Guidelines for Implementation: Ad Insertion in DASH

September 1, 2014

DASH Industry Forum

Version 0.9 (Community Review)

Until November 1st, 2014

Please submit comments

https://gitreports.com/issue/Dash-Industry-Forum/AdInsertion

Note: This document is a recommendation rather than a specification.

Scope

This document provides recommendations for implementing ad insertion in DASH. Second edition of DASH (ISO/IEC 23009-1:2014 2nd edition, including COR1) is the baseline for this document. Furthermore, this document defines several interoperability points.

Disclaimer

This document is not yet final. It is provided for public review until November 1st, 2014. If you have comments on the document, please submit comments at the following URL:

- https://gitreports.com/issue/Dash-Industry-Forum/AdInsertion

DASH-IF members may alternatively

  - submit comments at the github repository at https://github.com/Dash-Industry-Forum/AdInsertion/issues
  - or dashif+iop@groupspaces.com with a subject tag [AdInsertion]

Please add a detailed description of the problem and the comment. Based on the received comments a final document will be published by December 15-th, 2014.

Contents

List of Figures

Acronyms, abbreviations and definitions

Ad Break: A location or point in time where one or more ads may be scheduled for delivery; same as *avail* and *placement opportunity*.

Ad Decision Service: functional entity that decides which ad(s) will be shown to the user. It interfaces deployment-specific and are out of scope for this document.

Ad Management Module: logical service that, given cue data, communicates with the ad decision service and determines which advertisement content (if at all) should be presented during the ad break described in the cue data.

Cue: indication of time and parameters of the upcoming ad break. Note that cues can indicate a pending switch to and ad break, pending switch to a next ad within an ad break, and pending switch from an ad break to the main content.

CDN node: functional entity returning a segment on request from DASH client. There are no assumptions on location of the node.

Packager: functional entity that processes conditioned content and produces media segments suitable for consumption by a DASH client. This entity is also known as fragmentor, encapsulator, or segmentor. Packager does not communicate directly with the server – its output is written to the origin.

Origin: functional entity that contains all media segments indicated in the MPD, and is the fallback if CDN nodes are unable to provide a cached version of the segment on client request.

Splice Point: point in media content where

MPD Generator: functional entity returning an MPD on request from DASH client. It may be generating an MPD on the fly or returning a cached one.

XLink resolver: functional entity which returns one or more remote elements on request from DASH client.

1 Introduction

1.1 General

This document specifies the reference architecture and interoperability points for a DASH-based ad insertion solution.

The baseline reference architecture addresses both server-based and app-based scenarios. The former approach is what is typically used for Apple HLS, while the latter is typically used with Microsoft SmoothStreaming and Adobe HDS.

1.2 DASH Concepts

DASH ad insertion relies heavily on several DASH tools, which are introduced in this section. The correspondence between these tools and ad insertion concepts will be established later. Remote elements

1.2.1 Remote Elements

Remote elements are elements that are not fully contained in the MPD document but are referenced in the MPD with an HTTP-URL using a simplified profile of XLink.

A remote element has two attributes, `@xlink:href` and `@xlink:actuate`. `@xlink:href` contains the URL for the complete element, while `@xlink:actuate` specifies the resolution model. The value `"onLoad"` requires immediate resolution at MPD parse time, while `"onRequest"` allows deferred resolution at a time when an XML parser accesses the remote element. In this text we assume deferred resolution of remote elements, unless explicitly stated otherwise. While there is no explicit timing model for earliest time when deferred resolution can occur, the spec strongly suggests it should be close to the expected playout time of the corresponding period.

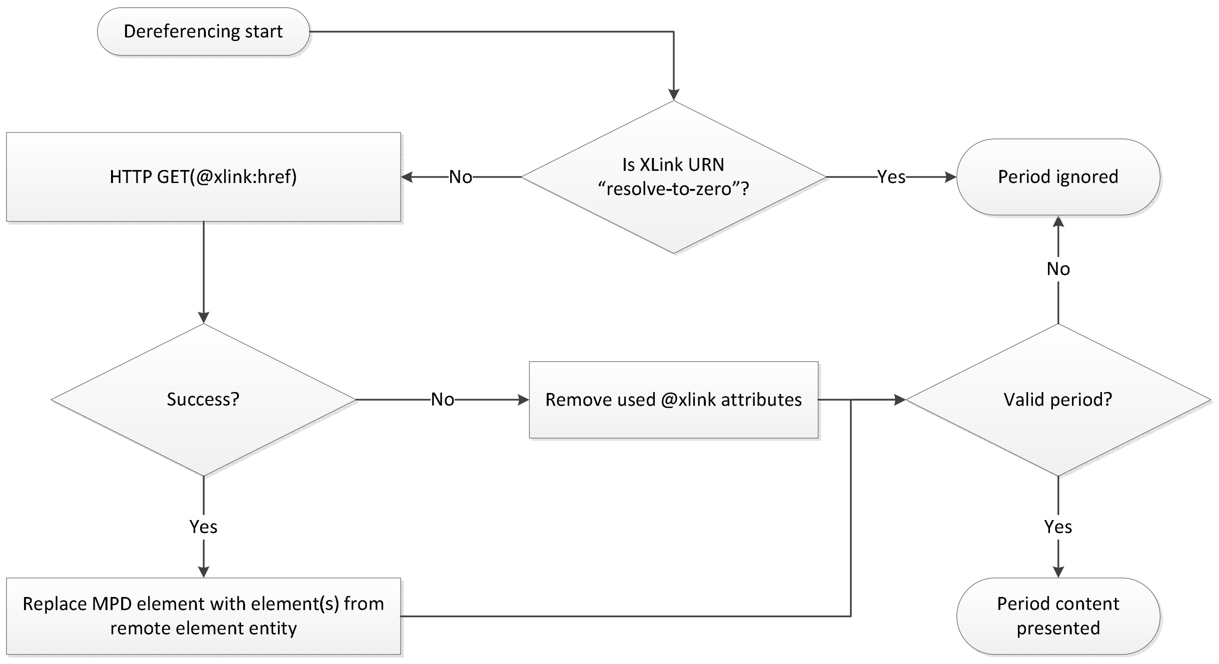

Dereferencing start
Is XLink URN "resolve-to-zero"?
No
Yes
HTTP GET(@xlink:href)
Period ignored
No
Success?
No
Remove used @xlink attributes
Valid period?
Yes
Yes
Replace MPD element with element(s) from remote element entity
Period content presented

Figure 1: XLink resolution

Resolution (a.k.a. dereferencing) consists of two steps. Firstly, a DASH client issues an HTTP GET request to the URL contained in the `@xlink:href`, attribute of the *in-MPD element*, and the XLink resolver responds with a *remote element entity* in the response content. In case of error response or syntactically invalid remote element entity, the `@xlink:href` and `@xlink:actuate` attributes are removed from the *in-MPD element*.

If the value of the `@xlink:href` attribute is `urn:mpeg:dash:resolve-to-zero:2013`, HTTP GET request is not issued, and the in-MPD element is removed from the MPD. This special case is used when a remote element can be accessed (and resolved) only once during the time at which a given version of MPD is valid.

If syntactically valid remote element entity was received, the DASH client will replace in-MPD element with remote period entity.

Once a remote element entity is resolved into a fully specified element, it may contain an `@xlink:href` attribute, which contains a new XLink URL allowing repeated resolution.

Note that the only information passed from the DASH client to the XLink resolver is encoded within the URL. Hence there may be a need to incorporate parameters into it, such as splice time (i.e., PeriodStart for the remote period) or cue message.

A specification of XLink behavior in DASH is provided in 0 sec. 5.5.3.

1.2.2 Periods

1.2.2.1 Timing

Periods are time-delimited parts of a media presentation, starting at offset *PeriodStart* from its start. *PeriodStart* can be explicitly stated using the **`Period@`**`start` attribute or indirectly computed using **`Period`**`@duration` of the previous periods.

Precise period duration of period *i* is given by *PeriodStart(i+1) – PeriodStart(i)*. This can accommodate the case where media duration of period *i* is slightly longer than the period itself, in which case a client will schedule the start of media presentation for period *i+1* at time *PeriodStart(i+1)*.

