Broadpeak

October 4, 2021

MSYNC
draft-bichot-msync-02

Abstract

This document describes the Multicast Synchronisation (MSYNC) Protocol that aims at transferring video media objects over IP multicast operating preferably RTP. Although generic, MSYNC has been primarily designed for transporting HTTP adaptive streaming (HAS) objects including manifest/playlists and media segments (e.g. MP4, CMAF) according to an HAS protocol such as Apple HLS or MPEG DASH between a multicast server and a multicast gateway.

Table of Contents

1 Introduction

MSYNC relies preferably on RTP that makes it particularly suited for transitioning IPTV legacy(MPEG2 TS/RTP) to the HAS ecosystem. MSYNC is simple (no flow control, no forward error correction) although reliable, flexible and extensible; it has been experimented and deployed over IPTV infrastructure (xDSL, cable, fiber) and satellite.

1.1 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 RFC 2119 [RFC2119].

1.2 Definitions

manifest: A file gathering the configuration for conducting a streaming session; corresponds to a play list as defined by HLS [RFC8216]. During a HAS streaming session, a manifest or playlist can be modified.

media chunk: A piece of a media segment of a fixed duration as specified in [MPEGCMAF].

media segment: A piece of a media sub-stream of a fixed duration (e.g. 2s) as specified in [MPEGCMAF].

init segment: A piece of a media sub-stream used to initialize the decoder as specified in [MPEGCMAF].

media: A digitalized piece of video, audio, subtitle, image, ...

media stream: Gathers one or more media sub-streams.

media sub-stream: A version of a media encoded in a particular bit-rate, format and resolution; also called representation or variant stream.

variant stream : A media sub-stream as defined by HLS [RFC8216]; corresponds to a representation as defined by [MPEGDASH].

representation: A media sub-stream as defined by [MPEGDASH]; Corresponds to a variant stream as defined by HLS [RFC8216].

HTTP Adaptive Streaming (HAS) session: Transport one or more media

streams (e.g. one video, two audios, One subtitle) according to HTTP. A HAS session is triggered by a player downloading first a manifest file(s), then init and/or media segments (belonging to possibly different sub-streams according to the selected representation) and possibly more manifest files according to the HAS protocol.

MSYNC object: As part of a HAS session carried over MSYNC, an MSYSNC object can be an addressable HAS entity like an init segment, a media segment (or fragment, or chunk), a manifest. An MSYNC object can also be a non-addressable transport entity like a part of a segment (an HTTP2 frame or an HTTP 1.1 CTE block). As part of the control channel, an MSYNC object may transport some control plane information (for the receiver as e.g. the multicast gateway configuration). An MSYNC object is typically associated with metadata (aka info), data and possibly an HTTP header.

MSYNC packet: The transport unit of MSYNC. Several MSYNC packets mays be used to transport an MSYNC object.

transport multicast session: Operating a transport protocol that is (possibly based on) UDP over IP multicast. A session is identified by the transport (UDP) port number, the source IP address and the IP multicast address.

RTP multicast session: A transport multicast session based on RTP as defined in [RFC3550].

IP multicast session: A session gathering transport multicast sessions having the same source IP address and destination multicast IP address.

MSYNC channel: The set of transport multicast sessions carrying a HAS session as a set of MSYNC objects.

MSYNC control channel: the transport multicast session carrying control plane MSYNC objects.

2. Overview

MSYNC is a simple protocol typically used between a multicast server (the MSYNC sender) and a multicast gateway (the MSYNC receiver). The multicast server gets ingested with a unicast HAS session conforming to a HAS protocol as e.g. MPEG DASH [MPEGDASH] or HLS [RFC8216] and sends the HAS session elements over a broadcast/multicast link according to MSYNC supporting [possibly RTP/] UDP/IP multicast up to

the multicast gateway(s) that serve the HAS player(s) in unicast conforming to the same HAS protocol. MSYNC can serve simultaneously multiple terminals conforming to one or several HAS protocols and formats.

