# IMPLEMENTING DYNAMIC AD INSERTION IN HTML5 USING MPEG DASH

Stefan Pham*, Kilroy Hughes**, Thorsten Lohmar***

*Fraunhofer FOKUS, Germany
**Microsoft, USA
***Ericsson, Germany

## ABSTRACT

Today, with W3C HTML5 premium media extensions MSE (Media Source Extensions) and EME (Encrypted Media Extensions) adaptive streaming formats such as MPEG DASH enable delivery of media content to many devices. Even televisions and settop boxes are adding Internet connections, and support HTML5 for GUI rendering and media processing. From a commercial point of view, dynamic ad insertion plays a crucial role. For broadcasters, cable and IPTV operators, content owners and advertisers new opportunities open up as advertisement can be personalized and delivered to any device. Complex signaling and back-end systems have been built for ad decision over years, and integrating with them is imperative for industry adoption of any new technology. We evaluate existing dynamic advanced advertisement techniques and present solutions for interoperable ad insertion using MPEG DASH for HTML5-based platforms and its integration with existing advertisement ecosystem.

We have extended the open source DASH-IF reference player “dash.js” with mechanisms for ad insertion. These mechanisms are based on the DASH-IF Interoperability Points guidelines and some of the recent SCTE work. We present different ad insertion workflows and highlight the flexibility that can be achieved with state-of-the-art technologies and standards.

## 1. INTRODUCTION

Content owners and broadcasters are increasingly using adaptive streaming (ABR) over HTTP to reach a multitude of devices at any time and place. The popularity of Internet audio and video services is increasing worldwide. According to Cisco’s Visual Networking Index, consumer internet video traffic is expected to be 80% of all consumer Internet traffic in 2019, up from 64% in 2014. IP video delivery to TVs is projected to grow at a rapid pace, increasing fourfold by 2019.

Requirements for adaptive streaming solutions are based on existing broadcast solutions. Users expect a comparable experience to what they are used to in broadcast TV or cable in terms of video quality, ease of playback, and features such as subtitles, trick-play, and alternative audio/video tracks. From a commercial point of view, content protection, audience measurement, and ad insertion are a must for many content providers.

Advertisement plays a key role in broadcast, cable and IPTV ecosystems Ads are spliced into the video stream at the broadcaster's site, or at any point in the video distribution chain, often in response to cue messages (e.g. SCTE 35 [5]) that mark intended insertion points, identify ad breaks and, possibly, identify the ads already inserted.

Web-based video uses industry standards such as IAB's Video Ad-Serving Template (VAST) [1], Video Multiple Ad Playlist (VMAP) and Video Player-Ad Interface Definition (VPAID) to establish a common interface between video players and ad decision servers, enabling targeted ad insertion during playback of streaming video. Ads are requested by each player and chosen by the ad decision server just in time. Ad decision servers usually select ads from their inventory based on the context (e.g. content being played, time of day, location) and various user information. Same techniques are being applied within the traditional broadcast, IPTV and cable ecosystems as a result of shift to IP delivery and emergence of hybrid broadcast/broadband systems, such as HbbTV and the emerging ATSC 3.0.

With the introduction of HTML5 Video around 2010 a cross-device alternative to third-party browser plugins (e.g. Flash or Silverlight) emerged. The <video> and <audio> tags enable media stream playback in the Web browser. In order to support adaptive bitrate (ABR) streaming another HTML5 extension is needed: The W3C Media Source Extension (MSE). MSE is implemented in all major Web browsers, and offers a JavaScript API to implement adaptive streaming, e.g. using MPEG DASH. The W3C Encrypted Media Extension (EME) enables playback of protected content by specifying a common API to discover, select and interact with each device's Digital Rights Management (DRM) system. The ISO standard, Dynamic Adaptive Streaming over HTTP (MPEG DASH), defines media presentation and formats optimized for HTML5 playback using MSE and EME. Streaming services are migrating to DASH and HTML5 from proprietary formats and plugins.

With Google's and Mozilla's decision to deprecate NPAPI [1] – the native interface that enables execution of plugins such as Flash or Silverlight – the relevance of HTML5 MSE and EME interfaces increases. Content providers have to transition to HTML5-based players in order to continue playback in Web browsers. Furthermore, features that have been available in NPAPI plugins, such as ad insertion, will now have to be enabled by HTML5-based players.

The DASH-IF provides dash.js [2], an open source HTML5/JavaScript DASH player. It takes advantage of W3C MSE and EME interfaces, and adds a wide range of features including DRM, captioning, and adaptive streaming.

