Integration of Networks, Applications and Contents
Practice 3: DASH

The development of a web application to provide basic video streaming service is divided into two parts: the creation of a server that organizes access to resources through the browser, and a client that offers the viewing environment to the user.

All the development for the practice will be carried out within the provided Docker container. This container has all the necessary software installed for the completion of the different sections.

The Docker container must be deployed with the following command:

```
docker run -it --name irac_p3 -p 8000:8000 -v
/home/your.user/files_p3/:/home/irac_p3/files_p3
rstiupm/irac_p3:2025
```

Inside the container you will find:

- Bento4 (folder with the Bento4 framework installed)
- gpac_public (folder with the gpac framework which contains MP4Box)
- x264 binary
- video_p3.mp4
- chart.min.js and dash.all.min.js libraries
- User folder "files_p3" mounted as a volume to exchange files between the laboratory PC and the container.

Important: you must create the "files_p3" folder before launching the container. You can deposit the different "index.html" files required for the practice in it.

When you create an index.html, verify that the Docker container has mapped it to the path "/home/irac_p3/files_p3" and in that case, move it to the path "/home/irac_p3", as it is the path where the practice will be carried out and the server that will offer the video on port 8000 will be executed."

The server-side will be implemented inside of docker container using the "**python3 -m http.server**" command [1]. This command allows starting a simple web server on port 8000 that will offer the resources of the directory on which it is invoked.On the other hand, the client will be based on the dash.js tool [2], a JavaScript-based framework that implements the necessary methods to offer high-quality adaptive video streaming output based on the ISO MPEG-DASH standard.

For a complete understanding of the functioning of this standard, an incremental practice based on four mandatory parts and an optional one is proposed, divided into two sub-parts:

1. Introduction to the dash.js web client;

2. Preparation of videos to be offered through dash.js and viewing associated metrics;
3. Management of digital rights through dash.js;
4. Viewing metrics through graphs;
5. Optional part: dynamic DRM key management and advanced DRM management.

The evaluation of this practice will be based on the creation of a report that includes the results indicated in each part, as well as the codes and commands used for their implementation, grouped into different directories, one for each part.

The source code (web application) generated in each section of this practice should be delivered separately (in the same batch of delivery), and not included in the reports (a thread can be created on Github if desired and included as a reference or appendix to the report)."

Part 1: dash.js introduction

The objective of this part of the practice is the deployment and use of the dash.js tool to offer a basic video streaming service. To do this, a video encoded with different qualities will be used, which will allow analyzing the operation of the DASH protocol. In this part of the practice, although not mandatory, the video offered by Envivo through the presentation document or Media Presentation Document (MPD) hosted at the following URL will be used:

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

For correct visualization, you must create a file “index.html” that includes the following code (the location of the index.html file must be in the path /home/irac_p3 of the Docker container, at the same height as the dash.min file .all.js) :

The HTML tag necessary to create the video element is the following:

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

On the other hand, the code necessary to associate dashjs with this HTML video element is the following:

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

1.1. Results

For the correct execution of this part of the practice, the following is required:

- Implementation of the web client (index.html) for viewing the video offered by envivo through the previously indicated MPD document;
- Analysis and explanation of the MPD document clearly indicating the following fields:
    - XML
    - MPD
    - Period
    - AdaptationSet
    - SegmentTemplate
    - Representation
- Screenshot of the client to visualize the behavior of MPEG-DASH and its explanation (it is necessary to run the python server to access localhost:8000 and see the video).

Part 2 – Generation of videos for streaming and basic display of metrics

The objective of this part is to generate different qualities of the same video to be offered in streaming through the MPEG-DASH standard and the visualization of metrics on the web client. The following tools will be used for this purpose:

- X264[1] y MP4Box[2] for configuring different resolutions and bitrates of a video (x264) and its final encoding to MP4 (MP4Box);
- Bento4 Framework (mp4dash) for splitting each video into streaming segments and creating the MPD.

