End-to-End DASH Platform including a Network-based and Client-based Adaptive Quality Switching Module

David Gómez[1], Fernando Boronat[1], Mario Montagud[1,2], Clara Luzón[1]
[1]Universitat Politècnica de València (UPV) – Campus de Gandia, Spain
Immersive Interactive Media (IIM) R&D
[2]Centrum Wiskunde & Informatica (CWI), The Netherlands
{dagomi@alumni., fboronat@dcom., mamontor@, clalual@epsg.}upv.es

ABSTRACT
This paper presents an end-to-end Dynamic Adaptive Streaming over HTTP (DASH) platform, developed by using open-source components. It includes and allows the configuration of all the required steps along the end-to-end media delivery chain, from the encoding, segmentation and storage of the media content at the server side, to the delivery and adaptive consumption of the media content at the client side. A key component of this platform is the DASH client, developed using the GStreamer framework. It includes a module with a novel adaptive algorithm for switching between the available representations (i.e., qualities) of the media content at the server side, based on the available bandwidth and on internal conditions and features of the client (such as the buffer occupancy level, the battery level, its charging state and the CPU load). It also includes different modules to simulate specific values for these parameters, and to visualize their values and the value of the selected quality in real-time. A preliminary evaluation has proved the satisfactory performance of the overall platform. Finally, links to demo videos are provided to demonstrate the satisfactory performance and capabilities of this platform.

CCS Concepts
• **Information systems → Information systems applications → Multimedia information systems → Multimedia streaming**

Keywords
Adaptive Streaming; DASH; HTTP-based Adaptive Streaming (HAS); GStreamer; QoS; QoE.

1. INTRODUCTION
Broadband media delivery is exponentially gaining momentum. Consumers, wherever they are and whichever the device or access technology they are using, demand high-quality media experiences (e.g., in terms of high resolution, low start-up delays, continuous playout, interaction features, trick mode support…). To ensure these demands, the whole media delivery ecosystem needs to be optimized.

Recently, a new wave of client-driven HTTP-based Adaptive Streaming (HAS) solutions have been specified with the goal of improving the adaptability and continuity of media playout, based on the changing network conditions and on the capabilities, resources and state of the consumption devices. Their main advantages are adaptability, scalability, reliability, reachability and cost efficiency. The basic concept of HAS solutions is to provide multiple versions (aka *representations*) of the media content (e.g., in different resolutions or bitrates) and chop each of these versions into a sequence of (small) segments (e.g., with a duration of 2s). These different segments can be decoded and consumed independently of the other segments. In addition, an index or manifest file is created, which contains the required metadata information to describe the relationships between the segments and the available representations.

Based on these resources, the client, by means of HTTP requests, will firstly download the manifest file and, after that, will dynamically decide which segment (of which representation) to download in each moment, based on the information contained in the manifest file and on its current specific context (e.g., available bandwidth or BW). Typically, the client will select, for each segment request, the highest quality possible to ensure the continuity of the playout, according to the (changing) available resources. A key aspect of HAS clients is the design of the algorithm to adaptively determine which segment to download in each iteration. It is commonly known as *adaptation algorithm*, and its behavior has a direct influence on the system performance and on the perceived Quality of Experience (QoE).

From the available HAS solutions, Dynamic Adaptive Streaming over HTTP (DASH) [1] is highlighted, as it is a standard solution specified by ISO/IEC MPEG [2], which has been adopted by TV standards (e.g., in HbbTV [3]), by mobile broadcasting standards (e.g., eMBMS [4]), and by popular media services. In [2], the format of segments and the manifest file in DASH are specified. In this case, the manifest file is an XML-based file, called Media Presentation Description (MPD).

The existing DASH clients mostly monitor the network conditions to adapt the quality switching process. Different works have proposed adaptation algorithms to optimize this process (e.g., [5]). Moreover, especially in mobile streaming scenarios, other (variable) environmental conditions can play a key role in this optimization process. For example, the works in [6] and in [7] have taken into account the location of the devices to plan future quality requests. Apart from the location of the device, the work in [6] takes into account its speed and surrounding humidity. Furthermore, the work in [8] proposes a system that leverages the availability of sensing capabilities of mobile devices to reduce the network load in situations in which non-optimal watching conditions exist (by sensing the device shakiness and the ambient noise) or users are not paying attention to the video watching experience (by using the front-facing camera to integrate a face detection and tracking functionality).