The multicast server is configured in order to get the unicast HAS feeds. Considering one among several possible ingest methods (e.g. HTTP GET), for each ingested feed, the multicast server behaves as a sort of player, reading the manifest, discovering the available representations and downloading concurrently media segments of all (or a subset) of the available representations. Finally the multicast server is configured for sending all those HAS session elements over [possibly RTP/]UDP/IP multicast according to a certain UDP flow arrangement (for example all the objects related to each video representation are sent over a separate multicast transport session (multicast IP address + port number) whereas all audio representations are sent over the same transport multicast session.

The multicast gateway is configured accordingly in order to be attached to the transport multicast sessions (in particular, it has to subscribe to the corresponding multicast IP group address). Note that the multicast gateway might not be capable of being attached to all the concurrent transport multicast sessions in the same time per bandwidth restriction (e.g. ADSL). In that case, the multicast gateway attaches to the transport multicast session corresponding to the player's request (and detaches from the other previous one).

The multicast gateway then receives the corresponding MSYNC objects and feeds a local cache. Whenever a HAS request is sent by a user terminal (e.g. the media player) and received by the multicast gateway, the latter reads first its local cache. In case of cache hit, it returns the object. In case of cache miss, the multicast gateway can retrieve the requested object from the associated CDN (or a dedicated server) over an unicast interface (if existing) through operating HTTP conventionally and forwards back to the terminal the object once retrieved.

At any time, the multicast gateway can detect corrupted and lost packets and attempt to repair using a repair protocol. This is possible thanks to the RTP protocol if used as the transport layer over UDP.

With MSYNC deployed over a multicast link/network, the end user media player gets basically the HAS content in full transparency (i.e. the player is absolutely unaware of getting the content through MSYNC or not).

Note that nothing precludes the multicast gateway to be co-located

with the media player and therefore embedded in the end-user terminal.

Note that nothing precludes application dependent features in the multicast server and/or the multicast gateway that may adapt/modify the content delivered to the end-user player.

3. MSYNC Protocol

3.1. MSYNC packet format

The MSYNC packet has the following format.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   version     |  packet type  |        object identifier      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                            sub-header                         |
|                               ....                            |
+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
|                             data                              |
|                             ....                              |
```

version: 8 bits
   version of the MSYNC protocol = 0x3

packet type: 8 bits
   Defines the MSYNC packet type. The sub-header and the associated data (if any) are dependent on the packet type. The following types are defined.
     0x01: object info
     0x02: object info redundancy packet
     0x03: object data
     0x04: Reserved
     0x05: object http header
     0x06: object data-part as a piece of an object data for transporting e.g. an MPEG CMAF chunk, an HTTP 1.1 chunk or yet an HTTP2 frame.

object identifier: 16 bits
   The field identifies the object being transferred. All MSYNC packets associated with the same object carry the same object identifier in their MSYNC packet header.

sub-header: series of N x 32 bits
   The packet sub-header is linked to the packet type. The details of

each packet type is given in the next sections.

data: series of D x 8 bits
This field is optional and is present depending on the packet type. D is bounded by the maximum size of a transport multicast session protocol packet size and the MTU (Maximum Transfer Unit) otherwise as depicted in 3.6.

3.2. Object info packet

The Object info packet is used to transport the meta-data associated with an object. It permits to characterize the object in term of e.g. size and type. The object information is carried over one object info packet only. The object info packet is typically sent along with the object data it describes.

The object identifier corresponds to the object identifier of the object data packets or the object data-part packets the object info packet relates to.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   version     |  0x1 or 0x2   |        object identifier      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           object size                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                     number of MSYNC packets                   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           object CRC                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| object type   |   Reserved    | mtype |    object URI size    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         media sequence                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
|                          object URI                           |
:                                                               :
:                                                               :
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

object size: 32 bits
The number of bytes that compose the transported object.

number of MSYNC packets: 32 bits
Number of MSYNC packets that compose the transported object.

object CRC: 32 bits
A CRC applied to the object data payload for corruption detection.

mtype: 4 bits
The manifest (playlist) type, the MSYNC INFO is compatible with.
The field can take the following values.
0x00: Not Applicable
0x01: MPEG Dash as specified in [MPEGDASH].
0x02: HLS as specified in [RFC8216].
0x03-0xF: Reserved

object URI size: 12bits
The size in bytes of the object URI field. The value must guarantee that the MSYNC info packet size is not greater than the network MTU.

object type : 8 bits
Defines the type of MSYNC data object associated with this MSYNC info packet
0x00: Reserved
0x01: media manifest (playlist)
0x02: Reserved
0x03: media data or data-part: Transport stream (MPEG2-TS)
0x04: media data or data-part: MPEG4 (CMAF)
0x05: control: control plane information (e.g. multicast gateway configuration)
0x06-0xFF: Reserved

media sequence: 32 bits
It is a sequence number associated with the MSYNC object data (segment or manifest). It is dependent on the mtype value. It is used to synchronize unicast and multicast receptions in the multicast gateway. The values and rules are detailed in the section 3.9 dedicated to the HAS protocol dependencies. The default value is 0x00.

object URI: Quotient(object URI size/32) bits
This the path name associated with the object. It may corresponds to the storage path in the multicast gateway. There MUST be a direct relationship between this URI and the URL associated with an addressable HAS object (segment or CMAF chunk) and/or a manifest request received by the multicast gateway from the terminal/media player. The rules are detailed in the section 3.9 dedicated to the HAS protocol dependencies.

3.3. Object data packet

This MSYNC packet carries part or all of the the object's data

payload. The type of data and the way to process the object's data packets is function of the associated object's info packet. Object payload is transported through a series of object data packets.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   version     |      0x3      |        object identifier      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          object offset                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                               data                            |
:                                                               :
:                                                               :
```

