Live Media and Metadata Ingest Protocol
draft-mekuria-mmediaingest-00.txt

Abstract

This Internet draft presents a protocol specification for ingesting live media and metadata content from a live media source such as a live encoder towards a media processing entity or content delivery network. It defines the media format usage, the preferred transmission methods and the handling of failovers and redundancy. The live media considered includes high quality encoded audio visual content. The timed metadata supported includes timed graphics, captions, subtitles and metadata markers and information. This protocol can for example be used advanced live streaming workflows that combine high quality live encoders and advanced media processing entities. The specification follows best current industry practice.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.
1. Introduction

This specification describes a protocol for media ingest from a live source (e.g. live encoder) towards media processing entities. Examples of media processing entities include media packagers, publishing points, streaming origins, content delivery networks and others. In particular, we distinguish active media processing entities and passive media processing entities. Active media processing entities perform media processing such as encryption, packaging, changing (parts of) the media content and deriving additional information. Passive media processing entities provide pass through functionality and/or delivery and caching functions that do not alter the media content itself. An example of a passive media processing entity could be a content delivery network (CDN) that provides functionalities for the delivery of the content. An example of an active media processing entity could be a just-in-time packager or a just in time transcoder.

Diagram 1: Example workflow with media ingest
Live Media Source -> Media processing entity -> CDN -> End User

Diagram 1 shows the workflow with a live media ingest from a live media source towards a media processing entity. The media processing entity provides additional processing such as content stitching, encryption, packaging, manifest generation, transcoding etc. Such setups are beneficial for advanced
media delivery. The ingest described in this draft includes the latest technologies and standards used in the industry such as timed metadata, captions, timed text and encoding standards such as HEVC [HEVC]. The media ingest protocol specification and associated requirements were discussed with stakeholders, including broadcasters, live encoder vendors, content delivery networks, telecommunications companies and cloud service providers. While this draft specification has also been extensively discussed and reviewed by these stakeholders representing current best practices. Nevertheless, this current draft solely reflects the point of view of the authors of this draft taking received feedback from these stakeholders into account. Some insights on the discussions leading to this draft can be found on [fmp4git].

2. Conventions and Terminology

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 described in BCP 14, RFC 2119 [RFC2119].

This specification uses the following additional terminology.

- ISOBMFF: the ISO Base Media File Format specified in [ISOBMFF].
- ftyp: the filetype and compatibility "ftyp" box as described in the ISOBMFF [ISOBMFF] that describes the "brand"
- moov: the container box for all metadata "moov" described in the ISOBMFF base media file format [ISOBMFF]
- moof: the movie fragment "moof" box as described in the ISOBMFF base media file format [ISOBMFF] that describes the metadata of a fragment of media.
- mdat: the media data container "mdat" box contained in an ISOBMFF [ISOBMFF], this box contains the compressed media samples
- kind: the track kind box defined in the ISOBMFF [ISOBMFF] to label a track with its usage
- mfra: the movie fragment random access "mfra" box defined in the ISOBMFF [ISOBMFF] to signal random access samples (these are samples that require no prior or other samples for decoding) [ISOBMFF].
- tfdt: the TrackFragmentDecodeTimeBox box "tfdt" in the base media file format [ISOBMFF] used to signal the decode time of the media fragment signalled in the moof box.

mdhd: The media header box "mdhd" as defined in [ISOBMFF], this box contains information about the media such as timescale, duration, language using ISO 639-2/T codes [ISO639-2]

pssh: The protection specific system header "pssh" box defined in [CENC] that can be used to signal the content protection information according to the MPEG Common Encryption (CENC)
sinf: Protection scheme information box "sinf" defined in [ISOBMFF] that provides information on the encryption
scheme used in the file

elng: extended language box "elng" defined in [ISOBMFF] that
can override the language information

nmhd: The null media header Box "nmhd" as defined in [ISOBMFF]
to signal a track for which no specific
media header is defined, often used for metadata tracks

HTTP: Hyper Text Transfer Protocol,
version 1.1 as specified by [RFC2626]

HTTP POST: Command used in the Hyper Text Transfer Protocol for
sending data from a source to a destination [RFC2626]

fragmentedMP4stream: stream of [ISOBMFF] fragments
(moof and mdat) see page 5 for definition

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 [RFC793]

URI_SAFE_IDENTIFIER: identifier/string
formatted according to [RFC3986]

