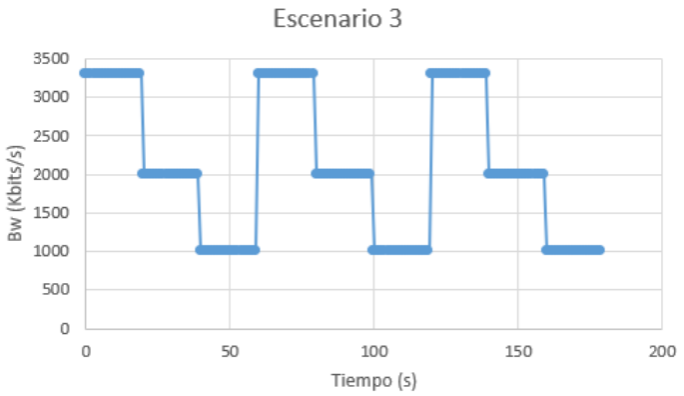
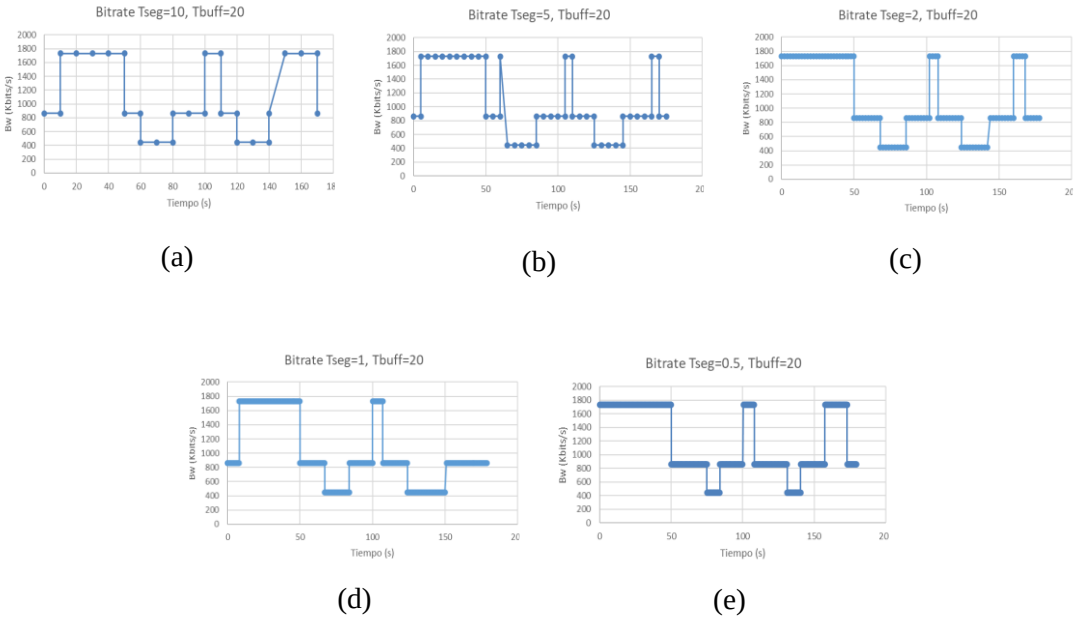


