TV graphics personalization using in-band events*

M. Ibrahim
Ericsson Research
P.O. Box 52134
Germany
mohamed.a.ibrahim@ericsson.com

T. Lohmar
Ericsson Research
P.O. Box 52134
Germany
thorsten.lohmar@ericsson.com

A. El-Essaili
Ericsson Research
P.O. Box 52134
Germany
ali.el.essaili@ericsson.com

A. d'Allonnes
Ericsson broadcast services
P.O. Box 92100
France
aurelien.revault@ericsson.com

ABSTRACT

This paper presents the concepts of overlaying personalized TV[1] graphics on the device side, controlled and triggered by Dynamic Adaptive Streaming over HTTP (DASH) in-band events. TV graphics personalization helps engaging viewers in the program and maximize the value of information shown to them. Today's TV graphics are encoded within the video making it difficult to modify it afterwards. Also for offerings using HTTP Adaptive Bitrate (ABR) technologies, the graphics are encoded with the video. Graphics become unreadable when the ABR algorithm switches to low quality representation e.g. due to bad network conditions. Overlaying the graphics on the player side decouples the quality of the graphics from the quality of the video. This paper describes how to control and personalize TV graphics using timed in-band events defined in MPEG-DASH. Each viewer can resolve the events to different auxiliary media according to its profile. The graphics handling is performed at the client side where each client fetches and overlays the auxiliary media to the video. This allows personalization of graphics and provides high quality overlays independent of the current video quality.

CCS CONCEPTS

• **Information systems → Multimedia information systems → Multimedia streaming**

ADDITIONAL KEYWORDS AND PHRASES

MPEG-DASH, Graphic Overlays, Personalization, sparse timed metadata

1 INTRODUCTION

TV broadcasters are offering content that fits the majority of viewers in order to captivate as much customers as possible. Linear TV services remain attractive as they bring the social experience to consumers [1]. In particular live events drive synchronized and linear viewing. With the technology advancements in media streaming (e.g., HTTP Live Streaming (HLS), Dynamic Adaptive Steaming over HTTP (MPEG-DASH) [2]), more video is consumed today using smartphones and tablets. Content providers are starting to use the Internet as the delivery platform for TV services in order to reach a wider audience through a higher variety of devices.

TV personalization has been a key factor in the past few years to offer a tailored experience for viewers and boost TV revenues. By looking at TV channels, channel overlays (e.g., graphic overlays, banners, etc.) play an essential role as a complementary media to the TV channel. Overlays are added in various scenarios and shapes. They can appear as complementary to the current TV program such as statistics in sports shows, or can be used as a news roll or remainder for important events on the channel, or as advertisement banner.

Nowadays graphic overlays are mixed with the content before encoding. With TV industry heading towards personalization, time-shift viewing and Internet delivery, there is a need to change the graphics according to the viewer profile, time of viewing or region. This will increase the value of the graphics by making it more relevant to viewers. For example, the statistics could change during a sport event according to the viewer interest, where viewers see more statistics on the player they are following. Another use-case is to update the news roll in case the viewer is watching in time-shift fashion. The state of the art on inserting graphics to TV is based on transcoding of video. The typical procedure is to decode the stream, overlay the graphics and re-encode it [3], where some tricks are used to reduce the computational complexity of the system. As an example is the usage of transform coefficient domain instead of the pixel domain has been proposed in [5]. Such solution, however, is still complex and not scalable. The authors in [6] use tiles in High Efficiency Video Coding (HEVC) profiles to perform encoded pictures composition into an HEVC video via temporal or spatial multiplexing. The approach in [6], however, is based on a certain type of encoding which limits the usage of other encoding schemes. Object-based broadcasting is considered one step towards personalization of streams through broadcast. The contents are broadcasted as separate objects accompanied by meta-data [7][8]. The client uses the meta-data to reconstruct the stream according to the client type, region or personal preference. Such solutions, however, require that the client

DOI: 00.000/000_0

receives all objects and reconstruct the stream using only the relevant objects, thus resulting in a waste of bandwidth.