Connection: connection setup between a host and a source.
Live stream event: the total media broadcast stream of the ingest.
(Live) encoder: entity performing live encoding and producing
a high quality encoded stream, can serve as Media ingest source
(Media) Ingest source: a media source ingesting media content,
typically a live encoder but not restricted to this,
the media ingest source could by any type of media ingest
source such as a stored file that is send in partial chunks
Publishing point: entity used to publish the media content,
consumes/receives the incoming media ingest stream
Media processing entity: entity used to process media content,
receives/consumes a media ingest stream.
Media processing function: Media processing entity

3. Media Ingest Protocol Behavior

The specification uses multiple HTTP POST and/or PUT requests
to transmit an optional manifest followed by encoded media data
packaged in fragmented [ISOBMFF]. The subsequent posted segments
correspond to those decribed in the manifest. Each HTTP POST sends
a complete manifest or media segment towards the processing entity.
The sequence of POST commands starts with the manifest and init
segments that includes header boxes (ftyp and moov boxes).
It continues with the sequence of segments
(combinations of moof and mdat boxes).

An example of a POST URL
targeting the publishing point is:
http://HostName/presentationPath/manifestpath
/rsegmentpath/Identifier

The PostURL the syntax is defined as follows using the
IETF RFC 5234 ANB [RFC5234] to specify the structure.

PostURL = Protocol ://BroadcastURL Identifier
Protocol = "http" / "https"
BroadcastURL = HostName "/" PresentationPath
HostName = URI_SAFE_IDENTIFIER
PresentationPath = URI_SAFE_IDENTIFIER
ManifestPath = URI_SAFE_IDENTIFIER
Rsegmentpath = URI_SAFE_IDENTIFIER
Identifier = segment_file_name

In this PostURL the HostName is typically the hostname of the media processing entity or publishing point. The presentation path is the path to the specific presentation at the publishing point. The manifest path can be used to signal the specific manifest of the presentation. The Rsegmentpath can be a different optional extended path based on the relative paths in the manifest file. The identifier describes the filename of the segment as described in the manifest. The live source sender first sends the manifest to the path http://hostname/presentationpath/ allowing the receiving entity to setup reception paths for the following segments and manifests. In case no manifest is used any POST_URL setup for media ingest such as http://hostname/presentationpath/ can be used. The fragmentedMP4stream can be defined using the IETF RFC 5234 ANB [RFC5234] as follows.

fragmentedMP4stream = headerboxes fragments
headerboxes = ftyp moov
fragments = X fragment
fragment = Moof Mdat

The communication between the live encoder/media ingest source and the receiving media processing entity follows the following requirements.

1. The live encoder or ingest source communicates to the publishing point/processing entity using the HTTP POST method as defined in the HTTP protocol [RFC2626], or in the case for manifest updates the HTTP PUT Method.
2. The live encoder or ingest source SHOULD start by sending an HTTP POST request with an empty "body" (zero content length) by using the same POSTURL. This can help the live encoder or media ingest source to quickly detect whether the live ingest publishing point is valid, and if there are any authentication or other conditions required.

3. The live encoder/media source SHOULD use secured transmission using HTTPS protocol as specified in [RFC2818] for connecting to the receiving media processing entity or publishing point.
4. In case HTTPS protocol is used, basic authentication HTTP AUTH [RFC7617] or better methods like TLS client certificates SHOULD be used to secure the connection.
5. As compatibility profile for the TLS encryption we recommend the mozilla intermediate compatibility profile which is supported in many available implementations [MozillaTLS].
6. Before sending the segments based on fragmentedMP4stream the live encoder/source
MAY send a manifest
with the following the limitations/constraints.

6a. Only relative URL paths to be used for each segment
6b. Only unique paths are used for each new presentation
6c. In case the manifest contains these relative paths, these paths MAY be used in combination with the POST_URL + relative URLs to POST each of the different segments from the live encoder or ingest source to the processing entity.

6d. In case the manifest contains no relative paths, or no manifest is used the segments SHOULD be posted to the original POST_URL specified by the service.
6e. In this case the tdft and trackids MAY be used by the processing entity to distinguish incoming segments instead of the target POST_URL.

7. The live encoder MAY send an updated version of the manifest, this manifest cannot override current settings and relative paths or break currently running and incoming POST requests. The updated manifest can only be slightly different from the one that was send previously, e.g. introduce new segments available or event messages. The updated manifest SHOULD be send using a PUT request instead of a POST request.

Note: this manifest will be useful for passive media processing entities mostly, for ingest towards active media processing entities this manifest could be avoided and information is signalled through the boxes available in the ISOBMFF.

8. The encoder or ingest source MUST handle any error or failed authentication responses received from the media processing entity such as 403 (forbidden), 400 bad request, 415 unsupported media type, 412 not fulfilling conditions.