MMSys'16, May 10–13, 2016, Klagenfurt, Austria.
ACM 978-1-4503-4297-1/16/05.
DOI: http://dx.doi.org/10.1145/2910017.2910638

Unlike the previous works which have extended existing DASH clients, this paper focuses on both the server and client sides, presenting an end-to-end DASH platform, developed by using open-source components. This platform includes and allows the configuration of all the steps along the end-to-end media delivery chain, from the generation, encoding, segmentation and storage of the media content, as well as the creation of the MPD, at the server side, to the delivery and adaptive consumption of the media content at the client side. A key component of this platform is the DASH client, developed using the GStreamer framework [9], which includes a module with an adaptive quality switching algorithm. This algorithm takes into account key network-based and client-based parameters (some of them not considered in other works), such as the available BW and various internal parameters of the client (e.g., buffer occupancy level, battery level, its charging state and CPU load). Apart from the DASH player, the client also includes a Graphical User Interface (GUI) with a configuration module to set the allowable thresholds for these parameters, a simulation module to dynamically force specific values to these parameters, and a representation module that allows a real-time visualization of these values and the selected quality in text labels, overlays and graphs.

A preliminary evaluation proves the satisfactory performance of the overall platform, when forcing specific values to the involved parameters. Links to demo videos are also provided to show the satisfactory performance and capabilities of this platform.

During the demo session, the attendees will be able to interact with our platform, will be guided in its use, and will be able to check how it performs in different situations. We also expect to get valuable feedback about its applicability, usability, design aspects and future functionalities.

2. END-TO-END DASH PLATFORM

This section describes the two main parts of the presented DASH platform: the server side and the client side. An overview of the platform and the HTTP-based communication process between the two parts is shown in Figure 1.

2.1 Server Side

The server side comprises four main processes (summarized in Figure 2). First, an input media file (whatever its format) is encoded into the supported codecs (e.g., H.264 or H.265 for video and AAC for audio) in DASH, in different representations, which are then encapsulated in MP4 format. Second, the representations are segmented. Third, the MPD is created.

The first three processes are automatically executed by running a developed Python script, called *DASH-Creator*, which internally makes use of:

- *FFmpeg*[1]: it is responsible of the encoding process. It allows configuring the codec settings, Group of Picture (GoP) settings, allowable resolutions and bitrates, frames per second (*fps*), among other relevant settings.
- *Bento4*[2]: it is responsible of creating segments of a specific duration, also according to the GoP and *fps* settings specified in the previous process.
- *MPD-Creator*: it is a Python script developed from scratch, by using the *xml.etree.ElementTree* library[3], to create the MPD, based on the DASH specification and on the settings of the previous two processes.

The *DASH-Creator* script performs a complete iteration for each of the first two processes for each one of the targeted representations and, once all the representations have been segmented, creates the MPD using *MPD-Creator*.

In order to assist non-expert users in these processes, a GUI for *DASH-Creator* has been designed. A demo video showing its performance can be watched at: https://goo.gl/IxMxWL

The fourth process consists of storing the DASH content and MPD file on an HTTP server (e.g., Apache).

2.2 Client Side

An overview of the GStreamer pipeline (i.e., chain of elements) required to create the DASH client can be seen in Figure 3, while a brief summary of each element's purpose is provided in Table 1.

The DASH client has been implemented using a combination of different programming languages. First, C, complemented with GObject and Glib libraries, which are the core languages in which the GStreamer backend is implemented, have been used to modify the source code of specific GStreamer elements (explained later). Second, the GStreamer pipeline and the adaptive quality switching module have been implemented by using the Gtreamer Python bindings (*Gst-Python*). Third, the different modules of the provided GUI have been implemented by using PyGtk.

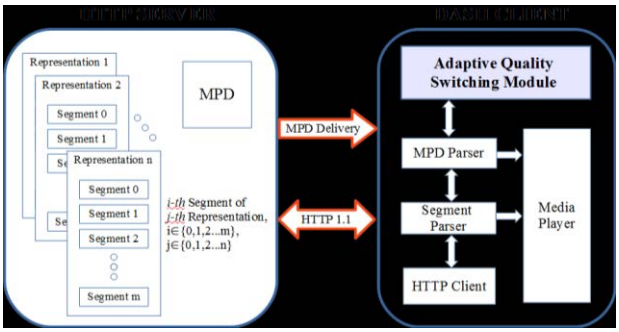

Figure 1. Overview of the End-to-End DASH Platform.