object offset: 32 bits
The index from which the MSYNC object data packet payload is to be written in order to compose the object data at the receiver side (i.e. the multicast gateway). The first data packet of an object has an offset equal to 0.

data: N x 8bits
The size N is not declared; it is bounded by the maximum size of the under-laying transport multicast session packet (e.g. RTP) as depicted in the section 3.6. The total size (number of bytes) of the object data is indicated in the associated object info (field object size).

3.4. Object HTTP header packet The HTTP header packet carries part or all of an HTTP header related to the object (data) to be sent.

There is at most one HTTP header per object that can be repeated.

The object identifier is the same than the one present in the object data packets or object data-part packets it relates to.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   version     |      0x5      |        object identifier      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      header size              |        header offset          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                               data                            |
:                                                               :
:                                                               :
```

header size: 16 bits
An object HTTP header can be transported over one or several under-laying transport packets. This field indicates the total size of the HTTP header in bytes and it is indicated in each the HTTP header's packet.

header offset: 16 bits
The index from which this HTTP header MSYNC packet payload data is to be written in order to complement the HTTP header at the receiver side (i.e the multicast gateway). The first packet of the HTTP header has an offset equal to 0.

data: N x 8bits
The size N is not declared; it is bounded by either the header size field value or by the maximum size of the under-laying transport packet(e.g. RTP)as depicted in the section 3.6.

3.5. Object data-part packet

This MSYNC packet carries part or all of the object data-part payload. The type of data and the way to process the object's data-part packets is function of the associated info packet. Object payload is transported through a series of object data-part packets. The data-part is used when the object corresponds to a "part" (a chunk) of a super object for which the size is unknown (a super object may correspond to a stream or a media segment not yet complete and for which the size is therefore unknown).

All data-part packets belonging to the same data part object has the same object identifier which is the same one present in the object info packet and HTTP header (if any) packets the data-part object relates too.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   version     |      0x6      |         object identifier     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          object offset                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       super object offset                     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                data                           |
:                                                               :
:                                                               :
```

object offset: 32 bits

The index from which the data-part packet payload is to be written in order to compose the object data-part at the receiver side (i.e. the multicast gateway). The first packet of the data-part has an offset equal to 0.

