Draft Media and Timed Meta-Data Ingest

**Status: Draft**

**Source: Unified Streaming Platform**

**Author: Rufael Mekuria, Dirk Griffioen, Arjen Wagenaar**

## 1. Overview

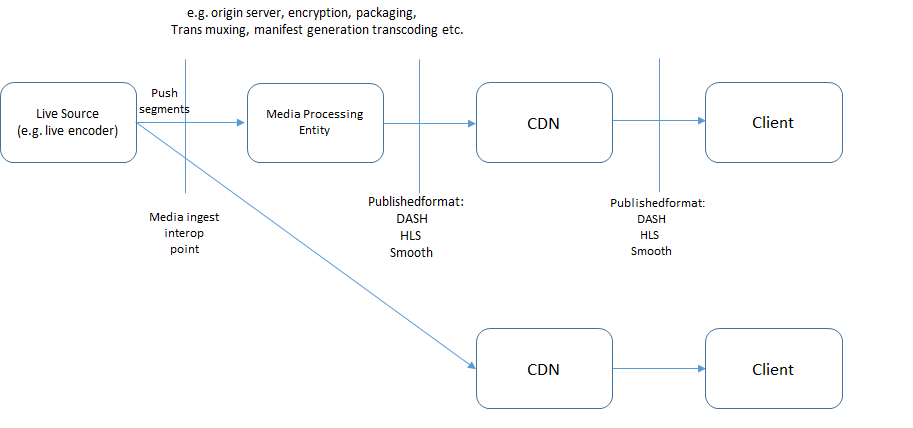


Figure 1 Live Media streaming workflow with media processing with live encoder ingest

This specification describes a protocol and format for live media ingest to media processing and CDN based on fragments of ISO Base media file format [[1](#ISO16)]. It serves as a starting point for an industry wide specification for live media ingest following current best practices.

The workflow architecture diagram is shown in Figure 1. A live streaming entity (e.g. a live encoder) pushes media to a media processing entity or a Content Delivery Network (CDN). The media processing entity delivers functionality for further delivery such as content stitching, encryption, packaging, manifest generation, transcoding, scalable delivery and many other media processing functions.

The connection between a live encoder and a cloud or network based media processing function is still often based on proprietary protocols, as unfortunately MPEG-DASH [[2](#ISO14)] is a client only protocol. MPEG DASH seems not suitable for pushing content from a live encoder to a media processing function, as it only supports pull based requests based on existing manifest. For live encoder ingest still legacy or proprietary protocols are often used leading to interop issues in implementations as many specification are not specified and based on the latest technologies and standards used in the industry (e.g. timed meta-data based, HEVC etc.).

Overall, this is hampering deployments with live encoding and advanced media processing functions (e.g. streaming with content insertion and DRM). In practice, interop problems arise both with the file format (MPEG-2-TS, fMP4), encoder settings, timed meta data insertion (ID3, SCTE 35) and the transmission protocol layer on top of protocols like TCP/USP/HTTP that are used to connect a live encoder to a media processing function. This is also problematic when using multiple live encoders as ingest to a media processing entity. Another issue is passing live meta-data from broadcast workflows into cloud media processing (e.g. ID3 tags, SCTE-35 markers).

To define standardized formats for future networks and cloud based architectures MPEG network based media processing (NBMP) aims to address this interop issue with a new ingest specification to improve interoperability between live encoders and cloud media processing services. The specification covers both media (encoded audio and video) and timed-meta data such as sparse tracks, ID3, SCTE 35 [[2](#Soc)] tracks and so on. The specification deals with both the data format and the connection protocol mechanism in networks.

The following diagram shows the high-level architecture and the scope of the live streaming ingest specification to be developed as part of the Network Based Media Processing framework (NBMP) of standards in MPEG. Other possible places to standardize this draft specification text include the streaming video alliance (SVA) and the DASH industry forum (DASH-IF).

