Internet-Draft
Intended Status: Informational
Expires: May 29, 2026

S. Bale
R. Brebion
G. Bichot
Broadpeak

November 25, 2025

MSYNC
draft-bichot-msync-19

Abstract

This document specifies the Multicast Synchronization (MSYNC) Protocol. MSYNC is intended to transfer video media objects over IP multicast. Although generic, MSYNC has been primarily designed for transporting HTTP adaptive streaming (HAS) objects including manifests/playlists and media segments (e.g., CMAF) according to a HAS protocol such as Apple HLS or MPEG DASH between a multicast sender and a multicast receiver.

Status of this Memo

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/1id-abstracts.html

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html

Copyright and License Notice

Table of Contents

1 Introduction

Transporting media content over multicast is known to be very effective for saving network resources (bandwidth). Multicast is used by Internet service providers for providing IPTV services. IPTV technology relies essentially on MPEG Transport Stream (MPEG TS) format, UDP transport, and IP multicast, whereas the HTTP adaptive bit-rate streaming (HAS), a unicast "Over The Top" technology relies on HTTP /TCP, new container formats such as MP4/CMAF, and signaling protocols such as Apple HLS and MPEG DASH. With the generalization of HAS streaming there is a need to operate an IPTV service in association with HAS streaming technology for unifying the two ecosystems. MSYNC allows transporting HTTP based ABR flows over multicast relying on IP/UDP and optionally RTP that makes it suited for transitioning IPTV legacy (MPEG2 TS) to the HAS ecosystem. Various IPTV infrastructures (xDSL, cable, fiber) and broadcast networks have experimented with, and deployed this protocol.

MSYNC is deployable within a controlled environment wherein multicast distribution relies on a pre-arranged capacity planning.

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 [RFC2119].

1.2 Definitions

ABR: Adaptive Bit Rate streaming is a method that consist of changing the media encoding bit-rate function of the network condition.

HTTP/1.1 CTE: Chunked Transfer Encoding. A method for object delivery over HTTP1.1 of unknown size. See Section 7.1 of [RFC9112]

HTTP Adaptive Streaming (HAS) protocol: an ABR method based on HTTP and signaling procedures described in [MPEGDASH] and in [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 initially downloading a manifest file, then an init segment 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.

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

manifest: A file containing 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: A digitalized piece of video, audio, subtitle, image, etc.

media stream: The aggregate of one or more media sub-streams. This should not be confused with the RTP media stream.

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

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

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

MSYNC object: An MSYSNC object can be an addressable HAS entity like an initialization segment, a media segment or chunk, a manifest or playlist. An MSYNC object can also be a non-addressable transport entity as an HTTP2 frame or an HTTP/1.1 CTE block.

MSYNC super object. An object composed of parts delivered on the fly when the size of this object to be transmitted is unknown in advance. A super object may correspond to a stream or a media segment not yet completely generated/received and the size of which is therefore unknown.

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

MSYNC receiver. The MSYNC end point that receives MSYNC objects over multicast.

MSYNC sender. The MSYNC end point that sends MSYNC objects over multicast according to MSYNC.

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

variant stream: A media sub-stream as defined by HLS [RFC8216];

corresponds to a representation as defined by [MPEGDASH].

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

transport multicast session: Operating a transport protocol that is based on UDP over IP multicast. A transport multicast session is identified by the destination 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].

2. Overview

2.1. A typical MSYNC deployment

MSYNC is a protocol typically used between a multicast server that hosts the MSYNC sender and a multicast gateway that hosts the MSYNC receiver. This is depicted in Figure 1. Arrows represent the HAS session elements directional flows. The multicast server acquires HAS session elements in unicast conforming to a HAS protocol as e.g., MPEG DASH [MPEGDASH] or HLS [RFC8216] and sends those HAS session elements over a multicast network, supporting possibly RTP and UDP/IP multicast, to the multicast gateways. A multicast gateway listens the corresponding multicast flows and serves the HAS player(s) in unicast conforming to the same HAS protocol. MSYNC permits a sender to serve simultaneously multiple receivers conforming to one or several HAS protocols and formats (e.g., assuming one shared multicast network, one sender could serve some receivers with MPEG DASH compliant content and other receivers with HLS compliant content).

The multicast server is configured (by e.g., the ISP operating the multicast network) in order to acquire HAS content from a Content Distribution Network (CDN) via a unicast protocol, typically over the Internet. Considering one among several possible content ingest methods (e.g., HTTP GET), for each HAS session, the multicast server behaves as a HAS player, reading the manifest, discovering the available representations and downloading concurrently media segments of all (or a subset) of the available representations. The multicast server is configured for sending all those HAS session elements over possibly RTP and UDP/IP multicast according to a certain UDP/IP flow arrangement. For example, the objects related to each video representation are sent over a separate transport multicast session (multicast IP address + port number) whereas all audio representations are sent over the same transport multicast session.

The Multicast gateway is configured by the same ISP having configured the multicast server for being aware of the same UDP/IP flow arrangement. Depending on this arrangement and on the HAS player request, the Multicast gateway joins the multicast IP group associated with the HAS representation requested by the HAS player. Note that the multicast gateway might not be capable of receiving all the concurrent transport multicast sessions at the same time due to bandwidth limitations (e.g., ADSL).

At any time, the multicast gateway can detect corrupted and/or lost packets and attempt to repair using a repair protocol. This is possible with the HAS server interacting with the HAS content delivery network (CDN) or thanks to RTP when used as the transport layer over UDP (See Section 3.9).

The multicast gateway receives the MSYNC objects and is ready to serve them (e.g., acts as a local cache). Whenever a HAS request is sent by a media player and received by the multicast gateway, the latter reads first its local cache. In case of hit, it returns the object. In case of miss, the multicast gateway can retrieve the requested object from the associated CDN (or a dedicated server) over a unicast interface through operating HTTP conventionally and forwards back to the HAS player the object once retrieved. If no unicast interface exists, the multicast gateway can wait some time for the local cache to be updated with the element requested by the media player and/or returns an error.

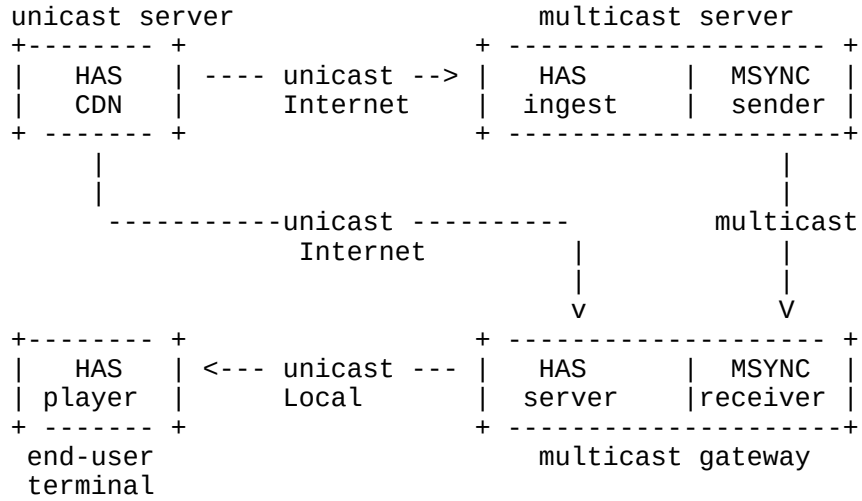

Figure 1: example of MSYNC deployment

Section 3.11 discloses a video streaming workflow example that involve MSYNC and application elements. It clarifies the roles and operations attached to the MSYNC sender and receiver and those attached to the application elements (i.e. HAS ingest and HAS server respectively).

With MSYNC deployed over a multicast network, the HAS player receives 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 MSYNC receiver or even the multicast gateway from be co-located with the media player and therefore embedded in the end-user terminal as shown in Figure 2.

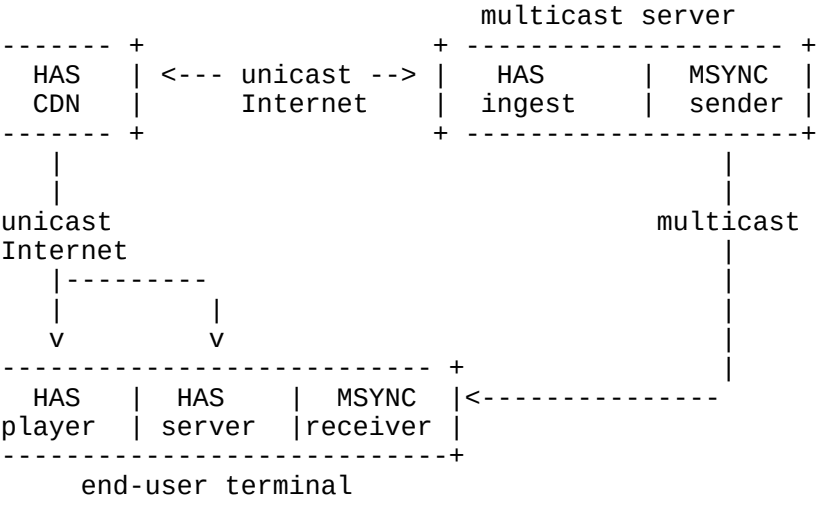

Figure 2: MSYNC receiver in the terminal

2.2. Unicast Networks

Figure 1 shows a typical MSYNC deployment where a HAS player interacts with a HAS server in an unicast way over e.g., Internet and interacts with a multicast gateway over e.g., a local network according to the same HAS protocol. Note that the multicast gateway may reside in the local area network (LAN) or upstream, in the ISP’s network premises.