This paper describes a novel technique to trigger and control client side overlays and personalize graphics in a linear TV stream. Since graphics events are very seldom, the graphics triggers are embedded as in-band metadata into the media stream. The client uses the metadata to fetch the graphics. The graphics are handled at the client side at the time of viewing. The metadata acts as trigger point and carries instructions for the client to construct and overlay the graphics. These instructions contain for example the Uniform Resource Locator (URL) of a graphics personalization server, which the client is contacting for detailed description of the graphics. The graphics server identifies the client request and provides an Extensible Markup Language (XML) file describing the personalized graphics and its URL. In summary, the main contributions of the paper are:

- Using timed MPEG-DASH in-band events to trigger and control the client-side application that overlay graphics. The solution is fully compatible with the MPEG-DASH standard [2].
- Using HTML5 on top of DASH video stream to display overlay graphics, making the full linear TV experience W3C compliant.
- Proposing a two-step principle to display graphics, using a graphics personalization server to produce unique overlays per clients or client group.

The paper is organized as follows: Section 2 explains the background for the technologies used in this paper. Section 3 describes the implementation of in-band events for personalized graphic overlays at the client. Section 4 explains the prototypical implementation, and Section 5 concludes the paper.

2 BACKGROUND

2.1 Dynamic adaptive streaming over HTTP:

MPEG-DASH [2] is the common technique for video streaming today. The media file is divided via a segmenter into small files called segments. In case of a live offering, segments are created immediately from the live feed. Each segment resembles a few seconds of the full period (typically 2s to 10s). The segments are hosted on a Web server where the client can fetch them via HTTP Get requests. Fetching segments is similar to hosting Web pages over the Internet, which brings a great advantage for DASH. The media distribution can be accelerated through content delivery network (CDN) which allows high scalability with lower cost compared to the conventional Real-time Transport Protocol (RTP) streaming. The segments locations in the CDN are described in an XML script file called Media Presentation Description (MPD).

The MPD contains all information necessary to download and play-out the segments (e.g. supported codec, availability time, bitrate, resolution). The MPD is organized in periods which represent a sequence of programs (Figure 1). Each period can be available at different quality levels which include the media segments. The DASH client uses the MPD to fetch the desired segment according to the segment availability time. Another advantage of DASH is the seamless dynamic adaptation of the video quality. The segmenter is capable with the help of transcoder to produce multiple qualities for a single stream. The client dynamically estimates the bandwidth via adaptation bitrate (ABR) algorithm and fetches the segment quality that fit its current bandwidth.

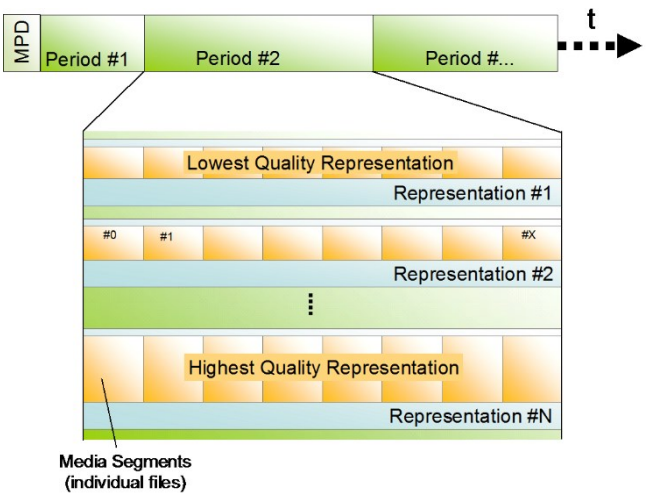

Figure 1: *Visualization of the MPEG-DASH MPD structure.*

2.1.1 In-band events. MPEG DASH [2] provides a technique to send timed in-band events either directly to the client or to an application running on the top of the client. In many players the application can register an event handler with the DASH client in order to receive custom events. An in-band message is coupled via the segment to the presentation time of the content. This message must exist in all representations of an adaptation set so that the ABR client can switch representations without missing in-band events. Each in-band message is uniquely identified by its *@scheme_id_uri* and *@value*. The client would ignore any redundant message with the same identifier and value. The client is informed about the presence of the in-band message in a given presentation through the MPD. The presentation or the AdaptationSet that carries the in-band message within its segment should contain an `InbandEventStream` element in the MPD. It should contain identical *@scheme_id_uri* and *@value* attributes to the one carried in the in-band message.