HTML5 and MPEG DASH [3] enable an interoperable and platform-agnostic streaming solution. In this paper we will focus on the available architectures and methods supported by HTML5 and DASH for targeted ad insertion.

---

[1] https://www.chromium.org/developers/npapi-deprecation
[2] https://github.com/Dash-Industry-Forum/dash.js

Based on the DASH-IF Interoperability Points we have contributed various ad insertion mechanisms to the open source dash.js project, which will be presented in this paper. These features allow the dash.js player to be compatible with both server-based as well as client-based ad insertion architectures as specified in the DASH-IF Interoperability Points.

## 2. WORKFLOW

The DASH-IF IOP guidelines describe two different architectures for ad insertion, namely the server-based and the app-based architecture [4].

Both server-based and app-based ad insertion require means to signal upcoming ad slots to the DASH packager and the MPD generator, or directly the DASH player app. The DASH-IF IOP guidelines recommend SCTE 35 cue messages for that purpose. SCTE 35 messages were specified as a technique for “carrying notification of upcoming splice points and other timing information” multiplexed in a transport stream [5]. Splice points are opportunities to either enter (In-point) or exit (Out-point) elementary streams and hence signal the start and the end of ad breaks in an MPEG-2MPEG2 transport stream. DASH has defined how SCTE 35 messages may be carried in DASH Media Segments and DASH Media Presentation Description (MPD) manifests to identify ad that have been or can be inserted.

### 2.1 Server-based Architecture

In a server-based architecture, ad insertion decisions are implemented on the server by splicing the video stream, or by “logically” splicing Media Segments in the DASH manifest. Video splicing is equivalent to broadcast delivery, and is inefficient for targeting because a separate video stream is required for each different ad sequence. Video splicing in the DASH manifest only requires a different manifest for each different ad sequence. This architecture is depicted in Figure 1.

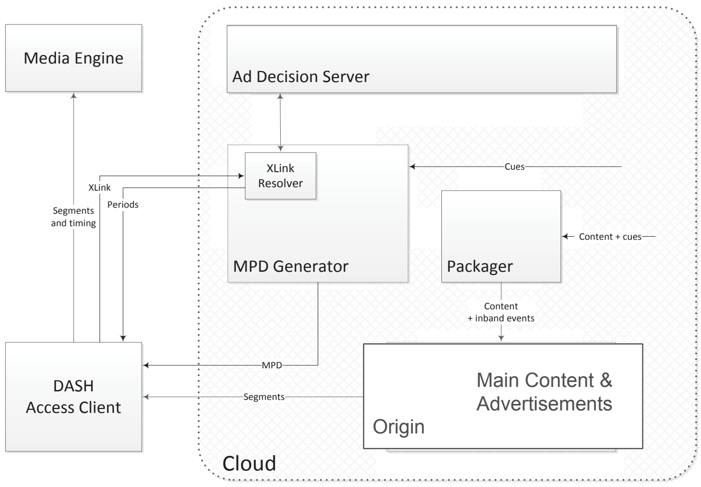

Figure 1 - Server based ad insertion architecture

When using an MPD for server ad insertion, one or more ads can be expressed as a single DASH Period element in an MPD. For instance, in a classic mid-roll scenario, period one and period three are continuous content Periods, while the Period in the middle references ad Media Segments. Continuous Periods of the main program can be marked with an optional asset identifier in order to allow the reuse of initialization data and DRM information across period boundaries and preserve UI consistency.

In order to integrate with existing deployments, actionable SCTE35 cue messages can be converted to inband MPD Validity Expiration events, which trigger an MPD reload at the DASH client. At the same time the MPD generator creates a new MPD adding ad Periods to the media presentation. As a result, the client player receives an updated MPD with additional ad Periods.

In both cases, the same MPD and Media Segments can be shared by all users, but an ad decision server, usually independent of the MPD server, selects targeted ads for each player based on various characteristics such as device, region, time of day, etc.

The simplest way of implementing server-based ad insertion is manifest manipulation: an MPD generator will provide a personalized MPD to each client with ads inserted as periods. While doable with straightforward MPD polling, this is better done with incoming actionable SCTE 35 cue messages translated into MPD Validity Expiration events and triggering MPD requests from clients. Remote periods make this implementation more scalable by having clients request personalized ad periods rather than a complete MPD, thus not requiring keeping state on the server side. Both of the alternatives are discussed in more detail below.