In theory, all interfaces labeled "unicast" in Figure 1 could be deployed over an Internet network, although practically, the interface between the end-user terminal and the multicast gateway corresponds to a broadband access network or a Local area network

(LAN) controlled by the ISP.

2.3. Multicast Network and congestion avoidance

In this document "multicast network" means a network supporting IP multicast in addition to supporting IP unicast.

A multicast network is typically provided and controlled by a broadband Internet service Provider following the design principles depicted in [BFTR145] and [BFTR178]. A multicast network is composed with one or several multicast sub-networks interconnected with multicast routers and/or layer 2 bridge/switches performing IGMP snooping (Multicast Listener Discovery in IPv6) as discussed in [RFC4541] allowing to duplicate/forward multicast IP packets based on IGMP messaging. In a broadband multicast infrastructure the multicast network interconnects a service end-point (e.g., an IPTV service) with a broadband gateway located in the end-user premises. The last multicast sub-network is typically a point-to point circuit/line between the end-user broadband gateway and the first access network infrastructure aggregation point (e.g., a DSL access module or DSLAM). It has a rather limited [bandwidth] capacity comparing with the other multicast sub-networks being part of the ISP’s access, aggregation and core networks.

The MSYNC sender is connected to the first multicast sub-network whereas the MSYNC receiver is connected to the last multicast sub-network. A multicast network provides a certain capacity (i.e., bandwidth) attached to the first sub-network (connected to the MSYNC sender) that may be different from the capacity attached to the last sub-network connected to the MSYNC receiver. The data transported (i.e., HAS session elements) by MSYNC is not assumed elastic, i.e., it SHOULD be ingested at a fixed rate, sharing the concerns expressed in [RFC3550], Section 10.

The multicast network MUST support pre-provisioning bandwidth resources. This assumption permits to have the MSYNC sender able to transmit one HAS session or concurrently several HAS sessions operating one or more transport multicast session up to a certain maximum bandwidth, said MAX_BW_SEND. MAX_BW_SEND corresponds to the maximum guaranteed bandwidth dedicated to MSYNC allowing to transport the provisioned HAS session(s) across all multicast sub-networks up to the last multicast sub-network ingress point (e.g., the last router or bridge) before reaching the MSYNC receiver.

The MSYNC sender MUST control the sending rate of each HAS media sub-stream (and generally speaking of all MSYNC object to be transmitted) in such a way the maximum bandwidth MAX_BW_SEND corresponds to the following:

1. the sum of all individual media sub-stream bit-rate composing the set of provisioned HAS session(s) and

2. an additional bandwidth reserve for supporting control (initialization segments, manifest file, configuration information) transmission.

In addition, the MSYNC sender MUST be configured in such way that the minimum bandwidth consumed by a HAS session as advertised by a manifest (the least bandwidth consuming combination of media sub-streams as e.g., video, audio, subtitling) remains within the smallest provisioned bandwidth dedicated to MSYNC over the last multicast sub-network (connected to the N MSYNC receivers), said min (MAX_BW_RECEIVE_1, MAX_BW_RECEIVE_2, MAX_BW_RECEIVE_3,..., MAX_BW_RECEIVE_N). There is one MAX_BW_RECEIVE restriction per MSYNC receiver as there might be up to one different multicast sub-network connected to each MSYNC receiver. With this approach, any MSYNC receiver (whatever the last multicast sub-network capacity) fed by the MSYNC sender is ensured to receive at least one HAS sub-streams combination for each HAS session. The MSYNC sender MAY send a manifest and related media sub-streams whose combination could result in a throughput higher than the MAX_BW_RECEIVE of some MSYNC receivers.

The MSYNC receiver is instructed to join one or more IP multicast sessions up to its maximum bandwidth constraint (MAX_BW_RECEIVE) that represents the provisioned capacity dedicated to MSYNC over the last multicast sub-network it is connected to. As an example, the capacity of the last multicast sub-network can be limited to a few Mbps with ADSL and up to several hundred of Mbps with fiber to the home (FTTH). In the case of a broadcast network (e.g., satellite) the capacity exposed to the MSYNC sender may be equivalent to the capacity exposed to the MSYNC receiver if the broadcast network is composed with only one sub-network.

The MSYNC receiver MUST support IGMP version 2 [RFC2236] or above versions in order to "join" and "leave" an IP multicast session, When source filtering (Source-Specific Multicast or SSM) is required the MSYNC receiver MUST support IGMP version 3 [RFC9776].

Sending and receiving MSYNC packets over a transport multicast session is detailed in 3.7.

2.4. Handling third party content

As introduced above, MSYNC is an enabler for allowing HAS content to be distributed over a controlled multicast network. Ideally any content provider or content delivery network provider on the Internet

should be able to benefit from MSYNC. Content Distribution Network Interconnection (CDNi) is a framework [RFC7336] for a content provider or an upstream CDN provider to delegate streaming to a downstream CDN. Regarding HAS streaming, CDNi is used to improve the user experience, allowing the third party content provider to operate a downstream CDN owned, shared and exposed by an ISP through the Open caching interfaces specified by the CDNi framework. The delegation is basically done through request routing where an upstream request router on the Internet redirects a request to a cache server located in the ISP network. Advantages and benefits are disclosed in [RFC6770] and in particular in Section 2.3 that discusses the mutual benefits for the ISP and the content/CDN provider in the context of video streaming.

Let’s now assume that the ISP desires to share and open its multicast delivery service and infrastructure powered by MSYNC in a similar way. This may be completely transparent for the content provider. According to the CDNi framework, HAS session request can be delegated to (i.e., routed) down to the ISP’s HAS server hosted by the multicast gateway in figure 1.

In summary with the CDNi framework and MSYNC combined together, HAS streaming over Internet can leverage the ISP’s multicast network delivery (powered by MSYNC) in an open/standard way.

3. MSYNC Protocol

The MSYNC protocol allows an MSYNC sender to transmit MSYNC objects (e.g. a media segment or a manifest) to an MSYNC receiver. An MSYNC object is composed with several elements (Info, HTTP header, data) where each element is composed with one or multiple MSYNC packets. For instance, sending a media segment object requires the sender to send: one object info element (The object info element always fits to only one MSYNC info packet), one object HTTP header (optional) that is composed with one or more object HTTP header packets and finally the object data element that is composed with one or more object data packets. When the object to be sent is known to be part of a bigger super object of unknown size then the sender uses the data-part element instead of the data element. An object data-part element is composed with one or more object data-part packets.

All MSYNC packets carrying elements or parts of elements associated with the same MSYNC object carries the same MSYNC object identifier.

The MSYNC sender sends an object over one transport multicast session (possibly a RTP multicast session). All objects belonging to the same media sub-stream are typically sent over the same multicast transport session although it is not mandatory. The MSYNC sender is instructed

to send media sub-streams over one or more multicast transport sessions according to a mapping configuration file that is shared at the application level (i.e., between the Multicast server and the Multicast gateway in Figure 1). The minimum set of required configuration parameters related to MSYNC is summarized in Section 3.10. The format of this MSYNC configuration document as well as parameters that are not directly linked to MSYNC are not specified. This is the subject of standard specification such as [DVB-MABR].

The MSYNC receiver behavior is driven by the application as envisaged in Section 2.1 for example. The end user application (e.g. a HAS player) may request a media object (e.g., a manifest or a segment) that makes the MSYNC receiver be instructed by the HAS Server to listen the corresponding multicast transport sessions (joining the corresponding IP multicast session if not already done); the mapping information (request URL to multicast transport session) is the responsibility of the Application. The MSYNC receiver delivers to the application (e.g., the HAS Server in Figure 1) any received object from any listening multicast transport session along with the useful metadata such as the object URI, an information of the object info element (see Section 3.2) that may be used as a cache key as envisaged in Section 2.1.

The details of the MSYNC protocol are disclosed in the following sub-sections.

3.1. MSYNC Packet Format

The MSYNC packet has the following format. All bytes are sent according to the conventional network order: big-endian.

