Integración de Redes, Aplicaciones y Contenidos
Práctica 3: DASH

El desarrollo de una aplicación web para proporcionar un servicio de *streaming* de vídeo básico se divide en la creación de un servidor que organice el acceso a los recursos a través del navegador y un cliente que ofrezca el entorno de visualización al usuario.

Todo el desarrollo de la práctica se realizará dentro del contenedor Docker facilitado. Dicho contenedor tiene instalado todo el software necesario para la realización de los distintos apartados.

El contenedor docker se debe desplegar con el siguiente comando:

```
docker run -it --name irac_p3 -p 8000:8000 -v
/home/vuestro.usuario/ficheros_p3/:/home/irac_p3/ficheros_p3
rstiupm/irac_p3:2025
```

Dentro del contenedor encontrareis:

- Bento4 (carpeta con el framework Bento4 instalado)
- gpac_public (carpeta con el framework gpac el cual contiene MP4Box)
- Binario de x264
- video_p3.mp4
- Librerías chart.min.js y dash.all.min.js
- Carpeta del usuario "ficheros_p3" montada como volumen para intercambiar ficheros entre el pc de laboratorio y el contenedor.

Importante: la carpeta "ficheros_p3" la debéis crear antes de lanzar el contenedor. En ella podéis depositar los diferentes ficheros "index.html" que se requieren para los apartados de la práctica.

Cuando creéis un index.html, comprobar que el contenedor docker lo ha mapeado en el path "/home/irac_p3/ficheros_p3" y en ese caso, moverlo al path "/home/irac_p3", ya que es el path donde se realizará la práctica y se ejecutará el servidor que ofrecerá el video en el puerto 8000.

La parte del servidor se implementará, dentro del contenedor Docker, mediante el comando ***"python3 -m http.server"*** [1]. Este comando permite arrancar un servidor web simple en el puerto 8000 que ofrecerá los recursos del directorio sobre el que se invoque. Por su parte, el cliente se basará en la herramienta dash.js [2], un *framework* basado en JAVASCRIPT que implementa los métodos necesarios para ofrecer salida de *streaming* de vídeo adaptable de alta calidad basado en la norma ISO MPEG-DASH.

Para una comprensión completa del funcionamiento de esta norma se plantea una práctica incremental basada en cuatro partes obligatorias y una opcional, dividida a su vez en dos sub-partes:

1. Introducción al cliente web dash.js;

2. Preparación de vídeos para ser ofrecidos mediante dash.js y visualización de métricas asociadas;
3. Gestión de derechos digitales a través de dash.js;
4. Visualización de métricas mediante gráficos;
5. Parte opcional: gestión dinámica de claves DRM y gestión DRM avanzada.

La evaluación de esta práctica se basará en la creación de una memoria donde se incluyan los resultados indicados en cada parte y los códigos y comandos utilizados para su realización, agrupados en diferentes directorios, uno por parte.

El código fuente (aplicación web) generado en cada sección de esta práctica debe ser **entregado a parte (en el mismo lote de entrega),** y no incluido en las memorias (se podrá crear un hilo en *github* si así se desea e incluirlo como referencia o apéndice a la memoria).

Parte 1: Introducción a dash.js

El objetivo de esta parte de la práctica es el despliegue y uso de la herramienta **dash.js** para ofrecer un servicio de *streaming* de vídeo básico. Para ello, se utilizará un vídeo codificado con diferentes calidades, lo que permitirá analizar el funcionamiento del protocolo DASH. En esta parte de la práctica se utilizará, aunque no es obligatorio, el video ofrecido por ***envivo*** a través del documento de presentación, o Media Presentation Document (MPD), alojado en la URL:

```
https://dash.akamaized.net/envivio/EnvivioDash3/manifest.mpd
```

Para la correcta visualización, deberéis crear un fichero “index.html” que incluya el siguiente código (la ubicación del fichero index.html debe estar en el path /home/irac_p3 del contenedor Docker, a la misma altura que el fichero dash.min.all.js) :

La etiqueta HTML necesaria para crear el elemento video es el siguiente:

```
<video class="dashjs-player" autoplay controls preload="auto">
 <source src="[MPD envivo]" type="application/dash+xml"/>
</video>
```