Figura 4.15. Perfil cambios de ancho de banda Escenario 3

4.3.1 Simulación con tamaño de buffer de 20 segundos

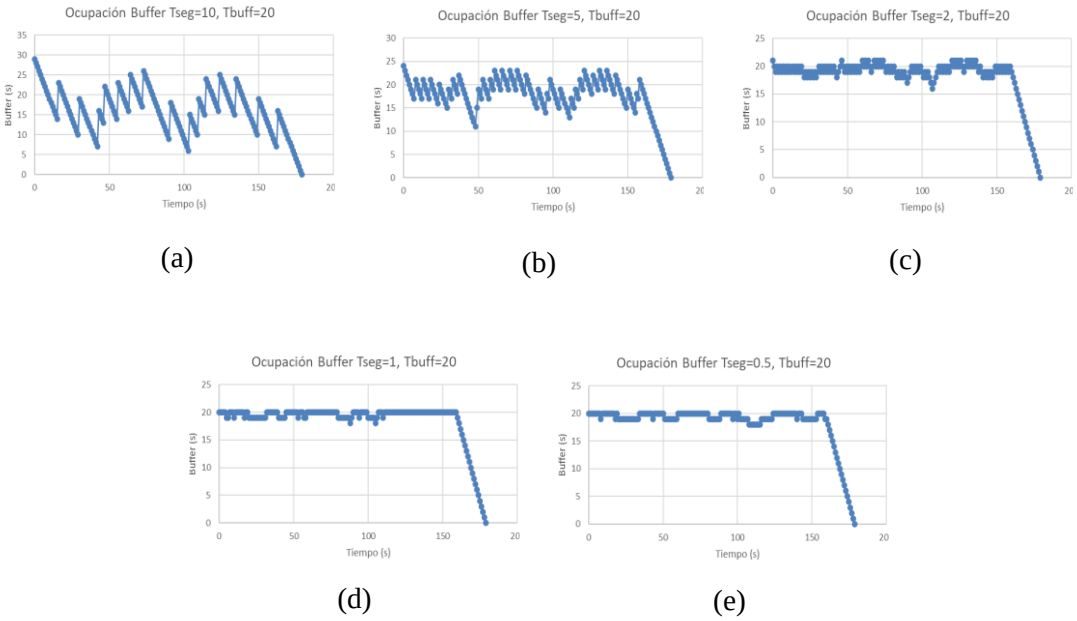


(a) (b) (c)

(d) (e)

Figura 4.16. Cambios de calidad para Tamaño de segmento = 0.5, 1, 2, 5, 10 segundos, Tamaño buffer = 20 segundos

En la Fig. 4.16 se pueden observar los cambios de calidad obtenidos cuando el tamaño de buffer o `bufferingGoal` es de 20 segundos.



(a) (b) (c)

(d) (e)

Figura 4.17. Ocupación del buffer para Tamaño de segmento = 0.5, 1, 2, 5, 10 segundos, Tamaño buffer = 20 segundos

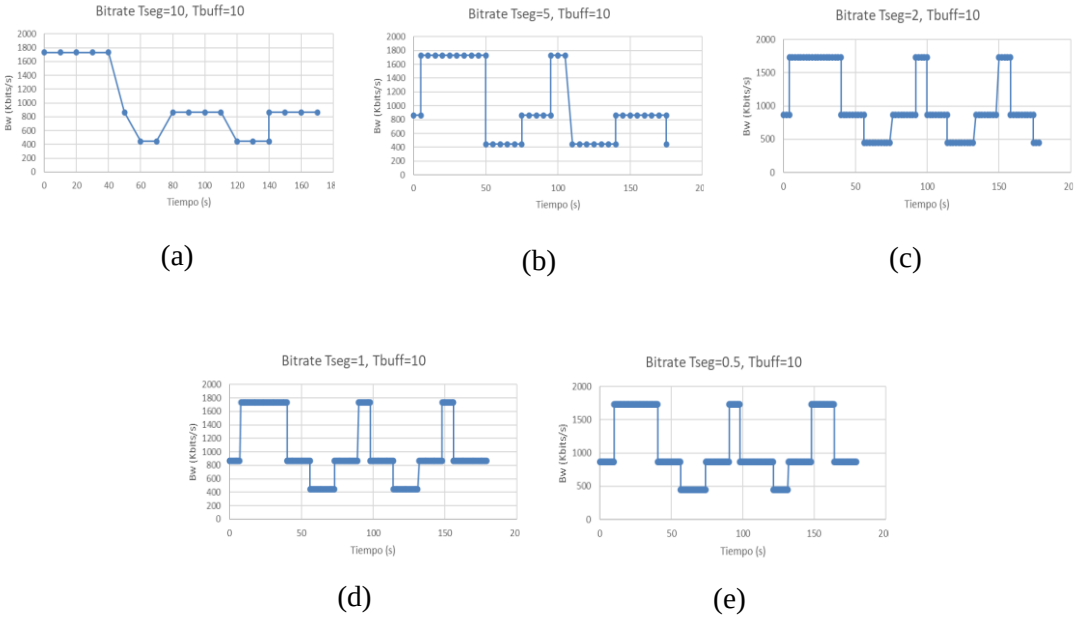
En cuanto a la ocupación del buffer se observa en la Fig. 4.17 que este no llega a 0 con lo que no se produce la interrupción del video.