2.1.1.2 Event message box. The event message box ("emsg") is embedded in the segment in order to signal DASH specific operations such as MPD updates which is processed directly by the client [2]. It can also contain custom information to be processed by an application running on the top of the DASH client, such as content program metadata [4].
The "emsg" box is encapsulated in ISO Base Media File Format (ISO-BMFF) before "moof" box and contains the following parameters:

Scheme_id_uri: identifies the Uniform Resource Identifier (URI) scheme. It has to be identical to the scheme in the MPD. Some values are reserved for DASH operations, while other URI can be used for custom messages.

Value: specifies the value of the event. It should also be identical to the value announced in the MPD. In case it is different, the client will ignore the message.

Timescale: provides the timescale for the duration and time [2].

Presentation_time_delta: provides the difference between the start up of the event and the processing of the segment.

Event_duration: provides the duration of the event.

Id: provides a unique id for the event message. The client can ignore "emsg" if the Id is repeated.

3 IMPLEMENTATION

In this section, we explain the architecture components for our solution. Specifically, we detail the components that we have implemented for realizing TV graphics personalization using in-band DASH events. Figure 2 depicts the architecture, where each component is typically deployed on a different virtual machine in a cloud.

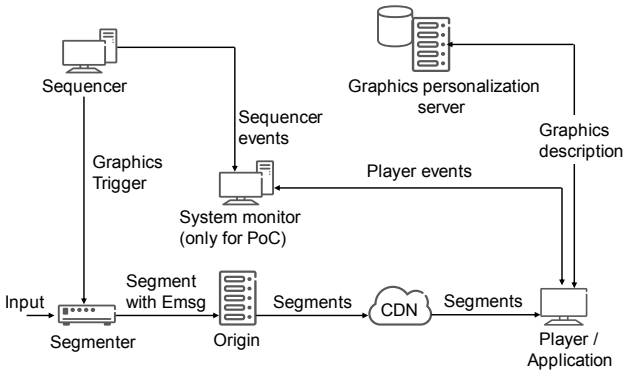

Figure 2: *Architecture for client side graphics control*

3.1 Sequencer

The sequencer is a Node.JS [14] server. The broadcaster connects to the sequencer via a Web interface where he can choose the type of graphics and the level of personalization. The sequencer constructs a JavaScript Object Notation (JSON) message [15] with all required details in order to trigger the segmenter to insert a DASH in-band event into the according segment. Figure 3 depicts a screen shot of the sequencer GUI of the proof of concept implementation. The broadcaster can define the type of channel related graphics. Channel graphics are inserted on the fly, while program related graphics can be inserted beforehand.

3.2 Segmenter

The segmenter receives either an MPEG2-TS stream or it can loop over an MP4 file. The segmenter generates DASH segmented content with its corresponding MPD as an output. The output segments are pushed to the HTTP origin using WebDAV. The segmenter listens to trigger commands (HTTP POST requests) from the sequencer. The trigger command carries a JSON message which contains needed information for the "emsg" to be inserted in the latest segment.

3.3 Graphic personalization server

The Graphics personalization server also uses Node.JS connected to a database (DB). It contains information about each user (or user group) and geographical areas. It also contains URLs for different HTML graphic objects. The database can be filled with information derived from a recommendation engine and or manual inputs from an editor. The graphics personalization server implements a group of HTTP Application Programming Interface (API) which are used to handle unique requests from each client. The server receives a GET request from the client requesting a certain type of graphics (e.g. http://10.0.0.1/Personal/Comingnext). The request is handled by the corresponding API, where the server identifies the user via cookies or login credentials for personal graphics, or it can identify the user location using the IP address. The server compiles the XML file using information derived from the DB. It describes in details the graphics location and necessary information for the client to fetch and assemble the graphics on top the video correctly.

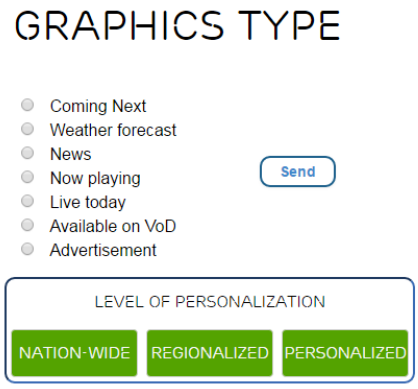

Figure 3: *Sequencer GUI*

3.4 System monitor

The system monitor is used to track and visualize live information about the system. The system monitor receives information from all players and also the sequencer. The sequencer updates the system monitor whenever the segmenter was instructed to insert an "emsg" in one of the segments. It gets a notification from the player when an "emsg" is received and when the graphics are overlayed. The system monitor gives a better understanding of the call flow and helps to monitor the players.