Remote Period elements allow standard DASH players that support XLink to trigger an ad decision and stream the ad, sequenced in a single DASH presentation and player. Remote Periods are XML elements in the manifest containing a URL that is resolved via XML Linking Language (XLink) by each DASH player. Only a subset of the XLink specification needs to be implemented by DASH players. Remote Periods must contain two attributes, namely “xlink:href” and “xlink:actuate”. The former is used to specify the URL of a remote fully populated Period element describing an ad, and the latter defines the XLink resolution time. The resolution can be done either when parsing the MPD (xlink:actuate=“onload”), or some other time before the Period element will be played (xlink:actuate=“onRequest”). A deferred resolution should always be close to the playout time of the corresponding period and should leave the player with enough time to parse and interpret the resolved element. The use of remote Periods allow a single MPD to target each viewer with different advertisements.

Three methods of Period resolution are:

1. MPD server ad decision (manifest manipulation): At each client request for the MPD, the MPD generator will create a custom manifest depending on the player requesting the MPD. The MPD generator will contact the ad decision server and include the ad Periods in the MPD. Each player receives a different MPD if they have different ads, and the server must maintain a separate session for each player if dynamic MPDs are offered, e.g. for live streaming.
2. Player ad insertion using MPD server supplied targeting information: The MPD server generates remote Periods with player-specific XLink URLs that contain targeting information. For instance, it might include a parameter specifying market or geolocation. The player passes the server generated URL parameters to the XLink server for targeted ad selection. The MPD server must generate a different MPD for each user with different parameters on the remote Period href, and update each separately for dynamic MPDs.
3. Player ad insertion using player supplied targeting information: The MPD server generates remote Periods and encodes ad break information, such as SCTE 35 or VMAP information into each XLink URL. The ad decision server must interpret the SCTE 35 or VMAP information and other player sent information to select targeted ads. Every player can download and update the same MPD, but can play a different sequence of ads.

The general workflow of the XLink resolution process is depicted in **Error! Reference source not found.**. An HTTP GET request is issued using the URL of the xlink:href attribute. The XLink resolver contacts an ad decision server with user specific parameters included in the request URL. The ad decision server responds with the ad content that should be presented. The XLink resolver creates one or more periods for that content and returns them to the DASH client.

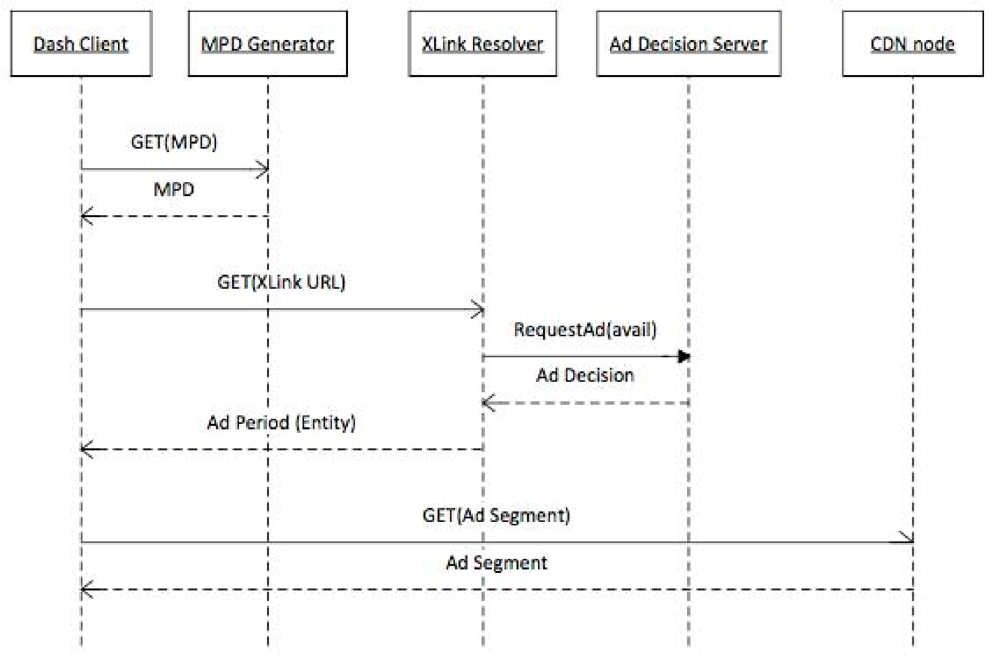

Figure 2 The XLink resolution process for the server-based ad insertion approach [4]

## 2.2 App-based Ad Insertion

In the app-based architecture, SCTE 35, VMAP, or live production messages are delivered in MPD Event Streams and optionally in Event Message boxes in Media Segments.