super object offset: 32 bits

The index from which the object part-data packet payload is to be written in order to compose the super object data at the receiver side (i.e. the multicast gateway). The first data-part object composing a super object has the super object offset equal to 0. The super object offset is the same for all object data-part packets composing the same object data-part. Having this field present in the object data-part header (and not in the associated object info header) permits to possibly recompose a super object without the corresponding object info packet.

data: N x 8bits

The size N is not declared; it is bounded by the maximum size of the under-laying transport packet (e.g. RTP) as depicted in the section 3.6. The total size (number of bytes) of the object data is indicated in the associated object info (field object size).

3.6. Maximum size of a MSYNC packet

An MSYNC packet is composed with a header part and a data part for which the size is bounded by the transport multicast session packet. In case there is no particular restriction as with RTP and/or UDP (which authorize up to 65235 bytes), then the maximum size is linked to the path MTU (Maximum Transfer Unit) as the largest transfer unit supported between the source (the multicast server) and the destination (the multicast gateway) without fragmentation. An MSYNC packet must fit within a link layer packet.

For Ethernet, as an example, the MTU is typically 1500 bytes, assuming a 20 bytes IPv4 header, a 8 bytes UDP header and the 8 bytes MSYNC object data packet header, it gives an MTU of 1464 bytes for the MSYNC object data packet. Operating RTP, the MSYNC object data MTU is decreased by 12 bytes (= 1452 bytes).

3.7. Sending MSYNC objects over IP/transport multicast sessions

The following considerations are linked to the multicast server configuration.

Per MSYNC channel, the way to map MSYNC objects related to a media stream with an IP or transport multicast session is not constrained. The arrangement is chosen function of the network

architecture and capacity. For example, in xDSL, the capacity dedicated to multicast is limited which may drive to an arrangement where each sub-stream/representation of a HAS session/MSYNC channel matches with one dedicated IP multicast session. The receiver (the multicast gateway) switches to the IP transport session corresponding to the sub-stream/representation it must serve to the user terminal/player. Alternatively, one would like to have one IP multicast session (with possibly multiple transport multicast sessions, each having a different destination port number) for the entire HAS session/MSYNC channel that is an arrangement a la "IPTV", less bandwidth efficient but where only one multicast IP address is allocated per HAS session/MSYNC channel.

Considering a satellite network, as all transport multicast sessions are carried simultaneously, all arrangements may make sense.

Regarding the mapping with a transport multicast session, the triplet: source IP address, destination multicast IP address and destination transport port number is the discriminator. It is recommended to carry media sub-streams and the control channel in separate transport multicast channels. It facilitates potential error correction procedures.

The following granularity is possible:

- One IP multicast session per media (audio or video or subtitle) sub-stream (representation); each transport multicast session having a different destination multicast IP address.

- One transport multicast session for the MSYNC control channel.

- It is perfectly possible to send all the MSYNC packets in only one transport multicast session.

For each MSYNC object to be sent, the sender MUST send one object info packet, 0 or more object info redundant packets, zero or more HTTP header packets and one or more object data packets or object data-part packets.

The sender MUST send the object's HTTP header along with the corresponding object's data to be sent through using the MSYNC object data packet(s). The sender MUST send the object's HTTP header along with the corresponding first object's data-part to be sent through using the MSYNC object data-part packet(s)

Whenever a manifest (playlist) has to be sent, the manifest

(playlist) object MUST be sent along with (duplicated in) all the transport multicast sessions related to the transmission of the video segment objects the manifest/playlist refers to.

3.9. HAS protocol dependency

A certain number of MSYNC packet header fields have a dependency on the HAS protocol and therefore on the manifest type. Similarly the sending rules may also depend from the HAS protocol.