`Period``@presentationTimeOffset` specifies the offset of the earliest presentation time of the first segment of period *i* from *PeriodStart(i)* .

1.2.2.2 Segment Availability

In case of dynamic MPDs, Period-level **`BaseURL`**`@availabilityTimeOffset` allow earlier availability start times. A shorthand notation `@availabilityTimeOffset="INF"` at a Period-level BaseURL indicates that the segments within this period are available at least as long as the current MPD is valid. This is the case with stored ad content. Note that DASH also allows specification of `@availabilityTimeOffset` at AdaptationSet and Representation level.

1.2.2.3 Seamless transition

The DASH specification says nothing about period transitions – i.e., there are no guarantees seamless continuation of playout across the period boundaries. Content conditioning and receiver capability requirements should be defined for applications relying on this functionality.

1.2.2.4 Period labeling

Period-level **`AssetIdentifier`** descriptors identify the asset to which a given period belongs. Beyond identification, this can be used for implementation of client functionality that depends on distinguishing between ads and main content (e.g. progress bar and random access).

1.2.3 DASH events

DASH events are messages having type, timing and optional payload. They can appear either in MPD (as period-level event stream) or inband, as ISO-BMFF boxes of type `emsg`. The `emsg` boxes appear at the very beginning of the segment, so that DASH client will need a minimal amount of parsing to detect them.

DASH defines three events that are processed directly by a DASH client: MPD Validity Expiration, MPD Patch and MPD Update. All signal to the client that the MPD needs to be updated – by providing the publish time of the MPD that should be used, by providing an XML patch that can be applied to the client's in-memory representation of MPD, or by providing a complete new MPD.

User-defined events are also possible. The DASH client does not deal with them directly – they are passed to an application, or discarded if there is no application willing to process these events. A possible client API would allow an application to register callbacks for specific event types. Such callback will be triggered when the DASH client parses the `emsg` box in a segment, or when it parses the Event element in the MPD.

In the ad insertion context, user-defined events can be used to signal information, such as cue messages (e.g. SCTE 35, see 4.1.2)

1.2.4 MPD Updates

If **`MPD`**`@type='dynamic'`, it can be periodically updated. These updates can be *synchronous*, in which case their frequency is limited by **`MPD`**`@minimumUpdatePeriod.` In the *asynchronous* case MPD updates are triggered by DASH events (see above). An `emsg` event box for the

MPD Validity Expiration event specifies the time after which current MPD is invalid and a new MPD (with a newer `@publishTime`) must be obtained to continue playout. Use of MPD Patch events provides an optimization that allows not requesting an MPD, and rather specifying the difference between the new and the old MPD. This is convenient when changes are relatively small – e.g., addition of a remote period. In case of more extensive changes, MPD Update message can be used to carry the complete MPD inband.

When new period containing stored ads is inserted into a linear program, and there is a need to unexpectedly alter this period the inserted media will not carry the `emsg` boxes – these will need to be inserted on-the-fly by proxies. In this case use of synchronous MPD updates may prove simpler.

`MPD@publishTime` provides versioning functionality: MPD with later publication times include all information that was included all MPDs with earlier publication times.

1.2.5 Session information

In order to allow fine-grain targeting and personalization, we should be able to know the identity of the viewer, i.e. maintain a notion of a session.

HTTP is a stateless protocol, however state can be preserved by the client and communicated to the server.

The simplest way of achieving this is use of cookies. According to RFC 6265, cookies set via 2xx, 4xx, and 5xx responses must be processed and have explicit timing and security model.

1.2.6 Tracking and reporting

The simplest tracking mechanism is server-side logging of HTTP GET requests. Knowing request times and correspondence of segment names to content constitutes an indication that a certain part of the content was requested. If MPDs (or remote element entities) are generated on the fly and identity of the requester is known, it is possible to provide more precise logging. Unfortunately this is a non-trivial operation, as same user may be requesting parts of content from different CDN nodes (or even different CDNs), hence log aggregation and processing will be needed.

Another approach is communicating with existing tracking server infrastructure using existing external standards. A VAST-based implementation is shown in 3.2.6 below.

2 Architectures

The possible architectures can be classified based on the location of component that communicates with the ad decision service: a *server-based* approach assumes a generic DASH client and all communication with ad decision services done at the server side (even if this communication is triggered by a client request for a segment, remote element, or an MPD. The *app-based* approach assumes an application running on the UE and controlling one or more generic DASH clients.

Yet another classification dimension is amount of media engines needed for a presentation – i.e., whether parallel decoding needs to be done to allow seamless transition between the main and the inserted content, or content is conditioned well enough to make such transition possible with a single decoder.

Workflows can be roughly classified into *linear* and *elastic*. Linear workflows (e.g., live feed from an event) has ad breaks of known durations which have to be taken: main content will only resume after the end of the break and the programmer / operator needs to fill them with some inserted content. Elastic workflows assume that the duration of an ad break at a given cue location not fixed, thus the effective break length can vary (and can be zero if a break is not taken).

3 Server-based Architecture

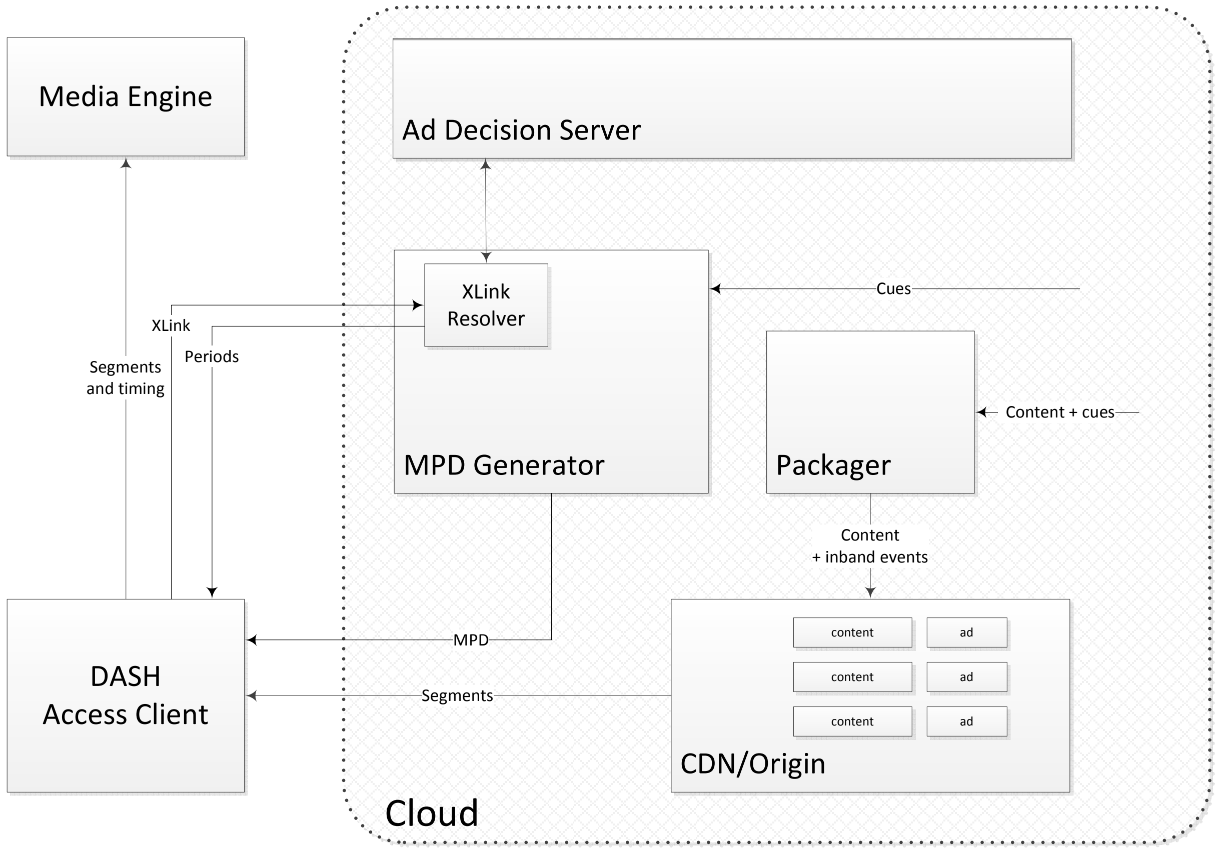


Figure 2: Server-based architecture

In the server-based model, all ad-related information is expressed via MPD and segments, and ad decisions are triggered by client requests for MPDs and for resources described in them (segments, remote periods).