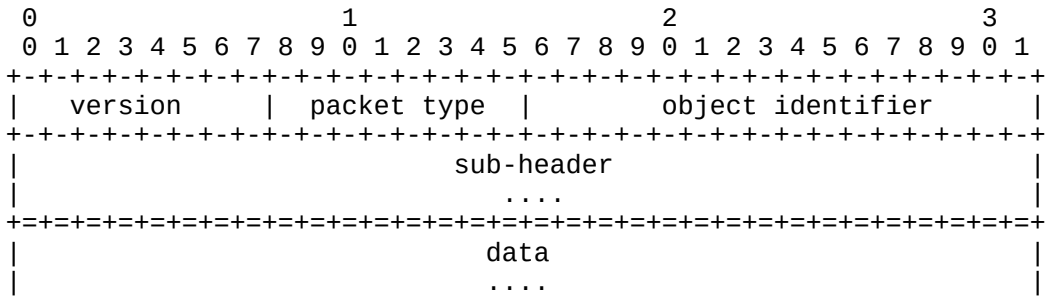

Figure 3: MSYNC Packet

version: 8 bits
Version of the MSYNC protocol = 0x03

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 HTTP/2 frame.

object identifier: 16 bits
: This field identifies the object being transferred in a multicast transport session. Considering one transport multicast session, all MSYNC packets associated with the same object carry the same object identifier in their MSYNC packet header. Whenever this object ID change that means the sending of the previous object is finished but not necessarily the reception (packets might have been possibly reordered). Depending on the deployment, un-ordered packet reception is either not possible or acceptable within a certain time limit. When transmitting a new object, the MSYNC sender MUST NOT reuse an object ID that corresponds to an ongoing MSYNC object transmission. The way to deal with packet reordering is discussed in Section 3.7.

sub-header: series of N x 32 bits
: The packet sub-header is linked to the packet type. The details of each packet type are specified in the next sections.

data: series of D x 8 bits
: The presence and contents of field is optional and is present depends 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 explained in Section 3.6.

3.2. Object Info Packet

The Object info packet is used to transport meta-data associated with an object. It is used to describe the object. 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 that the object info packet relates to.

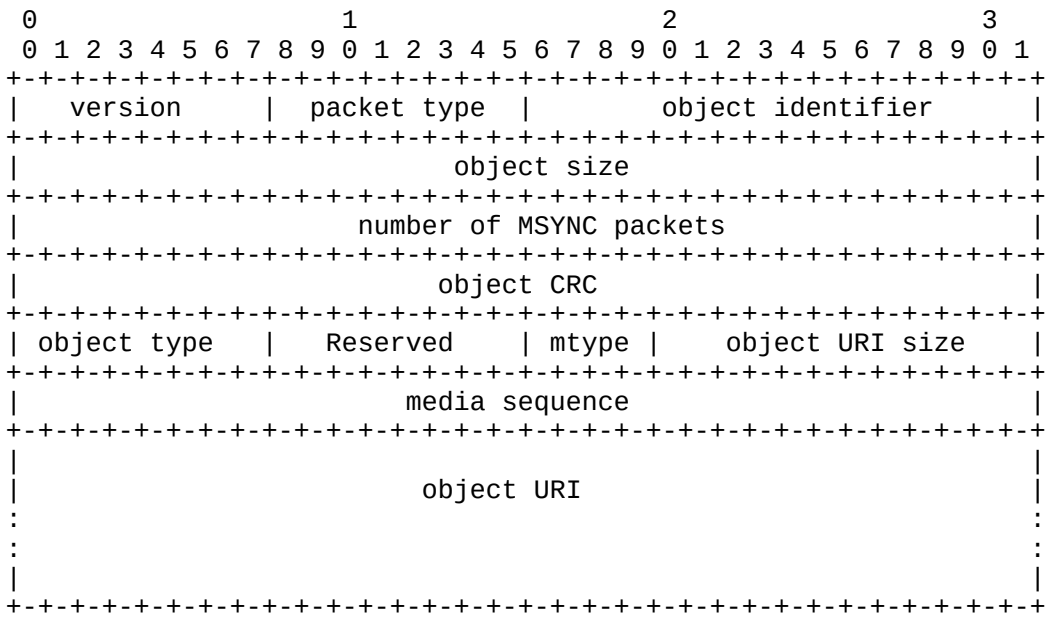

Figure 4: Object Info packet

packet type: 0x01 or 0x02
: Redundant object INFO packets (packet type 02) MAY be sent in addition to the "main" object info packet according to Section 3.7.

object size: 32 bits
: The number of bytes that compose the object data payload transported with one or more MSYNC object data packets (Section 3.3) or MSYNC object data-part packets (Section 3.5).

The size may be 0 indicating that there is no corresponding object’s data payload transmission foreseen (i.e., no expected MSYNC data packet or MSYNC data-part packet). In case of a super object transmission (Section 3.5), if the object URI of an object info with an object size set to 0 matches the super object URI then it MUST be interpreted as the end of the super object transmission (Section 3.8.1.2).

Note that 32 bits is sufficient when transporting HAS elements.The maximum size of an object (4.4 GBytes) authorizes the transfer of a video segment of several tens of seconds, 4K encoded.

number of MSYNC packets: 32 bits
: Number of MSYNC packets that compose the transported object. If the object size is null (set to 0) then the number of MSYNC packets MUST be null (set to 0).

object CRC: 32 bits
A Cyclic Redundancy Check applied to the object data payload or the object data-part payload for corruption detection according to the CRC-32 algorithm defined in the ISO/IEC 3309:1999 specification revised by the ISO/IEC 13239:2002 specification. The object data payload is composed with the "data" field of all data packets associated with the object (see section 3.3). The object data-part payload is composed with the "data" field of all data-part packets associated with the object (see Section 3.5).

object type: 8 bits
Defines the type of object, i.e., the content type transported with Object data (or data-part) packets, associated with this MSYNC Object info packet.
0x00: Unknown.
0x01: Media manifest (playlist).
0x02: Reserved.
0x03: Media content. Data plane elements formatted according to e.g. MPEG-TS [MPEG2TS], MP4 or yet fragmented MP4 [MPEGCMAF].
0x04: Reserved.
0x05: control: control plane information (e.g., MSYNC configuration elements as discussed in Section 3.10).
0x06-0xFF: Reserved.

mtype: 4 bits
Characterizes the media manifest. This field MUST only be used in association with the object type 0x01 (media manifest). It MUST be set to 0x0 (not applicable) otherwise. The field can take the following values.
0x0: Not Applicable
0x1: MPEG Dash as specified in [MPEGDASH].
0x2: Master HLS playlist as specified in [RFC8216].
0x3: Media HLS playlist as specified in [RFC8216].
0x4-0xF: Reserved

object URI size: 12 bits
The size in bytes (as an unsigned integer)of the object URI field (including 0x00 padding bytes). The object URI maximum size depends on the network MTU as discussed in Section 3.7.

media sequence: 32 bits
A sequence number (as an unsigned integer) associated with the MSYNC objects data and data-part (for transporting a segment or a manifest) that depends on the mtype value. It is used by the application operating MSYNC (e.g. the multicast gateway) to facilitate/accelerate the synchronization between unicast and MSYNC reception. The multicast gateway may operate jointly MSYNC/multicast and conventional HTTP/unicast for retrieving HAS

elements as indicated in Section 2 and illustrated in Figure 1. The values and rules are detailed in the Section 3.8 dedicated to the HAS protocol dependencies. If this field is unused, it MUST be set to 0x00, and MSYNC receivers MUST ignore it.

object URI: object URI size * 8bits
: This is the path name associated with the object as the URI reference to be resolved according to [RFC3986]. It MAY corresponds to a storage/Cache path. There SHOULD be a direct relationship between this URI and the URL associated with the addressable object (e.g., HAS segment or CMAF chunk and/or a manifest). The rules for HAS delivery are detailed in Section 3.8 dedicated to the HAS protocol dependencies.

The object URI is coded as a series of string characters conforming to UTF-8 [STD63]. Remaining unused bytes of the last 32 bits word MUST be filled with the 0x00 value.

3.3. Object Data Packet

The Object Data Packet carries part or all of the object's data payload. The type of data and the way to process the object's data packets are determined by the associated object info packet.

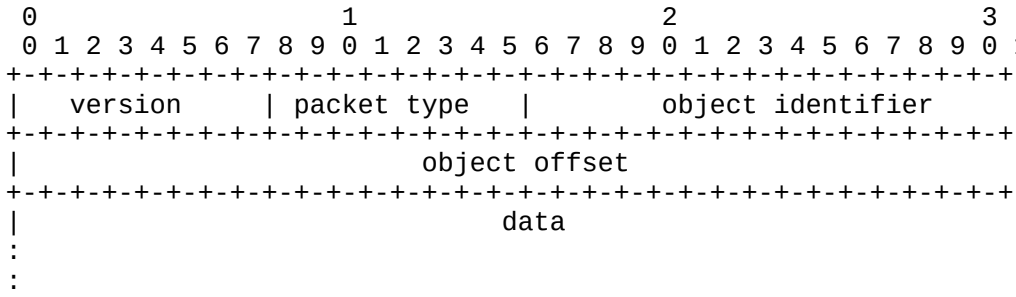

Figure 5: Object Data packet

packet type: 0x03

object offset: 32 bits
: The index from which the MSYNC object data packet payload (data field) is to be written in order to compose the object data payload at the receiver side (i.e., the multicast gateway). The first data packet of an object has an offset equal to 0. The MSYNC sender MUST NOT send a data packet that exhibit an object offset that makes the data range overlapping with a previously transmitted data packet belonging to the same object. Likewise,the MSYNC receiver MUST ignore a data packet that exhibit an object

offset that makes the data range overlapping with a previously received data packet belonging to the same object.

data: N x 8 bits
The data payload (or part of data payload) related to the carried object (e.g., part or all of a HAS segment or a manifest). The maximum size of the object data packet depends on the network MTU as discussed in Section 3.7. This data payload represents all or part of the object data payload. The total size (number of bytes) of the object data payload is indicated in the associated object info packet (field object size). The object CRC in the associated object info packet applies to the object data payload (i.e. composed with one or more data payloads).