3.9.1. Object info packet

3.9.1.1. media sequence

The media sequence is used by the multicast gateway to synchronize the MSYNC (i.e. multicast) reception with unicast reception. The multicast gateway may operate jointly MSYNC and unicast retrieval of HAS objects. This is useful in some occasions like processing a new streaming session request (i.e. a manifest request after a channel switch) or in the case of segment repair. The multicast gateway may attempt to retrieve a manifest object or segment(s) through a unicast mean (e.g. a CDN server or a repair server) in order to speed up the start of the session or to repair damaged object(s). Consequently, the multicast gateway needs to understand the freshness of the HAS object received through multicast with regard to unicast.

If no unicast reception is used jointly with MSYNC in the multicast gateway (e.g. like in one way delivery only), the default value MAY be used: 0x00

HLS master playlist: 0x00

HLS variant playlist; MUST contain the value of EXT-X-MEDIA-SEQUENCE added with the position in the playlist of the last segment transmitted.

HLS segment: MUST contain the value of EXT-X-MEDIA-SEQUENCE added with the position of the segment in the playlist.

DASH manifest: MUST contain $time$/scale or $Number$ corresponding to the last segment transmitted or under transmission (and possibly received partially) and declared by the manifest.

DASH segment: MUST contain the $time$/scale or $Number$ value

3.9.1.2. object URI

In the context of HTTP adaptive streaming, if the object is a HAS addressable entity (e.g. a segment or a CMAF chunk), the path name MUST match the absolute path present in the incoming segment request.

The segment S_2: tvChannel1/Q1/S_2.
The CMAF chunk C_3 of the segment S_2: tvChannel11/Q1/S_2/C_3.

if the object is a non-addressable HAS entity (e.g. a HTTP 1.1 CTE block), the URI MUST hierarchically match with the related incoming segment request.
The HTTP CTE 3rd chunk of the segment S_2
tvChannel11/Q1/S_2/3;

3.9.2. Sending rules

When a manifest/play-list is sent, it must reference addressable objects (segment or CMAF chunk) that have already been sent or for which the transmission has started.

3.10. RTP as the transport multicast session protocol

RTP [RFC3550] can be used as part of the transport multicast session protocol. Depending on the deployment case (e.g. unidirectional) and the infrastructure in place, the companion RTCP protocol MUST be operated according to the following.

- RTCP usage SHALL conform to [RFC5506]

- RTCP sender report MAY be switched off

- RTCP receiver report MAY be switched off

- RCTP destination port number must be configurable but it must be different than the associated RTP destination port number, i.e. the RTCP destination port number is not necessarily the RTP destination port number + 1 as recommended in [RFC3550].

- RTCP MAY be used for packet loss recovery (aka "RTP Repair"). If packet loss recovery through RTCP is activated then the RTP Repair client and server MUST be compliant with [RFC4585] and [RFC5506]. The RTP Repair client that submit the feedback (FB) messages (according to [RFC5506] and [RFC4585]) is the MSYNC receiver (i.e. the multicast gateway). The RTP Repair server that receives, processes and responds to the feedback messages (FB) MAY be the MSYNC sender (i.e. the multicast server) or it MAY be any intermediate entity acting as a multicast RTP receiver (i.e. capable of receiving the multicast RTP packets).

In any case, the RTP Repair server and the RTP Repair client MUST operate a unicast interface.

Note that instead of relying on "RTP repair", an MSYNC receiver (i.e. the multicast gateway) could attempt to recover HAS elements (segments, manifest) through HTTP (aka "HTTP repair"). However the latter method requires a CDN and is less reactive than operating RTCP.

In addition, each RTP multicast session must operate a different [RFC3550] SSRC number. This guaranties a reparation on the RTP transport multicast session basis.

-RTCP MAY be used for Fast Channel change according to [RFC6585]. The way to operate [RFC6585] is out of scope of this document.

4. IANA Considerations

This document has no actions for IANA.

5. Security Considerations

The multicast communication between the MSYNC sender (multicast server) and the MSYNC receiver (the multicast gateway) should be protected for confidentiality, message corruption and replay attacks. The MSYNC protocol does not gather any security mechanism. MSYNC relies on possibly content protection (Digital Right Management) and on the underlying transport layer and security extensions for providing message integrity/authentication and replay. Secure RTP (SRTP) [RFC3711] and IPSec applied to multicast [RFC5374] are potential candidates for providing such extensions.