La calidad media obtenida para cada duración de segmento es:

Duración de segmento	Ancho de banda medio
10 s	1107,265 kbps
5 s	999,2988 kbps
2 s	1077,919 kbps
1 s	994,2516 kbps
0.5 s	1171,508 kbps

Tabla 4.7. Calidad media obtenida escenario 3, tamaño de buffer = 20 segundos

4.3.2 Simulación con tamaño de buffer de 10 segundos



(a) (b) (c)

(d) (e)

Figura 4.18. Cambios de calidad para Tamaño de segmento = 0.5, 1, 2, 5, 10 segundos, Tamaño buffer = 10 segundos

En la Fig. 4.18 se pueden observar los cambios de calidad obtenidos cuando el tamaño de buffer o `bufferingGoal` es de 10 segundos. De nuevo, se observan pequeñas desviaciones en el seguimiento del perfil de cambios de anchos de banda en el caso de duración de segmento de 10 y 5 segundos.

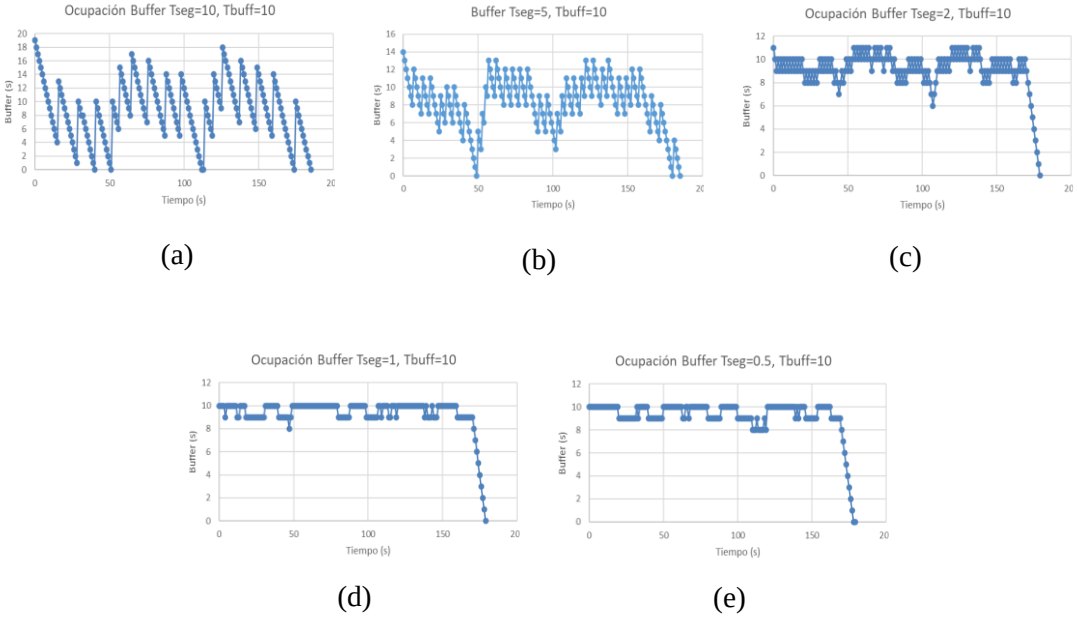


Figura 4.19. Ocupación del buffer para Tamaño de segmento = 0.5, 1, 2, 5, 10 segundos, Tamaño buffer = 10 segundos