Por su parte, el código necesario para asociar *dashjs* a este elemento HTML de vídeo es el siguiente:

```
<script>
 var player;
 document.addEventListener("DOMContentLoaded", function () {
  init();
 });
 function init(){
  player = dashjs.MediaPlayerFactory.create(   document.querySelector(".dashjs-
player"));
 };
</script>
```

1.1. Resultados

Para la realización correcta de esta parte de la práctica se pide:

- Implementación del cliente web (index.html) para la visualización del vídeo ofrecido por *envivo* a través del documento MPD indicado previamente;
- Análisis y explicación del documento MPD indicando claramente los siguientes campos:
  - XML
  - MPD
  - Period
  - AdaptationSet
  - SegmentTemplate
  - Representation
- Captura de pantalla del cliente para visualizar el comportamiento de MPEG-DASH y su explicación (es necesario ejecutar el servidor python para poder acceder a localhost:8000 y ver el video).

Parte 2 – Generación de videos para *streaming* y visualización básica de métricas

El objetivo de esta parte es la generación de diferentes calidades de un mismo vídeo para ser ofrecido en *streaming* a través de la norma MPEG-DASH y la visualización de las métricas en el cliente web. Para ello se utilizarán las siguientes herramientas:

- X264[1] y MP4Box[2] para la configuración de diferentes resoluciones y *bitrates* de un video (x264) y su codificación final a MP4 (MP4Box);
- Framework Bento4 (mp4dash) para la división de cada video en segmentos de streaming y la creación del MPD.

2.1. Generación de diferentes configuraciones de video (resolución y *bitrate*)

Dependiendo del sistema operativo, la instalación de x264 y MP4Box puede variar. Una vez instaladas las herramientas se puede proceder a la creación de los videos.

El vídeo utilizado para esta parte está contenido en el contenedor docker (video_p3.mp4). Se debe generar diferentes codificaciones de éste mediante x264. Se recomienda generar, al menos, 3 configuraciones (estas configuraciones están creadas para forzar un funcionamiento determinado de la norma DASH y poder visualizar su comportamiento):

- Calidad baja con una resolución de 160x90 y bitrate de 100K;
- Calidad media con una resolución de 640x360 y bitrate de 600K;
- Calidad alta con una resolución de 1280x720 y bitrate de 2400K.

El comando utilizado para generar la configuración de baja calidad se especifica a continuación (se ha resaltado los puntos de interés que variarán dependiendo de la configuración):

[1] Página oficial de x264 accesible en https://www.videolan.org/developers/x264.html

[2] Página oficial de MP4Box accesible en https://gpac.wp.imt.fr/

```
[COMANDO DE EJECUCIÓN DE x264] --output low_config.264 --fps 24 --preset slow --
bitrate 100 --vbv-maxrate 4800 --vbv-bufsize 9600 --min-keyint 48 --keyint 48 --
scenecut 0 --no-scenecut --pass 1 --video-filter "resize:width=160,height=90"
video_original.mp4
```

PD: No copiar el comando directamente de la práctica, puede producir errores al pegarlo. Revisar correctamente la sintaxis del comando, así como los flags, guiones y espacios.

[COMANDO DE EJECUCIÓN DE X264] debe ser sustituido por el comando de ejecución de la herramienta x264 y que dependerá del sistema operativo, así como las opciones/parámetros3.

En este caso, el comando para lanzar desde el contenedor docker es "**./x264**". (ejecutado desde el path /home/irac_p3, en otro caso, utilizar el path absoluto)

Por su parte, Low_config.264 es el nombre del fichero 264 de salida usado en el ejemplo, se podrá dar el nombre que se desee respetando el tipo de fichero (264).

Una vez realizadas las 3 configuraciones es necesario codificarlas en MPEG. Para ello se utiliza el siguiente comando para cada una de ellas:

```
[COMANDO DE EJEC. MP4Box] -add [nombre_entrada].264 -fps 24 [nombre_salida].mp4
```

PD: No copiar el comando directamente de la práctica, puede producir errores al pegarlo. Revisar correctamente la sintaxis del comando, así como los flags, guiones y espacios.