5. References

5.1. Normative References

[RFC2119] Key words for use in RFCs to Indicate Requirement Levels. S. Bradner. March 1997. (Format: TXT, HTML) (Updated by RFC8174) (Also BCP0014) (Status: BEST CURRENT PRACTICE) (DOI: 10.17487/RFC2119)

[RFC3550] RTP: A Transport Protocol for Real-Time Applications. H. Schulzrinne, S. Casner, R. Frederick, V. Jacobson. July 2003. (Format: TXT, PS, PDF, HTML) (Obsoletes RFC1889) (Updated by RFC5506, RFC5761, RFC6051, RFC6222, RFC7022, RFC7160, RFC7164, RFC8083, RFC8108) (Also STD0064) (Status: INTERNET STANDARD) (DOI: 10.17487/RFC3550)

[MPEGDASH] "Information technology - Dynamic adaptive streaming over HTTP (DASH) - Part1:Media presentation description and segment formats",ISO/IEC23009-1

[MPEGCMAF] "Information technology - Multimedia application format (MPEG-A) - Part 19:Common media application format (CMAF) for segmented media"ISO/IEC 23000-19

[RFC5506] Support for Reduced-Size Real-Time Transport Control Protocol(RTCP): Opportunities and Consequences. I. Johansson, M. Westerlund. April 2009. (Format: TXT, HTML) (Updates RFC3550, RFC3711, RFC4585)(Status: PROPOSED STANDARD) (DOI: 10.17487/RFC5506)

[RFC4585] Extended RTP Profile for Real-time Transport Control Protocol(RTCP)-Based Feedback (RTP/AVPF). J. Ott, S. Wenger, N. Sato, C. Burmeister, J. Rey. July 2006. (Format: TXT, HTML) (Updated by RFC5506, RFC8108) (Status: PROPOSED STANDARD) (DOI:10.17487/RFC4585)

5.2. Informative References

[RFC3711] The Secure Real-time Transport Protocol (SRTP). M. Baugher, D. McGrew, M. Naslund, E. Carrara, K. Norrman. March 2004. (Format: TXT, HTML) (Updated by RFC5506, RFC6904) (Status: PROPOSED STANDARD) (DOI: 10.17487/RFC3711)

[RFC5374] Multicast Extensions to the Security Architecture for the Internet Protocol. B. Weis, G. Gross, D. Ignjatic. November 2008. (Format: TXT, HTML) (Status: PROPOSED STANDARD) (DOI: 10.17487/RFC5374)

[RFC6585] Unicast-Based Rapid Acquisition of Multicast RTP Sessions. B. VerSteeg, A. Begen, T. Van Caenegem, Z. Vax. June 2011. (Format: TXT, HTML) (Status: PROPOSED STANDARD) (DOI: 10.17487/RFC6285)

[RFC8216] HTTP Live Streaming. R. Pantos, Ed., W. May. August 2017. (Format:TXT, HTML) (Status: INFORMATIONAL) (DOI: 10.17487/RFC8216)

Acknowledgments

The authors will be ever grateful to their late colleague Arnaud Leclerc who has been the initiator of that work.

The authors would like to thank the following people for their feedback: Yann Barateau (Eutelsat).

Authors' Addresses

Sophie Bale
Broadpeak
15 rue Claude Chappe
Zone des Champs Blancs
35510 Cesson-Sevigne
France

Email: sophie.bale@broadpeak.tv

Remy Brebion
Broadpeak
15 rue Claude Chappe
Zone des Champs Blancs
35510 Cesson-Sevigne
France

Email: remy.brebion@broadpeak.tv

Guillaume Bichot (Editor)
Broadpeak
15 rue Claude Chappe
Zone des Champs Blancs
35510 Cesson-Sevigne
France

Email: guillaume.bichot@broadpeak.tv