A Seamless Web Integration of Adaptive HTTP Streaming

Benjamin Rainer, Stefan Lederer, Christopher Müller, and Christian Timmerer
Alpen-Adria-Universität Klagenfurt
Universitätsstraße 65-67
9020 Klagenfurt am Wörthersee, Austria
+43 (0) 463 2700 3600
{firstname.lastname}@itec.aau.at

ABSTRACT

Nowadays video is an important part of the Web and Web sites like YouTube, Hulu, etc. count millions of users consuming their content every day. However, these Web sites mainly use media players based on proprietary browser plug-ins (i.e., Adobe Flash) and do not leverage adaptive streaming systems. This paper presents a seamless integration of the recent MPEG standard on Dynamic Adaptive Streaming over HTTP (DASH) in the Web using the HTML5 video element. Therefore, we present DASH-JS, a JavaScript-based MPEG-DASH client which adopts the Media Source API of Google's Chrome browser to present a flexible and potentially browser independent DASH client. Furthermore, we present the integration of WebM based media segments in DASH giving a detailed description of the used container format structure and a corresponding Media Presentation Description (MPD). Our preliminary evaluation demonstrates the bandwidth adaption capabilities to show the effectiveness of the system.

Index Terms—MPEG-DASH, Dynamic Adaptive Streaming over HTTP, HTML5, WebM, JavaScript, World Wide Web

1 INTRODUCTION

The delivery of video content within the Web has become omnipresent nowadays, which is mostly based on the hypertext transfer protocol (HTTP) and consequently on the transmission control protocol (TCP). Online video portals like YouTube or Hulu count millions of users watching their content every day. Most of these platforms adopt proprietary solutions based on progressive download via HTTP. Recently, adaptive HTTP streaming has been introduced, including deployments based on Microsoft Smooth Streaming [1], Apple HTTP Live Streaming [2] and Adobe Dynamic HTTP Streaming [3]. In this context, ISO/IEC developed the MPEG-DASH standard allowing for dynamic adaptive streaming over HTTP (DASH) [4].

All these systems can leverage the same advantages over traditional video streaming, i.e., using the real time transport protocol (RTP) which is based on the user datagram protocol (UDP). First and foremost, adaptive HTTP streaming is able to adapt the video stream to the varying bandwidth conditions, which is especially important when it comes to mobile scenarios using smartphones or tablets in 3G/4G cellular networks. Using this adaptation it is possible to deliver continuous video without stalls or long buffering periods to the user. Furthermore, it is possible to leverage existing content delivery networks (CDN) and proxy cache infrastructures, which are optimized for HTTP delivery and which costs are significantly lower than dedicated streaming infrastructures.

All the industry solutions, as well as DASH, follow the same approach of chunk-based HTTP streaming. The basic idea is that the media content will be encoded in different versions, which differ in bitrates, resolutions, etc. and will then be chopped into segments that could be accessed individually by the client via HTTP GET requests. In MPEG-DASH a version of the media content with a specific characteristic (e.g., bitrate, resolution) is referred to as representation. A representation may consist of several segments of a given length, which correspond to the chopped media content. Thus, the client has the possibility to switch between those representations at segment boundaries to adjust the media bitrate to the current throughput capabilities of the client's Internet connection. These segments are transferred over the top (OTT) of the current Internet infrastructure following a client driven pull model. Furthermore, all adaptive HTTP streaming systems maintain some kind of manifest file, like the media presentation description (MPD) of DASH, which is downloaded by the client in the beginning of the streaming session to get the information about the media bitrate, resolution, etc., of each representation as well as the URLs of the segments [5][6].

The upcoming HTML5 standard [7] offers new ways to integrate video and audio in Web sites, leveraging built-in capabilities of the Web browsers. For example, YouTube [8] is already experimenting with the video element at a large scale, leveraging their WebM container format [9] in combination with the VP8 video and Vorbis audio codecs. With the Media Source API [1] it is now also possible to use adaptive HTTP streaming in combination with the video element. Therefore, we present an approach of implementing MPEG-DASH with JavaScript with the use of the HTML 5 video element.