3.4. Object HTTP Header Packet

Using the object HTTP header is optional (see 3.7). The MSYNC sender and the MSYNC receiver do not exploit directly the HTTP header. HTTP header fields can be used by the application operating MSYNC. For example, considering the Figure 1, the HAS Ingest component in the Multicast server may ingest some HTTP headers useful for the HAS server in the Multicast gateway and/or the end user application. As an example a security mechanism based on HTTP may exploit this possibility (see Section 5).

The HTTP header packet carries part or all of HTTP header fields related to the object to be sent. There is at most one object HTTP header per object data (or per object data-part) that means the MSYNC sender MUST NOT send more than one object HTTP header element with a different content (i.e. different set of HTTP header fields) associated with the same object identifier. The exact same object HTTP header element can be repeated (sent several times) according to the sending rules detailed in Section 3.7.

The transport of the HTTP header fields MUST be conformed to HTTP/1.1 Section 5 of [RFC9112]. Carrying HTTP header fields of a version of HTTP greater than HTTP/1.1, the MSYNC sender MUST convert the format according to HTTP/1.1 Section 5 of [RFC9112].

The object HTTP header MAY also be used in association with Object Data-part. The fields (name/value pairs) MUST be transported according to HTTP/1.1 Section 5 of [RFC9112].

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

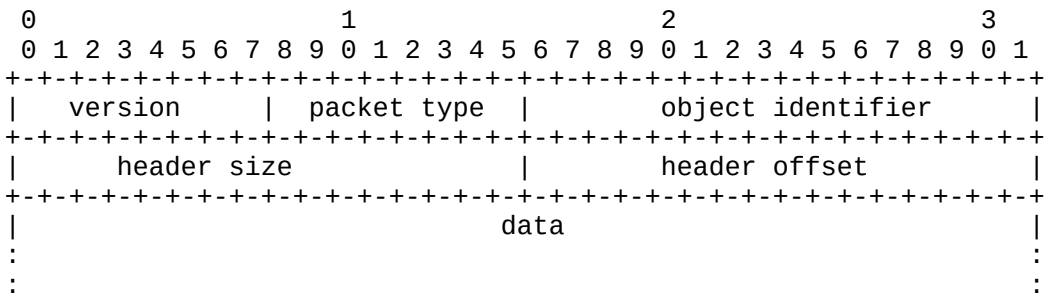

Figure 6: Object HTTP Header packet

packet type: 0x05

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 8 bits
The payload data related to the HTTP header. The maximum size of the Object HTTP header packet depends on the network MTU as discussed in Section 3.7.

3.5. Object Data-part Packet

This MSYNC packet carries part or all of the media data-part object payload. The type of data and the way to process the object’s data-part packets are determined by the associated object info packet. Object data-part payload is transported through a series of object data-part packets. The data-part is used when the object corresponds to a "part" (a block) 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 have the same object identifier that is the same one present in the object info packet and HTTP header (if any) packets the data-part object relates to.

All data-part objects composing a super object have a different object identifier. The object info packet (object URI) links a data-part object with a super object as explained in Section 3.8.1.2.

The end of super-object transmission is signaled with an object info packet having both the object size and the number of MSYNC packets set to 0 and having the object URI matching the object URI of the already received parts according to Section 3.8.1.2.

Detecting missing data-part packets of a data-part object is based on detecting the end of an object transmission as depicted in Section 3.7.2 possibly assisted with the CRC. The receiver detects missing packets through the Object Info packet that indicate the number of MSYNC packets composing the object. The receiver detects corrupted packets through processing the CRC. Regarding the super object transmission, the MSYNC receiver does not know how many data-part objects composes a super object but knows when the super object transmission ends and knows the super object offset as well as the object URI of each of the already received data-part objects (object info contains the object size set to zero and the object URI referencing the last block +1) that permits the MSYNC receiver to avoid missing data-part objects belonging to the same super object.

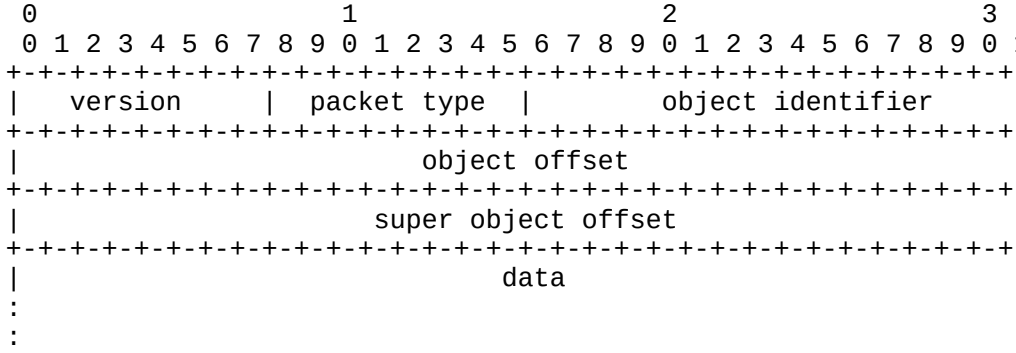

Figure 7: Object Data-part packet

packet type: 0x06

object offset: 32 bits
The index from which the data-part packet payload (data field) 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. The MSYNC sender MUST NOT send a data-part packet that exhibit an object offset that makes the data range overlapping with a previously transmitted data-part packet belonging to the same object. Likewise,the MSYNC receiver

MUST ignore a data-part packet that exhibit an object offset that makes the data range overlapping with a previously received data-part packet belonging to the same object.

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. The MSYNC sender MUST NOT send a data packet that exhibit a super object offset that makes the data range overlapping with a previously transmitted data-part object belonging to the same super object. Likewise,the MSYNC receiver MUST ignore a data-part packet that exhibit a super object offset that makes the data range overlapping with a previously received data-part object belonging to the same super object. All data-part objects belonging to the same super object share the same object URI prefix (see Section 3.2)

data: N x 8 bits
The data payload related to the carried object (e.g., part or all of a HAS segment or a manifest). The maximum size of the object data-part packet depends on the network MTU as discussed in Section 3.7. The total size (number of bytes) of the object data-part payload is indicated in the associated object info packet (field object size). The object CRC in the associated object info packet applies to the object data-part payload (i.e. composed with one or more "data" payloads).

3.6. Maximum Size of an MSYNC Packet

An MSYNC packet MUST fit within the underlying protocol packet. As detailed in Section 3, an MSYNC packet is composed with a header part and a data part for which the size is limited by the transport multicast protocol. With RTP and/or UDP (which authorize up to 65535 bytes), the maximum size is linked to the path MTU (Maximum Transfer Unit) as the largest transfer unit supported between the source (the multicast sender) and the destination (the multicast receiver) without fragmentation. The mean to compute the MTU is out of scope of this document.

3.7. Sending and Receiving MSYNC Objects

The following considerations are linked to the MSYNC configuration (3.10).

3.7.1. Mapping over Transport Multicast Sessions

The mapping of MSYNC objects onto transport and IP multicast sessions is not constrained by the MSYNC protocol but by the multicast network capacity (i.e., the bandwidth) provisioned for MSYNC as indicated in Section 2.3. For example, with ADSL (Asymmetric Digital Subscriber Line), the capacity dedicated to multicast is limited which may drive to an IP multicast flow arrangement where one IP multicast session carries the elements related to only one video sub-stream and another one that carries the elements related to all audio sub-streams (each of the audio sub-stream being associated with a different transport multicast session). In that case, the MSYNC receiver MUST join at most three IP multicast sessions (one for the video representation packets, another one for the audio representations packets and the last one for the control information).

Another arrangement could dedicate one IP multicast session per HAS stream gathering all media sub-streams (one transport multicast session per sub-stream).

Considering a satellite network, as all transport multicast sessions are carried simultaneously, all IP multicast flow arrangements may make sense. The MSYNC receiver may be configured to join all IP multicast sessions in advance (see Section 3.10).

In general, the MSYNC receiver is instructed to join the IP multicast session associated with the media sub-stream(s) the application (the HAS server in figure 1) wants to listen/receive that is itself linked with the incoming requests from the end user media player.