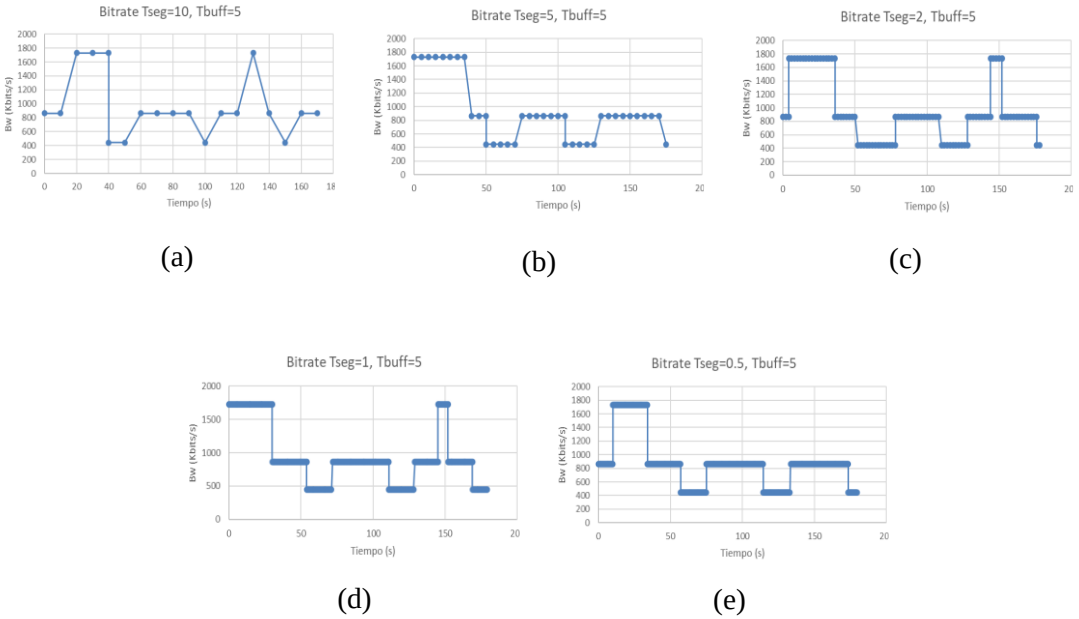
En cuanto a la ocupación del buffer se observa en la Fig. 4.19 que este llega a 0 y por tanto se producen interrupciones cuando la duración de segmento es de 10 y 5 segundos.

La calidad media obtenida para cada duración de segmento es:

Duración de segmento	Ancho de banda medio
10 s	891,5082 kbps
5 s	964,4351 kbps
2 s	1020,493 kbps
1 s	1019,983 kbps
0.5 s	1056,739 kbps