The remainder of this paper is organized as follows. Section 2 discusses related work. The integration of WebM into MPEG-DASH, our DASH-JS implementation, and a preliminary evaluation thereof is presented in Section 3. Section 4 concludes the paper and including future work.

2 RELATED WORK

Apples' HTTP Live Streaming (HLS) [2] is fully integrated in the Safari Web browser using the HTML5 video element and, thus, it is possible to define the m3u8 manifest file as source of the video element. That is, the parsing of the m3u8 file and download of the segments is handled within the Web browser. Unfortunately, this functionality works on Apple systems only as Safari versions for Windows or Linux do not support HLS.

Other existing adaptive HTTP streaming systems do not make use of the HTML5 video element but deploy browser independent platforms like Microsoft Silverlight (Smooth Streaming [1]) and Adobe Flash (Adobe HTTP Dynamic Streaming [3]). The advantage is that the video player application is downloaded on-demand by the browser, which makes it easy to publish updates. The disadvantage is that these platforms are proprietary and poorly supported on mobile platform like smartphones or tablets.

As DASH is a rather new standard there are currently only a few implementations publicly available, e.g., our DASH plug-in for the VLC player [10]. Although VLC is available as a Firefox Web browser plug-in, this approach does not make use of HTML5 and is limited to Firefox only. Another publicly available DASH implementation is part of the GPAC project [11] within the Osmo4 player which supports basic browser integration.

In [12] several approaches are proposed on how to integrate MPEG-DASH into the HTML 5 video element. This includes configuration possibilities via attributes of the element to make use of the different media representations in the DASH MPD, e.g., selecting the appropriate audio language, or subtitles. However, the presented approaches are only proposals and are not supported by todays' Web browsers. Accessing the HTML 5 video element directly is currently only possible with the Media Source Application Programming Interface (API) available in the Google Chrome browser [13]. This API provides access to the decoder unit of this element and events issued by this interface (i.e., *webkitsourceopen*, *webkitsourceended*, and *progress*) via JavaScript. In particular, the *progress* event is used to push byte chunks (or in our case segments) into the decoder. Hence, this API enables the integration of various streaming formats such as MPEG-DASH.

3 MPEG-DASH FOR WEB

In this section we present our Web integration of MPEG-DASH using JavaScript and the Media Source API.

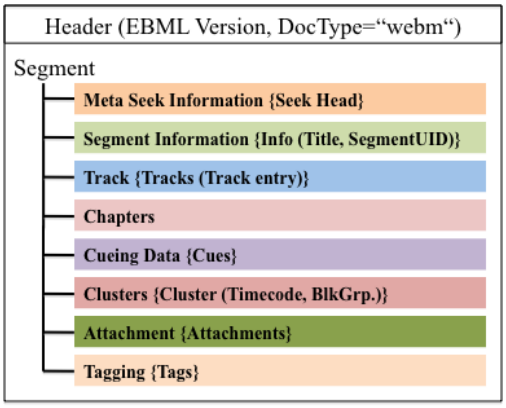

Figure 1. WebM container format.

Therefore, we present how to use MPEG-DASH with WebM [9], a subset of Matroska [14], that mandates the usage of the VP8 video and Vorbis audio codecs but is accessible via the Chromes' Media Source API and provides built-in browser support (i.e., decoding and rendering is handled natively by the browser). To the best of our knowledge, that is currently the only option that allows for a seamless Web integration of adaptive HTTP streaming.

3.1 Integrating WebM with MPEG-DASH

WebM is a container format (cf. Figure 1) based on the Extensible Binary Meta Language (EBML), which is a binary extension of XML and enables hierarchical file structures[14].

A major change to the Matroska format is that the *DocType* attribute in the header has to be set to "webm". Furthermore, *Seek Head* has to be present to reflect whether a *Cue* element is present within the *Segment*. The definition of the *Cueing Data* is very strict within WebM. This element has to be placed before any clusters and has to include only references to key frames or I-frame to lower the overall size of the *Segment* header. This allows seek operations before any clusters have been downloaded. A cluster comprises a time code and one or more block groups. The time code increases monotonically and is associated with the start time of the block. Furthermore, the clusters included in the *Clusters* element should start with a key frame. Finally, video and audio blocks referring to the same time stamps are stored in the same clusters [9].