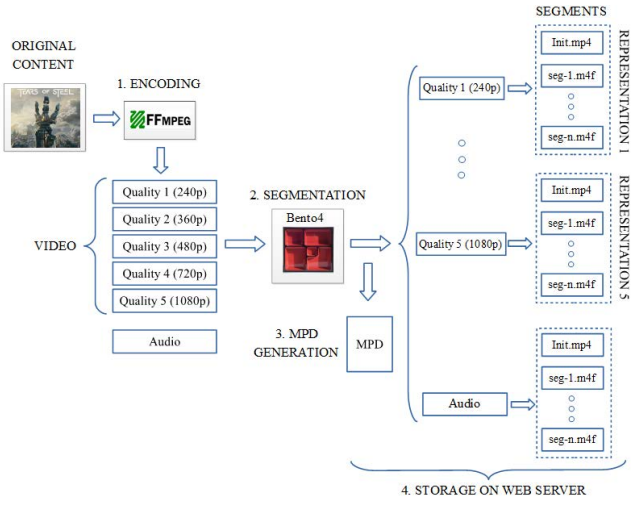

Figure 2. DASH Content Preparation and Storage.

[1] https://ffmpeg.org/

[2] https://www.bento4.com/

[3] https://docs.python.org/2/library/xml.etree.elementtree.html

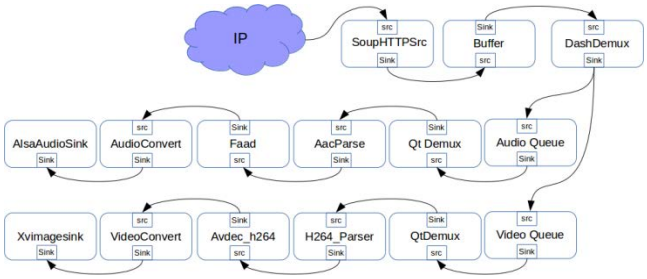

Figure 3. GStreamer pipeline for the DASH CLIENT.

Table 1. GStreamer elements used to create the DASH Client

Element	Description
SoupHttpSrc	Data Reception over the network via HTTP
Buffer / Queue	Queue element that acts as a reception buffer for a specific (configurable) number of (DASH) segments
DashDemux	It does not act as a traditional de-multiplexer (demuxer), instead it analyzes the MPD and requests the most proper representation, based on the available BW. Then, it exposes the incoming segments to the actual demuxers.
Audio / Video Queues	Queues are commonly used as buffers, but can also be used after demux elements to force independent execution threads for each branch, allowing for a better sync between the data in each branch.
QtDemux	Demux to extract (QuickTime) H264 and AAC components from an incoming MP4 stream
H264_Parser	H264 video stream parser
Avdec_h264	H264 video decoder
Xvimagesink	Rendering of raw video frames to an output window
Aac Parse	AAC audio stream parser
Faad	(Free) Audio AAC decoder
Alsasink	Output raw audio to a sound card

A reception buffer has been added to the DASH client. Its capacity can be set in bytes, seconds or in number of segments. The latter option has been chosen. The buffer plays a key a role at the beginning of the media session, as it is used to perform a pre-buffering process, which means that media playout will not start until a target (configurable) buffer fullness level is reached. This contributes to a smoother playout during the media session's lifetime. During the pre-buffering process, the lowest quality will be requested in order to minimize the start-up delay.

In addition, the client includes a module integrating an adaptive algorithm for switching between the available representations of the media content at the server side. For that purpose, the source code of the Dashdemux element (Table 1) has been slightly modified to allow for a more complete analysis and interpretation of the MPD and to be able to take into account other parameters than the available BW in the quality switching process.

Concretely, this algorithm takes into account the following key network-based and client-based parameters:

- CPU Load. If the CPU load exceeds a specific threshold (α), a lower quality will be requested.
- Battery Level and its Charging State. When the battery is not being charged, if its level is below a specific threshold (β), a lower quality will be requested, and if its level is even lower than a more restrictive threshold (ε), the lowest quality will be requested.
- *Buffer Occupancy*. If the buffer occupancy is lower than a specific threshold (δ), a lower quality will be requested to prevent from buffer underflow situations. Buffer overflow situations cannot occur in the developed platform, because the DashDemux element only requests for a new segment when an outgoing segment is sent to the decoders. That is the reason why an upper threshold has not been considered.
- Available BW or effective throughput. Rather than the available BW, as in other works, the adaptive quality switching module measures the effective throughput by calculating the time interval needed to download a segment of a specific size. Besides, a (configurable) sliding window of S segments can be used to average this calculation, thus somehow overcoming short term BW fluctuations. The bitrate of the selected quality will be always equal or lower than the measured effective throughput.