Tabla 4.8. Calidad media obtenida escenario 3, tamaño de buffer = 10 segundos

4.3.3 Simulación con tamaño de buffer de 5 segundos

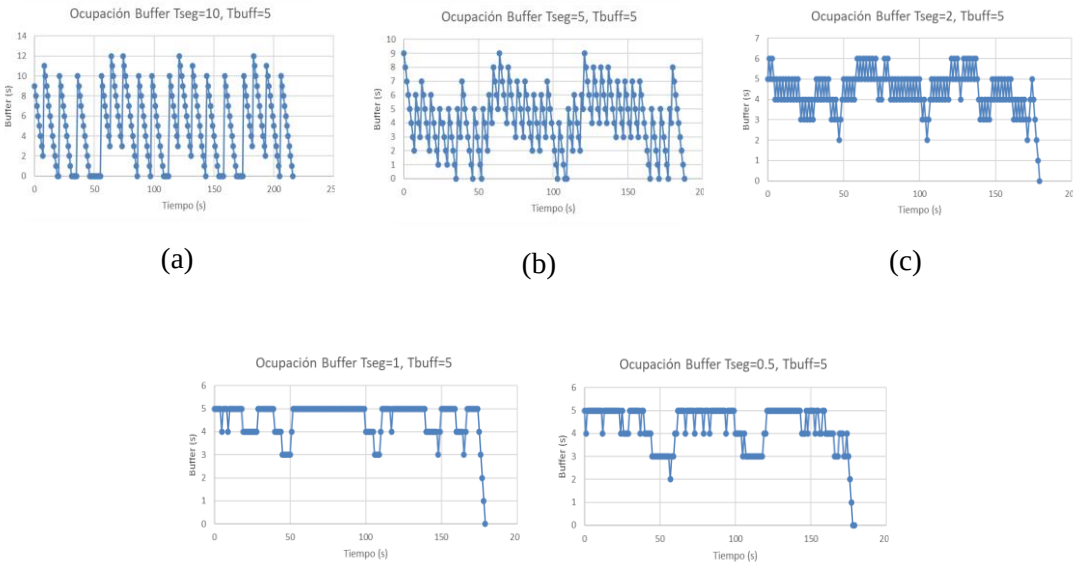


(a) (b) (c)

(d) (e)

Figura 4.20. Cambios de calidad para Tamaño de segmento = 0.5, 1, 2, 5, 10 segundos, Tamaño buffer = 5 segundos

En la Fig. 4.20 se pueden observar los cambios de calidad obtenidos cuando el tamaño de buffer o `bufferingGoal` es de 5 segundos.



(a) (b) (c)

(d) (e)

Figura 4.21. Ocupación del buffer para Tamaño de segmento = 0.5, 1, 2, 5, 10 segundos, Tamaño buffer = 5 segundos

En cuanto a la ocupación del buffer se observa en la Fig. 4.21 que este llega a 0 y por tanto se producen interrupciones cuando la duración de segmento es de 10 y 5 segundos.

La calidad media obtenida para cada duración de segmento es:

Duración de segmento	Ancho de banda medio
10 s	937,9932 kbps
5 s	903,0418 kbps
2 s	929,8708 kbps
1 s	934,1731 kbps
0.5 s	877,4901 kbps

Tabla 4.9. Calidad media obtenida escenario 3, tamaño de buffer = 5 segundos

4.4 Análisis prestaciones DASH

Tras el estudio realizado mediante diferentes perfiles de cambios de anchos de banda analizaremos cuáles son la duración de segmento y tamaño de buffer óptimos. En primer lugar veremos algunas ventajas e inconvenientes de las diferentes duraciones de los segmentos.

Duración del segmento	Ventajas	Inconvenientes
Corta duración	- Adecuado para el servicio de *live streaming*. - Alta granularidad en el proceso de switching a nivel de segmento.	- Elevado número de ficheros y URLs. - En entornos donde el ancho de banda es fluctuante puede empeorar la QoE.
Larga duración	- Pequeño número de ficheros y URLs. - Menos cambios de calidad en entornos de ancho de banda fluctuante.	- Reacción lenta ante estados de congestión. - No es válido para el servicio de *live streaming*