De nuevo, [COMANDO DE EJEC. MP4BOX] hace referencia al comando de ejecución de la herramienta MP4Box y que dependerá del sistema operativo. Además, [nombre_entrada] variará dependiendo de los ficheros generados anteriormente (p.ej. para la configuración de baja calidad se utilizaría Low_config.264). Como nombre de salida se podría optar por Low_config.mp4, en el caso de estar gestionando la configuración de baja calidad.

En este caso, el comando para lanzar desde el contenedor docker es "**./gpac_public/bin/gcc/MP4Box**" (ejecutado desde el path /home/irac_p3, en otro caso utilizar el path absoluto)

Una vez terminado este paso, se debería disponer de 3 vídeos codificados en MPEG-4, uno para cada configuración (si se han generado más configuraciones se dispondrá de tantos vídeos MP4 como configuraciones).

2.2. Preparación de los vídeos para DASH y generación del fichero MPD

El siguiente paso se centra en la segmentación de cada vídeo mediante las herramientas ofrecidas por el *framework* **Bento4**. La instalación y uso de este *framework* puede ser estudiada en [3].

[3] El sistema operativo utilizado para la planificación de esta práctica ha sido UNIX y, por tanto, la elaboración de las guías/códigos mostrados pueden variar con respecto a otros sistemas operativos.

Bento4 está ya instalado en el contenedor docker, en la carpeta "**/home/irac_p3/Bento4**", cada grupo deberá estudiar las herramientas ofrecidas por este *framework* necesarias para realizar la segmentación de los vídeos codificados previamente y prepararlos así para su presentación mediante DASH (mp4fragmented, mp4dash, mp4info), las cuales se encuentran en forma de binarios en el path "**/home/irac_p3/Bento4/bin**".

Como resultado de este proceso se generará un directorio llamado output (si no se especifica lo contrario) cuyo contenido se detalla a continuación:

- Directorio video que contiene otro directorio (si sólo se ha codificado vídeo sin audio) denominado avc1. Este directorio contiene a su vez un directorio por cada uno de los ficheros de video segmentado (si no se especifica nombre se utilizan los números 1, 2 y 3 por defecto. Cada uno de estos directorios contienen, a su vez, un fichero de inicialización y un fichero por cada segmento.
- Documento MPD asociado a la segmentación

Figura 1. Estructura de directorios resultante del proceso de segmentación de 3 configuraciones de video.

2.3. Implementación del cliente de visualización y análisis de métricas

Finalmente se debe integrar lo aprendido en la primera parte con lo realizado en las secciones 2.1 y 2.2. mediante la implementación de un cliente web capaz de ofrecer el video codificado en *streaming*. Para ello se debe incluir en el fichero HTML index.html el siguiente código:

```
<div class="code">
 <video class="dashjs-player" autoplay controls preload="auto" muted>
</video>
</div>
 <div class="code">
  <p>Video Bitrate: <span id="bitrate"></span> kbps</p>
  <p>Video Buffer: <span id="buffer"></span> seconds</p>
  <p>Video Representation: <span id="representation"></span></p>
</div>
```

Esto nos permite crear una sección para alojar el reproductor de vídeo (<video>…</video>) y otra para presentar las métricas obtenidas (*bitrate*, *buffer* y representación visualizada).

Por su parte, el código JAVASCRIPT necesario para el correcto funcionamiento de la visualización del vídeo y de las métricas asociadas es el siguiente:

```
document.addEventListener("DOMContentLoaded", function () {
 init();
});

function init(){
 var video,
  player,
  mpd_url = "./output/stream.mpd";

 video = document.querySelector("video");
 player = dashjs.MediaPlayer().create();
 player.initialize(video, mpd_url, true);
 player.on(dashjs.MediaPlayer.events["PLAYBACK_ENDED"], function() {
  clearInterval(eventPoller);
 });

 var eventPoller = setInterval(function() {
  var streamInfo = player.getActiveStream().getStreamInfo();
  var dashMetrics = player.getDashMetrics();
  var dashAdapter = player.getDashAdapter();

  if (dashMetrics && streamInfo) {
   const periodIdx = streamInfo.index;
   var repSwitch = dashMetrics.getCurrentRepresentationSwitch('video', true);
   var bufferLevel = dashMetrics.getCurrentBufferLevel('video', true);
   var bitrate = repSwitch ? Math.round(dashAdapter.
    getBandwidthForRepresentation(repSwitch.to,
    periodIdx) / 1000) : NaN;
   document.getElementById('buffer').innerText = bufferLevel + " secs";
   document.getElementById('bitrate').innerText = bitrate + " Kbps";
   document.getElementById('representation').innerText = repSwitch.to;
  }
 }, 500);
}
```