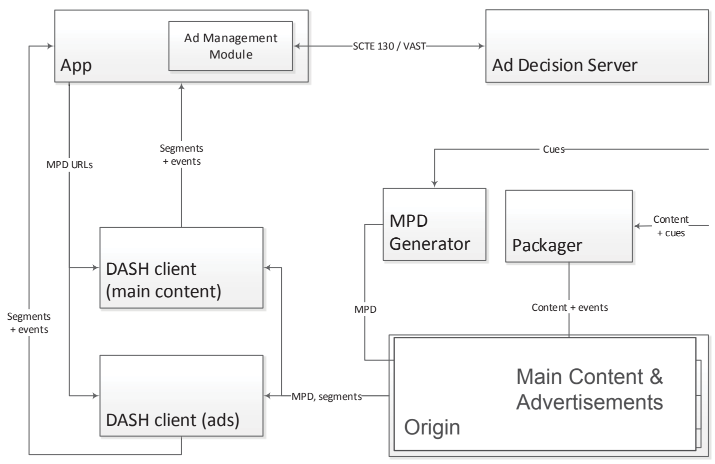

Figure 3 - App-based ad insertion architecture

The App is responsible for fetching the correct advertisement and ensures the rendering of the ad. The App may include a dedicated Ad Management module, which is implements the needed functions and transactions with the ad decision server. The App is leveraging a separate DASH client for rendering of the Advertisements. The main content is paused while playing the Ad.

On arrival of an Event Message, a generic DASH player, such as dash.js, passes it to a registered event handler, like an ad player in the service provider's customized HTML5 Web application and video player. The ad management module in the App contacts the ad decision server, forwarding the message and possibly other player information, and the ad server responds with the URL of the ad to be played.

The ad can be a short video file in any format that is downloaded and played in a separate <video> tag in parallel with the DASH presentation. Or, the ad player can use a second DASH instance and MPD to stream the ad in Segments. In either case, the ad can be displayed full screen starting on any frame of the main content, then hidden on any frame without interrupting the main content or requiring segmentation of the main content at in/out points. The ad can also be visible at the same time as the main content, e.g. split screen or picture in picture. The ability to play ads in any format and duration, starting and ending on any frame of the main program is particularly useful for live presentations.

There are other reasons for a separate ad insertion component, including verifying that it is a trusted ad player before authorizing the presentation, control of player behaviour such as ad skipping, custom targeting information, user interaction information, and authenticated audience measurement and ad play reporting. Ad networks often use the same ad insertion application module on a variety of players, streaming formats, and services to allow large scale aggregation of ad availabilities for programmatic advertising. Programmatic ad exchanges combine thousands of services and advertisers in a bidding system that result in a large ad inventory, the highest revenue per availability for services, and the highest return on investment for advertisers who can control and measure the effectiveness of targeting.

With separate <audio> and <video> elements, multiple Periods are not required. When it is time to play the ad, the app pauses (VOD case) or continues in background (live case) the main content and shows the ad content already initialized and cued on the ad player. After a pod of ads is finished, the client resumes continuous playback of the main content without changing the state of buffers, decoders, decryption, DRM, etc. In contrast, segmentation of live video at Period boundaries often requires “splice conditioning”, which requires re-encoding the video sequence overlapping the splice point to create new video sequences at the correct out and in points in the main content, then packaging those video sequences in new DASH Segments. DASH audio Segments usually don't align with video Segments, so there is audio discontinuity when Periods are spliced, but that is avoided with separate audio and video elements and buffers for the ad player that can be cued in parallel.

## 2.3 Hybrid approach

App-driven ad insertion allows direct client-ADS communication, thus improving application scalability and eliminating the need in XLink-to-ADS translation proxy. The downside of app-driven ad insertion is the need for multiple DASH clients, which is not always acceptable. Assuming content is conditioned (which is typically the situation in controlled broadcast/IPTV/cable systems), period sequencing is viable. Whith that said, service and content providers may want to do direct client-ADS communication (e.g., using SCTE 130 or VAST), rather than proxying through XLink resolver. In this case an approach being considered is using locally resolved remote periods – i.e., non-HTTP XLink URLs with resolution done locally on device, in a manner similar to iOS custom URL schemes.