Server-based model is inherently MPD-centric – all data needed to trigger ad decision is concentrated in the MPD. In case where ad break location (i.e., its start time) is unknown at the MPD generation time, it is necessary to rely on MPD update functionality. The two possible ways of achieving these are described in 1.2.4.

In the live case, packager receives feed containing inband cues, such as MPEG-2 TS with SCTE 35 cue messages 0. Packager ingests content segments into the CDN, passing manifest and cue data to the ad management module. In the on demand case, cues can be provided out of band.

Ad management is located at the server side (i.e., in the cloud), thus all manifest and content conditioning is done at the server side.

3.1 Mapping into DASH

3.1.1 Period elements

A single ad is expressed as a single `Period` element.

Periods with content that is expected to be interrupted as a result of ad insertion should contain explicit start times (`Period@start`), rather than durations. This allows insertion of new periods without modifying the existing periods. If a period has media duration longer then the distance between the start of this period and the start of next period, use of start times implies that a client will start the playout of the next period at the time stated in the MPD, rather than after finishing the playout of the last segment.

An upcoming ad break is expressed as a single Period element, possibly remote.

Remote Period will be resolved on demand into one or more than one Period elements. It is possible to embed parameters from the cue message into the XLink URL of the corresponding remote period, in order to have them passed to the ad decision system via XLink resolver at resolution time.

In an elastic workflow, when an ad break is not taken, the remote period will be resolved into a period with zero duration. This period element will contain no adaptation sets.

If a just-in-time remote Period dereferencing is required by use of @xlink:actuate="onRequest", MPD update containing a remote period should be triggered close enough to the intended splice time. This can be achieved using MPD Validity events and full-fledged MPD update, or using MPD Patch and MPD Update events (see sec. 1.2.4 and 1.2.3).

It may be operationally simpler to avoid use of onRequest dereferencing in case of linear content.

3.1.2 Asset Identifiers

`AssetIdentifier` descriptors identify the asset to which a period belongs. This can be used for implementation of client functionality that depends on distinguishing between ads and main content (e.g. progress bar).

Periods with same `AssetIdentifier` should have identical adaptation sets, initialization segments and same DRM information (i.e., DRM systems, licenses). This allows reuse of at least some initialization data across periods of the same asset, and ensures seamless continuation of playback if inserted periods have zero duration

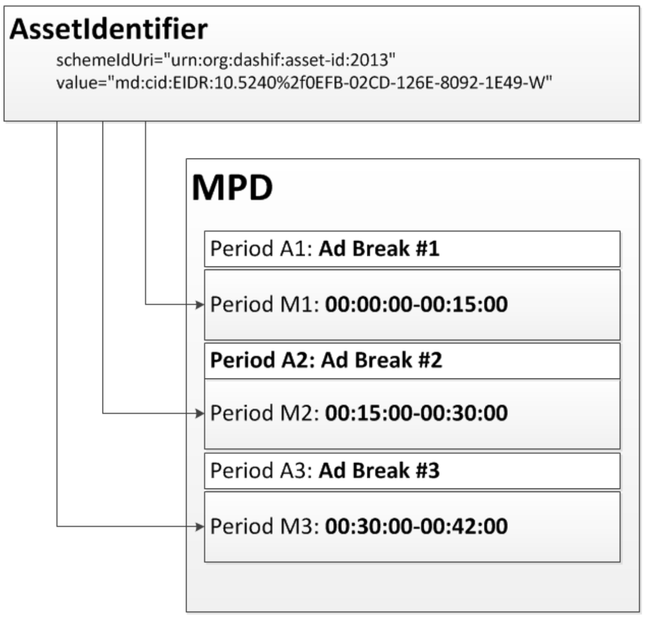

AssetIdentifier
schemeIdUri="urn:org:dashif:asset-id:2013"
value="md:cid:EIDR:10.5240%2f0EFB-02CD-126E-8092-1E49-W"
MPD
Period A1: Ad Break #1
Period M1: 00:00:00-00:15:00
Period A2: Ad Break #2
Period M2: 00:15:00-00:30:00
Period A3: Ad Break #3
Period M3: 00:30:00-00:42:00

Figure 3: Using an asset identifier

3.1.3 MPD updates

MPD updates are used to implement dynamic behavior. An updated MPD may have additional (possibly – remote) periods. Hence, MPD update should be triggered by the arrival of the first cue message for an upcoming ad break. Ad breaks can also be canceled prior to their start, and such cancellation will also trigger an MPD update.

Frequent regular MPD updates are sufficient for implementing dynamic ad insertion. Unfortunately they create an overhead of unnecessary MPD traffic – ad breaks are rare events, while MPD updates need to be frequent enough if a cue message is expected to arrive only several seconds before the splice point. Use of HTTP conditional GET requests (i.e., allowing the server to respond with "304 Not Modified" if MPD is unchanged) is helpful in reducing this overhead, but asynchronous MPD updates avoid this overhead entirely.

DASH events with scheme `"urn:mpeg:dash:event:2013"` are used to trigger asynchronous MPD updates.

The simple mapping of live inband cues in live content into DASH events is translating a single cue into an MPD Validity expiration event (which will cause an MPD update prior to the splice time). MPD Validity expiration events need to be sent early enough to allow the client request a new MPD, resolve XLink (which may entail communication between the resolver and ADS), and, finally, download the first segment of the upcoming ad in time to prevent disruption of service at the splice point.

3.1.4 MPD events

In addition to tracking events (ad starts, quartile tracking, etc.) the server may also need to signal additional metadata to the video application. For example, an ad unit may contain not only inline linear ad content (that is to be played before, during, or after the main presentation), it may also contain a companion display ad that is to be shown at the same time as the video ad. It is important that the server be able to signal both the presence of the companion ad and the additional tracking and click-through metadata associated with the companion.

With that said, there is no need to have a generic DASH client implement this functionality – it is enough to provide opaque information that the client would pass to an external module. Event schemeIdUri provides us with such addressing functionality, while MPD events allow us to put opaque payloads into the MPD.

3.2 Workflows

In the workflows below we assume that our inputs are MPEG-2 transport streams with embedded SCTE 35 cue messages. In our opinion this will be a frequently encountered deployment, however any other in-band or out-of-band method of getting cue messages and any other input format lend themselves into the same model.

3.2.1 Linear

A real-time MPEG-2 TS feed arrives at both packager and MPD generator. While real-time multicast feeds are a very frequently encountered case, the same workflow can apply to cases such as ad replacement in a pre-recorded content (e.g., in time-shifting or PVR scenarios).

MPD generator generates dynamic MPDs. Packager creates DASH segments out of the arriving feed and writes them into the origin server. Client periodically requests the MPDs so that it has enough time to transition seamlessly into the ad period.

Packager and MPD generator may be tightly coupled (e.g. co-located on the same physical machine), or loosely coupled as they both are synchronized only to the clock of the feed.