De esta forma, el directorio debería disponer de una sección de ficheros relativa a las distintas configuraciones del vídeo a ofrecer y un fichero index.html, como se muestra en la siguiente figura:

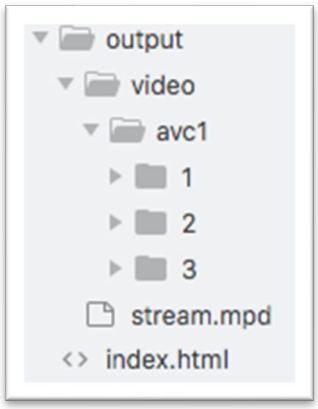

Figura 2. Estructura de directorios resultante la implementación de la parte 2 de la práctica.

2.4. Resultados

Los resultados obtenidos a lo largo de la realización de esta parte deben ser incluidos en la memoria de la práctica. De tal forma, que para la realización correcta de esta parte de la práctica se pide:

- Tabla que refleje las configuraciones de video utilizadas (calidad/nombre/bitrate/resolución/path)
- Codificación de un vídeo en, mínimo, tres calidades diferentes. Adjuntar los comandos ejecutados para la obtención de cada calidad.
- Preparación de los vídeos para su presentación DASH: explicación de los comandos/herramientas utilizados y explicación del documento MPD resultante;
- Captura de pantalla del inspector del cliente (pestaña de Red) para visualizar el comportamiento de MPEG-DASH y explicación de la captura con relación al fichero MPD;
- Captura de pantalla de los mensajes ofrecidos por el servidor y explicación general de dichos mensajes.
- Captura de la reproducción de cada uno de los videos generados.

Parte 3 – Gestión DRM con dash.js

El objetivo de esta última parte de la práctica se centra en afianzar el conocimiento adquirido con relación a la gestión de derechos digitales (DRM). Para ello se utilizarán las herramientas mp4fragment y mp4encrypt ofrecidas por el *framework* Bento4.

En primer lugar, se debe fragmentar cada una de las diferentes configuraciones del vídeo (calidad baja, media y alta).

Una vez finalizada la fragmentación se debe proceder con la encriptación de cada vídeo fragmentado mediante el comando mp4encrypt. Un ejemplo de uso de dicho comando es:

```
[Binario_mp4encrypt] --method MPEG-CENC --key
1:87237D20A19F58A740C05684E699B4AA:random --property
1:KID:A16E402B9056E371F36D348AA62BB749 --global-option mpeg-cenc.eme-pssh:true
```

PD: No copiar el comando directamente de la práctica, puede producir errores al pegarlo. Revisar correctamente la sintaxis del comando, así como los flags, guiones y espacios.

De esta forma se está encriptando [INPUT] mediante el método MPEG-CENC y definiendo la KEY y KID necesarias para la gestión del DRM como:

- 87237D20A19F58A740C05684E699B4AA
- A16E402B9056E371F36D348AA62BB749

La herramienta mp4info ejecutada sobre un vídeo nos ofrecerá toda la información asociada al fichero MP4. Un atributo importante de esta información es el número de *tracks* del fichero de vídeo, ya que se deberá añadir encriptación (KEY y KID) para cada uno de ellos. El ejemplo de uso de mp4encrypt anterior sólo contempla un *track*, por lo que el mismo ejemplo para un vídeo fragmentado con 2 *tracks* podría ser el siguiente:

```
[Binario_mp4encrypt] --method MPEG-CENC --key
1:87237D20A19F58A740C05684E699B4AA:random --property
1:KID:A16E402B9056E371F36D348AA62BB749 --key
2:87237D20A19F58A740C05684E699B4AA:random --property
2:KID:A16E402B9056E371F36D348AA62BB749 --global-option mpeg-cenc.eme-pssh:true
```

PD: No copiar el comando directamente de la práctica, puede producir errores al pegarlo. Revisar correctamente la sintaxis del comando, así como los flags, guiones y espacios.