A transport multicast session is identified with the triplet: source IP address (MSYNC supports Source Specific Multicast), destination multicast IP address and destination transport port number. It is RECOMMENDED to carry media sub-streams and the control information (Section 3.2) in separate transport multicast sessions; it allows the deployment of different error correction (see Section 3.9) or content protection procedure (e.g., one ISP may decide to encrypt the transport multicast session dedicated to the transmission of control information).

The following arrangement is typical in ADSL:

- 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 the control information.

It is perfectly possible to send the MSYNC packets attached to different objects in one transport multicast session only (and

therefore one IP multicast session) as long as the following statement is respected.

For each MSYNC object (see object type in 3.2) to be sent over a transport multicast session, the MSYNC sender MUST send the following MSYNC packets in the specified order:

- One object info packet

- Zero or more object info redundant packets

- Zero or one object HTTP header, the object HTTP header being compose with one or more packets (sent in a sequential order),

- Zero, one or more redundant object HTTP header elements (all identical to the previously sent object HTTP header if any), each repeated object HTTP header being composed with one or more packets (sent in a sequential order),

- Zero, one or more object data packets (or object data-part packets) in a sequential order.

The MSYNC receiver MUST continuously control that it does respect its MAX_BW_RECEIVE constraint (see Section 2.3) and therefore the MSYNC receiver MUST NOT attempt to join a new IP multicast group if that condition cannot be respected.

When the MSYNC object is of size null (used to signal the end of the transmission of a super object) then only one object info packet is sent (see Section 3.2).

3.7.2. Detecting the End of an Object Reception

Detecting the end of an MSYNC object (or super object) transmission is done thanks to the Object Info (see 3.2) information. However, packet loss is possible and MSYNC packets related to an MSYNC object may be received out of order. Packet re-ordering may be acceptable or not depending on the deployment scenario (it is generally bounded by the potential latency introduced by un-ordered MSYNC packets reception). As a consequence, the detection of the end of the MSYNC object reception MUST NOT be based solely on the detection of the end of the object transmission.

An MSYNC receiver implementation MAY rely on a timer associated with the maximum transmission time of a particular MSYNC object type in order to detect the end of the MSYNC object transmission. The MSYNC receiver MAY arm a timer when the reception starts (e.g., first received packet related to a new object) and MAY stop the timer

whenever the object is completely received. When the timer reaches the time limit, the MSYNC receiver SHOULD consider the transmission of that object done while the object being partially received.

Note that the MSYNC sender MAY use the same maximum transmission time of a particular MSYNC object type for controlling the object identifier (re-)allocation (see Section 3.1).

Assuming receiving unordered packets is not possible, an MSYNC implementation MAY rely on the detection of a new object transmission and decide that the previous object transmission (and reception) is done while the object being possibly partially received.

After the transmission of an object is considered done, The MSYNC receiver MUST consider subsequent packets related to the same object identifier as being part of new object transmission only if the previously received object associated with the object identifier has been completely received or partially received but after a maximum transmission time. If this condition is not verified the MSYNC receiver MUST discard the packets.

When a new object transmission is detected (an object data or data-part with a new object identifier) and there is no associated object info packet received within a certain time limit (linked to the support of packet re-ordering) the object MUST be ignored and related packets MUST be discarded.

In the case of a partially received MSYNC object, this is up to the application (e.g., the HAS server in Figure 2) to react, triggering, for instance, an object repair procedure.

Note that packet repair and packet reordering can be performed at the underlying RTP, based on the RTP sequence number (see Section 3.9).

3.7.3. Congestion Control

MSYNC is applicable and deployable in a controlled environment according to Section 3.1.9 of [RFC8085]. MSYNC MUST be used in a single operator network that operates network capacity provisioning.

As indicated in Section 2.3, the MSYNC sender MUST control its sending rate according to a pre-provisioned capacity (i.e., bandwidth) dedicated to MSYNC. The deployment SHOULD prevent any potential "leaks out into unprovisioned Internet paths" in conformance with Section 3.1.9 of [RFC8085]. This can be achieved through logical and physical traffic isolation and filtering as commonly implemented in broadband networks following the design principles depicted in [BFTR145] and [BFTR178]. This may also be

complemented with the support of a circuit breaker as disclosed in [RFC8084].

The MSYNC receiver or more probably the application exploiting the MSYNC receiver (e.g. the multicast gateway in Figure 1) may detect and mitigate potential congestions according to the receiver-driven congestion control method as detailed in Section 4.1 of [RFC8085]. When congestion occurs, the received objects are subject to a growing number of missing bytes and therefore a growing number of repair procedures (the MSYNC receiver repairs the packets possibly based on RTP - see 3.9). On congestion detection, the MSYNC receiver, under the control of the application SHOULD leave one or more IP multicast groups and may even terminate the multicast reception. Regarding HAS streaming, one mitigation action would be to switch to a less bandwidth consuming IP multicast session, forcing the end-user terminal/player somehow to request HAS sub-stream elements related to that less bandwidth consuming IP multicast session.

3.7.4. Object repair

MSYNC MAY be used in association with a unicast Internet link in order to repair an MSYNC object (corrupted/lost packets) as overviewed in Section 2.1 and Section 3.11. This is the responsibility of the Application (the HAS Server in Figure 1) to perform this reparation. However MSYNC receiver SHALL be able to deliver the necessary information for allowing the application to request the object from a CDN. Such information is the corrupted object’s URI (see Section 3.2) along with the missing byte range list. The Application MAY use this information along with some configuration parameters to build the object repair request (operating e.g. HTTP). This is out of the scope of MSYNC and addressed by standard specification such as [DVB-MABR].

Note that MSYNC objects MAY also be repaired (recover) through the Transport multicast session protocol RTP according to Section 3.9.2.

3.8. 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 on the HAS protocol.

3.8.1. Object Info Packet

3.8.1.1. Media Sequence

The media sequence (an object Info Packet header field presented in the Section 3.2) is used by the application operating MSYNC (e.g. the

multicast gateway) to facilitate the synchronization of the MSYNC (i.e., multicast) reception with the unicast reception. The multicast gateway may operate jointly MSYNC/multicast and conventional HTTP/unicast for retrieving HAS elements as indicated in section 3.2. For example, considering a HAS session, the application (e.g. the multicast gateway) starts to fetch a manifest from unicast as the MSYNC receiver is not ready. When the MSYNC receiver becomes ready and is receiving segments related to the HAS session, the application MUST quickly determine the level of freshness of the last segment received via MSYNC, meaning whether this segment is more recent than the last one fetched from unicast. Without the media sequence information field present in the Object info packet, the multicast gateway would need to parse the manifest received from MSYNC that takes time and consumes CPU. Similarly, the application may also need to understand the level of freshness of a manifest file received via MSYNC versus the last one received over 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 of 0x00 MAY be used.

If unicast reception is used jointly with MSYNC then the media sequence MUST be set depending on the object type and mtype values (Info Packet header fields presented in Section 3.2.) as listed below.

HLS master playlist: 0x00

HLS variant playlist; MUST contain the Media Sequence Number according to Section 3 of [RFC8216] of either the last segment transmitted (i.e. the last declared segment in the playlist) or the last segment under transmission, i.e. the last declared segment in the playlist plus one (+1) also called parent segment in the the case of partial segment(s) declared after the last declared segment in the playlist (see Section 3.2 of [ID-HTTPLIVESTREAM]).

HLS segment: MUST contain the Media Sequence Number according to Section 3 of [RFC8216] corresponding to either the declared segment in the associated playlist or the last declared segment in the playlist plus one (+1) also called parent segment in the the case of partial segment(s) declared after the last declared segment in the playlist (see Section 3.2 of [ID-HTTPLIVESTREAM]).

DASH manifest: MUST contain $time$/(integer division)@timescale or $Number$ corresponding to the last segment transmitted or under transmission (and possibly received partially) and declared in the manifest. see [MPEGDASH] for the definition of $time$, @timescale and $Number$.

DASH segment: MUST contain the $time$/(integer division)@timescale or $Number$ value corresponding to the segment declared in the manifest.

3.8.1.2. Object URI

In the context of HTTP adaptive streaming, the object URI is a URI reference.

If the object is a HAS addressable entity (e.g., a segment or a CMAF chunk), the object URI MUST match (be a substring) with the URL announced in the corresponding manifest/playlist.

Examples:

- The object URI: /tvChannel1/Q1/S_2 matches with the segment’s URL that is computed from the associated manifest/playlist: ".../tvChannel1/Q1/S_2.mp4"

- The object URI /tvChannel11/Q1/S_2_3 matches with the CMAF chunk URL that is computed from the associated manifest/playlist: ".../tvChannel11/Q1/S_2_3.mp4".

If the object is a non-addressable HAS entity (e.g., a HTTP/1.1 CTE block), the object URI is composed with a sub-string (that MUST match with the URL announced in the corresponding manifest) and a suffix composed with the hash sign/character (#) and the block number).

Example:

- The object URI of the 3rd HTTP/1.1 CTE block of the segment S_2: tvChannel11/Q1/S_2.mp4#2 matches with the segment’s request URL that terminates with ".../tvChannel1/Q1/S_2.mp4"

The block number of an object URI attached to a media data-part object MUST be incremented for each subsequent transmission.