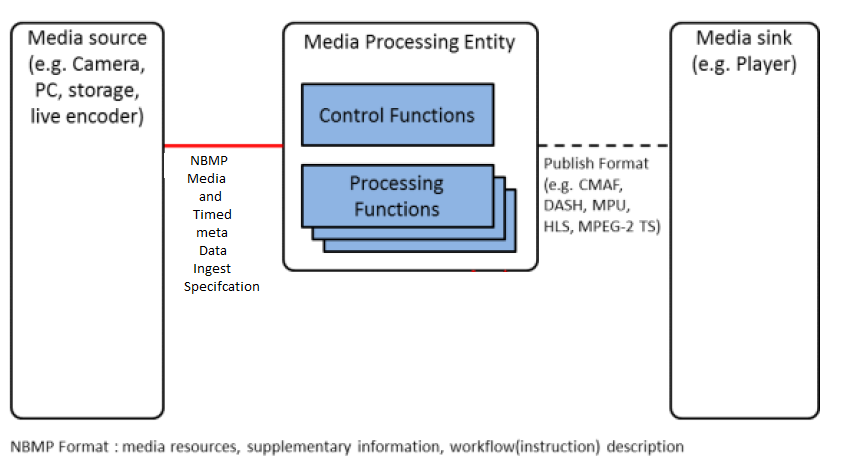


Figure 2 Media and Timed Meta data ingest in the NBMP framework defined by MPEG

The live ingest to a streaming end point comprises the following steps:

1. A live encoder pushes live feeds to a media processing entity (e.g. streaming origin server )
2. The media processing entities handle additional live streaming processing such as formatting, cloud DVR, security, scalability, and redundancy. Alternative media processing functions in the entity could handle more advanced conversions and operations.
3. Optionally, one could choose to deploy a Content Delivery Network layer between the media processing entity and the client endpoints.
4. Clients receive their streams from the streaming endpoint by using HTTP Adaptive Streaming protocols. Examples include Microsoft Smooth Streaming, Dynamic Adaptive Streaming over HTTP (DASH, or MPEG-DASH), and Apple HTTP Live Streaming (HLS), possibly through a content delivery network (CDN).
5. More advanced media processing functions could be applied by media processing entities such as video content insertion/ stitching, graphics overlay, transcoding, High dynamic range conversion of segments, ultra-resolution in future deployments.

This draft is provided for improvement by Unified Streaming Platform based on the original Azure media ingest specification [[4](#Mic2)] and it was shared with Microsoft Azure team in early November with a request for feedback.

## 2. Conformance notation

The key words "MUST," "MUST NOT," "REQUIRED," "SHALL," "SHALL NOT," "SHOULD," "SHOULD NOT," "RECOMMENDED," "MAY," and "OPTIONAL" in this document are to be interpreted as they are described in RFC 2119.

## 3. Terminology

This specification uses the following terminology

ISO BMFF: ISO BMFF in this document refers to part 12 of the ISO/IEC specification which describes the ISO Base media file format and the box structures related in this file format [[1](#ISO16)].

Ftyp: the filetype and compatibility “*ftyp”* box as described in the ISO BMFF [[1](#ISO16)] that describes the brand of the media stream file brand.

Moov: the container for all metadata *“moov”* box as described in the ISO BMFF base media file format [[1](#ISO16)] that describes the metadata of the media and tracks in the presentation.

Moof: the movie fragment “*moof”* box as described in the ISO BMFF base media file format [[1](#ISO16)] that describes the meta data of a fragment of media.

Mdat : the media data container “*mdat”* box contained in an ISO BMFF [[1](#ISO16)], this box contains the physical media samples based on a codec such as for example the HEVC [[2](#IEC15)] codec.

Kind box: the track kind “*kind*” box defined in the ISO BMFF [[1](#ISO16)] to label a track with its usage (i.e. user defined data)

Mfra: the movie fragment random access “*mfra”* box defined in the ISO BMFF [[1](#ISO16)] to signal random access samples [[1](#ISO16)].

Tfdt: the TrackFragmentDecodeTimeBox box “*tfdt”* in MPEG-4 ISO base media file format [[1](#ISO16)] used to signal decode time of the media fragment moof box.

nmhd: The null media header Box as defined in [[1](#ISO16)] to signal a track for which no specifc media header is defined

HTTP: Hyper Text Transfer Protocol, version 1.1 as specified by RFC 2626 [[2](#IET)]

HTTP POST: Command used in the Hyper Text Transfer Protocol for pushing data from a source to a destination [[2](#IET)]

fragmentedMP4stream: stream of ISO BMFF [[1](#ISO16)] fragments (moof and mdat), the payload of the live media ingest.

POST\_URL: Target URL of a post command in the HTTP protocol for pushing data from a source to a destination.