Así sería posible aplicar diferentes claves para cada uno de los *tracks* del vídeo procesado.

Una vez encriptados todas las configuraciones de los vídeos, se deben procesar del mismo modo que en la parte anterior para su presentación mediante el protocolo DASH.

3.1. Implementación del cliente de visualización

El código necesario para procesar DRM en el cliente es similar al ya utilizado en las secciones anteriores, pero añadiendo lo relativo a la protección de la información. Para ello se debe crear una estructura de datos donde se especifiquen la KEY y la KID en cadenas de texto base 64 (*base64 text*). Para la traducción de hexadecimal a texto base 64 se puede utilizar la herramienta web CRYPTII, accesible en https://cryptii.com/pipes/binary-to-base64. Utilizando las claves especificadas anteriormente se obtendrían las siguientes cadenas de texto base 64 (y como se muestra también en la figura 3):

- 87237D20A19F58A740C05684E699B4AA → hyN9IKGfWKdAwFaE5pm0qg
- A16E402B9056E371F36D348AA62BB749 → oW5AK5BW43HzbTSKpiu3SQ

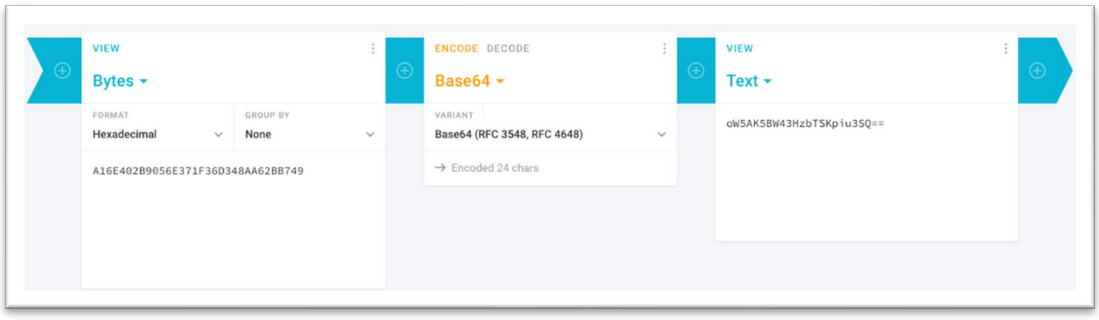

Figura 3. Ejemplo de codificación de la clave KID en texto Base 64

La estructura de datos mencionada y relativa a la protección DRM se puede construir mediante el siguiente código JAVASRIPT:

```
const protData = {
 "org.w3.clearkey": {
  "clearkeys": {
   "oW5AK5BW43HzbTSKpiu3SQ": "hyN9IKGfWKdAwFaE5pm0qg"
  }
 }
};
```

Finalmente, esta información de protección debe ser asociada al objeto *MediaPlayer* creado durante la inicialización (*player = dashjs.MediaPlayer().create();*). El siguiente código muestra cómo realizar esta asociación:

```
player.setProtectionData(protData);
```

Se recomienda utilizar Firefox en lugar de Chrome para realizar las pruebas de funcionamiento de esta parte.

3.2. Resultados

Los resultados obtenidos a lo largo de la realización de esta parte deben ser incluidos en la memoria de la práctica. De tal forma, que para la realización correcta de esta parte de la práctica se pide:

- Encriptación de un vídeo en, mínimo, tres calidades diferentes, y explicación del comando/herramienta utilizado para ello. Incluyendo las capturas de los comandos utilizados.
- Segmentación de los vídeos y explicación de los comandos/herramientas utilizados, así como del documento MPD resultante;
- Captura de pantalla del cliente para visualizar el comportamiento de MPEG-DASH con DRM;
- Realizar una alteración de las claves especificadas en protData para visualizar lo que ocurre cuando no se tiene acceso a un vídeo encriptado. Será necesario recoger los mensajes de error ofrecidos por el navegador (Firefox) a través de su consola.