9. In case of a 412 not fullfilling conditions or 415 unsupported media type, the live source/encoder MUST resend the init segment consisting of a "moov" and "ftyp" box.

10. The live encoder or ingest source SHOULD start a new HTTP POST segment request sequence with the init segment including header boxes "ftyp" and "moov"

11. Following media segment requests SHOULD be corresponding to the segments listed in the manifest if a manifest was sent.

12. The payload of each request MAY start with the header boxes "ftyp" and "moov", followed by segments which consist of a combination of "moof" and "mdat" boxes.

Note that the "ftyp", and "moov" boxes (in this order) MAY be transmitted with each request, especially if the encoder must reconnect because the previous POST request was terminated prior to the end of the stream with a 412 or 415 message. Resending the "moov" and "ftyp" boxes allows the receiving entity to recover the init segment and the track information needed for interpreting the content.
13. The encoder or ingest source MAY use chunked transfer encoding option of the HTTP POST command [RFC2626] for uploading as it might be difficult to predict the entire content length of the segment. This can be used for example to support use cases that require low latency.

14. The encoder or ingest source SHOULD use individual HTTP POST commands [RFC2626] for uploading media segments when ready.

15. 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 segments that were already sent again.

16. In case fixed length POST Commands are used, the live source entity MUST resend the segment to be posted described in the manifest entirely in case of responses HTTP 400, 412 or 415 together with the init segment consisting of "moov" and "ftyp" boxes.

17. In case the live stream event is over the live media source/encoder should signal the stop by transmitting an empty "mfra" box towards the publishing point/processing entity.

18. The trackFragmentDecodeTime box "tfdt" box MUST be present for each segment posted.

19. The ISOBMFF media fragment duration SHOULD be constant, to reduce the size of the client manifests. A constant MPEG-4 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) this issue can be avoided.

20. The MPEG-4 fragment duration SHOULD be between approximately 2 and 6 seconds.

21. The fragment decode timestamps "tfdt" of fragments in the fragmentedMP4stream and the indexes base_media_decode_time SHOULD arrive in increasing order for each of the different tracks/streams that are ingested.

22. The segments formatted as fragmented MP4 stream SHOULD use a timescale for video streams based on the framerate and 44.1 KHz or 48 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.

23. The manifest MAY be used to signal the language of the stream, which SHOULD also be signalled in the "mdhd" box or "elng" boxes in the init segment and/or moof headers ("mdhd")

24. The manifest SHOULD be used to signal encryption specific information, which SHOULD also be signalled in the "pssh", "schm" and "sinf" boxes in the segments of the init segment and media segments.

25. The manifest SHOULD be used to signal information about the different tracks such as the durations, media encoding types, content types, which SHOULD also be signalled in the "moov" box in the init segment or the "moof" box.
26. The manifest SHOULD be used to signal information about the timed text, images and sub-titles in adaptation sets and this information SHOULD also be signalled in the "moov" box in the init segment, for more information see the next section.

27. Segments posted towards the media processing entity MUST contain the bitrate "btrt" box specifying the target bitrate of the segments and the "tfdt" box specifying the fragments decode time and the "tfhd" box specifying the track id.

28. The live encoder/media source SHOULD repeatedly resolve the Hostname to adapt to changes in the IP to Hostname mapping such as for example by using the dynamic naming system DNS [RFC1035] or any other system that is in place.

29. The Live encoder media source MUST update the IP to hostname resolution respecting the TTL (time to live) from DNS query responses, this will enable better resilience to changes of the IP address in large scale deployments where the IP address of the publishing point media processing nodes may change frequently.

30. To support the ingest of live events with low latency, shorter segment and fragment durations MAY be used such as segments with a duration of 1 second.

31. The live encoder/media source SHOULD use a separate TCP connection for ingest of each different bit-rate tracks ingested

4. Formatting Requirements for Timed Text, Captions and Subtitles

The specification supports ingest of timed text, images, captions and subtitles. we follow the normative reference [MPEG-4-30] in this section.

1. The tracks containing timed text, images, captions or subtitles MAY be signalled in the manifest by an adaptationset with the different segments containing the data of the track.

2. The segment data MAY be posted to the URL corresponding to the path in the manifest for the segment, else they MUST be posted towards the original POST_URL.

3. The track will be a sparse track signalled by a null media header "nmhd" containing the timed text, images, captions corresponding to the recommendation of storing tracks in fragmented MPEG-4 [CMAF].