TCP: Transmission Control Protocol (TCP) as defined in RFC 793 [[3](#DAR81)]

URI\_SAFE\_IDENTIFIER: identifier/string formatted according to [[4](#TBe98)]

Connection: connection setup between a host and a source using the TCP protocol and the HTTP POST method.

Live stream event: the total media broadcast stream of the encoder ingest or source to be pushed to the destination.

## 4. Ingest Protocol Requirements

The media and time meta-data ingest specification uses a standard long-running HTTP POST request to transmit encoded media data packaged in fragmented ISO BMFF [[1](#ISO16)] format to the media processing entity. Each HTTP POST sends a complete fragmented MP4 bitstream ("stream"), starting from the beginning with header boxes (**ftyp** and **moov** boxes), and contin-uing with a sequence of fragments (**moof** and **mdat** boxes). An example of the Fragmented Media Ingest POST URL targeting the media processing entity is:

http://customer.channel.mediaservices.windows.net/ingest.isml/streams(720p)

The POST URL and syntax is defined as follows using the RFC 5234 ANB [[9](#IET08)] to specify the structure.

PostURL = Protocol “://” BroadcastURL Identifier

Protocol = "http" / "https"

BroadcastURL = ServerAddress "/" PresentationPath

ServerAddress = URI\_SAFE\_IDENTIFIER

PresentationPath = URI\_SAFE\_IDENTIFIER ".isml"

Identifier = [EventID] StreamID

EventID = "/Events(URI\_SAFE\_IDENTIFIER)”

StreamID = "/" Streams(URI\_SAFE\_IDENTIFIER)"

In this PostURL the server address is typically the hostname or IP address of the media processing entity and the presentation path is the path to the specific media function instance. The identifier, Event ID and stream ID are used to signal the stream and can be generated by various means such as by the system administrator, by the live encoder of by the control plane of the cloud setting, or manually by assigning a number to a stream or service.

The payload and content of the media ingest stream in the long running post operation is a fragmentedMP4stream defined using the RFC 5234 ANB [[9](#IET08)] as follows.

fragmentedMP4stream = HeaderBoxes Fragments

HeaderBoxes = FileType Moov

Fragments = \*1Fragment

Fragment = moof mdat

During operation the communication between the media ingest encoder and the streaming end point follows the following requirements.