These guidelines and restrictions allow it to push clusters into the decoder regardless which representation is currently selected. However, the order of the clusters must be preserved, which is determined by the time codes. Due to the fact that clusters carry no dependency information with them it is possible to simply switch the representation of a video stream to lower or higher bitrates.

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <MPD xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3.  xmlns="urn:mpeg:DASH:schema:MPD:2011"
4.  xsi:schemaLocation="urn:mpeg:DASH:schema:MPD:2011"
5.  profiles="urn:mpeg:dash:profile:full:2011"
6.  type="static"
7.  mediaPresentationDuration="PT10M"
8.  minBufferTime="PT15.0S">
9.  <BaseURL>http://www.example.org/</BaseURL>
10. <Period start="PT0S">
11.  <AdaptationSet bitstreamSwitching="true">
12.   <Representation id="0" codecs="vp8" mimeType="video/webm"
13.    width="854" height="480" startWithSAP="1" bandwidth="213068">
14.    <SegmentBase>
15.     <Initialization sourceURL="test.webm" range="0-441"/>
16.    </SegmentBase>
17.    <SegmentList duration="2">
18.     <SegmentURL media="test.webm" mediaRange="442-51909"/>
19.     <SegmentURL media="test.webm" mediaRange="51910-107084"/>
20.     <SegmentURL media="test.webm" mediaRange="107085-157899"/>
21.     <SegmentURL media="test.webm" mediaRange="157900-219804"/>
22.    </SegmentList>
23.   </Representation>
24.  </AdaptationSet>
25. </Period>
26. </MPD>
```

Figure 2. MPD with WebM content.

Figure 2 depicts an example of an MPD using WebM segments signaled by the *@mimeType* attribute value *video/webm* of the *Representation* element. Each segment consists of a cluster with an I-frame at the beginning, which is required by the WebM specification as already mentioned. Additionally, each cluster comprises media data, which has a length of two seconds in our implementation. However, the segment length can also be longer or shorter as it is not restricted by DASH. These segments are located in a continuous WebM file and described by byte ranges within the MPD using the *@mediaRange* attribute of the *SegmentURL* element (cf. Figure 2, line 18). The client requests the segments using partial HTTP GET requests.

As these segments are not self-initializing an initialization segment is needed (cf. Figure 2, line 15). It contains the WebM header information signaling the *DocType* as well as the seek information and cueing data for all subsequent clusters. The initialization segment is the first segment that has to be downloaded and pushed into the decoder. Additionally, this segment contains the timescale and track information as well as relevant initialization parameters for the video and audio decoders. After the initialization the following segments can be subsequently downloaded and handed over to the decoders.

The WebM compliant media files were generated using FFMPEG [15], which generates a *Cluster* for each group of pictures (GoP). By adjusting the framerate and GoP parameters of the encoder it is possible to generate media segments of a fixed time length. Note that FFMPEG does not support all container elements, such as *Cueing Data* (cf. Figure 1) and, thus, this element is currently missing when generating WebM media files. For generating MPEG-DASH compliant MPDs based of WebM media files we developed a Python script that extracts the necessary segment information such as the associated byte ranges of the segments in the media file.

3.2 DASH in JavaScript (DASH-JS)

Figure 3 depicts the architecture of our JavaScript implementation of MPEG–DASH using Chromes' Media Source API (specification version 0.3). DASH-JS comprises the following components:

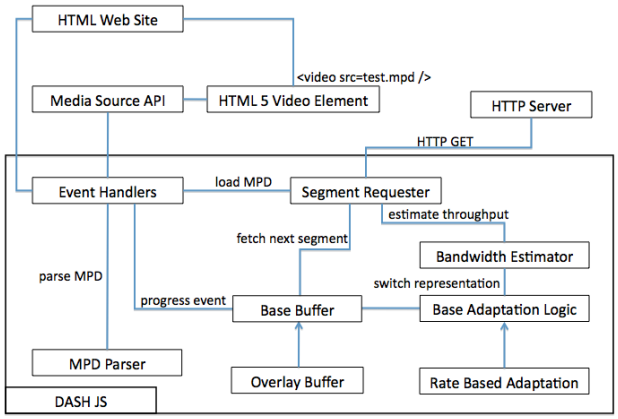

Figure 3. DASH-JS architecture.