Parte 4 – Visualización avanzada de métricas

Esta última parte obligatoria tiene como objetivo la mejora de la visualización de las métricas asociadas al *streaming* de video implementado hasta el momento. Cada grupo de práctica puede elegir si utilizar el código generado en la parte 2 o la parte 3 como base para esta sección. La captura mostrada en la Figura 4 muestra un ejemplo del resultado esperado en esta parte de la práctica. Para su realización se pueden utilizar librerías JAVASCRIPT desarrolladas ad-hoc para la implementación de gráficas como *ChartJS* (accesible en https://www.chartjs.org/) o *VisJS* (accesible en https://visjs.org/).

Figura 4. Ejemplo de visualización de métricas de streaming de video mediante gráficos en tiempo real.

Los resultados obtenidos a lo largo de la realización de esta parte deben ser incluidos en la memoria de la práctica. Para la realización correcta de esta parte de la práctica se pide:

- Implementación del cliente web con la visualización avanzada de las métricas DASH;
- Captura de pantalla del cliente para visualizar el comportamiento de las gráficas ante cambios de la calidad del vídeo ofrecido en *streaming* y su explicación.

Parte 5 – Opcional

El objetivo de esta parte OPCIONAL es profundizar en los conocimientos adquiridos a lo largo de la realización de la práctica **(La parte Opcional no admite resolución de dudas durante las sesiones de laboratorio).** Para ello se proponen las siguientes mejoras aplicables a los clientes implementados anteriormente:

- Utilización de otro modelo o método de encriptación para la gestión de DRM (widevine, PlayReady, etc.);
- Gestión dinámica de las claves de acceso a contenido DRM mediante la implementación de una API REST;

La memoria de esta parte opcional deberá contener una explicación detallada del trabajo realizado, tanto a nivel de diseño como de implementación, así como de los resultados obtenidos. ***El código fuente generado debe ser entregado a parte, no incluido en la memoria.***

5.1. Gestión de DRM avanzada

La gestión de DRM mediante modelos de encriptación como Widevine y PlayReady puede resultar complejo y se presenta como parte opcional de esta práctica. Sin embargo, existen recursos como los ofrecidos por Axinom en el repositorio github https://github.com/Axinom/drm-quick-start.

La realización de esta parte opcional quedará condicionada a una correcta implementación de un modelo de gestión DRM basado en, al menos, uno de los dos métodos de encriptación mencionado. De tal forma, la ejecución correcta de esta parte implica incluir los siguientes puntos en la memoria:

- Explicación del modelo de encriptación utilizado;
- Detalle del proceso de implementación con capturas probatorias del correcto funcionamiento del sistema DRM.

5.2. Gestión dinámica de claves de acceso a contenido DRM mediante API REST

La gestión básica de DRM realizada a lo largo de la parte 3 de esta práctica utiliza un conjunto de claves-valor fijo e incrustado en el código fuente del cliente. Una mejora para esta implementación es la inclusión de un servidor de licencias que ofrezca, a cada usuario dado de alta, la posibilidad de acceder al *streaming* de video protegido mediante un intercambio dinámico de claves.

La realización de esta parte opcional quedará condicionada a una correcta implementación de un modelo de intercambio de claves dinámico para un modelo DRM tipo ClearKey. De esta forma, la ejecución correcta de esta parte implica incluir los siguientes puntos en la memoria:

- Explicación del modelo de intercambio de claves propuesto y la tecnología empleada para la implementación;
- Captura de pantalla probatorias del correcto funcionamiento de la implementación.

Referencias

[1] Mozilla, «¿Cómo se configura un servidor de prueba local?,» [En línea]. Available: https://developer.mozilla.org/es/docs/Learn/Common_questions/set_up_a_local_testing_server.

[2] DASH Industry Forum, «dash.js Wiki,» DASH Industry Forum, [En línea]. Available: https://github.com/Dash-Industry-Forum/dash.js/wiki. [Último acceso: 2019].

[3] Bento4, "MPEG DASH Adaptive Streaming," Bento4, [Online]. Available: https://www.bento4.com/developers/. [Accessed 2019].