4. Based on this recommendation the trackhandler "hdlr" shall be set to "text" for WebVTT and "subt" for TTML.

5. In case TTML is used the track must use the XMLSampleEntry to signal sample description of the sub-title stream.

6. In case WebVTT is used the track must use the WVTTSampleEntry to signal sample description of the text stream.

7. These boxes SHOULD signal the mime type and specifics as described in [CMAF] sections 11.3, 11.4 and 11.5.

8. The boxes described in 3-7 must be present in the init segment ("ftyp" + "moov") for the given track.

9. subtitles in CTA-608 and CTA-708 can be transmitted.
following the recommendation section 11.5 in [CMAF] via SEI messages in the video track

10. The "ftyp" box in the init segment for the track containing timed text, images, captions and sub-titles can use signalling using CMAF profiles based on [CMAF]

10b. TTML IMSCL Text Specified in 11.3.3 [MPEG-4-30] IMSCL Text Profile ‘im1t’
10c. TTML IMSCL Image Specified in 11.3.4 [MPEG-4-30] IMSCL Image Profile ‘im1i’
10d. CEA CTA-608 and CTA-708 Specified in 11.4 [MPEG-4-30] Caption data is embedded in SEI messages in video track; ‘ccel’

11. The segments of the tracks containing Timed Text, Images, Captions and Sub-titles SHOULD use the bit-rate box "btrt" to signal bit-rate of the track in each segment.

5. Formatting Requirements for Timed Metadata

This section discusses the specific formatting requirements for ingest of timed metadata related to events and markers for ad-insertion or other timed metadata relating to the media content such as information about the content. When delivering a live streaming presentation with a rich client experience, often it is necessary to transmit time-synced events, metadata or other signals in-band with the main media data. An example of these are opportunities for dynamic live ad insertion signalled by SCTE-35 markers. 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 metadata are ID3 tags [ID3v2], SCTE-35 markers [SCTE-35] and DASH emsg messages defined in section 5.10.3.3 of [DASH]. For example, DASH Event messages contain a schemeIdUri that defines the payload of the message. Table 1 provides some example schemes in DASH event messages and Table 2 illustrates an example of a SCTE-35 marker stored in a dash emsg. The presented approach allows ingest of timed metadata from different sources, possibly on different locations by embedding them in sparse metadata tracks.

<table>
<thead>
<tr>
<th>Scheme URI</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>-------------</td>
<td>-----------</td>
</tr>
</tbody>
</table>

Table 1 Example of DASH emsg schemes URI
Table 2 example of a SCTE-35 marker embedded in a DASH emsg

<table>
<thead>
<tr>
<th>Tag</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>scheme_uri_id</td>
<td>&quot;urn:scte:scte35:2013:bin&quot;</td>
</tr>
<tr>
<td>Value</td>
<td>the value of the SCTE 35 PID</td>
</tr>
<tr>
<td>Timescale</td>
<td>positive number</td>
</tr>
<tr>
<td>presentation_time_delta</td>
<td>non-negative number expressing splice time relative to tfdt</td>
</tr>
<tr>
<td>event_duration</td>
<td>duration of event</td>
</tr>
<tr>
<td>Id</td>
<td>unique identifier for message</td>
</tr>
<tr>
<td>message_data</td>
<td>splice info section including CRC</td>
</tr>
</tbody>
</table>

The following steps are recommended for timed metadata ingest related to events, tags, ad markers and program information:

1. Create a fragmentedMP4stream that contains only a sparse metadata track which are tracks without audio/video.
2. Metadata tracks MAY be signalled in a manifest using an adaptationset with a sparse track, the actual data is in the sparse media track in the segments.
3. For a metadata track the media handler type is "meta" and the tracks handler box is a null media header box "nmhd".
4. The URI MetaSampleEntry entry contains, in a URIBox, the URI following the URI syntax in [RFC3986] defining the form of the metadata

(see the ISO Base media file format specification [ISOBMFF]). For example, the URIBox could contain for ID3 tags [ID3v2] 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 e-msg, 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 e-msg, the kind box (contained in the udta) MUST be used to signal the scheme URI of the type 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 [ISOBMFF], 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:

12. At the beginning of the live event, the encoder or media ingest source sends the initial header boxes to the processing entity/publishing point, which allows the service to register the sparse track.

13. When sending segments, the encoder SHOULD start sending from the header boxes, followed by the new fragments.

14. The sparse track segment becomes available to the publishing point/processing entity when the corresponding parent track fragment that has an equal or larger timestamp value is made available. For example, if the sparse fragment has a timestamp of \( t=1000 \), it is expected that after the publishing point/processing entity sees "video" (assuming the parent track name is "video") fragment timestamp 1000 or beyond, it can retrieve 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 is possible that the sparse fragment of \( t=1000 \) has an XML payload, which is for inserting an ad in a position that is a few seconds later.