Tabla 4.10. Ventajas e inconvenientes diferentes duraciones de segmento

Para elegir el tamaño de segmento óptimo se debe tener en cuenta por un lado la codificación de los videos. Para garantizar el cambio, sin interrupciones, entre las diferentes representaciones de calidad del video es importante que las tramas de tipo intra (I) estén en posiciones fijas en el video. Por ejemplo para un video con duración de segmento de 2 segundos y 24 imágenes por segundo (FPS) se debe colocar una trama I cada 48 segundos. De esta manera se garantiza que cada segmento tiene una trama I disponible al comienzo de este, lo cual es necesario para poder cambiar entre las representaciones de calidad. Cuando el decodificador recibe un segmento que comienza con una trama I no necesita referencias a tramas o segmentos anteriores para poder decodificar el contenido y, por lo tanto, el nuevo segmento puede ser reproducido sin problemas aunque tenga una calidad o bitrate diferente al anterior. Las posiciones fijas de tramas I se pueden lograr restringiendo el tamaño del grupo de imágenes (GoP) del codificador al tamaño de segmento deseado del contenido. Como consecuencia, desde el punto de vista de la codificación, los segmentos de corta duración producen un mayor número de segmentos y debido a esto, también se necesitan más tramas I para garantizar el cambio entre representaciones de calidad a nivel de segmento. Esto conduce a una menor eficiencia de codificación debido a que las tramas I no aprovechan la predicción temporal y por tanto necesitan más bits para la codificación de los mismos. La calidad del video empeora puesto que se están utilizando más bits de los que se utilizarían si la duración de segmento fuese mayor. Este problema es bien conocido y debe

considerarse en la generación de contenido. Según el siguiente estudio [13] realizado sobre cómo afecta el GoP al PSNR [dB] segmentos con duración inferior a 2 segundos deben ser evitados. El PSNR es la métrica utilizada con mayor frecuencia como medida de la calidad objetiva de video.

Por otro lado, desde la perspectiva de la red, también hay factores influyentes que debemos considerar. En primer lugar, segmentos de larga duración pueden causar interrupciones cuando existen cambios bruscos en el ancho de banda disponible. Este efecto se observa claramente en el escenario 1 cuando la duración de segmento es de 10 segundos y el tamaño de buffer de 10 segundos. Cuando la duración de segmento es corta se produce una sobrecarga de solicitudes GET dando lugar a un rendimiento de transmisión deficiente. Este efecto es también observado en todas las simulaciones de duraciones de segmento pequeñas ya que aparecen muchos mas puntos indicando que se han realizado más peticiones GET. Teniendo en cuenta el estudio anterior [13] en el cual se realizan simulaciones del ancho de banda medio en función de la duración del segmento se llega a la conclusión de que el tamaño de segmento óptimo al utilizar un servidor HTTP 1.1 con conexiones persistentes está entre 2 y 3 segundos.

En este caso, teniendo en cuenta los resultados obtenidos, el tamaño de segmento óptimo es de 2 segundos con un tamaño de buffer de 20 segundos. Además, se cumple que el bitrate medio es mayor que en las duraciones de segmento adyacentes de 5 y 1 segundos. En cuanto a la ocupación del buffer en los 3 escenarios es parecida sin llegar este a vaciarse en ningún momento.

Capítulo 5. Conclusiones y propuestas para trabajo futuro

En los últimos 15 años los servicios de transmisión en vivo han evolucionado a un ritmo exponencial. Los primeros servicios de streaming adaptativo HTTP fueron ofrecidos por Apple con *HTTP Live Streaming* (HLS) utilizando segmentos de duración 10 segundos con un tamaño de buffer mínimo para iniciar la reproducción de 3 segmentos. Estos sistemas ofrecían latencias de alrededor de 40 segundos desde la generación del video hasta su reproducción por el usuario. La primera solución para disminuir la latencia fue reducir el tamaño de los segmentos, aunque esto ha sido solo una solución parcial debido a que segmentos de corta duración hacen que la longitud del grupo de imágenes (GoP) sea menor y por tanto empeora la calidad visual del video además de hacer la infraestructura de entrega de las CDN más inestable.

Los contenedores de archivos CMAF [14] aparecen como solución al problema de la pérdida de calidad al utilizar segmentos de corta duración, además de ofrecer una latencia del orden de varios segundos. CMAF ofrece la posibilidad de transmitir contenidos tanto HLS como MPEG-DASH manteniendo una latencia muy baja. También ofrece un formato común de manera que los servidores no necesitan almacenar una copia de los contenidos en formato compatible con HLS y otra para DASH.