The algorithm is computed for each segment request. Its flow chart can be seen in Figure 4. On the one hand, it adopts a slow decrease adaptation strategy to prevent from abrupt adjustments (except when the battery level is below ε), which may lead to QoE degradation. In case that bad conditions persist, a lower quality will be requested in the next iteration. On the other hand, if all the conditions are good, the algorithm requests the best quality possible according to the available BW.

The client side also includes a GUI with: 1) a DASH player; 2) a configuration module to set the thresholds for each of the considered parameters; 3) a simulation module that allows dynamically forcing specific values for each of these parameters; and 4) a representation module that allows a real-time visualization of the evolution of these values and of the selected representation via text labels, screen overlays and graphs. A screen capture of the DASH client can be seen in Figure 5, while a demo video showing its performance can be watched at: https://goo.gl/oEig1v

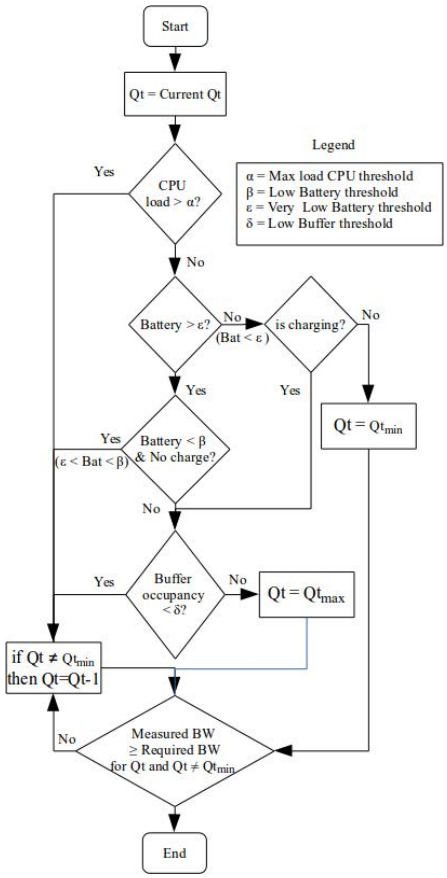

Figure 4. Flow chart of the Quality Adaptation Algorithm.

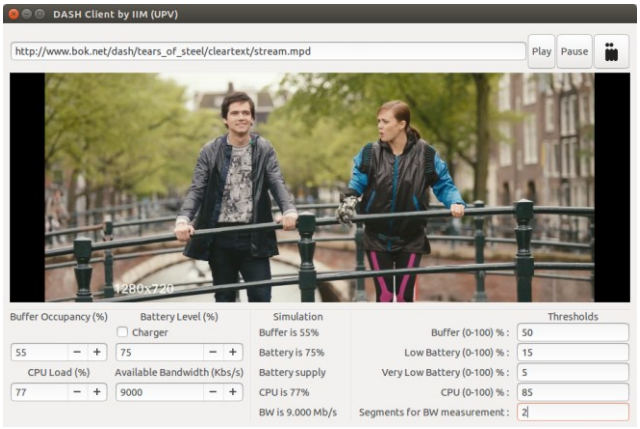

Figure 5. Screen Capture of the DASH Client.

3. EVALUATION

This section presents the results of a preliminary evaluation to show the satisfactory behavior and performance of the developed platform. For the evaluation, the *Teers of Steel* open movie[4] has been used as the input media file, and it was encoded into five video representations (H.264 codec, 24 *fps*, and resolutions of 240p, 360p, 480p, 720p and 1080p) and one audio representation (AAC codec with a sampling rate of 44100 Hz). The segments' length was set to 3s. The values for the different thresholds were set to: α=85%, β=15%, ε=5%, δ=50% and S=2. In order to check how the platform performs in the presence of critical situations, the evolution of the values of the considered parameters was simulated, forcing extreme values (exceeding the allowable thresholds) at some instants. The simulation can be done either by manually and dynamically setting these values through the controls provided in the DASH client (see Figure 5) during the session evolution or by preparing and loading text-based traces with the targeted values at the beginning of the session.