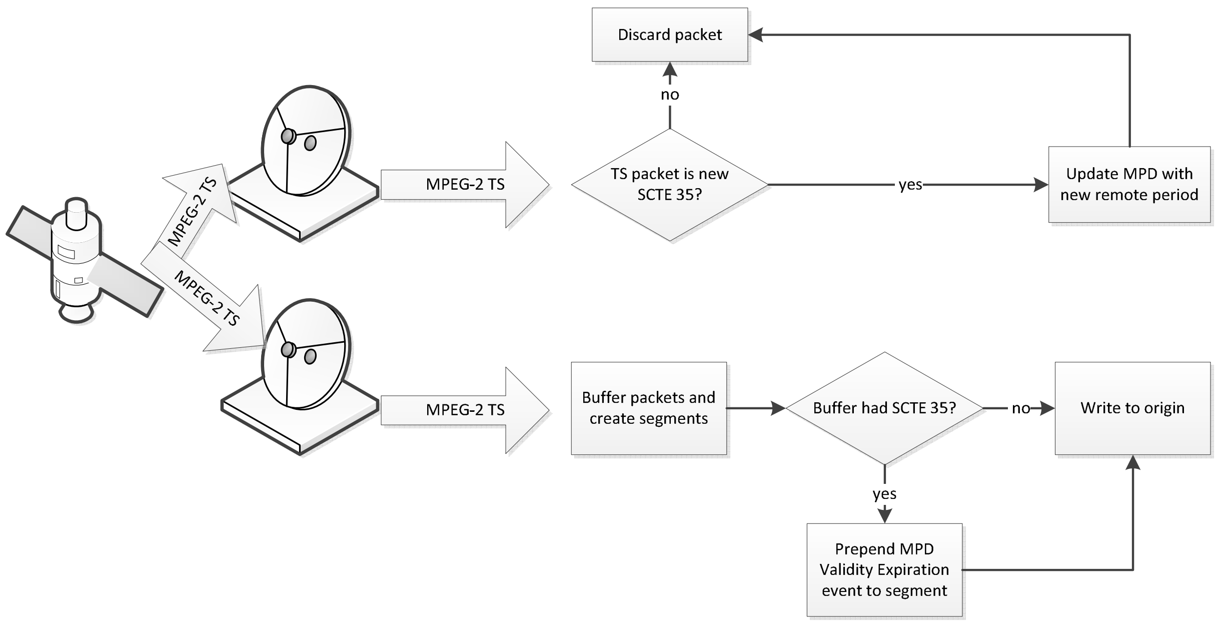

MPEG-2 TS
MPEG-2 TS
MPEG-2 TS
MPEG-2 TS
Discard packet
no
TS packet is new SCTE 35?
yes
Update MPD with new remote period
Buffer packets and create segments
Buffer had SCTE 35?
no
Write to origin
yes
Prepend MPD Validity Expiration event to segment

Figure 4: Live workflow

3.2.1.1 Cue Interpretation by the MPD generator

When an SCTE 35 cue message indicating an upcoming splice point is encountered by the MPD generator, the latter creates a new MPD for the same program, adding a remote period to it.

The **`Period`**`@start` attribute of the inserted period has `splice_time()` translated into the presentation timeline. Parameters derived from the cue message are inserted into the **`Period`**`@xlink:href` attribute of the inserted period. Examples below show architectures that allow finer targeting.

3.2.1.1.1 Example 1: Immediate ad decision

MPD generator keeps an up-to-date template of an MPD. At each cue message arrival, the generator updates its template. At each MPD request, the generator customizes the request based on the information known to it about the requesting client. The generator contacts ad decision server and produces one or more non-remote ad periods. In this case XLink is not needed.

3.2.1.1.2 Example 2: Stateful cue translation

MPD generator keeps an up-to-date template of an MPD. At each cue message arrival, the generator updates its template. At each MPD request, the generator customizes the request based on the information known to it about the requesting client.

The operator targets separately male and female audiences. Hence, the generator derives this from the information it has regarding the requesting client (see 1.2.5), and inserts an XLink URL with the query parameter `?sex=male` for male viewers, and `?sex=female` for the female viewers.

3.2.1.1.3 Example 3: Stateless cue translation

At cue message arrival, the MPD generator extracts the entire SCTE 35 `splice_info_section` (starting at the `table_id` and ending with the `CRC_32`) into a buffer. The buffer is then encoded into URL-safe base64url format [RFC 4648], and inserted into the XLink URL of a new remote Period element. `splice_time` is translated into **`Period`**`@start` attribute. The new MPD is pushed to the origin.

Note: this example follows the technique defined for HLS in [OATC], but uses slightly different encoding as the section is included in a URI.

3.2.1.2 Cue Interpretation by the packager

Cue interpretation by the packager is optional and is an optimization, rather than core functionality.

On reception of an SCTE 35 cue message signaling an upcoming splice, an `` `emsg` `` with MPD Validity Expiration event is inserted into the first available segment. This event triggers an MPD update, and not an ad decision, hence the sum of the earliest presentation time of the `emsg`-bearing segment and the `` `emsg`.presentation_time_delta `` should be sufficiently earlier than the splice time. This provides the client with sufficient time to both fetch the MPD and resolve XLink.

`splice_time()` of the cue message is translated into the media timeline, and last segment before the splice point is identified. If needed, the packager can also finish the segment at the splice point and thus having a segment shorter than its target duration.

3.2.1.3 Multiple cue messages

There is a practice of sending several SCTE 35 cue messages for the same splice point (e.g., the first message announces a splice in 6 seconds, the second arrives 2 seconds later and warns about the same splice in 4 seconds, etc.). Both the packager and the MPD generator react on the same first message (the 6-sec warning in the example above), and do nothing about the following messages.

3.2.1.4 Cancelation

It is possible that the upcoming (and announced) insertion will be canceled (e.g., ad break needed to be postponed due to overtime). Cancelation is announced in a SCTE 35 cue message.

When cancelation is announced, the packager will insert the corresponding `emsg` event and the MPD generator will create a newer version of the MPD that does not contain the inserted period or sets its duration to zero. This implementation maintains a simpler less-coupled server side system at the price of an increase in traffic.

3.2.1.5 Early termination

It is also possible that a planned ad break will need to be cut short – e.g., an ad will be cut short and there will be a switch to breaking news. The DASH translation of this would be creating an `emsg` at the packager and updating the MPD appropriately. Treatment of early termination here would be same as treatment of a switch from main content to an ad break.

It is easier to manipulate durations when **Period**@duration is absent and only **Period**@start is used – this way attributes already known to the DASH client don't change.

3.2.1.6 Ad decision

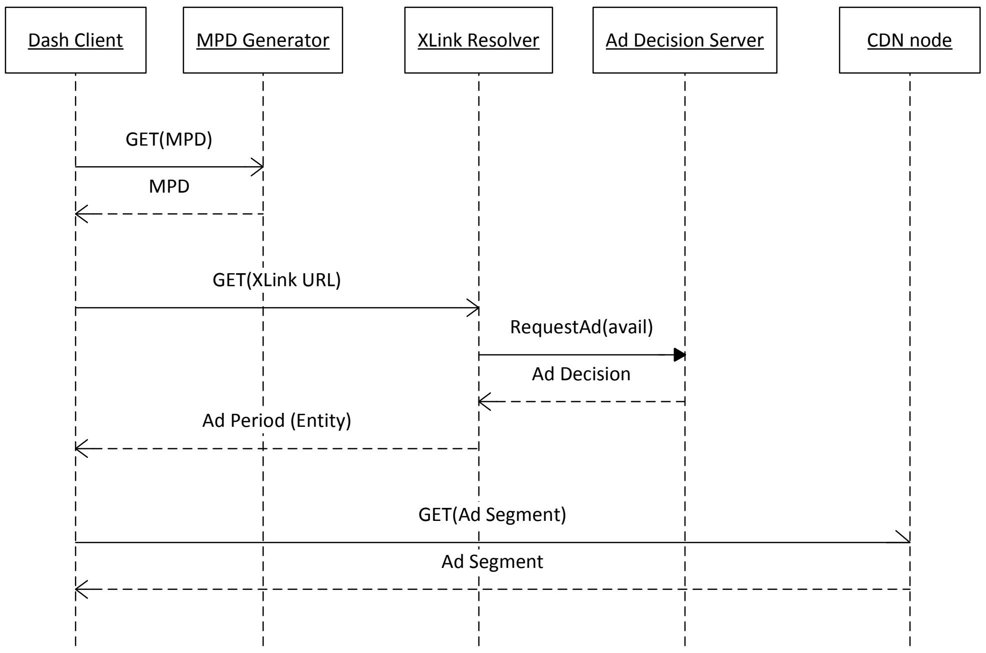


Figure 5: Ad Decision

A client will attempt to dereference a remote period element by issuing an HTTP GET for the URL that appears in **Period**@xlink:href. The HTTP server responding to this request (XLink resolver) will contact the ad decision service, possibly passing it parameters known from the request URL and from client information available to it from the connection context. In case described in 3.2.1.1.3, the XLink resolver has access to a complete SCTE 35 message that triggered the splice.