La idea principal de los contenedores CMAF es que trocean los segmentos en subsegmentos de duración más corta con lo que son enviados antes reduciendo así la latencia. A pesar de las ventajas que ofrece CMAF, el uso de segmentos muy pequeños es una alternativa hasta que esté implementado CMAF en todos los sistemas de DASH que requieran baja latencia.

En cuanto a los objetivos del presente proyecto, estos se han realizado con éxito. Se ha creado un entorno de pruebas automatizado, flexible y sencillo que abarca desde la codificación de contenidos hasta la extracción de parámetros de interés. Con dicho sistema se han analizado las prestaciones de DASH con diferentes tamaños de segmentos.

Se propone como propuesta de trabajo futuro emplear el entorno automatizado de pruebas desarrollado en el proyecto en un sistema donde se utilice CMAF para evaluar las prestaciones del mismo.

Capítulo 6. Bibliografía

[1] Videoconferencia y vídeo en streaming doblan el tráfico de datos en España https://www.redestelecom.es/comunicaciones/noticias/1117640000303/videoconferencia-y-video-streaming-doblan-trafico-de-datos-espana.1.html

[2] Vídeo adaptativo a través de MPEG-DASH https://www.comm.upv.es/es/dash/

[3] Apuntes de la asignatura Comunicaciones Multimedia, ETSIT

[4] Paola Guzmán, Pau Arce, Juan Carlos Guerri, Automatic QoE evaluation for asymmetric encoding of 3D videos for DASH streaming service, Ad Hoc Networks 106 (September 2020) [Online] Available:

https://www.sciencedirect.com/science/article/abs/pii/S1570870520301748?via%3Dihub

[5] Docker definición, Wikipedia https://es.wikipedia.org/wiki/Docker_(software)

[6] FFMPEG definición, Wikipedia https://es.wikipedia.org/wiki/FFmpeg

[7] Documentación FFMPEG https://ffmpeg.org/ffmpeg.html

[8] MP4BOX definición, Wikipedia https://es.wikipedia.org/wiki/MP4Box

[9] Documentación MP4BOX https://github.com/gpac/gpac/wiki/mp4box-dash-opts

[10] "Puppeteer."[Online]. Available: https://github.com/puppeteer/puppeteer

[11] Network throttling in Puppeteer https://fdalvi.github.io/blog/2018-02-05-puppeteer-network-throttle/

[12] Shaka Player Demo [Online]. Available: https://shaka-player-demo.appspot.com/

[13] Optimal Adaptive Streaming Formats MPEG-DASH & HLS Segment Length https://bitmovin.com/mpeg-dash-hls-segment-length/

[14] Ultra-Low-Latency Streaming Using Chunked-Encoded and ChunkedTransferred CMAF

https://www.akamai.com/us/en/multimedia/documents/white-paper/low-latency-streaming-cmaf-whitepaper.pdf