Figure 6 shows the evolution of the selected quality, represented by the associated bitrates, according to the BW availability. It can be seen that the algorithm adaptively selects the best quality (bitrate) possible without exceeding the available BW. In addition, by setting S>1 (in this case S=2), the algorithm can perform better when short term BW fluctuations occur (see the interval between seconds *90-th* and *100-th* in Figure 6), but at the cost of a slower reaction if these BW variations persist. Figure 7 shows the evolution of the selected quality (bitrate) according to the evolution of the considered parameters, assuming an unlimited BW availability (the evaluation was performed in a LAN scenario). It can be seen that the algorithm performs as expected when their values exceed the allowable (configured) thresholds.

4. CONCLUSION & FUTURE WORK

We believe that our platform is an outstanding contribution and that it will be relevant for interested researchers, practitioners and trainees, thus having potential for academic purposes. Several objectives are planned for future work. First, we want to determine the dependences between the considered parameters and the impact that the selected representation has on each of them. Second, we plan to explore the suitability of considering additional features and conditions as input parameters to the adaptive algorithm (e.g., device stability, screen resolution, ambient noise, aspects that somehow reflect the users' attention…). Moreover, we also plan to refine the quality adaptation algorithm, based on the previous tasks and on the insights from objective (QoS) and subjective (QoE) evaluations we will conduct. Finally, we plan to include network-aware strategies to take into account situations in which multiple clients are requesting the same content and/or competing for BW.

[4] https://mango.blender.org/

5. ACKNOWLEDGMENTS

This work has been funded, partially, by the "Fondo Europeo de Desarrollo Regional (FEDER)" and the Spanish Ministry of Economy and Competitiveness, under its R&D&I Support Program, in project with reference TEC2013-45492-R.

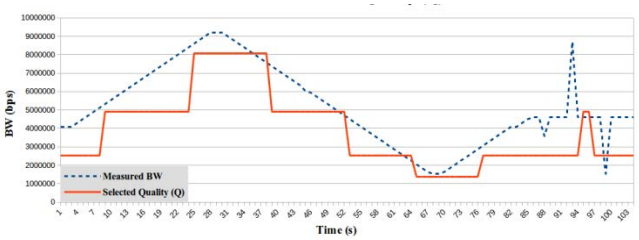

Figure 6. Selected Quality according to the BW availability.

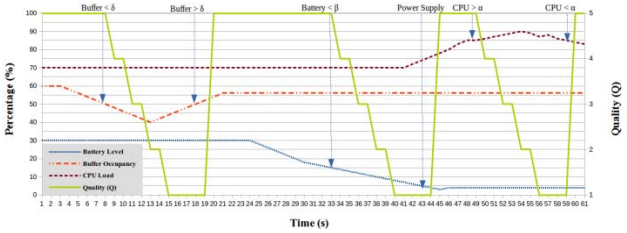

Figure 7. Selected Quality according to the Input Parameters.

6. REFERENCES

[1] Stockhammer, T. Dynamic Adaptive Streaming over HTTP: Standards and Design Principles. ACM MMSYS 2011, San Jose, California (USA), February 2011.

[2] ISO/IEC 23009-1: 2012. Information Technology. Dynamic Adaptive Streaming over HTTP (DASH). Part 1: Media Presentation Description and Segment Formats. April 2012.

[3] ETSI TS 102 796 V1.3.1, Hybrid Broadcast Broadband TV, www.hbbtv.org, November 2015.

[4] Lecompte, D., Gabin, F. *Evolved multimedia broadcast/multicast service (eMBMS) in LTE-advanced: overview and Rel-11 enhancements*. IEEE Communications Magazine, 50(11), pp.68-74, November 2012.

[5] Wang, X., et al. 2013. AMES-Cloud: A Framework of Adaptive Mobile Video Streaming and Efficient Social Video Sharing in the Clouds. IEEE Transactions on Multimedia, 15(4), 811–820, 2013.

[6] Han, D., et al. 2013. MASERATI: Mobile Adaptive Streaming Based on Environmental and Contextual Information. ACM WiNTECH, Miami(USA), October 2013.

[7] Hao, J., Zimmermann, R., and Ma, H. 2014. GTube: Geo-predictive Video Streaming over HTTP in Mobile Environments. ACM MMSYS 2014, Singapore, March 2014.

[8] Wilk, S., et al. 2015. EnvDASH: An Environment-Aware Dynamic Adaptive Streaming over HTTP System. ACM TVX '15. Brussels (Belgium), June 2015.

[9] Gstreamer Framework, http://gstreamer.freedesktop.org/