1. The encoder or ingest source communicates to the end point using the HTTP POST method as defined in HTTP protocol [[2](#IET)]
2. The encoder SHOULD start the broadcast by sending an HTTP POST request with an empty “body” (zero content length) by using the same ingestion POSTURL. This can help the encoder quickly detect whether the live ingestion endpoint is valid, and if there are any authentication or other conditions required. Per HTTP protocol, the server can't send back an HTTP response until the entire request, including the POST body, is received. Given the long-running nature of a live event, without this step, the encoder might not be able to detect any error until it finishes sending all the data.
3. The encoder MUST handle any errors or authentication challenges because of (1). If (1) succeeds with a 200 response, continue.
4. The encoder MUST start a new HTTP POST request with the fragmented MP4 stream. The payload MUST start with the header boxes **ftyp and moov**, followed by fragments signalled by the moof and mdat boxes. Note that the **ftyp**, and **moov** boxes (in this order) MUST be sent with each request, even if the encoder must reconnect because the previous request was terminated prior to the end of the stream.
5. The encoder SHOULD use chunked transfer encoding option of the HTTP POST command [[2](#IET)] for uploading as it is impossible to predict the entire content length of the live stream.
6. The encoder MAY use individual HTTP POST commands [[2](#IET)] for uploading media fragments when ready if it is possible to predict the entire content length after the fragment became available. The encoder must send the **ftyp** and **moov** boxes (in this order) with each individual request, followed by the media fragments **moof** and **mdat**.
7. When the live stream event is over, after sending the last fragment, the encoder MUST gracefully end the chunked transfer encoding message sequence (most HTTP client stacks handle it automatically) by signalling the stop. The encoder MUST wait for the service to return the final response code, and then terminate the connection.
8. The stop message can be sent by an ingesting entity such as a live encoder to signal an end of stream (end of the live stream event) by sending a movie fragment random access “mfra” box in the stream.
9. If the HTTP POST request terminates or times out with a TCP error prior to the end of the stream, the encoder MUST issue a new POST request by using a new connection, and follow the preceding requirements. Additionally, the encoder MAY resend the previous two MP4 fragments for each track in the stream, and resume without introducing a discontinuity in the media timeline. Resending the last two ISO BMFF fragments for each track ensures that there is no data loss. In other words, if a stream contains both an audio and a video track, and the current POST request fails, the encoder must reconnect and resend the last two fragments for the audio track, which were previously successfully sent, and the last two fragments for the video track, which were previously successfully sent, to ensure that there is no data loss. The encoder MAY maintain a “forward” buffer of media fragments, which it resends when it reconnects.
10. In case of an encoder restart and none of the last two ISO BMFF fragments can be send, the timestamps of **tdft**of fragments that are sent MUST be higher than previously sent media fragments.
11. The **ftyp**, and **moov** boxes MUST be sent with each request (HTTP POST). These boxes MUST be sent at the beginning of the stream and any time the encoder must reconnect to resume stream ingest.
12. The **Tfdt** box MUST be present for each fragment.
13. Version 2 of the **tfdt** box SHOULD be used to generate media segments that have identical URLs in multiple datacentres. The fragment index field is REQUIRED for cross-datacenter failover of index-based streaming formats such as Apple HLS and index-based MPEG-DASH. To enable cross-data centre failover, the time stamps MUST be synced across multiple encoders and be increased by 1 or a multiple of 1 for each successive media fragment, even across encoder restarts or failures. Encoders should use the timing information (Universal Time Stamps) from the original SDI input signal if possible in order to allow exact synchronization of the Universal Time Stamps in the streams. Reconnecting encoders should transmit in sync with other encoders.
14. The ISOBMFF media fragment duration SHOULD be constant, to reduce the size of the client manifests. A constant MP4 fragment duration also improves client download heuristics through the use of repeat tags. The duration MAY fluctuate to compensate for non-integer frame rates. By choosing an appropriate timescale (a multiple of the frame rate is recommended) as this should avoid this issue.
15. The ISO BMFF fragment duration SHOULD be between approximately 2 and 6 seconds.
16. The fragment decode timestamps **tdft** of fragments in the fragmentedMP4stream and the indexes base\_media\_decode\_ time SHOULD arrive in increasing order.
17. The fragmented MP4 stream SHOULD use a 90-KHz timescale for video streams and 44.1 KHz or 48.1 KHz for audio streams or any another timescale that enables integer increments of the decode times of fragments signalled in the **tfdt** box based on this scale.

## 5. Formatting Requirements for Timed Meta-Data Ingest

This section discusses the specific formatting requirements for ingest of timed meta-data.

When delivering a live streaming presentation with a rich client experience, often it is necessary to transmit time-synced events, meta-data or other signals in-band with the main media data. An example of this are opportunities for dynamic live ad insertion signalled by SCTE-35 markers for example. This type of event signalling is different from regular audio/video streaming because of its sparse nature. In other words, the signalling data usually does not happen continuously, and the interval can be hard to predict.

Examples of timed meta-data are ID3 tags ([*http://www.id3.org/*](http://www.id3.org/)), SCTE-35 markers [[3](#Soc)] and DASH emsg messages defined in section 5.10.3.3 of [[2](#ISO14)]. For example, DASH Event messages contain a schemeIdUri that defines the payload of the message. Table 1 provide some example schemes in DASH event messages and Table 2 illustrates an example of a SCTE-35 marker stored in a dash emsg.

*Table 1 Example of DASH emsg schemes URI*

|  |  |  |  |
| --- | --- | --- | --- |
| **Scheme URI** | **Value** | **Description** | **Reference** |
| urn:mpeg:dash:event:2012 | 1 | Signals DASH specific events for DASH clients | ISO / IEC 23009-1 (2014), §5.10.4 |
| urn:dvb:iptv:cpm:2014 | 1 | Basic metadata relating to current program | ETSI TS 103 285, §9.1.2.1 ([pdf](http://www.etsi.org/deliver/etsi_ts/103200_103299/103285/01.01.01_60/ts_103285v010101p.pdf)) |
| urn:scte:scte35:2013:bin | “” | Contains a binary SCTE-35 message | ANSI / SCTE 14-3 (2015), §7.3.2 |
| [www.nielsen.com:id3:v1](http://www.nielsen.com:id3:v1/) | 1 | Contains a Nielsen ID3 tag | [Nielsen ID3 in MPEG-DASH](http://engineeringportal.nielsen.com/docs/ID3_in_MPEG-DASH) |
| <application provider specific> | " |  |  |