3.5 Application/Player

The client side implementation is based on javascript (JS) players. We used two players in the implementation to study the differences and demonstrate the applicability of our solution for different media clients, namely, the Shaka player V2.0 library from Google [9] and the Dash.js library V2.1 from the DASH industry forum (DASH-IF) [10] have been considered. The two players are the very prominent open source javascript players for DASH playout.

Both, Shaka player and Dash.js support DASH in-band event message parsing. There is, however, a difference in the implementation which results in variations in the client behaviour. In the following section, we explain how in-band events are implemented in the clients. Both players use the Document Object Model (DOM) event listener [11], which allows the client to trigger a sequence of custom functions once a certain event occurs.

3.5.1 Shaka player based implementation. The Shaka player requires a generic event listener in order to trigger the player to listen for event messages. The library provides an event listener API for DASH events. The handler requires two parameters: The text string "emsg" and the function to be triggered once the according "emsg" box is found in a segment. If the in-band stream element exists in the MPD, the player applies a network filter which inspects all downloaded segments for "emsg" box. If

an in-band event is found, then the player notifies the application through the registered listener function. Such implementation raises two issues: first, the player triggers the listener regardless of the scheme URL in the "emsg" box. It is the application responsibility to check whether the "emsg" complying with the scheme in the MPD and whether the "emsg" is actually intended for this application. The Shaka player allows the application to register for a single event listener and handle different "emsg" separately. The second issue is that the in-band event is triggered once the segment is downloaded and not when the rendering of the segment starts. Thus, the application needs to figure out the buffer level to predict the presentation time. The buffer level typically varies between different clients, since the download of the segments depends on the available bandwidth. Further the application needs to queue the received events until the presentation starts.

Figure 4 summarizes the event call flow for the Shaka player.

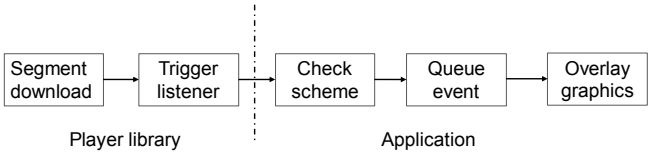

Figure 4*: Shaka player event call flow*

3.5.2 Dash.js player based implementation. The Dash.js player handles events call flow slightly different than the Shaka player. The player initiates an event listener with the specific scheme URI and the function to be triggered once the event occurs. The library creates an array of all the expected event schemes from the MPD. When a segment is downloaded, the segment parser checks for the existence of "emsg" box. If the box is found the library checks the scheme URI against the list of events. If the schemes match, the library queues the event and triggers the listener function at the time of rendering the segment. The application triggers the graphics overlay function to fetch and overlay the graphics. The issue of the Dash.js player is that the application has literally no time to prepare for the event, since the application is only notified upon presentation of the segment.

Figure 5 summarizes the event call flow for Dash.js.

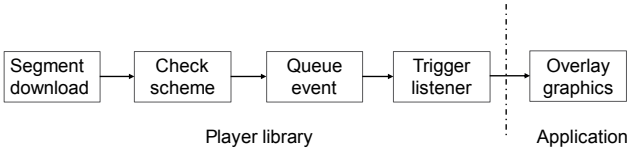

Figure 5: *Dash.js event call flow*

3.6 Call flow

Figure 6 depicts the call flow for the system. It includes all the steps from the start of playout until graphics are rendered at the client. Below are the detailed steps for the process:

1. The player downloads the MPD once the client tunes-in to a given channel.
2. The player parses the MPD and recognizes the presence of in-band event stream element. It assigns the event listener according to the given scheme id and value.
3. The player starts fetching segments and inspects each segment for the presence of "emsg" box
4. The sequencer decides to overlay graphics; it sends a POST request to the segmenter with JSON payload.
5. The segmenter parse the JSON message, compiles the "emsg" and inserts it in the currently generated segment.
6. The player downloads the segment and detects the "emsg". It validates the scheme id and value against the parameters in the MPD. Afterwards, it calls the graphics function.
7. The graphics function requests the graphic meta-data from the graphics personalization server.
8. The personalization server composes the XML file that describes the personal graphics and sends it to the player.
9. The player parses the XML, downloads the graphics elements and renders it on the top of the video.