The ad decision service response identifies the content that needs to be presented, and given this information the XLink resolver can generate one or more Period elements that would be then returned to the requesting DASH client.

A possible optimization is that resolved periods are cached – e.g. in case of 3.2.1.1.1 "male" and "female" versions of the content are only generated once in *T* seconds, with HTTP caching used to expire the cached periods after *T* seconds.

3.2.2 On Demand

In a VoD scenario, cue locations are known ahead of time. They may be available multiplexed into the mezzanine file as SCTE 35 or SCTE 104, or may be provided via an out-of-band EDL.

In VoD workflows both cue locations and break durations are known, hence there is no need for a dynamic MPD. Thus cue interpretation (which is same as in 3.2.1) can occur only once and result in a static MPD that contains all remote elements with all Period elements having `Period@start` attribute present in the MPD.

In elastic workflows ad durations are unknown, thus despite our knowledge of cue locations within the main content it is impossible to build a complete presentation timeline. `Period@duration` needs to be used. Remote periods should be dereferenced only when needed for playout. In case of a "jump" – random access into an arbitrary point in the asset – it is a better practice not to dereference Period elements when it is possible to determine the period from which the playout starts using `Period@duration` and asset identifiers. The functionality described in 3.2.1 is sufficient to address on-demand cases, with the only difference that a client should be able to handle zero-duration periods that are a result of avails that are not taken.

3.2.3 Capture to VoD

Capture to VoD use case is a hybrid between pure linear and on demand scenarios: linear content is recorded as it is broadcast, and is then accessible on demand. A typical requirement is to have the content available with the original ad for some time, after which ads can be replaced

There are two possible ways of implementing the capture-to-VoD workflow.

The simplest is treating capture-to-VoD content as plain VoD, and having the replacement policy implemented on the XLink resolver side. This way the same Period element(s) will be always returned to the same requester within the window where ad replacement is disallowed; while after this window the behavior will be same as for any on-demand content. An alternative implementation is described in 3.2.4 below.

[[Ed.: address blackout use cases]]

3.2.4 Slates and ad replacement

A content provider (e.g., OTT) provides content with ad breaks filled with its own ads. An ISP is allowed to replace some of these with their own ads. Conceptually there is content with slates in place of ads, but all slates can be shown and only some can be replaced.

An ad break with a slate can be implemented as a valid in-MPD Period element that also has XLink attributes. If a slate is replaceable, XLink resolution will result in new Period element(s), if not – the slate is played out.

3.2.5 Blackouts and Alternative content

In many cases broadcast content cannot be shown to a part of the audience due to contractual limitations (e.g., viewers located close to an MLB game will not be allowed to watch it, and will be shown some alternative content). While unrelated to ad insertion per se, this use case can be solved using the same "default content" approach, where the in-MPD content is the game and the alternative content will be returned by the XLink resolver if the latter determines (in some unspecified way) that the requester is in the blackout zone.

3.2.6 Tracking and reporting

A Period, either local or a remote entity, may contain an EventStream element with an event containing IAB VAST 3.0 Ad element. DASH client does not need to parse the information and act accordingly – if there is a listener to events of this type, this listener can use the VAST 3.0 Ad element to implement reporting, tracking and companion ads. The processing done by this listener does not have any influence on the DASH client, and same content would be presented to both "vanilla" DASH client and the player in which a VAST module registers with a DASH client a listener to the VAST 3.0 events.

3.3 Examples

3.3.1 MPD with mid-roll ad breaks and default content

In this example, a movie ("Top Gun") is shown on a linear channel and has two mid-roll ad breaks. Both breaks have default content that will be played if the XLink resolver chooses not to return new Period element(s) or fails.

In case of the first ad break, SCTE 35 cue message is passed completely to the XLink resolver, together with the corresponding presentation time. The syntax in this case is similar to what is proposed in 0 for HLS.

In case of the second ad break, proprietary parameters u and z describe the main content and the publishing site.

```
<?xml version="1.0"?>
<MPD xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns="urn:mpeg:dash:schema:mpd:2011"
    xsi:schemaLocation="urn:mpeg:dash:schema:mpd:2011 DASH-MPD.xsd"
    type="dynamic"
    minimumUpdatePeriod="PT2S"
    timeShiftBufferDepth="PT600S"
    minBufferTime="PT2S"
    profiles="urn:mpeg:dash:profile:isoff-live:2011"
    availabilityStartTime="2012-12-25T15:17:50">
    <BaseURL>http://cdn1.example.com/</BaseURL>
    <BaseURL>http://cdn2.example.com/</BaseURL>

    <!-- Movie -->
    <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">
        <AdaptationSet mimeType="video/mp4" codecs="avc1.640828"
            frameRate="24000/1001" segmentAlignment="true" startWithSAP="1">
            <BaseURL>video_1/</BaseURL>
            <SegmentTemplate timescale="90000" initialization="$Band-
width%/init.mp4v"
                media="$Bandwidth$/$Number%05d$.mp4v"/>
            <Representation id="v0" width="320" height="240" bandwidth="250000"/>
            <Representation id="v1" width="640" height="480" bandwidth="500000"/>
            <Representation id="v2" width="960" height="720" bandwidth="1000000"/>
        </AdaptationSet>
    </Period>

    <!-- Mid-roll advertisement, passing base64url-coded SCTE 35 to XLink resolver -
```

```
->
    <Period duration="PT60.6S" id="ad break #1"
            xlink:href="https://adserv.com/avail.mpd?time=54054000&id=1234567&
                 cue=DAIAAAAAAAAAAAQAAZ_I0VniQAQAgBDVUVJQAAAAH+cAAAAAA=="
            xlink:actuate="onRequest" >

        <!-- Default content, replaced by elements from remote entity -->
        <AdaptationSet mimeType="video/mp4" codecs="avc1.640828"
                       frameRate="30000/1001"
            segmentAlignment="true" startWithSAP="1">
            <BaseURL availabilityTimeOffset="INF">default_ad/</BaseURL>
            <SegmentTemplate timescale="90000" initialization="$Band-
width%/init.mp4v"
                media="$Bandwidth%/$Time$.mp4v"/>
            <Representation id="v0" width="320" height="240" bandwidth="250000"/>
            <Representation id="v1" width="640" height="480" bandwidth="500000"/>
            <Representation id="v2" width="960" height="720" bandwidth="1000000"/>
        </AdaptationSet>
    </Period>

    <!—Movie, cont'd -->
    <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">
        <AdaptationSet mimeType="video/mp4" codecs="avc1.640828"
                       frameRate="24000/1001"
            segmentAlignment="true" startWithSAP="1">
            <BaseURL>video_2/</BaseURL>
            <SegmentTemplate timescale="90000" initialization="$Band-
width%/init.mp4v"
                media="$Bandwidth%/$Time$.mp4v"/>
            <Representation id="v0" width="320" height="240" bandwidth="250000"/>
            <Representation id="v1" width="640" height="480" bandwidth="500000"/>
            <Representation id="v2" width="960" height="720" bandwidth="1000000"/>
        </AdaptationSet>
    </Period>

    <!-- Mid-roll advertisement, using proprietary parameters  -->
    <Period start="PT60.6S" id="ad break #2"
        xlink:href=”https://adserv.com/avail.mpd?u=0EFB-02CD-126E-8092-1E49-
W&z=spam”
        xlink:actuate="onRequest" >

        <!-- Default content, replaced by elements from remote entity -->
        <AdaptationSet mimeType="video/mp4" codecs="avc1.640828"
                       frameRate="30000/1001"
            segmentAlignment="true" startWithSAP="1">
            <BaseURL availabilityTimeOffset="INF">default_ad2/</BaseURL>
            <SegmentTemplate timescale="90000" initialization="$Band-
width%/init.mp4v"
                media="$Bandwidth%/$Time$.mp4v"/>
            <Representation id="v0" width="320" height="240" bandwidth="250000"/>
            <Representation id="v1" width="640" height="480" bandwidth="500000"/>
            <Representation id="v2" width="960" height="720" bandwidth="1000000"/>
        </AdaptationSet>
    </Period>
</MPD>
```