Capítulo 7. Anexo A – Aplicación

```
let data = require(process.argv[4]);
let shaka_config = require(process.argv[5]);
var puppeteer = require('puppeteer');
var shaka_parameters = '';


async function run(){
	var options = {headless : false, executablePath: process.argv[2], 
devtools: true};
	var browser = await puppeteer.launch(options);
	var page = await browser.newPage();
//Conectamos con Chrome DevTools
	const client = await page.target().createCDPSession();
//Añadimos parámetros relativos al buffer
	shaka_parameters= shaka_parameters + 
'rebufferingGoal='+parseInt(shaka_config[0]['rebufferingGoal'], 10) + ';';
	shaka_parameters= shaka_parameters + 
'bufferingGoal='+parseInt(shaka_config[0]['bufferingGoal'], 10) + ';';
	shaka_parameters= shaka_parameters + 
'bufferBehind='+parseInt(shaka_config[0]['bufferBehind'], 10) + ';';
//Abrimos la página de  Shakaplayer
	await page.goto('http://localhost:8081/shaka-player-
2.5.12/demo/#audiolang=es-ES;textlang=es-ES;' + shaka_parameters + 
'alwaysStreamText=true;uilang=es-ES;asset='+ process.argv[3] + 
';panel=CUSTOM%20CONTENT;build=uncompiled;v');
// Esperamos 2 segundos para que la página cargue completamente
	await page.waitFor(2000);
//Utilizamos windows.isPlaying para saber cuando terminar la simulación
	await page.evaluate(() => window.isPlaying = true);
	await page.evaluate(() => 
document.getElementById("video").addEventListener('ended', function() {
		window.isPlaying = false;
	}));
//Hacemos click en el botón de reproducir
    await page.click('[aria-label="Reproducir"]');
//Se define la async function para obtener el valor del buffer y se definen
las variable timestamp y buffer haciendo un page evaluate
```

```
async function getBuffer() {
//Define la variable timestamp
	var timestamp = new Date().getTime();
//Define la variable buffer
	var buffer = await page.evaluate(() => {
		var video = shakaDemoMain.video_;
		var behind = 0;
		var ahead = 0;
		var currentTime = video.currentTime;
		var buffered = video.buffered;
			for (var i = 0; i < buffered.length; i++) {
				if (i == (buffered.length-1)) {
				ahead = buffered.end(i) - currentTime;
				behind = currentTime - buffered.start(i);
				break;
			}
		}
// Retorna el valor de ahead sin decimales (calculado con la función del
shaka player modificado)
		return  ahead.toFixed(0);
	});
	console.log(buffer*1000);
	}
//Se usa setInterval para repetir la ejecución de la función cada 1000ms, en
este caso leer el buffer
	var bufferInterval = setInterval(getBuffer, 1000);
	while (await page.evaluate(() => window.isPlaying)) {
//Función que para cuando termina el video
// Aplicamos condiciones de red
		for ( var j = 0; j <data.length; j++){
			await client.send('Network.emulateNetworkConditions', {
			'offline': false,
			'downloadThroughput': parseInt(data[j]['bw'],10)* 1024/ 8,
			'uploadThroughput': parseInt(data[j]['bw'],10) * 1024 / 8,
			'latency': 5
				})
			if (!(await page.evaluate(() => window.isPlaying))) break;
				await page.waitFor(parseInt(data[j]['time'], 10));
		}
```

```
        }
        clearInterval(bufferInterval);
        console.log('Simulacion finalizada');
}
run();
```

Capítulo 8. Anexo B – Escenarios y configuración Shaka Player

- escenario1.json

```
[
  {
    "time": 20000,
    "bw": 3300
  },
  {
    "time": 20000,
    "bw": 1000
  },
  {
    "time": 20000,
    "bw": 3300
  },
  {
    "time": 20000,
    "bw": 1000
  },
  {
    "time": 20000,
    "bw": 3300
  },
  {
    "time": 20000,
    "bw": 1000
  },
  {
    "time": 20000,
    "bw": 3300
  },
  {
    "time": 20000,
    "bw": 1000
  }
]
```

escenario2.json

```
[
  {
    "time": 36000,
    "bw": 3300
  },
  {
    "time": 36000,
    "bw": 2000
  },
  {
    "time": 36000,
    "bw": 1000
  },
  {
    "time": 36000,
    "bw": 2000
  },
  {
    "time": 36000,
    "bw": 3300
  }
]
```

escenario3.json

```
[
  {
    "time": 20000,
    "bw": 3300
  },
  {
    "time": 20000,
    "bw": 2000
  },
  {
    "time": 20000,
    "bw": 1000
  },
  {
    "time": 20000,
    "bw": 3300
  },
  {
    "time": 20000,
    "bw": 2000
  },
  {
    "time": 20000,
    "bw": 1000
  },
  {
    "time": 20000,
    "bw": 3300
  },
  {
    "time": 20000,
    "bw": 2000
  },
  {
    "time": 20000,
    "bw": 1000
```

```
  }
]
```

shaka_config.json

```
[
  {
    "rebufferingGoal": 2,
    "bufferingGoal": 5,
    "bufferBehind": 30
```