*Table 2 example of a scte-35 marker embedded in a DASH emsg*

|  |  |
| --- | --- |
| **Tag** | **Value** |
| scheme\_uri\_id | ["urn:scte:scte35:2013:bin"](urn:scte:scte35:2013:bin) |
| Value | the value of the SCTE 35 PID |
| Timescale | positive number |
| presentation\_time\_delta | non-negative number expressing splice time relative to track fragment base media decode time (tfdt) expressed in timescale |
| event\_duration | duration of event in media presentation time, 0xFFFFFFFF indicates unknown duration |
| Id | unique identifier for message |
| message\_data | splice info section including CRC |

The following steps are a recommended for timed metadata ingest:

1. Create a fragmentedMP4stream that contains only sparse tracks meta-data track, i.e. timed metadata streams which are streams without audio/video tracks.
2. For this meta-data track the media handler type is ("meta") and the tracks handler box is a null media header box ("**nmhd**").
3. The URIMetaSampleEntry entry contains, in a URIbox, the URI following the URI syntax in [[8](#TBe98)] defining the form of the metadata (see the ISO Base media file format specification [[1](#ISO16)]).
4. For example, the URIBox could contain for ID3 tags the URL <http://www.id3.org>
5. For the case of ID3, a sample contains a single ID3 tag. The ID3 tag may contain one or more ID3 frames.
6. For the case of DASH emsg, a sample may contain one or more event message ("emsg") boxes. Version 0 Event Message should be used. The presentation\_time\_delta field is relative to the absolute timestamp specified in the TrackFragmentBaseMediaDecode-TimeBox ("**tfdt**"). The timescale field should match the value specified in the media header box **mdhd**.
7. For the case of a DASH emsg, the **kind** box (contained in the **udta**) shall be used to signal the scheme URI of the types of metadata
8. A BitRateBox ("btrt") should be present at the end of MetaDataSampleEntry to signal the bit rate information of the stream.
9. If the specific format uses internal timing values, then the timescale must match the timescale field set in the media header box **mdhd**.
10. All Timed Metadata samples are sync samples [[**1**](#ISO16)], defining the entire set of metadata for the time interval they cover. Hence, the sync sample table box is not present.
11. When Timed Metadata is stored in a TrackRunBox ("**trun**"), a single sample is present with the duration set to the duration for that run.

Given the sparse nature of the signalling event, the following is recommended:

1. At the beginning of the live event, the encoder sends the initial header boxes to the service, which allows the service to register the sparse track in the client manifest.
2. The encoder SHOULD terminate the HTTP POST request when data is not being sent. A long-running HTTP POST that does not send data can prevent Media Services from quickly disconnecting from the encoder in the event of a service update or server reboot. In these cases, the media server is temporarily blocked in a receive operation on the socket.
3. During the time when signalling data is not available, the encoder SHOULD close the HTTP POST request. While the POST request is active, the encoder SHOULD send data.
4. When sending sparse fragments with a new connection, the encoder SHOULD start sending from the header boxes, followed by the new fragments. This is for cases in which failover happens in-between, and the new sparse connection is being established to a new server that has not seen the sparse track before.
5. The sparse track fragment becomes available to the client when the corresponding parent track fragment that has an equal or larger timestamp value is made available to the client. For example, if the sparse fragment has a timestamp of t=1000, it is expected that after the client sees "video" (assuming the parent track name is "video") fragment timestamp 1000 or beyond, it can download the sparse fragment t=1000. Note that the actual signal could be used for a different position in the presentation timeline for its designated purpose. In this example, it’s possible that the sparse fragment of t=1000 has an XML payload, which is for inserting an ad in a position that’s a few seconds later.
6. The payload of sparse track fragments can be in different formats (such as XML, text, or binary), depending on the scenario

## 6. Live Stream Ingest Options

The fragmented MP4 Stream is the basic unit of operation in live ingestion for composing live presentations, handling streaming failover, and redundancy scenarios. It is defined as one unique, fragmented MP4 bitstream that might contain a single track or multiple tracks. A full live presentation might contain one or more streams, depending on the configuration of the live encoders. The following examples illustrate various options of using streams to compose a full live presentation.