## 3. DASH CLIENT IMPLEMENTATION

Based on the server-based and the app-based architecture we enabled ad insertion in the DASH-IF dash.js player. The GitHub repository contains the source code, bug tracking and documentation. We have added support for XLink, MPD validity expiration events and custom events to the player. Consequently, sample applications and server side components, which utilize the additions made to the dash.js player, were implemented.

### 3.1 XLink Support

For XLink support we concentrated on the implementation of the onLoad behavior. The main reason for that decision is the huge complexity that comes in hand with onRequest elements. A time-based resolution of parts of the MPD requires a sophisticated timing model. Late resolution or even delayed responses led to non-trivial issues. Moreover, specifications like DVB-DASH (used in HbbTV 2.0) explicitly exclude the use of onRequest elements [6] and DASH-IF IOP limits their use to VOD scenarios. **Error! Reference source not found.** illustrates the workflow of the player with and without XLink support.

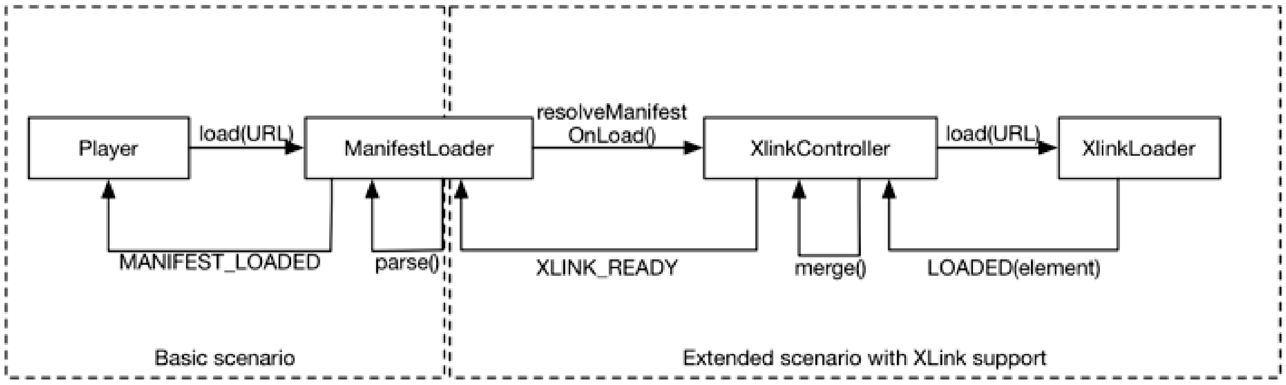

Figure 4 - The XLink implementation workflow. In addition to the basic scenario the XLink elements are resolved and merged after the manifest has been loaded.

### 3.2 Event Support

DASH specific MPD validity expiration events tell the client to perform an MPD reload at a certain point of time and have to be processed directly by the client player. **Error! Reference source not found.** shows the relation between the different classes involved. Both inband and inline events are only processed if a corresponding EventStream element with the specified @schemeIdUri and @value exist in the MPD.

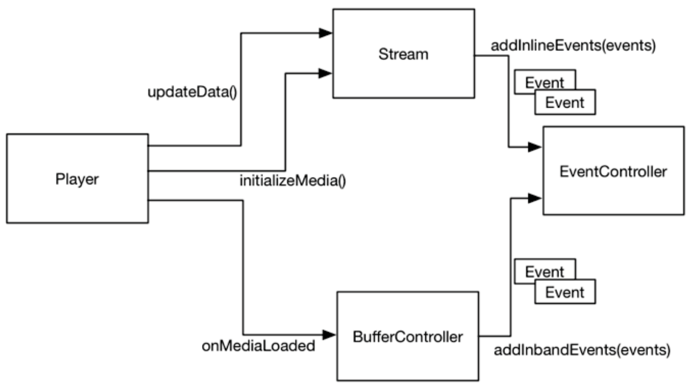

Figure 5 - The dash.js workflow for DASH events

Custom events are not handled by the DASH player itself, instead the events are transferred to the application logic for further processing (e.g. in case of SCTE35 or VAST metadata). Based on the existing implementation for DASH specific events, the EventController is extended so that it is able to throw a notification whenever a custom event is triggered. Thereby the application logic can register for a specific event and handle it appropriately.

### 3.3 Ad Insertion Sample Applications

Next we used the implemented changes made to the dash.js player to develop different ad insertion sample applications [3].

#### *3.3.1 XLink*

In the XLink sample application the focus is set on different possibilities to resolve period elements via XLink. The test cases include:

1. Resolve two Period elements.
2. Resolve three Period elements.
3. Resolve one Period element into two Period elements.
4. Resolve two Period elements into three Period elements.