1 **Figure 6: Example of MPD for "Top Gun" movie**

4 App-based Architecture

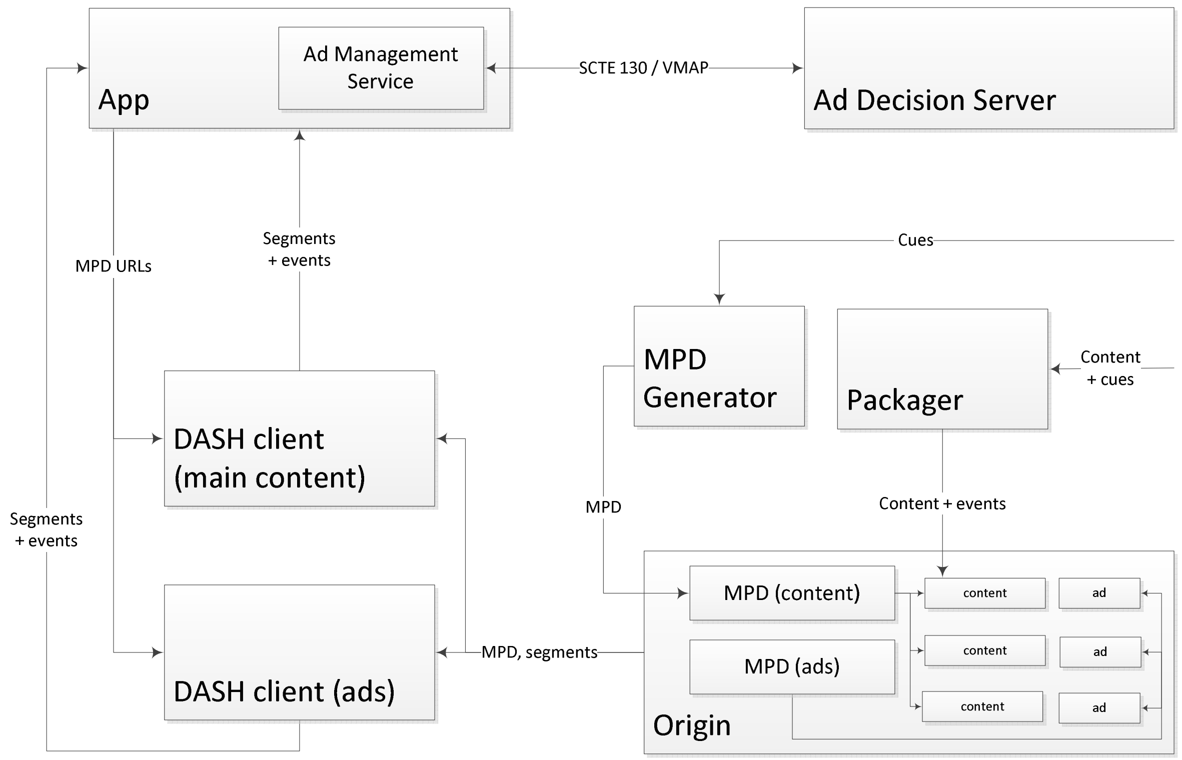


Figure 7: App-based architecture

Inputs in this use case are same as the ones described in sec. 3. At the packaging stage, cues are translated into a format readable by the app or/and DASH client and are embedded into media segments or/and into the manifest

Ad management module is located at the client side. The DASH client receives manifest and segments, with cues embedded in either one of them or in both.

Cue data is passed to the ad management module, which contacts the ad decision service and receives information on content to be played. This results in an MPD for an inserted content and a splice time at which presentation of main content is paused and presentation of the inserted content starts.

Note that this architecture does not assume multiple decoders – with careful conditioning it is possible to do traditional splicing where inserted content is passed to the same decoder. In this case it is necessary to keep a player state and be able to initialize a player into this state.

4.1 Mapping into DASH

This section details mapping of elements of the reference architecture into DASH concepts per the 2nd edition of the specification (i.e., ISO/IEC 23009-1:2014).

4.1.1 MPD

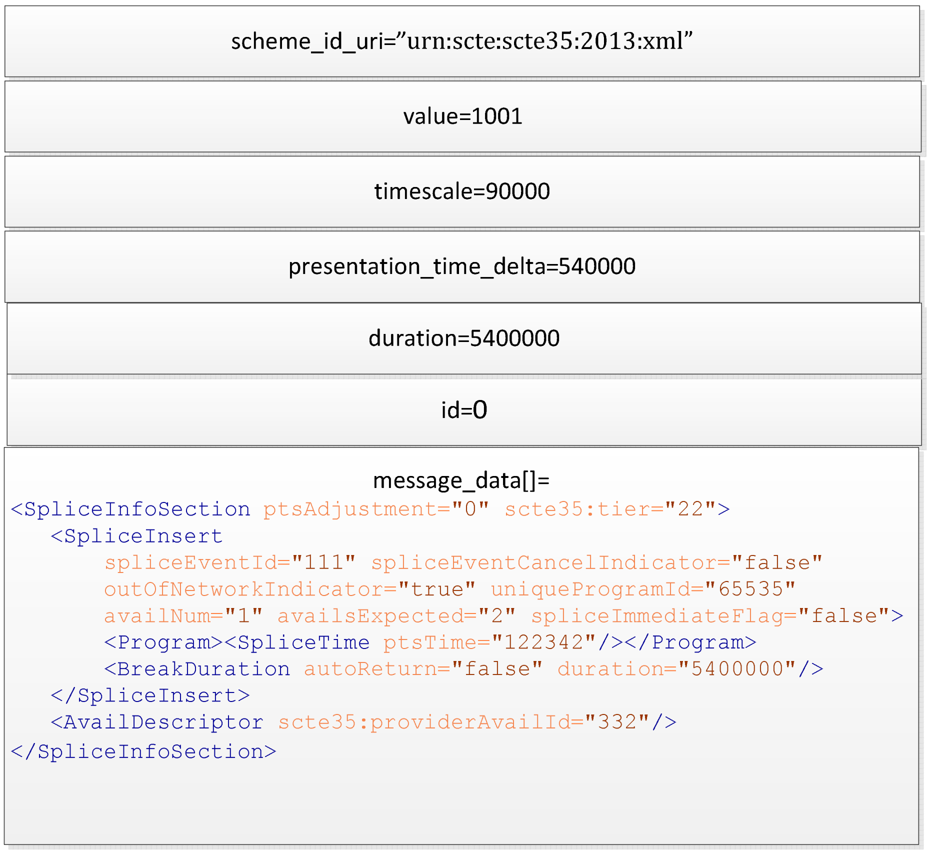
Each ad decision results in a separate MPD. A single MPD contains either main content or inserted content; existence of multiple periods or/and remote periods is possible but not essential.

4.1.2 DASH events

Cue messages are mapped into DASH events, using inband `emsg` boxes and/or in-MPD events. Note that SCTE 35 cue message may not be sufficient by itself.

The examples below show use of SCTE 35 in user-defined events, and presentation time indicates the timing in within the Period.