When all the MSYNC data-part packets for all the media data-part objects (e.g., HTTP/1.1 CTE blocks) composing a super object (e.g., a media segment) have been sent, the MSYNC sender MUST signal the end of the MSYNC super object transmission through sending an MSYNC object info packet with the object size set to zero (0). In addition, the object URI MUST contain the URI reference of the next block (never transmitted). see Section 3.2.

Example:

- The object URI of the object info packet associated with the 1st HTTP/1.1 CTE block of the segment S_2:

tvChannel11/Q1/S_2.mp4#0

- The object URI of the object info packet associated with the 2nd HTTP/1.1 CTE block of the segment S_2: tvChannel11/Q1/S_2.mp4#1

- The object URI of the object info packet associated with the 3rd HTTP/1.1 CTE block of the segment S_2: tvChannel11/Q1/S_2.mp4#2

- The object URI of the object info packet associated with the 4st HTTP/1.1 CTE block of the segment S_2: tvChannel11/Q1/S_2.mp4#3

- The object URI of the object info packet associated with the 5st HTTP/1.1 CTE block (of size null) signaling the end of the super object (i.e., segment) transmission: tvChannel11/Q1/S_2.m4s#4

3.9. RTP Multicast Session

RTP [RFC3550] being popular in the IPTV ecosystem is a motivation to consider RTP as an MSYNC transport multicast session protocol. The RTP support would allow an MSYNC deployment to benefit from the RTP infrastructure already in place such as RTP based monitoring tools.

3.9.1. RTP as the MSYNC packet container format

With MSYNC, RTP is used as a container format wrapping MSYNC packets. As such, There is no need to use RTP for media multiplexing or source/receiver synchronization. The following lines list the restrictions in using RTP based on [RFC3550] and Section 2 of [RFC3551].

- One RTP multicast session corresponds to one Transport multicast session as indicated in Section 1.2.- RTCP usage is not required if packet retransmission (see Section 3.9.2.) is not used. - There is no support, no need to announce/describe the RTP multicast session using SDP or any other equivalent announcement protocol.- MSYNC does not support RTP translators and mixers. Accordingly, the MSYNC sender (the RTP sender) MUST NOT list contributing source (CSRC) identifiers in the RTP header (see below).

The RTP packet header used to wrap a MSYNC packet is represented in Figure 8.

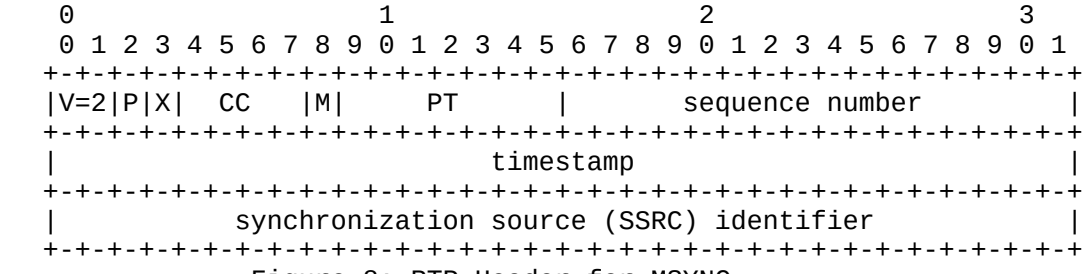

Figure 8: RTP Header for MSYNC

P: Padding.
Not used. MUST be set to 0.
X: Extension.
Not used. MUST be set to 0.
CC: CSRC Count..
MSYNC does not support contributing sources. The RTP header contains 0 (zero) contributing source identifier (CSRC) fields. MUST be set to 0.
M: Marker.
Not used. MUST be set to 0.
PT: Marker.
The RTP header payload type (PT) field SHOULD correspond to one of the dynamic value specified in [RFC3551]. Its value should be communicated to the MSYNC receiver as part of the MSYNC receiver configuration (Section 3.10). This payload type MUST be the same whatever the multicast transport session.
sequence number: see [RFC3550].
timestamp:
The RTP header timestamp field is computed as indicated in [RFC3550]; it corresponds to the instant the MSYNC sender starts the MSYNC packet transmission.
SSRC:
Each RTP multicast session MUST operate a unique different SSRC number [RFC3550]. This allows packet retransmission (if used) on the RTP multicast session basis.

3.9.2. RTP packet retransmission

Packet retransmission (see Figure 9 below) MAY be used in association with the RTP multicast session for packet loss recovery. If this is the case then the RTP Repair client and RTP repair server MUST be compliant with [RFC4585], [RFC4588] and [RFC5506] according to the followings.

- Reduced sized RTCP (Section 4 of [RFC5506]) MUST be used with sending feedback NACK messages operating the immediate Feedback mode [RFC4585]. The RTP Repair client does not send Receiver

Report (RR). There is no regular RTCP transmission but only feedback messages when appropriate (RTP packet loss detected).

- The RTP Repair client (coupled to the MSYNC receiver) submits transport layer feedback (FB) messages in NACK mode (Generic NACK) to the RTP Repair Server according to [RFC4585] and [RFC5506]. The format of the feedback message is based on Section 6.1 and Section 6.2 of [RFC4585] with the following setting: padding bit cleared (0), payload type set to "RTPFB" (205), FMT set to "Generic NACK" (1). The SSRC of packet sender MUST be unique. It is attached to the MSYNC receiver. It MAY be based on the IPv4 or IPv6 address of the MSYNC receiver unicast interface. The SSRC of media source MUST correspond to the SSRC of the RTP multicast session to be repaired (see Section 3.9.1). The Feedback Control Information (FCI) format is detailed in Section 6.2.1 of [RFC4585].

- The RTP Repair server receives, processes and responds to the feedback NACK messages (FB) according to [RFC4588]. It matches the SSRC media source present in the feedback (FB) message with the SSRC or the original RTP multicast session. The RTP Repair server MAY be located within the multicast server or it MAY be hosted by any intermediate entity acting as a multicast RTP receiver (i.e., capable of receiving the multicast RTP packets gathering MSYNC packets). In any case, the RTP Repair server and the RTP Repair client MUST operate a unicast interface. The format of the retransmission packets is based on Section 4 of [RFC4588] with the following setting: when sending a retransmission packet (an RTP packet indeed), the RTP Repair Server MUST use the same payload type (PT) as the one used in the original RTP header transporting the MSYNC packets (see Section 3.9.1). The SSRC value MUST be identical to the SSRC of media source value, present in the corresponding feedback message.

- The Session-multiplexing scheme [RFC5761] MUST be applied. The RTP retransmission (repair) stream MUST be sent over a unicast interface to the Repair client and therefore MUST NOT be part of the RTP multicast session used to transmit the original (multicast) RTP stream.

- The RTP Repair client and the RTP repair server SHOULD use the same UDP port numbers arrangement in order to facilitate the firewall traversal. The source port number of a RTCP packet (feedback NACK message) SHOULD correspond to the destination port number of the RTP retransmission packet.

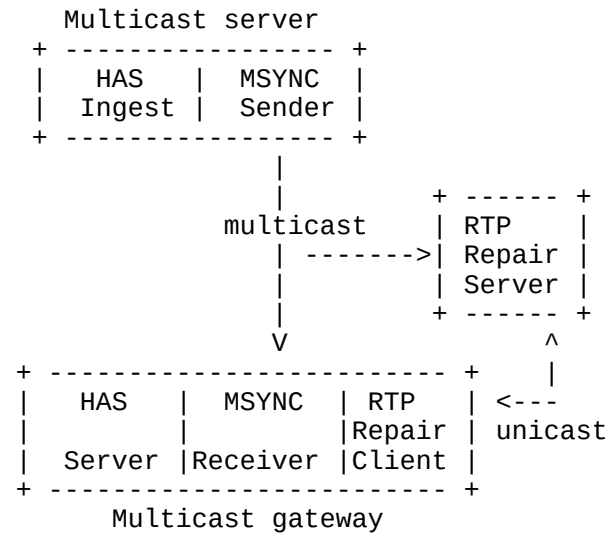

Figure 9: RTP repair

The Figure 10 represents a typical repair workflow. The interface between the MSYNC Receiver and the RTP Repair Client is implementation dependent.

Whenever the MSYNC receiver detects one or more packet losses attached to a RTP multicast session, it generates a Repair request (step 1 in Figure 10). One possible implementation of this interface consists into communicating the parameters SSRC (of the media source, i.e. the RTP multicast session), PID and BLP as defined in Section 6.2.1 of [RFC4585]. PID is the RTP sequence number of the first lost RTP packet of a potential series of lost RTP packets to be retrieved. BLP is a 16 bits field reporting loss of any of the 16 RTP packets following the first RTP packet represented by the PID parameter. Note that implementation dependent interface could allow a request with several PID/BLP parameters pairs as this is possible with [RFC4585].