The *Event Handlers* act as an interface for the Media Source API and process events issued by the Media Source API. When the Media Source API issues the `webkitsourceopen` event the *Event Handlers* will trigger the download of the MPD and afterwards the MPD will be handed over to the *MPD Parser*. The *progress event* is used to call the *Buffer* which is responsible for retrieving and buffering the segments of a media representation. Further details on the buffer are provided below.

The *MPD Parser* is used to parse the requested MPD. This will generate an object which holds all relevant information about the MPD such as the defined representations with their segments, the bitrate of each representation, the resolution of the video frames, and whether the segments are aligned throughout all representations. This information is used by the adaptation logic for determining which representations are available. Requesting the MPD is done with the *Segment Requester*, which uses the `xmlHttpRequest` to generate HTTP requests. Furthermore, the *Segment Requester* provides an asynchronous and a synchronous method for requesting and receiving HTTP requests and responses. The *Bandwidth Estimator* is used by the *Segment Requester* to measure and estimate the current effective throughput. The throughput at the n^{th} segment b_n is estimated using Equation (1).

$$b_n = \frac{w_1 b_{n-1} + w_2 b_m}{w_1 + w_2} \qquad (1)$$

Where b_{n-1} is the throughput calculated at the $n\text{-}1^{th}$ segment, b_m denotes the throughput measured when the $n\text{-}1^{th}$ segment is being downloaded, and w_1 and w_2 are the weighting factors. Instead of using all measured bandwidth values we used the calculated throughput with the $n\text{-}1^{th}$ segment. The weights allow adjusting the influence of the recently measured segment on the previous estimated throughput. As initialization (i.e., b_0) we used the bandwidth measured when downloading the MPD. This estimation is done with every segment that is requested and retrieved.

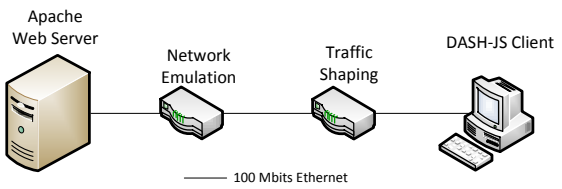

Figure 4. Evaluation Network.

The class *Base Buffer* provides the base class for buffer implementations. It offers the possibility to register event handlers for specific events (e.g., `criticalFillLevel`). Furthermore, two implementations of buffer types are provided. First, a buffer that operates on bytes, which can be used to store segments and query byte ranges of the stored segments and, second, a buffer that operates on the length of a segment.

Due to the fact that the Media Source API does not allow accessing the buffer of the HTML5 video element we implemented a so-called *Overlay Buffer*. This buffer inherits the *Base Buffer* and mimics the actual buffer of the video element. With each progress event issued by the Media Source API the method `bufferFillStateListener()` will be called. This method keeps track of the progress of the media being played back by subtracting the amount of time that has passed between the last call of this method and the current timestamp from the buffer. Additionally, it will trigger the download of further segments, which are then handed over to the video element and the buffer level is increased by the length of each segment. This buffer does not store byte chunks. It just keeps track of how many segments have been pushed into the video element in seconds and the current playback time of the video.

The buffer and the estimated bandwidth can be used to decide which representation of the media stream should be selected by making the decision based upon the bandwidth and/or the fill state of the buffer. To allow the implementation of different adaptation logics the class *Base Adaptation Logic* is provided. Every adaptation logic must inherit this class in order to be used. The adaptation logic is called after each downloaded segment. The base class implements only a single method called `switchRepresentation()`, which should be overridden. This method shall contain the adaptation decision logic. For our DASH-JS client we have implemented a simple adaptation logic (*Rate Based Adaptation Logic*), which switches the representation of the media stream based on the estimated bandwidth presented in Equation (1).