15. The payload of sparse track fragments can be in different formats (such as XML, text, or binary), depending on the scenario.

6. Guidelines for Handling of Media Processing Entity 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 requests. If the target segment duration is N seconds, use a send timeout between N and 2N seconds; for example, if the segment 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. This is needed to avoid latency introduced by failing connectivity in the workflow.

3. Completely resend segments from the ingest source for which a connection was terminated early.

4. We recommend that the encoder or ingest source 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 the segment to be ingested.
   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 segment MAY be retransmitted. Other ISOBMFF fragment timestamps MUST increase continuously, even across HTTP POST requests.

6. The encoder or ingest source SHOULD terminate the HTTP POST request if data is not being sent at a rate commensurate with the MP4 segment duration. An HTTP POST request that does not send data can prevent publishing points or media processing entities from quickly disconnecting from the live encoder or media ingest source 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.

In addition this draft defines responses to the POST requests in order to signal the live media source its status.

7. In case the media processing entity cannot process the manifest or segment POST request due to authentication or permission problems then it can return a permission denied HTTP 403.

8. In case the media processing entity can process the manifest or segment POSTED to the POST_URL it returns HTTP 200 OK or 202 Accepted.

9. In case the media processing entity can process the manifest or segment POST request but finds the media type cannot be supported it returns HTTP 415 unsupported media type.

10. In case an unknown error happened during the processing of the HTTP POST request a HTTP 400 Bad request is returned.
11. In case the media processing entity cannot process a segment posted due to missing init segment, a HTTP 412 unfulfilled condition is returned.

12. In case a media source receives an HTTP 412 response, it SHOULD resend the manifest and "ftyp" and "moov" boxes for the track.

An example of media ingest with failure and HTTP responses is shown in the following figure:
7. Guidelines for Handling of Live Media Source Failover

Encoder or media ingest source 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 or media ingest source instance SHOULD be created to continue streaming.
2. The new encoder or media ingest source MUST use the same URL for HTTP POST requests as the failed instance.
3. The new encoder or media ingest source POST request MUST include the same header boxes moov and ftyp as the failed instance.
4. The new encoder or media ingest source 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 media fragment levels.
6. The new encoder or media ingest source SHOULD try to minimize data loss. The basemediacontime tdt of media fragments SHOULD increase from the point where the encoder last stopped. The basemediacontime in the "tdf" box SHOULD increase in a continuous manner, but it is permissible to introduce a discontinuity, if necessary. Media processing entities or publishing points can ignore fragments that it has already received and processed, so it is better to error on the side of resending fragments than to introduce discontinuities in the media timeline.

8. Security Considerations

No security considerations except the ones mentioned in the preceding text. Further security considerations will be updated when they become known.

9. IANA Considerations

This memo includes no request to IANA.

10. Contributors

Arjen Wagenaar, Dirk Griffioen, Unified Streaming B.V.
We thank all of the individual contributors to the discussions in [fmp4git] representing major content delivery networks, broadcasters, commercial encoders and cloud service providers.
11. References

11.1. Normative References


<Mekuria> Expires November 7 2018 [Page15]

Coding of audio-visual objects -- Part 30":
Timed text and other visual overlays in
ISO base media file format

of Languages -- Part 2 ISO 639-2:1998"

[DVB-DASH] ETSI Digital Video Broadcasting
"MPEG-DASH Profile for Transport of ISOBMFF
Based DVB Services over IP Based Networks"
ETSITs 103 285

[RFC7617] J Reschke "The 'Basic' HTTP Authentication Scheme"
IETF RFC 7617 September 2015

11.2. Informative References

[RFC2626] R. Fielding et al
"Hyper Text Transfer Protocol HTTP/1.1",
RFC 2626 June 1999

[RFC2818] E. Rescorla RFC 2818 HTTP over TLS
IETF RFC 2818 May 2000

11.3. URL References

[fmp4git] Unified Streaming Github fmp4 ingest,
"https://github.com/unifiedstreaming/fmp4-ingest".

[MozillaTLS] Mozilla Wikie Security/Server Side TLS
https://wiki.mozilla.org/Security/Server_Side_TLS
#Intermediate_compatibility_.28default.29
(last accessed 30th of March 2018)

[ID3v2] M. Nilsson "ID3 Tag version 2.4.0 Main structure"
http://id3.org/id3v2.4.0-structure
November 2000 (last accessed 2nd of May 2018)

Author’s Address

Rufael Mekuria (editor)
Unified Streaming
Overtoom 60 1054HK

Phone: +31 (0)202338801
E-Mail: rufael@unified-streaming.com