**Example:**

A company wants to create a live streaming presentation that includes the following audio/video bitrates:

Video – 3000 kbps, 1500 kbps, 750 kbps

Audio – 128 kbps

Option 1: All tracks in one stream

In this option, a single encoder generates all audio/video tracks, and then bundles them into one fragmented MP4 bitstream. The fragmented MP4 bitstream is then sent via a single HTTP POST connection. In this example, there is only one stream for this live presentation.

Option 2: Each track in a separate stream

In this option, the encoder puts one track into each fragment MP4 bitstream, and then posts all of the streams over separate HTTP connections. This can be done with one encoder or with multiple encoders. The live ingestion sees this live presentation as composed of four streams.

Option 3: Bundle audio track with the lowest bitrate video track into one stream

In this option, the customer chooses to bundle the audio track with the lowest-bitrate video track in one fragment MP4 bitstream, and leave the other two video tracks as separate streams.

This is not an exhaustive list of all possible ingestion options for this example. As a matter of fact, any grouping of tracks into streams is supported by live ingestion. However given the recent specification of Common Media Application Format (CMAF) [[5](#MPE17)] a single CMAF track per stream could be a preferred option for future media systems. Customers and encoder vendors can choose their own implementations based on engineering complexity, encoder capacity, and redundancy and failover considerations. However, in most cases, there is only one audio track for the entire live presentation. So, it’s important to ensure the healthiness of the ingest stream that contains the audio track. This consideration often results in putting the audio track in its own stream (as in Option 2) or bundling it with the lowest-bitrate video track (as in Option 3). Also, for better redundancy and fault tolerance, sending the same audio track in two different streams (Option 2 with redundant audio tracks) or bundling the audio track with at least two of the lowest-bitrate video tracks (Option 3 with audio bundled in at least two video streams) is highly recommended for fragmented media ingest.

## 7. Service (ingest destination) failover

Given the nature of live streaming, good failover support is critical for ensuring the availability of the service. Typically, media services are designed to handle various types of failures, including network errors, server errors, and storage issues. When used in conjunction with proper failover logic from the live encoder side, customers can achieve a highly reliable live streaming service from the cloud.

In this section, we discuss service failover scenarios. In this case, the failure happens somewhere within the service, and it manifests itself as a network error. Here are some recommendations for the encoder implementation for handling service failover:

1. Use a 10-second timeout for establishing the TCP connection. If an attempt to establish the connection takes longer than 10 seconds, abort the operation and try again.
2. Use a short timeout for sending the HTTP request message chunks. If the target fragment duration is N seconds, use a send timeout between N and 2 N seconds; for example, if the fragment duration is 6 seconds, use a timeout of 6 to 12 seconds. If a timeout occurs, reset the connection, open a new connection, and resume stream ingest on the new connection.
3. Maintain a rolling buffer that has the last two fragments for each track that were successfully and completely sent to the service. If the HTTP POST request for a stream is terminated or times out prior to the end of the stream, open a new connection and begin another HTTP POST request, resend the stream headers, resend the last two fragments for each track, and resume the stream without introducing a discontinuity in the media timeline. This reduces the chance of data loss.
4. We recommend that the encoder does NOT limit the number of retries to establish a connection or resume streaming after a TCP error occurs.
5. After a TCP error:

a. The current connection MUST be closed, and a new connection MUST be created for a new HTTP POST request.

b. The new HTTP POST URL MUST be the same as the initial POST URL for fragmented media ingest.

c. The new HTTP POST MUST include stream headers (**ftyp**, and **moov** boxes) that are identical to the stream headers in the initial POST request for fragmented media ingest.

d. The last two fragments sent for each track may be resent. Other ISO BMFF fragment timestamps must increase continuously, even across HTTP POST requests.

1. The encoder SHOULD terminate the HTTP POST request if data is not being sent at a rate commensurate with the MP4 fragment duration. An HTTP POST request that does not send data can prevent Media Services from quickly disconnecting from the encoder in the event of a service update. For this reason, the HTTP POST for sparse (ad signal) tracks SHOULD be short-lived, terminating as soon as the sparse fragment is sent.

## 8. Ingest source (Encoder) failover

Encoder failover is the second type of failover scenario that needs to be addressed for end-to-end live streaming delivery. In this scenario, the error condition occurs on the encoder side.