The base classes for buffers and adaptation logics offer an easy way to extend the implemented methods and encourage to experiment with new approaches for selecting the appropriate representation of the media stream. Furthermore, our implementation does not need any third-party software or browser plug-ins. The DASH-JS client is purely written in JavaScript and makes use of the Media Source provided by Google Chrome. Due to this new or

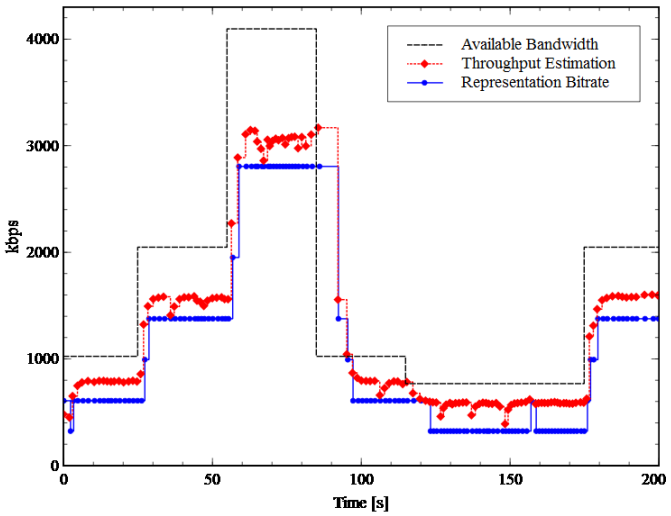

Figure 5. Media bitrate adaption of DASH-JS.

alternative adaption logics can be integrated very fast. It is also thinkable to change the adaption logic on-demand during the streaming session by requesting a more appropriate one, e.g., an adaption logic especially optimized for mobile scenarios, by loading a new JavaScript file. This implementation is publicly available on our DASH research Web site [16] providing all sources of DASH-JS licensed under the GNU Lesser General Public License.

3.3 Preliminary Evaluation

For the evaluation of DASH-JS we have used a simulation environment depicted in Figure 4. It comprises an Ubuntu host running an *Apache Web Server*, a *Network Emulation* as well as a *Traffic Shaper* node acting as gateways and a client running DASH-JS in the Google Chrome browser. The network emulation is realized using NetEm [17] to simulate a 50 ms round trip time (RTT). For the bandwidth shaping the Linux traffic control system (tc) is used in combination with a hierarchical token bucket (htb) packet scheduler. The segment length of the WebM content for this evaluation is two seconds, which also works well in high delay networks like evaluated in [18].

In Figure 5 one can see the bandwidth adaption ability of our DASH-JS client. The points on the line for the estimated throughput depict the measurement points of the DASH-JS client. For the representation bitrate the dots show the time point where a HTTP GET request is issued for the next segment, which is also the time point where the adaption decision is done. The weights for the throughput estimation presented in Equation (1) were w_1=0.7 and w_2=1.3. That is, taking 1.3 times the throughput of the last downloaded segment leads to a fast reaction of the estimated overall throughput on the recently measured throughput. We simulated various scenarios using our traffic shaping node. In particular, between seconds 0 and 70 the available bandwidth is increased stepwise. The client adapts quickly to the new bandwidth situation because this new condition

affects the effective media throughput of the segments. As we used the effective media throughput measurement of the last two segments for the bitrate adaption the client chooses the appropriate representation to the new bandwidth situation at least after two seconds. In the period between 70 and 100 we simulated a significant bandwidth drop, resulting in a long download time of the segment, which was requested just before the bandwidth change. Thus, the buffer of the HTML5 video element is leveraged during this delayed segment download. However, no stalls occur due to the high buffer level accumulated in the previous periods. A smaller bandwidth drop is simulated at approximately second 125 where the client adapts very well to the new situation. Interestingly, the estimated media throughput at approximately second 158 allows it to choose a higher bitrate representation for one segment. In the end of the simulation another bandwidth increase happens and in this case the chosen representation is selected properly in comparison to the available bandwidth.

4 CONCLUSIONS AND FUTURE WORK

In this paper we presented an implementation of MPEG-DASH using JavaScript, which exploits the HTML5 video element and the Media Source API provided by Google Chrome. Thus it is possible to provide DASH support for browsers without any further plugin necessary. Furthermore, we have illustrated how WebM can be integrated into MPEG-DASH by giving a detailed description of the segment format as well as the corresponding MPD.