2.1. Generation of different video configurations (resolution and bitrate)

Depending on the operating system, the installation of x264 and MP4Box may vary. Once the tools are installed, the video creation process can be initiated.

The video used for this part is contained in the docker container (video_p3.mp4). Different encodings of this must be generated using x264. It is recommended to generate at least 3 configurations (these configurations are created to force a certain operation of the DASH standard and to be able to visualize its behavior)

- Low quality with a resolution of 160x90 and a bitrate of 100K;
- Medium quality with a resolution of 640x360 and a bitrate of 600K;
- High quality with a resolution of 1280x720 and a bitrate of 2400K.

The command used to generate the low quality configuration is specified below (the points of interest that will vary depending on the configuration are highlighted):

[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
```

PS: Do not copy the command directly from the practice, it may cause errors when pasting it. Correctly review the syntax of the command, as well as flags, hyphens, and spaces.

[X264 RUN COMMAND] should be replaced by the x264 tool run command and it will depend on the operating system as well as the options/parameters.

In this case, the command to launch from the docker container is "**./x264**". (executed from the path /home/irac_p3, otherwise use the absolute path)

On the other hand, Low_config.264 is the name of the output 264 file used in the example, and any desired name can be given while respecting the file type (264).

Once the 3 configurations are created, it is necessary to encode them in MPEG. To do this, the following command is used for each of them:

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

PS: Do not copy the command directly from the practice, it may cause errors when pasting it. Correctly review the syntax of the command, as well as flags, hyphens, and spaces.

Again, "[MP4BOX EXECUTION COMMAND]" refers to the execution command for the MP4Box tool, which will depend on the operating system. In addition, "[input_name]" will vary depending on the files generated previously (e.g. for the low-quality configuration, Low_config.264 would be used). As an output name, one could choose Low_config.mp4, for example, when working with the low-quality configuration.

In this case, the command to launch from the docker container is "**./gpac_public/bin/gcc/MP4Box**" (executed from the path /home/irac_p3, otherwise use the absolute path)

Once this step is completed, there should be 3 MPEG-4 encoded videos, one for each configuration (if more configurations were generated, there will be as many MP4 videos as configurations).

2.2. Preparation of videos for DASH and generation of the MPD file

The next step focuses on the segmentation of each video using the tools provided by the Bento4 framework. The installation and use of this framework can be studied in [3].

Bento4 is already installed in the docker container, in the folder "/home/irac_p3/Bento4", each group must study the tools offered by this framework necessary to segment the previously

encoded videos and prepare them for presentation using DASH (mp4fragmented, mp4dash, mp4info), which are found in binary form in the path "/home/irac_p3/Bento4/bin".

As a result of this process, a directory called output (unless otherwise specified) will be generated, whose contents are detailed below:

- A directory named video that contains another directory (if only video without audio has been encoded) called avc1. This directory in turn contains a directory for each of the segmented video files (if no name is specified, the default numbers 1, 2, and 3 are used). Each of these directories contains, in turn, an initialization file and a file for each segment.
- Associated MPD document for the segmentation.

Figure 1. Directory structure resulting from the video segmentation process of 3 different configurations.

2.3. Implementation of the client for visualization and analysis of metrics

Finally, what was learned in the first part must be integrated with what was done in sections 2.1 and 2.2 through the implementation of a web client capable of offering the encoded video in streaming. To do this, the following code must be included in the HTML file index.html:

```
<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>
```

This allows us to create a section to host the video player (<video>…</video>) and another one to display the obtained metrics (bitrate, buffer, and visualized representation).

On the other hand, the necessary JAVASCRIPT code for the correct functioning of the video visualization and associated metrics is the following:

```
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);
}
```

In this way, the directory should have a section of files related to the different video configurations to be offered and an index.html file, as shown in the following figure:

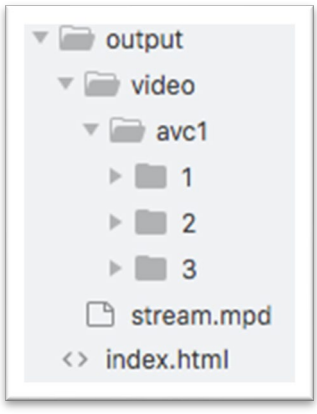

Figure 2. Directory structure resulting from the implementaion of part 2 of the practice.

2.4. Results

The results obtained throughout the completion of this part should be included in the practical report. Thus, for the correct completion of this part of the practical, the following is required:

- Table that reflects the video configurations used (quality/name/bitrate/resolution/path)
- Encoding a video in at least three different qualities. Attach the commands executed to obtain each quality.
- Preparation of the videos for DASH presentation: explanation of the commands/tools used and explanation of the resulting MPD document;
- Screenshot of the client inspector (Network tab) to visualize the behavior of MPEG-DASH and explanation of the capture in relation to the MPD file;
- Screenshot of the messages offered by the server and general explanation of said messages.
- Capture of the playback of each of the generated videos

Part 3 –DRM management with dash.js

The objective of this last part of the practice focuses on consolidating the knowledge acquired regarding digital rights management (DRM). To do so, the mp4fragment and mp4encrypt tools offered by the Bento4 framework will be used.

Firstly, each of the different video configurations (low, medium, and high quality) must be fragmented. Once the fragmentation is completed, each fragmented video must be encrypted using the mp4encrypt command. An example of using this command is:

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

PS: Do not copy the command directly from the practice, it may cause errors when pasting it. Correctly review the syntax of the command, as well as flags, hyphens, and spaces

In this way, [INPUT] is being encrypted using the MPEG-CENC method and defining the necessary KEY and KID for DRM management as:

- 87237D20A19F58A740C05684E699B4AA
- A16E402B9056E371F36D348AA62BB749

Executing the mp4info tool on a video file provides us with all the information associated with the MP4 file. An important attribute of this information is the number of tracks in the video file, as encryption (KEY and KID) must be added for each of them. The previous example of mp4encrypt usage only considers one track, so the same example for a video fragment with 2 tracks could be the following:

```
[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
```

PS: Do not copy the command directly from the practice, it may cause errors when pasting it. Correctly review the syntax of the command, as well as flags, hyphens, and spaces

This would make it possible to apply different keys for each of the tracks in the processed video.

Once all video configurations have been encrypted, they must be processed in the same way as in the previous section for their presentation using the DASH protocol.

3.1. Implementation of the visualization client

The code necessary to process DRM on the client is similar to that used in previous sections, but with the addition of information protection. To do this, a data structure must be created specifying the KEY and KID as base64 text strings. The hexadecimal to base64 text conversion can be done using the web tool CRYPTII, accessible at https://cryptii.com/pipes/binary-to-base64. Using the keys specified above, the following base64 text strings would be obtained (as also shown in Figure 3):

- 87237D20A19F58A740C05684E699B4AA → hyN9IKGfWKdAwFaE5pm0qg
- A16E402B9056E371F36D348AA62BB749 → oW5AK5BW43HzbTSKpiu3SQ

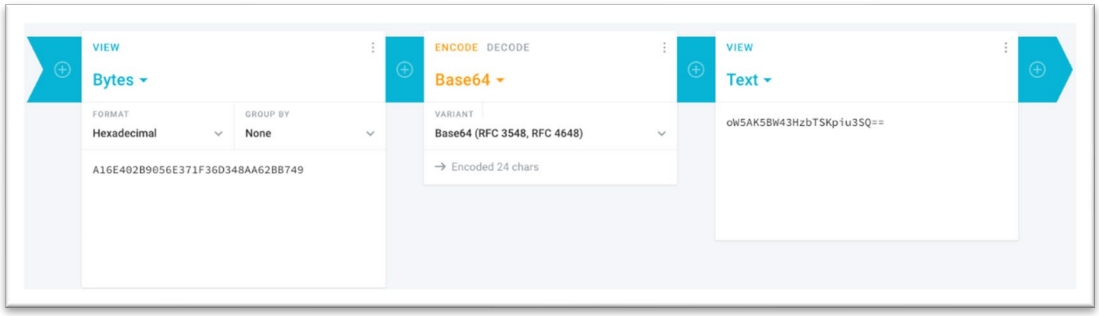

Figure 3. Example of encoding the KID key in Base 64 text

The mentioned data structure related to DRM protection can be built using the following JAVASCRIPT code:

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

Finally, this protection information must be associated with the MediaPlayer object created during initialization (player = dashjs.MediaPlayer().create();). The following code shows how to perform this association:

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

It is recommended to use Firefox instead of Chrome to perform the functionality tests of this part.

3.2. Results

The results obtained throughout the completion of this part must be included in the practice report. Thus, in order to successfully complete this part of the practice, the following is required:

- Encryption of a video in at least three different qualities, and an explanation of the command/tool used for this;
- Segmentation of the videos and explanation of the commands/tools used, as well as the resulting MPD document;
- Screenshot of the client to visualize the behavior of MPEG-DASH with DRM;
- Make an alteration to the keys specified in protData to see what happens when access to an encrypted video is not available. It will be necessary to collect error messages offered by the browser (Firefox) through its console.

Part 4 – Advanced metrics visualization

This mandatory last part aims to improve the visualization of the metrics associated with the video streaming implemented so far. Each practice group can choose whether to use the code generated in part 2 or part 3 as a basis for this section. The screenshot shown in Figure 4 shows an example of the expected result in this part of the practice. For its realization, JavaScript libraries developed ad hoc for the implementation of graphs such as ChartJS (accessible at https://www.chartjs.org/) or VisJS (accessible at https://visjs.org/) can be used.

Figure 4. Example of visualization of video Streaming metrics through real-time graphics.

The results obtained throughout the completion of this part must be included in the practice report. To successfully complete this part of the practice, the following is required:

- Implementation of the web client with advanced DASH metric visualization;
- Screenshot of the client to visualize the behavior of the charts when changing the quality of the video offered in streaming and its explanation.

Part 5 – Optional

The objective of this OPTIONAL part is to delve deeper into the knowledge acquired throughout the practice **(The Optional part does not allow resolution of doubts during the laboratory sessions).** To do so, the following improvements applicable to the clients implemented previously are proposed:

• Using another encryption model or method for DRM management (widevine, PlayReady, etc.);
• Dynamic management of access keys to DRM content through the implementation of a REST API;

The report for this optional part must contain a detailed explanation of the work carried out, both at the design and implementation levels, as well as the results obtained. The generated source code should be submitted separately, not included in the report.

5.1. Advanced DRM management

The management of DRM using encryption models such as Widevine and PlayReady can be complex and is presented as an optional part of this practice. However, resources such as those offered by Axinom in the github repository https://github.com/Axinom/drm-quick-start are available. The completion of this optional part will be conditioned on a correct implementation of a DRM management model based on at least one of the two mentioned encryption methods. Thus, the correct execution of this part implies including the following points in the report:

- Explanation of the encryption model used;
- Details of the implementation process with probative screenshots of the correct functioning of the DRM system.

5.2. Dynamic management of Access keys for DRM content through REST API

The basic DRM management implemented throughout Part 3 of this lab uses a fixed set of key-value pairs embedded in the client's source code. An improvement for this implementation is the inclusion of a license server that offers each registered user the ability to access the protected video streaming through a dynamic exchange of keys. The completion of this optional section will be conditioned to a correct implementation of a dynamic key exchange model for a ClearKey DRM model. Thus, the correct execution of this part implies including the following points in the report:

- Explanation of the proposed key exchange model and the technology used for implementation;
- Captured screenshots proving the correct functioning of the implementation.

References

[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].