On the client side, no further additions or modifications are done. On the server side appropriate content is required to support the aforementioned test cases. In the following, a closer look at test case 3 is taken, in which one period element is resolved into two period elements. The manifest file is structured as shown Figure 6.

```
<MPD>
 <Period start="PT0.00S" duration="PT600.6S" id="movie
period #1">
  <AssetIdentifier schemeIdUri="urn:org:dashif:asset-
id:2013" value="md:cid:EIDR:10.5240%2f0EFB-02CD-126E-
8092-1E49-W">
</Period>
 <Period xlink:href="http://se-
mashup.fokus.fraunhofer.de:8080/mws15/
period_timescapes_komp?country=us&zip=92309 "
xlink:actuate="onLoad"></Period>
  <Period duration="PT600.6S" id="movie period #2">
        <AssetIdentifier
schemeIdUri="urn:org:dashif:asset-id:2013"
value="md:cid:EIDR:10.5240%2f0EFB-02CD-126E-8092-1E49-
W">
</Period>
```

Figure 6 - Sample MPD to resolve one Period element into two Period elements

It contains two content periods and one midroll ad period element with XLink attributes. The @xlink:href attribute contains an additional parameter “gender” which is used on the server side to return gender specific ad content. It is important to set the return value of the response to “text/plain” when returning multiple elements that are not wrapped inside a parent container. Otherwise the content is interpreted as non valid XML. In the final step the client merges the resolved content back into the MPD and starts with the playback. This approach is an example of a stateful cue translation.

#### *3.3.2 MPD Validity Expiration Events*

The goal of this sample application is to insert an ad period into live content. The MPD reload can be done using either an inline or an inband event.

[3] http://se-mashup.fokus.fraunhofer.de:8080/ad-insertion

As illustrated in Figure 7, in a common live scenario the requests for a manifest file is always performed on the same URL.

### *3.3.3 Integrating SCTE 35*

SCTE 35 is the lifeblood of advanced advertising in live linear services. It can be used in both server-driven and app-driven architectures.

In app-driven case, SCTE 35 cue message is translated into a DASH event. Both inline an inband SCTE 35 events are defined in SCTE 214 and used in DASH-IF IOP.

In the server-driven case when used in conjunction with XLink, SCTE 35 message can be embedded into the XLink URL as a base64-coded string. An example of this approach is documented in the recent version of SCTE 214-1.

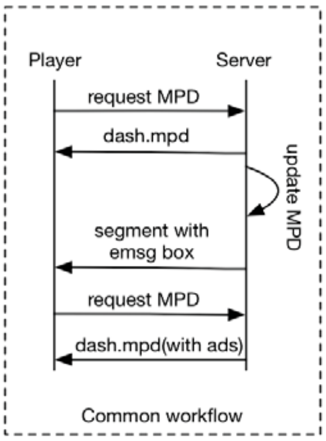

Figure 7 - Postroll server-side ad insertion using an MPD Validity Expiration event

In both of these syntactical representations there is a need for some additional intelligence. Not all SCTE 35 messages in the incoming linear feed are actionable. Repeated messages need to be filtered out. Same applies to messages used for conveying time (e.g. time descriptor) or identifying content.

MPEG-2 TS expresses timestamps on an abstract 90KHz clock not anchored to any absolute or relative timestamp. SCTE 35 signals the in-point and out-point timing in terms of their PTS (presentation time stamp) values. In case of inband SCTE 35 events, splice time needs to be translated into a time offset from the start of the segment. In case of remote periods, the offset is from the start of the presentation.

This is achieved by keeping track of the PTS value of the earliest presentation time of the first segment of the current period and its offset from the start of the current presentation. Care should, though, be taken to account for the 33-bit unsigned arithmetics involved in calculating PTS differences.