As JavaScript is available on a wide range of devices, it enables the integration of DASH-JS, e.g., within mobile devices such as smartphone and tablets using, e.g., Google Chrome, which – at the time of writing of this paper – is also available on the newest Android version. Furthermore, implementing MPEG-DASH with JavaScript provides platform and browser independency, however, currently only Google Chrome provides the Media Source API.

Future work items include the implementation of more intelligent adaptation logics and buffer strategies for evaluating the performance of DASH-JS on real mobile devices and in real environments. As we used JavaScript, our solution is very flexible and these variations as well as improvements can be integrated and deployed very easily, which may also be an important factor for content providers in practice. Furthermore, we will investigate the impact on the CPU usage of DASH-JS implemented in Flash vs. HTML 5. Finally, the specification of the Media Source API has been recently updated with respect to segment formats including the ISO base media file format (ISOBMFF) compliant within MPEG-DASH [19]. Once implemented within Chrome, we plan to evaluate DASH-JS using ISOBMFF segments.

5 ACKNOWLEDGMENT

This work was supported in part by the EC in the context of the ALICANTE (FP7-ICT-248652), SocialSensor (FP7-ICT-287975) projects and partly performed in the Lakeside Labs research cluster at AAU.

6 REFERENCES

[1] Microsoft Smooth Streaming, http://www.iis.net/download/smoothstreaming (last access: Mar., 2012).

[2] R. Pantos, W. May, "HTTP Live Streaming", IETF draft, http://tools.ietf.org/html/draft-pantos-http-live-streaming-07 (last access: Mar, 2012).

[3] Adobe HTTP Dynamic Streaming, http://www.adobe.com/products/httpdynamicstreaming/ (last access: Mar., 2012).

[4] ISO/IEC DIS 23009-1.2, "Information technology — Dynamic adaptive streaming over HTTP (DASH) — Part 1: Media presentation description and segment formats"

[5] T. Stockhammer, "Dynamic Adaptive Streaming over HTTP – Standards and Design Principles", *ACM Multimedia Systems*, San Jose, California, USA, Feb. 2011, pp. 133-143.

[6] I. Sodagar, "The MPEG-DASH Standard for Multimedia Streaming Over the Internet", IEEE Multimedia, vol. 18, no. 4, Oct.-Dec. 2011, pp. 62-67

[7] I. Hickson, HTML5 - W3C Working Draft 25 May 2011, http://www.w3.org/TR/html5/ (last access: Mar. 2012)

[8] YouTube HTML5 Video Player, http://www.youtube.com/html5 (last access: Mar. 2012).

[9] WebM Project, http://www.webmproject.org, (last access: Mar. 2012).

[10] HTML 5 Media Source API, http://code.google.com/p/html5-mediasource-api/, (last access: Mar. 2012).

[11] C. Müller, C. Timmerer, "A VLC Media Player Plugin enabling Dynamic Adaptive Streaming over HTTP", *ACM Multimedia*, Scottsdale, Arizona, November 28, 2011.

[12] J. Le Feuvre, C. Concolato, J. C. Dufourd, R. Bouqueau, J.-C. Moissinac, "Experimenting with Multimedia Advances using GPAC", *ACM Multimedia*, Scottsdale, USA, Nov. 2011.

[13] C. Concolato, J. Le Feuvre, R. Bouqueau, "Usages of DASH for Rich Media Services", *ACM Multimedia Systems*, San Jose, California, USA, Feb. 2011, pp. 265–270.

[14] Matroska Media Container, http://www.matroska.org, (last access: Mar. 2012).

[15] FFMPEG, http://ffmpeg.org/, (last access: Mar. 2012).

[16] DASH-JS at ITEC/Alpen-Adria-Universität Klagenfurt, http://dash.itec.aau.at (last access: Mar. 2012)

[17] NetEm,http://www.linuxfoundation.org/en/Net:Netem (last access: Mar. 2012)

[18] S. Lederer, C. Müller, C. Timmerer, "Dynamic Adaptive Streaming over HTTP Dataset", *ACM Multimedia Systems Conference 2012*, Chapel Hill, North Carolina, February 22-24, 2012.

[19] A. Colwell, A. Bateman, M. Watson (eds.), "Media Source Extensions v0.5", *Draft Proposal*, http://dvcs.w3.org/hg/html-media/raw-file/tip/media-source/media-source.html (last access: Jul. 2012).