The following expectations apply from the live ingestion endpoint when encoder failover happens:

1. A new encoder instance SHOULD be created to continue streaming
2. The new encoder MUST use the same URL for HTTP POST requests as the failed instance.
3. The new encoder’s POST request MUST include the same fragmented MP4 header boxes as the failed instance.
4. The new encoder MUST be properly synced with all other running encoders for the same live presentation to generate synced audio/video samples with aligned fragment boundaries. This implies that UTC timestamps for fragments in the tdft match between decoders, and encoders start running at an appropriate segment boundary.
5. The new stream MUST be semantically equivalent with the previous stream, and interchangeable at the header and fragment levels.
6. The new encoder SHOULD try to minimize data loss. The basemediadecodetime *tdft* of media fragments SHOULD increase from the point where the encoder last stopped. The basemediadecodetime in the tdft box SHOULD increase in a continuous manner, but it is permissible to introduce a discontinuity, if necessary. Media Services can ignore fragments that it has already received and processed, so it's better to err on the side of resending fragments than to introduce discontinuities in the media timeline.

## 9. Encoder/Ingest Source redundancy

For certain critical live event streams that demand even higher availability and quality of experience, it is recommended to use active-active redundant encoders to achieve seamless failover with no data loss. When two groups of encoders push two copies of each stream simultaneously into the live service. This setup is supported because Media Services can filter out duplicate fragments based on stream ID and fragment timestamp. The resulting live stream and archive is a single copy of all the streams that is the best possible aggregation from the two sources. For example, in a hypothetical extreme case, as long as there is one encoder (it doesn’t have to be the same one) running at any given point in time for each stream, the resulting live stream from the service is continuous without data loss.

The requirements for this scenario are as the requirements in the "Encoder failover" case in most cases, with the exception that the second set of encoders are running at the same time as the primary encoders.

## 10. Service/destination redundancy

For highly redundant global distribution, sometimes you must have cross-region backup to handle regional disasters. Expanding on the “Encoder redundancy” topology, customers can choose to have a redundant service deployment in a different region that's connected with the second set of encoders. Customers also can work with a Content Delivery Network provider to deploy a Global Traffic Manager in front of the two service deployments to seamlessly route client traffic. The requirements for the encoders are the same as the “Encoder redundancy” case. The only exception is that the second set of encoders needs to be pointed to a different live ingest endpoint.

## 11. References

x

|  |  |
| --- | --- |
| [1] | ISO/IEC JCT1/SC29 MPEG, "ISO/IEC 23009-1:2014: Dynamic adaptive streaming over HTTP (DASH) -- Part 1: Media presentation description and segment formats," 2014. |
| [2] | Society of Cable Television Engineers, "SCTE-35 (ANSI/SCTE 35 2013) Digital Program Insertion Cueing Message for Cable," SCTE-35 (ANSI/SCTE 35 2013),. |
| [3] | ISO/IEC JTC 1/SC 29, "Information technology -- Coding of audio-visual objects -- Part 12: ISO base media file format," ISO/IEC 14496-12:2012 , 2016. |
| [4] | IEC/ISO, "Information technology -- High efficiency coding and media delivery in heterogeneous environments -- Part 2: High efficiency video coding," ISO/IEC 23008-2:2015, 2015. |
| [5] | IETF. RFC 2626 Hypertext Transfer Protocol -- HTTP/1.1. [Online]. <https://tools.ietf.org/html/rfc2616> |
| [6] | DARPA, "TRANSMISSION CONTROL PROTOCOL," IETF, request for comments (international standards track) RFC 793, 1981. |
| [7] | R. Fielding, L. Masinter, T. Berners Lee, "Uniform Resource Identifiers (URI): Generic Syntax," IETF, IETF Request for comments (international standards track) RFC 2396, 1998. |
| [8] | "(MPEG-A) -- Part 19: Common media application format (CMAF) for segmented media," MPEG, ISO/IEC Draft International standard ISO/IEC FDIS 23000-19, 2017. |
| [9] | ISO/IEC JTC1 SC29 WG11, "Information technology -- Generic coding of moving pictures and associated audio information: Systems," ISO/IEC 13818-1:2007, 2007. |
| [10] | Microsoft Azure. Media ingest workflow with live encoder in Microsoft Azure. [Online]. <https://docs.microsoft.com/en-us/azure/media-services/media-services-manage-live-encoder-enabled-channels> |

x