Figure 8 below shows the content of an `emsg` box at the beginning of a segment with earliest presentation time *T*. There is a 6-sec warning of an upcoming splice – delta to splice time is indicated as 6 seconds – and duration is given as 1 minute. This means that an ad will start playing at time *T + 6* till *T + 66*. This example follows a practice illustrated in 0.



scheme_id_uri="urn:scte:scte35:2013:xml"

value=1001

timescale=90000

presentation_time_delta=540000

duration=5400000

id=0

message_data[]=

```
<SpliceInfoSection ptsAdjustment="0" scte35:tier="22">
   <SpliceInsert
       spliceEventId="111" spliceEventCancelIndicator="false"
       outOfNetworkIndicator="true" uniqueProgramId="65535"
       availNum="1" availsExpected="2" spliceImmediateFlag="false">
       <Program><SpliceTime ptsTime="122342"/></Program>
       <BreakDuration autoReturn="false" duration="5400000"/>
   </SpliceInsert>
   <AvailDescriptor scte35:providerAvailId="332"/>
</SpliceInfoSection>
```

Figure 8: Inband carriage of SCTE 35 cue message

Figure 9 below shows the same example with an in-MPD SCTE35 cue message. The difference is in the in-MPD event the splice time is relative to the Period start, rather than to the start of the event-carrying segment. This figure shows a one-minute ad break 10 minutes into the period.

```
<EventStream schemeIdUri="urn:scte:scte35:2013:xml">
    <Event timescale="90000" presentationTime="54054000" duration="5400000" id="1">
        <scte35:SpliceInfoSection scte35:ptsAdjustment="0" scte35:tier="22">
             <scte35:SpliceInsert
                  scte35:spliceEventId="111"
                  scte35:spliceEventCancelIndicator="false"
                  scte35:outOfNetworkIndicator="true"
                  scte35:uniqueProgramId="65535"
                  scte35:availNum="1"
                  scte35:availsExpected="2"
                  scte35:spliceImmediateFlag="false">
                  <scte35:Program>

                      <!-- Event timing is given by Event@presentationTime,  -->

                      <!-- splice_time() processing is up to the application -->
                      <scte35:SpliceTime scte35:ptsTime="122342"/>
                  </scte35:Program>
                  <scte35:BreakDuration

                          scte35:autoReturn="false" scte35:duration="5400000"/>
             </scte35:SpliceInsert>
             <scte35:AvailDescriptor scte35:providerAvailId="332"/>
         </scte35:SpliceInfoSection>
    </Event>
</EventStream>
```

Figure 9: In-MPD carriage of SCTE 35 cue message

4.1.3 Asset Identifiers

See sec. 3.1.2 for details.

4.2 Workflows

4.2.1 Linear

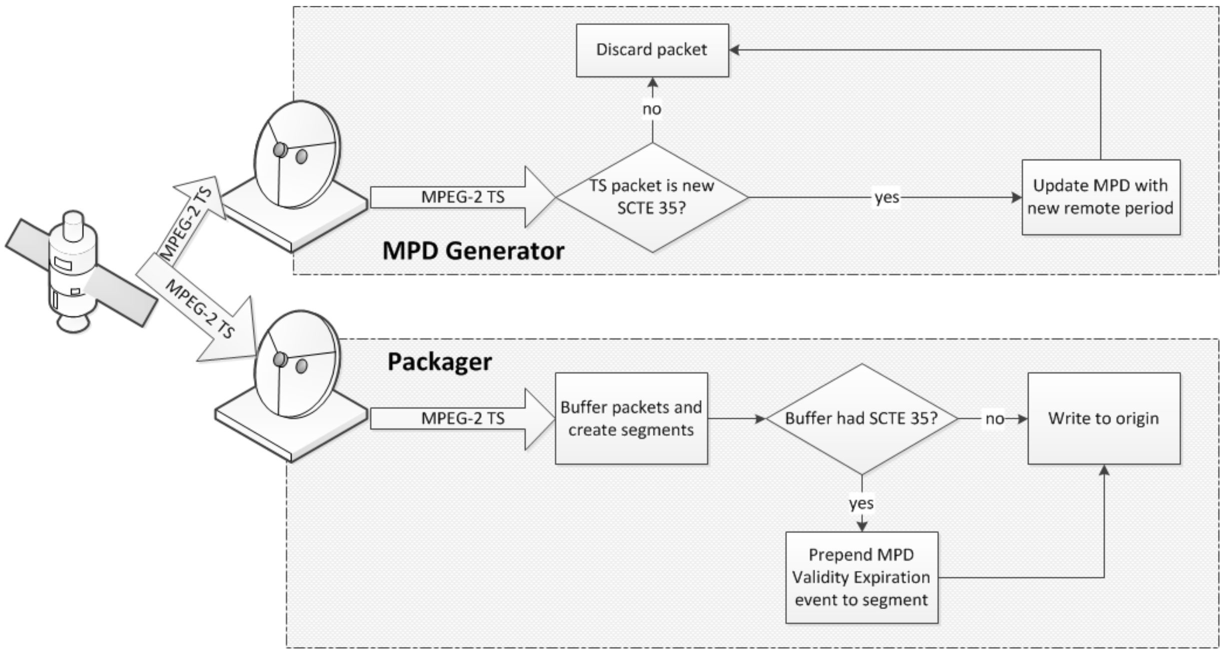

Discard packet
no
MPEG-2 TS
TS packet is new SCTE 35?
yes
Update MPD with new remote period
MPD Generator
MPEG-2 TS
MPEG-2 TS
Packager
MPEG-2 TS
Buffer packets and create segments
Buffer had SCTE 35?
no
Write to origin
yes
Prepend MPD Validity Expiration event to segment

Figure 10: Linear workflow for app-driven architecture

A real-time MPEG-2 TS feed arrives at a packager. While real-time multicast feeds are a very frequently encountered case, the same workflow can apply to cases such as ad replacement in a pre-recorded content (e.g., in time-shifting or PVR scenarios).

Packager creates DASH segments out of the arriving feed and writes them into the origin server. The packager translates SCTE 35 cue messages into inband DASH events, which are inserted into media segments.

MPD generator is unaware of ad insertion functionality and the packager does the translation of SCTE 35 cue messages into inband user-defined DASH events. On reception of an SCTE 35 cue message signaling an upcoming splice, a `` `emsg` `` with a translation of the cue message in its `` `emsg`.message_data[] `` field is inserted into the most recent segment. This event triggers client interaction with an ad decision server, hence the sum of the earliest presentation time of the `` `emsg` ``-bearing segment and the `` `emsg`.presentation_time_delta `` should be a translation of `splice_time()` into the media timeline.

An alternative implementation which is more compatible with server-based architecture in sec. 3: an MPD generator can generate separate MPDs for both server-based and app-based architectures creating remote periods for server-based and in-MPD SCTE 35 events for app-based architectures, while a packager can insert inband MPD Validity Expiration

A DASH client will pass the event to the app controlling it (e.g., via a callback registered by the app). The app will interpret the event and communicate with the ad decision server using some interface (e.g., VAST). This interface is out of the scope of this document.

The communication with ad decision service will result in an MPD URL. An app will pause the presentation of the main content and start presentation of the inserted content. After presenting the inserted content the client will resume presentation of the main content. This assumes either proper conditioning of the main and inserted content or existence of separate client and decoder for inserted content. The way pause/resume is implemented is internal to the API of the DASH client. Interoperability can be achieved by using the DASH MPD fragment interface 0.

4.2.2 On Demand

As in the server-based case, functionality defined for the live case is sufficient. Moreover, the fact that that app-based implementation relies heavily on app's ability to pause and resume the DASH client, support for elastic workflows is provided out of the box.

In the on demand case, as cue locations are well-known, it is advantageous to provide a static MPD with SCTE 35 events than run a dynamic service that relies on inband events.

5 Interoperability points

Signaling of ad insertion interoperability point should be done using **`MPD.SupplementalProperty`** descriptor with `@schemeIdUri="urn:org:dashif:iop:adin"`. The `@value` attribute shall always be present, and shall carry the IOP identifier values defined in this section.

5.1 Server-driven interoperability points

5.1.1 Multiperiod

5.1.1.1 General

This interoperability point assumes multi-period DASH content, with each Period corresponding either to inserted content or a single ad. This IOP assumes all elements are non-remote, i.e. no XLink implementation is needed. The downside is that ad decisions need to be made by the time MPD is generated or updated.

In order to facilitate asynchronous MPD updates in a dynamic service the content may contain DASH MPD Validity Expiration events.

Inband MPD Validity Expiration events are expected to occur in some of the media segments. Alternatively, MPD Validity Expiration events may appear as MPD events. Either one of the mechanisms – MPD events or inband events – may be used in a single period.

NOTE: additional user-defined events (either inband or MPD) may be present, but the client is not expected to process them

The compliance to Server-Driven Multiperiod IOP may be signaled by a `@profiles` attribute with the value `http://dashif.org/guidelines/adin/multiperiod`

5.1.1.2 Guidelines for Content Authoring

5.1.1.2.1 Multi-period content

The DASH-IF DASH-AVC/264 Live Services document sec. 4.3.3 provides guidelines for multi-period content. These guidelines should be followed.

5.1.1.2.2 Asset Identifiers

`AssetIdentifier` descriptor shall be used for distinguishing parts of the same asset within a multi-period MPD, hence it shall be used for main content.