The RTP Repair Client submits a RTCP packet to the RTP Repair Server (step 2 of Figure 10) that contains the Feedback NACK message with the Feedback Control Information (FCI) gathering the PID and BLP parameters as specified in [RFC4585].

The RTP Repair Server processes the request and retrieves in its cache the set of requested RTP packets and send them back to the RTP Repair Client conforming to [RFC4588] and the restrictions specified above (step 3 of Figure 10). The size of the RTP Repair server cache is application dependent.

Finally, the RTP Repair Client communicates the retransmitted packets to the MSYNC receiver.

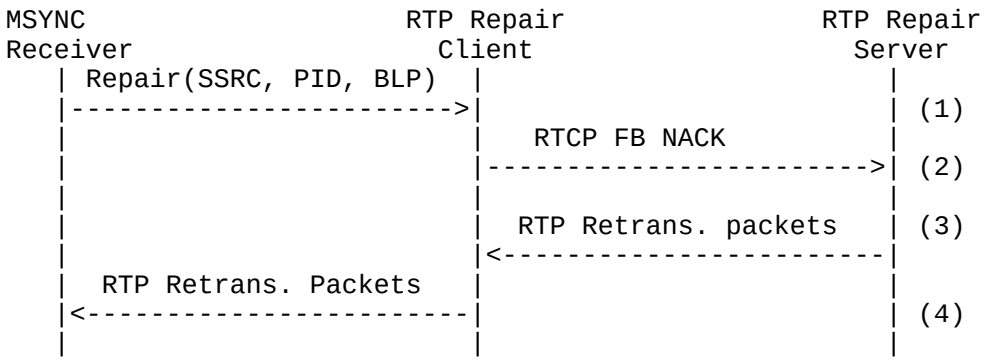

Figure 10: RTP Repair workflow

Note that instead of relying on MSYNC "RTP retransmission", the Application (i.e. the HAS server in Figure 1) could take care on recovering an MSYNC damaged received object (e.g. an HAS element like a segment or a manifest) through e.g. HTTP (see Section 3.7.4). This method requires a CDN, relies on HTTP Byte-range request for which the support is not harmonized and is less reactive than operating RTCP (UDP transactions over a dedicated path are typically much quicker than HTTP/TCP transactions over the unicast broadband data path).

3.10. Configuration

3.10.1 MSYNC Configuration

This section lists the essential configuration parameters related to the MSYNC sender and the MSYNC receiver. The goal is neither to specify the parameters list attached to the whole application exploiting MSYNC such as the Multicast Server or the Multicast Gateway (see Figure 1) nor the format of such configuration list that are both the subject of external specifications as [DVB-MABR].

Congestion avoidance: parameters for avoiding congestion as discussed in Section 2.3.

- Maximum Sender bandwidth (MAX_BW_SEND): corresponds to the maximum guaranteed bandwidth dedicated to MSYNC allowing to transport the provisioned HAS session(s) across all multicast sub-networks up to the last multicast sub-network ingress point (e.g., the last router or bridge) before reaching the MSYNC receiver.

- Maximum Receiver bandwidth (MAX_BW_RECEIVE): Represents the provisioned capacity dedicated to MSYNC over the last multicast sub-network the MSYNC receiver is connected to.

RTP Encapsulation: This is optional but required for relying on RTP retransmission as discussed in Section 3.9.

- RTP Repair Server: If RTP retransmission is available, IP address (or fully qualified domain name) and UDP port number of the RTP Repair Server.

- RTP Payload type: One parameter that cannot be fixed whatever the deployment is the RTP Payload type. This parameter concerns the MSYNC Sender when operating RTP.

3.11. MSYNC workflow example

In this section we depict a sequence diagram that illustrate a full MSYNC sequence as part of a media streaming application example that involves a multicast server and a multicast gateway (Figure 1). The workflow allows to better understand and separate the operations attached to MSYNC from those attached to the application elements. An application programmable interface (API) exposed by the MSYNC sender and MSYNC receiver is assumed for the sake of clarity but is purely informative.

In the following Figure 11, the '*' character indicates a repetition of the transaction. The number in parenthesis refers to an explanation of the transaction given below the figure.

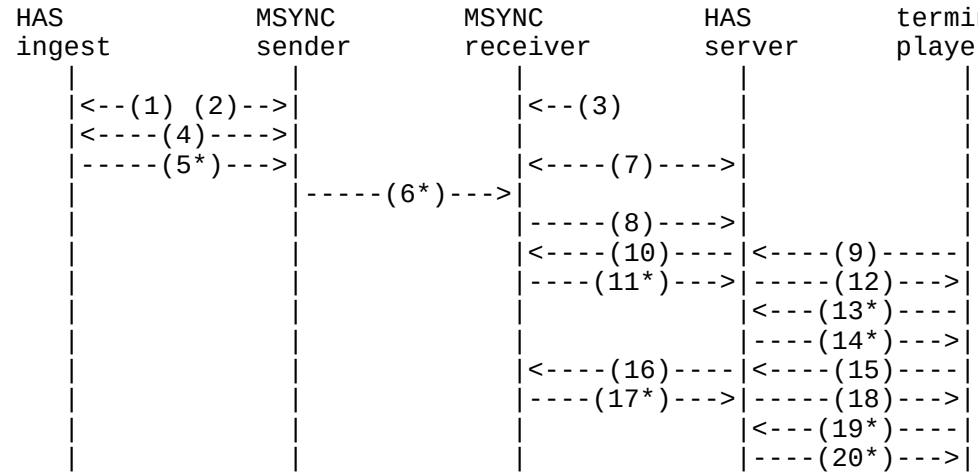

Figure 11: MSYNC interactions example

(1): The multicast server is configured for delivering a new HAS live service over multicast that means configuring the HAS ingest and the MSYNC sender. The HAS ingest gets configured to fetch all the elements announced in the manifest file associated with the new service. We assume a DASH [MPEGDASH] manifest file that describes a service with 2 adaptation sets (audio, video) and 2 representations for each of those adaptation sets. The HAS ingest is configured for delivering the new elements related to that live service. A transport multicast session arrangement (said HAS configuration) is provided targeting ADSL access, according to the following (the bandwidth and relative URL parameters are extracted from the manifest file).

- Service Live1: manifest url: https://provider.cdn.com/live/channel2/dash/manifest.mpd.

- Video1: transport multicast session = 233.252.0.1:17000; relative URL= "video1/$number$.mp4"; bandwidth="1000000".

- Video2: transport multicast session = 233.252.0.2:17000; relative URL= "video2/$number$.mp4"; bandwidth="2500000"

- Audio1: transport multicast session = 233.252.0.3:17000; relative URL= "audio1/$number$.mp4"; bandwidth="192000"

- Audio2: transport multicast session = 233.252.0.3:17001; relative URL= "audio2/$number$.mp4"; bandwidth="384000"

In addition , The HAS ingest is configured to operate a dedicated Control channel according to the following.

- Control: transport multicast session = 233.252.0.4:17002. Note that the Control session is used by the application to communicate to the HAS Server the transport multicast session arrangement.

(2): The MSYNC sender is configured with the following information. RTP encapsulation with RTP retransmission (see 3.10). The Maximum MSYNC Sender bandwidth is configured: MAX_BW_SEND = 4500000 bps.

(3): The MSYNC receivers are configured for supporting RTP reception as well as RTP retransmission conforming to Section 3.10. The maximum MSYNC receiver bandwidth is configured: MAX_BW_RECEIVE = 3500000 bps that is the same configuration for all MSYNC receivers assuming they are all connected to the same access network that guaranties this minimum available bandwidth.

(4): The HAS ingest and the MSYNC sender establish a connection

wherein the MSYNC sender communicates its limitation (MAX_BW_SEND) and in return gets the new provisioned service "Live1" and the provisioned Control channel that is accepted.

(5): The HAS ingest sends continuously (carousel) the transport multicast session arrangement for all configured services (here only one service: Live1) to the MSYNC sender along with the transmission parameters (Control transport multicast session, object type = "Control", Object URI="HAS/config"). In parallel, the HAS Ingest fetches the representation elements concurrently and according to the manifest. Those elements and the manifest itself are sent to MSYNC sender with, for each, the indication of the channel (i.e. the transport multicast session) to be used for the transmission, the object type, the media sequence and the manifest type (only for the manifest) and the object URI. The objects are sent of the transport multicast session according to the following arrangement.

- initialization segments: sent over the corresponding video transport multicast session and repeated after every segment. Note that the HAS ingest could also decide to send the initialization segment over the Control multicast session or even decide not to rely on MSYNC but on the CDN (i.e. the HAS server, in the multicast gateway when the initialization segment is requested would fetch it from the unicast path). Object URI = "provider.cdn.com/live/channel2/dash/video[x]/init.mp4"

- video segments: sent over the corresponding video transport multicast session. Object URI = "provider.cdn.com/live/channel2/dash/video[x]/[segment name]"

- audio segment: sent over the corresponding video transport multicast session. Object URI = "provider.cdn.com/live/channel2/dash/audio[x]/[segment name]"

- manifest file: sent (duplicated) over each of the 2 video transport multicast sessions, after a video segment is sent. Note that the HAS ingest could also decide to send the manifest file over the Control multicast session. Object URI = "provider.cdn.com/live/channel2/dash/manifest.mpd"

(6): The MSYNC sender transmits the objects on demand (from the HAS ingest) and according to Section 3.8 in each of the indicated Transport multicast session. The MSYNC sender MUST continuously adapt the transmission rate to its maximum permitted MAX_BW_SEND.