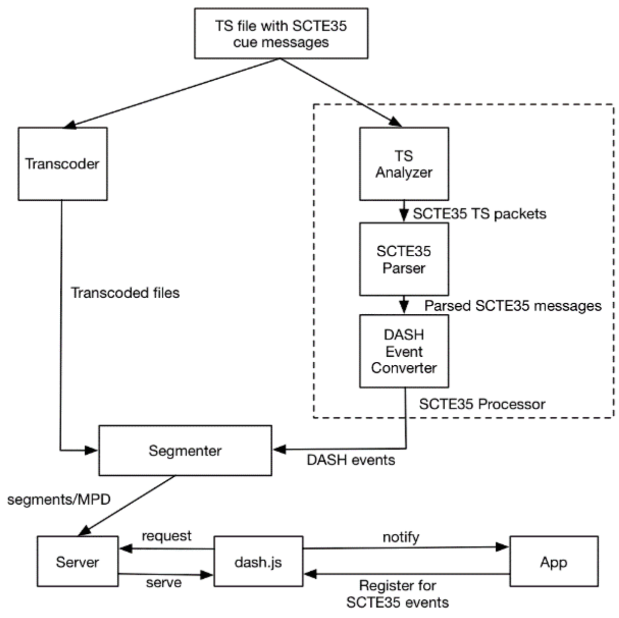

Figure 8 A SCTE 35-based advertisement workflow. The SCTE 35 cue messages included in the MPEG-2 TS file are parsed and mapped to DASH events.

As a result of the PTS to MPD time conversion, the presentation times for the cue in and cue out points are identified and can be used to create inline DASH events. Moreover, the complete SCTE 35 message is included as part of the event. The resulting DASH events and the transcoded files serve as input for the segmentation tool, which creates the DASH segments and the manifest file. On the client side an application registers for the SCTE 35 events and gets a notification from the DASH client, whenever such an event is triggered. The out_of_network indicator, which is included in the SCTE 35 message, is used to determine whether an ad should be started or removed. Afterwards either a second instance of the player is launched and put on top of the current content, or the ad is removed in order to resume the playback of the original content.

In a real system an application would use SCTE 35 cue message, possibly along with additional proprietary information, to request an ad decision from the ADS, using technologies such as VAST or SCTE 130. In our implementation this step is not simulated.

## 4. SECURITY CONSIDERATIONS

Major issues facing HTML5 advertising applications in Web pages today include ad blockers, malware, unverified play counting, and user privacy. The prospect of large scale IP video delivery using connected TVs and other devices running HTML5 web pages in browsers raises serious security considerations.

Services need to authenticate applications before authorizing playback to trust that they will play and report the ads as intended. Ad reporting also relies on proof of application authentication so that advertisers can trust and pay for reported ad exposures.

Users and devices need to authenticate and authorize applications that might otherwise track their activity, copy their personal information, monitor them through microphones and cameras, copy key strokes such as passwords, copy their data files, run background tasks such as denial of service attacks and network intrusions, take their data hostage, etc. Studies show that a large percentage of web browsers today have an ad blocker installed, and some of these ad blockers are malicious apps that allow monitoring and remote control of those devices.

It is difficult to authenticate and detect modification of Web pages written in ECMAScript and HTML5 mark-up because they are interpreted languages that are dynamically generated on servers and clients, so Web pages are easy to modify, and cannot be effectively signed like compiled code to detect if they have been tampered with.

Installed applications verified and distributed by platform “stores” provide a higher level of trust when running in secure software environments or “sand boxes” that protect them from other applications. But, it is difficult for content providers to develop and test different apps for each user experience, video, and type of device.

The benefits of HTML5 and script app development, and Web deployment can be combined with the benefits of trusted native applications using a technology called “Hosted Web Applications”, such as the Apache Cordova™ project [4].

A hosted Web application can be written as a web site in HTML, CSS, and script, then compiled to run on all major platforms. The W3C specifies a mobile application manifest format that allows hosted Web apps to configure themselves for each device they run on, ranging from mobile phones to TVs, and hosted Web apps can use native APIs such as

[4] https://cordova.apache.org/

touch screens, voice, location, search, etc. when available. Hosted Web apps may be a practical solution to cross platform video and advertising playback using Web technology, but with a higher level of security than Web pages.

Downloaded of manifests and media segments is also an attack point because ads could be easily removed from the manifest, or substituted in media segments. However, HTTPS provides server authentication and download protection, as long as the application can be trusted not to modify manifests or ignore ad markers in the manifest or media once it has arrived.