In order to enable better tracking and reporting, unique IDs should be used for different assets. Hence it is recommended to use `AssetIdentifier` descriptors in inserted content as well.

1.) The value of @schemeIdUri shall be "urn:org:dashif:asset-id:2014"
2.) The value of `@value` attribute descriptor shall be a MovieLabs ContentID URN (0, 2.2.1) for the content. It shall be the same for all parts of an asset. Preferred scheme is EIDR.
3.) If the period is the last period of a multi-period asset, the author may add "one-off" self-contained period (e.g., an advertisement), the value of the `AssetIdentifier@id` attribute shall be a UUID.

NOTE: if (3) above is not done, the random access logic for a repeated ad will consider every repeated appearance of the ad as a continuation of its previous appearance(s). This way, a second appearance of an ad will have playout bar start at 50%.

The `AssetIdentifier` descriptor should be used for all multi-period assets.

If a period has single-period semantics (i.e., an asset is completely contained in a single period, and its continuation is not expected in the future), the author shall not use asset identifier on these assets.

Periods that do not contain non-remote `AdaptationSet` elements, as well as zero-length periods shall not contain the `AssetIdentifier` descriptor.

5.1.1.2.3 On Demand content

In case the main content complies with ISO-BMFF On Demand profile, we can assume it is stored as a single file. There is no need to create per-period files if this content is offered as multi-period. All periods for this asset will have same `BaseURL` values and different `SegmentBase@presentationTimeOffset` values, each corresponding to the media time equivalent to *PeriodStart*. The file will contain `sidx` box(es), and the client will read the index information to calculate the segment offsets taking the value of `SegmentBase@presentationTimeOffset` into account. *Note: as of August 2014, this is insufficiently clear from the standard.*

5.1.1.2.4 Events

5.1.1.2.4.1 Signaling

Presence of inband MPD Validity Expiration events shall always be signaled using `AdaptationSet.InbandEventStream` element with `@schemeIdUri=" urn:mpeg:dash:event:2012"`.

Inband MPD Validity expiration events shall be present in video. If these events are used, all video adaptation sets shall carry them.

5.1.1.2.4.2 Placement

`emsg` boxes shall be aligned: if segment $S_{R1}(i)$ is the i^{th} segment of a representation *R1* within an adaptation set containing N representations, then if $S_{R0}(i)$ contains an `emsg` box, identical `emsg` box will be carried in segments $S_{R2}(i)$ …, $S_{RN}(i)$. This means that irrespective of representation selected, all `emsg` boxes will be read if media from an adaptation set is played out.

Note: this definition will be superseded by the event alignment definition in ISO/IEC 23009-1:2014 AMD1 when published by ISO.

If several `emsg` boxes are present in a segment and one of them is the MPD Validity Expiration event, `emsg` carrying it shall always appear first.

5.1.2 Static just-in-time

5.1.2.1 General

This interoperability point assumes multi-period DASH content, MPD that contains **Period**@xlink:href and **Period**@xlink:actuate. It is a superset of Static Multiperiod IOP (see 5.1.1), with the addition of XLink.

The compliance to Static Just-in-time IOP may be signaled by a @profiles attribute with the value "http://dashif.org/guidelines/adin/static#jit"

5.1.2.2 Guidelines for Content Authoring

5.1.2.2.1 General

Guidelines in 5.1.1.2 above apply

5.1.2.2.2 Remote Periods

MPD contains remote periods, some of which may have default content. Some of which are resolved into multiple Period elements.

After dereferencing MPD may contain zero-length periods or/and remote Periods.

In case of **Period**@xlink:actuate="onRequest", MPD update and XLink resolution should be done sufficiently early to ensure that there are no artefacts due to insufficient time given to download the inserted content.

If deterministic XLink resolution timing is used, the in-MPD remote element can contain a SupplementalProperty descriptor at the Period level (note: this is a COR1 addition). The @schemeIdUri attribute value shall be "urn:org:dashif:dereferencing-offset" and @value shall be always present and contain an offset, in seconds, from *PeriodStart* till the preferred XLink resolution time. For @value=*t*, XLink dereferencing should be done at time *PeriodStart-t* or later.

If a Period element is contained in a remote element entity (i.e., contained within the XLink response, which is one or more Period elements) it shall contain **Period**@start

5.1.3 Dynamic just-in-time

5.1.3.1 General

This interoperability point is a combination of Dynamic Multiperiod IOP and Static Just-in-time IOP (see 5.1.2 above). MPD may contain remote Period elements (i.e., **Period**@xlink:href and

Period@xlink:actuate may be present) and MPD updates may be triggered by MPD Validity Expiration events.

The compliance to Dynamic Just-in-time IOP may be signaled by a @profiles attribute with the value "http://dashif.org/guidelines/adin/dynamic#jit"

5.1.3.2 Guidelines for Content Authoring

5.1.3.2.1 Timing aspects

Recommendations 5.1.2.2.2 applies. Care needs to be taken so that the client is given a sufficient amount of time to (a) request and receive MPD update, and (b) dereference the upcoming remote period.

5.2 App-driven interoperability point

5.2.1 General

This interoperability point assumes separate MPDs for main and inserted content.

Thus DASH is only used as means of transport for cue messages. This profile further relies on SCTE 35 cue messages embedded in the MPD or in media segments. While there is no dependency on ISO-BMFF Live profile, we expect inband events to be used in conjunction with it.

The compliance to App-driven IOP may be signaled by a @profiles attribute with the value "http://dashif.org/guidelines/adin/app"

5.2.2 Guidelines for Content Authoring

5.2.2.1 Events

5.2.2.1.1 MPD Events

Period elements may contain an **EventStream** element with **EventStream**@schemeI-dUri="urn:scte:scte35:2013:xml". **Event** elements contain XML representation of an SCTE 35 command. This XML representation is defined in 0. Note that SCTE 35 2014 schema allows both more verbose detailed representation of SCTE 35 cue messages and a more compact base64-coded one. The latter can be used if verbosity is a concern.

Sum of **Event**@presentationTime and **Event**@duration shall never exceed period duration.

5.2.2.1.2 Inband Events

Media segments contain inband events with scheme "urn:scte:scte35:2013:bin" a complete SCTE 35 section as a payload. Sum of the earliest presentation time of the `emsg`-bearing segment and `emsg`.presentation_time_delta is a translation of the splice time into the media time-line.

Presence of inband SCTE 35 events shall always be signaled using **AdaptationSet.InbandEventStream** element with @schemeIdUri="urn:scte:scte35:2013:xml".

Event message boxes shall be aligned: if segment $S_{R1}(i)$ is the i^{th} segment of a representation *R1* within an adaptation set containing N representations, then if $S_{R0}(i)$ contains an `emsg` box, identical `emsg` box will be carried in segments $S_{R2}(i) \ldots, S_{RN}(i)$. This means that irrespective of representation selected, all `emsg` boxes will be read if media from an adaptation set is played out.

6 References

DASH Industry Forum, "Guidelines for Implementation: DASH264 Interoperability Points", http://dashif.org/w/2013/08/DASH-AVC-264-v2.00-hd-mca.pdf .

ISO/IEC 23009-1:2014 2nd Ed., Information technology -- Dynamic adaptive streaming over HTTP (DASH) -- Part 1: Media presentation description and segment formats.

IAB Video Multiple Ad Playlist (VMAP),http://www.iab.net/media/file/VMAPv1.0.pdf

IAB Video Ad Serving Template (VAST), http://www.iab.net/media/file/VASTv3.0.pdf

ANSI/SCTE 35 2013a, Digital Program Insertion Cueing Message for Cable

ANSI/SCTE 67 2014, Recommended Practice for SCTE 35 Digital Program Insertion Cueing Message for Cable

ANSI/SCTE 172, Constraints on AVC Video Coding for Digital Program Insertion

EIDR ID Format - EIDR_ID_Format_v1.02_Jan2012-1.pdf, www.eidr.org

Common Metadata, TR-META-CM, ver. 2.0, January 3, 2013, available at , http://www.movielabs.com/md/md/v2.0/Common_Metadata_v2.0.pdf