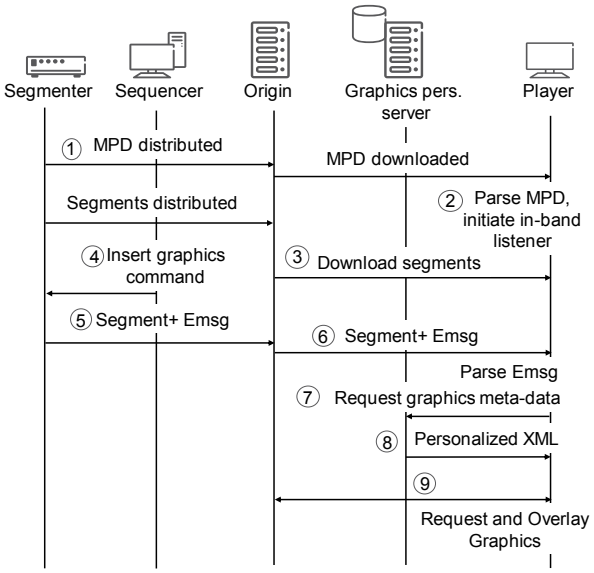

Figure 6: *System call flow*

4 PROTOTYPICAL IMPLEMENTAION

We have demonstrated our solution using the LORA Solution [12] graphic engine and big buck bunny movie [16]. LORA Solutions provides a JavaScript library that allows to display broadcast grade graphics in pure HTML5. As part of the library, LORA Solutions provides a set of reusable components like text-crawlers, picture, video overlay, and captions. A typical TV-grade graphic will be a composition of several components, described in a unique XML file. Each component can declare "dynamic fields" whose values are defined at loading time. An example is a *news crawler* (also called *news ticker*) where the text changes over time.

Displaying an overlay graphic is a three-step process:

1. A composition is constructed in the background.
2. Dynamic fields (data) are loaded for each component (data comes from the graphics personalization server).
3. The final graphic is then put in the foreground.

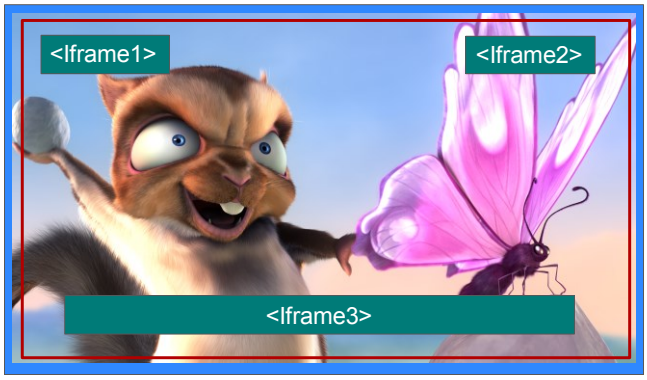

Figure 7: *Graphics objects composition*

Here is a detailed explanation of this process:
The client fetches the XML file from the personalization server and forwards it to the LORA Solution library. It uses the XML file to define how many objects are defined in the graphics, where each object is described as an element. Figure 7 shows the W3C elements alignment in the HTML document of the application. A W3C video element in full-screen mode takes always the highest priority and is rendered on top of the HTML document. Consequently, it is not possible to overlay any element on the top of it. Therefore, a background empty element is introduced to host the video element and allow other elements overlays. This background element is indicated as blue frame in Figure 7. Each graphic object is downloaded in a separate inline frame (W3C Iframe element [13]). Once all Iframe elements are fully downloaded, the library renders them in a transparent element (indicated as red frame in Figure 7) on the top of the background element.

Figure 8 depicts the resulting view at two players, rendering the same TV channel. Each player is showing different announcement for the next program. On the left side of the rendered TV channel, there is a poster and rolling brief on the upcoming movie and on the right side showing a picture inside picture with the movie trailer.