Additional possible security tools for cases where the playback app isn't trusted to authenticate and perform ad playback and reporting as intended:

- Encode ads and programs in a continuous video stream that makes it harder to automatically detect and skip ads
- Stream "live" without caching of media segments so that untrusted players can't skip ahead past ads, unless they record and playback later
- HTTPS delivery of manifests and media segments to avoid man in the middle attacks that could substitute malicious HTTP responses
- Signed or encrypted manifests that can be verified by a trusted player
- W3C Web Cryptography keys stored in the browser to identify the UA or device
- FIDO user and device OAuth authentication (W3C draft recommendation)
- Clear Key authorization of trusted ad players, until keys are redistributed
- Access tokens in ad segments that interrupt playback if ads are not download
- Encryption key changes in the video that interrupt playback if ads are not decrypted
- DRM client embedded key and ID that can authenticate the app and device
- Hosted Web app access to device stored advertising ID and location

## 5. EVALUATION

In order to evaluate the feasibility of the proposed ad insertion solutions, we have tested dash.js playback and availability of W3C Geolocation (for targeted ad insertion), Web Cryptography (security and authentication) and Encrypted Media Extensions API (DRM) on current representative browsers and devices from three categories: Desktop browsers, smartphones and TV devices. Dash.js playback (requires W3C Media Source Extensions) was evaluated with an actual playback using the developed sample application, while W3C APIs were tested using html5test.com.

| Browser/Device | dash.js | W3C Geolocation | W3C Web Cryptopgraphy | W3C EME |
|---|---|---|---|---|
| Chrome 50.0.2661.102 | ✓ | ✓ | ✓ | ✓ |

| | | | | |
|---|---|---|---|---|
| Safari 9.1 | ✓ | ✓ | ✗ | prefixed |
| Firefox 46.0.1 | ✓ | ✓ | ✓ | ✓ |
| Edge 25.10586.0.0 | ✓ | ✓ | ✓ | ✓ |
| Chrome Mobile Nexus 4 50.0.2661.89 Android 6.0.1 * | ✓ | ✓ | ✓ | ✓ |
| Chrome Mobile Nexus 7 50.0.2661.89 Android 5.1.1 * | ✓ | ✓ | ✓ | ✓ |
| Nexus Player | ✓ | ✓ | ✓ | ✓ |
| Chromecast (2nd generation) | ✓ | ✓ | ✓ | ✓ |
| LG UF8559 WebOS (2015 model) | ✓ | ✗ | ✗ | prefixed |
| Samsung UE48JU7090 Tizen 2.3 (2015 model) | ✓ | ✓ | ✗ | prefixed |
| Sony KD - 55X9305C Android 5.1.1 (2015 model) | ✓ | ✓ | ✓ | ✓ |

* same behaviour for Android apps using a WebView

## 6. CONCLUSION

Ad insertion is one of the last remaining challenges to be solved by content providers in order to move from browser-plugins such as Flash or Silverlight to platform-agnostic HTML5-based video players. We have integrated support for the server-based and app-based ad insertion workflows as specified in the DASH-IF IOP guidelines into the open source dash.js player and provide implementation considerations in this paper.

Our evaluation shows that already today it is possible to enable ad insertion using HTML5-based players and MPEG DASH across various browsers and platforms. For secure ad insertion playback W3C Web Cryptograpy is not widely available yet. However, with the finalization of the W3C media extensions MSE and EME specifications existing interoperability issues with regard to MPEG DASH playback should vanish.

## REFERENCES

1. IAB, “Digital Video Ad Serving Template (VAST) 3.0” Tech. Rep., July 2012. [Online]. Available: http://www.iab.com/guidelines/digital-video-ad-serving-template-vast-3-0/

2. HbbTV Association, “HbbTV 2.0 Specification,” Tech. Rep., Feb 2015. [Online] Available: https://www.hbbtv.org/pages/about_hbbtv/HbbTV_specification_2_0.pdf Ltd., Natick, MA.

3. T. Stockhammer, “Dynamic Adaptive Streaming over HTTP – Design Principles and Standards”, in MMSys '11 Proceedings of the second annual ACM conference on Multimedia systems, New York, NY, USA, February 2011, pp. 133-144.

4. Dash Industry Forum, “Guidelines for Implementation: DASH-IF Interoperability Points Version 3.0,” Tech. Rep., Apr 2015. [Online]. Available:http://dashif.org/wp-content/uploads/2015/04/DASH-IF-IOP-v3.0.pdf

5. Society of Cable Telecommunications Engineers, “ANSI/SCTE 35 2013,”Tech. Rep., 2013. [Online]. Available: http://www.scte.org/documents/pdf/Standards/Top%20Ten/ANSI_SCTE%2035%202013.pdf

6. EBU and DVB, “TETSI TS 103 285: Digital Video Broadcasting (DVB); MPEG-DASH Profile for Transport of ISO BMFF Based DVB Services over IP BasedNetworks,” Tech. Rep., Mai 2015. [Online]. Available: http://www.etsi.org/deliver/etsi_ts/103200_103299/103285/01.01.01_60/ts_103285v010101p.pdf