5 CONCLUSIONS

This paper presents a solution to trigger client side rendering of TV channel related graphics using DASH in-band events. Separation of graphics from encoded video allows personalization of the graphics and have high quality graphics even with lower bitrate video. Channel graphics personalization brings important value for live TV by broadcasting more relevant information for each client. Frame accurate rendering of graphics is in many cases required by the TV channel. Typically, the graphics are encoded within the video due to frame accurate timing, but that makes any later modification difficult. We implemented a proof of concept system to demonstrate the usage of in-band events specified in MPEG-DASH to overlay personalized graphics on the top of the channel stream. The client receives graphics metadata ingested in-band within the segment. It uses the metadata to fetch the HTML graphics object and overlay it on the top of the video.

Figure 8: *Screenshot two clients with different banners*

We have used both, Shaka and DASH.js for the implementation, both players support in-band events through application defined event listeners. However, they have different implementations when the listener is triggered. In case of Dash.js, the application is notified at the presentation time of the segment that contains the in-band event. This realization requires either to fetch and render the graphics very fast in order to provide time accurate graphics or to introduce a separate cue message. In case of Shaka, the application is notified at the time of reception of the in-band event. The application needs to somehow determine the buffered media time to calculate the presentation of the segment. Here, the client has some head-room to fetch the needed elements and compose the graphics in a hidden frame.

However, with such variable clients and browsers behavior, synchronized layover of graphics become an issue, some graphics requires frame accurate overlay. With the current solution the overlay could be out of sync, if the browser doesn't have sufficient bandwidth to fetch the graphics on the required time of overlay. For frame accurate graphics overlay, further work and assumptions need to be done at the client side.

REFERENCES

[1] Ericsson Consumer report, "TV and Media 2016," Nov. 2016, online: https://www.ericsson.com/networked-society/consumerlab/consumer-insights/reports/tv-and-media-2016.

[2] ISO/IEC 23009-1, MPEG-DASH, "Information technology — Dynamic adaptive streaming over HTTP (DASH) — Part 1: Media presentation description and segment formats, May 2014.

[3] Casparcg. (2016, May 9). Retrieved March 20, 2017, from http://casparcg.com/wiki/CasparCG_Server#Image_Producer

[4] ETSI TS 103 285, 9.1.2.1, v1.1.1 "Digital Video Broadcasting (DVB); MPEG-DASH Profile for Transport of ISO BMFF Based DVB Services over IP Based Networks," May, 2015.

[5] Shih-Fu Chang and D. G. Messerschmitt, "Manipulation and compositing of MC-DCT compressed video," in *IEEE Journal on Selected Areas in Communications*, vol. 13, no. 1, pp. 1-11, Jan 1995.

[6] R. Skupin, Y. Sanchez and T. Schierl, Compressed domain video compositing with HEVC, *Picture Coding Symposium (PCS)*, Cairns, QLD, 2015, pp. 287-291.

[7] T. Churnside, Object-based broadcasting, Nov. 2015, online: http://www.bbc.co.uk/rd/blog/2013-05-object-based-approach-to-

broadcasting.
[8] M. Armstrong, M. Brooks et al., Object-based broadcasting – curation, responsiveness and user experience, white paper, Jan. 2015.
[9] SHAKA PLAYER [JavaScript library] (Version 2.0,2016). Retrieved from: https://github.com/google/shaka-player
[10] DASH.JS [JavaScript library] (Version 2.2,2016). Retrieved from: https://github.com/Dash-Industry-Forum/dash.js
[11] JavaScript HTML DOM EventListener. Retrieved March 21, 2017, from https://www.w3schools.com/js/js_htmldom_eventlistener.asp
[12] LORA Solutions - Solutions for Broadcast Engineering. (n.d.). Retrieved March 22, 2017, from http://www.lora-solutions.com/index.php?lang=en&rub=nous
[13] HTML iframe tag. Retrieved March 23, 2017, from https://www.w3schools.com/tags/tag_iframe.asp
[14] Foundation, N. (n.d.). About Node.js. Retrieved March 31, 2017, from https://nodejs.org/en/about/
[15] Introducing JSON. (n.d.). Retrieved March 31, 2017, from http://www.json.org/
[16] (2008, April 10). Retrieved March 31, 2017, from http://www.bigbuckbunny.org/