(7): The HAS server and MSYNC receiver establish a connection wherein the MSYNC receiver communicates its bandwidth restriction

(MAX_BW_RECEIVE) and in return, the HAS server instructs the MSYNC Receiver to join the control multicast session.

(8): The MSYNC receiver joins the control multicast session, receives the transport multicast session arrangement ("HAS/config") and communicates that HAS configuration file to the HAS server.

(9): The player requests the manifest: https://provider.cdn.com/live/channel2/dash/manifest.mpd. In this example, the application indicates the original CDN FQDN (full qualified domain name) in the URL per convention.

(10): The HAS server matches the requested URL with the manifest URL attached to the configured service and instructs the MSYNC receiver to join one of the representations combination (i.e. one of the video transport multicast session and the audio transport multicast session) by default. Note that the HAS server can decide to hold the player’s request until the requested manifest is received through MSYNC or it can fetch the manifest from the CDN over the unicast path to accelerate the work flow.

- Video1: transport multicast session = 233.252.0.2:17000; relative URL= "video2/$number$.mp4"; bandwidth="2500000"

- Audio1: transport multicast session = 233.252.0.3:17001; relative URL= "audio2/$number$.mp4"; bandwidth="384000"

(11) The MSYNC receiver joins the Video1 and Audio IP multicast groups, receives MSYNC packets and delivers the objects (media manifests and segments) to the HAS Server along with their Object URI. The HAS Server stores the element in its cache along with its cache key (built with the Object URI). As overviewed in Section 3.7.4, it is possible that the object delivered by the MSYNC receiver is corrupted (i.e. incomplete) ; in that case the MSYNC receiver delivers the partial object along with missing bytes ranges to the HAS server. The latter attempts to repair the object through e.g. HTTP (not represented in the figure).

(12) The HAS server detects the presence of the requested manifest in its cache and returns it to the player. Note that the manifest object should be complete; potential transmission errors have been managed by the MSYNC receiver through RTP retransmission (Section 3.9.2). Note that without RTP retransmission, the MSYNC receiver would have communicated the received object to the HAS server with possibly the list of missing byte ranges.

(13) The player parses the manifest and requests the segments from video1/Audio1 representations as well as the manifest (possibly

updated).

(14) The HAS server either find the requested element in its cache or fetch it from the CDN and return them to the Player. Note that the element should be complete. Potential transmission errors would have been managed by the MSYNC receiver through RTP retransmission (Section 3.9.2). Note that without RTP retransmission, the MSYNC receiver would have communicated each received object to the HAS server with the list of any missing byte ranges.

(15) The player wishes to switch to an higher sub-stream representation and requests an element from the Video2 representation.

(16) The HAS server gets a cache miss, detects that the MSYNC receiver is not tuned to the right channel. It instructs the MSYNC receiver to first leave Video1 and then join Video2. The HAS server does that as it knows the bandwidth restriction announced by the MSYNC receiver. Note that the MSYNC receiver would have refused to Join Video2 while being still attached to Video1. Note also that the HAS server can decide to hold the player’s request until the requested object is received through MSYNC or it can fetch the object from the CDN over the unicast path to accelerate the work flow.

(17) The MSYNC receiver leaves the current IP multicast group corresponding to Video1 and joins the IP multicast group corresponding to Video2, receives MSYNC packets and sends media manifests/segments to the HAS Server along with their Object URI.

(18) The HAS server receives the objects, store them in its local cache and returns the requested element to the player.

(19) The player requests the segment from video2/Audio2 representations or the manifest (possibly updated).

(20) The HAS server either find the requested element in its cache or fetch it from the CDN and return it to the Player.

4. IANA Considerations

This document has no actions for IANA.

5. Security Considerations

MSYNC is exposed to the risks linked to the underlying transport protocols: UDP and RTP. An attacker can spoof the source and destination addresses, modify any MSYNC headers and, because MSYNC applies to IP multicast, the MSYNC sender has no control about the MSYNC receivers which may represent a non-authorized party.

The multicast communication between the MSYNC sender and the MSYNC receiver SHOULD be protected against confidentiality leaks, message tampering and replay attacks. The MSYNC protocol does not specify 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 encryption. Secure RTP (SRTP) [RFC3711] and IPsec applied to multicast [RFC5374] are potential candidates for providing such extensions.

As MSYNC supports transporting HTTP headers, it MAY also supports HTTP based mechanisms for integrity protection and authentication as specified by [RFC9530] and [RFC9421] respectively and profiled for multicast by [DVB-MABR]. Based on asymmetric cryptography, the authentication method specifies a digital signature [RFC9421] protecting a content digest [RFC9530] and a certain number of HTTP headers complemented with derived components such as the MSYNC Object URI. The signature itself is computed with a well identified/permitted algorithm and a private key attached to a X.509 certificate [RFC5280] . A dedicated HTTP header transports the algorithm identifier and the public key identifier that corresponds to the subject key identifier of the same X.509 certificate [RFC5280]. The content digest [RFC9530] is transported as a dedicated HTTP header field and is applied to the payload data of an MSYNC object transported with MSYNC Object data packets or MSYNC Object data-part packets. .

6. References

6.1. Normative References

[ID-HTTPLIVESTREAM] R Pantos, IETF Internet draft: "HTTP Live Streaming 2nd Edition", draft-pantos-hls-rfc8216bis-18, 20 August 2025

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

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

[RFC2119] S. Bradner, "Key words for use in RFCs to Indicate Requirement Levels", RFC 2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC2236] W. Fenner, "Internet Group Management Protocol, Version 2", RFC 2236, November 1997, <https://www.rfc-editor.org/info/rfc2236>

[RFC3550] H. Schulzrinne, S. Casner, R. Frederick, V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", RFC 3550, July 2003, <https://www.rfc-editor.org/info/rfc3550>.

[RFC9776] B. Haberman, "Internet Group Management Protocol, Version 3", RFC 9776, March 2025, <https://www.rfc-editor.org/info/rfc9776>

[RFC3986] T. Berners-Lee, R. Fielding, L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", RFC 3986, January 2005, https://www.rfc-editor.org/rfc/rfc3986.html

[RFC5506] I. Johansson, M. Westerlund. "Support for Reduced-Size Real-Time Transport Control Protocol(RTCP): Opportunities and Consequences", RFC 5506, April 2009, <https://www.rfc-editor.org/info/rfc5506>.

[RFC5761] Perkins, C. and M. Westerlund, "Multiplexing RTP Data and Control Packets on a Single Port", RFC 5761, April 2010, <https://www.rfc-editor.org/info/rfc5761>.

[RFC8216] R. Pantos, Ed., W. May, "HTTP Live Streaming", RFC 8216, August 2017, <https://www.rfc-editor.org/info/rfc8216>

[RFC9112] R. T. Fielding, M. Nottingham, J. Reschke, " HTTP/1.1", RFC 9112, June 2022, <https://www.rfc-editor.org/info/rfc9112>.

[STD63] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, November 2003.

6.2. Informative References

[BFTR145] "TR-145 Multi-service Broadband Network Functional Modules and Architecture, Issue: 1, Iddue date: November 2012", Technical report, Broadband Forum.

[BFTR178] "TR-178 Multi-service Broadband Network Architecture and Nodal Requirements, Issue: 2, Issue Date: September 2017", Technical report, Broadband Forum.

[MPEG2TS] "ISO/IEC 13818-1 Generic coding of moving pictures and associated audio information: MPEG2 Systems"

[RFC3551] Schulzrinne, H. and S. Casner, "RTP Profile for Audio and Video Conferences with Minimal Control", RFC 3551, July 2003, <https://www.rfc-editor.org/info/rfc3551>.

[RFC3711] M. Baugher, D. McGrew, M. Naslund, E. Carrara, K. Norrman. "The Secure Real-time Transport Protocol (SRTP)", RFC 3711, March 2004, <https://www.rfc-editor.org/info/rfc3711>.

[RFC4541] M. Christensen, K. Kimball, F. Solensky, "Considerations for Internet Group Management Protocol (IGMP) and Multicast Listener Discovery (MLD) Snooping Switches", RFC 4585, July 2006, <https://www.rfc-editor.org/info/rfc4541>

[RFC4585] J. Ott, S. Wenger, N. Sato, C. Burmeister, J. Rey. "Extended RTP Profile for Real-time Transport Control Protocol(RTCP)-Based Feedback (RTP/AVPF)", RFC 4585, July 2006, <https://www.rfc-editor.org/info/rfc4585>.

[RFC4588] Rey, J., Leon, D., Miyazaki, A., Varsa, V., and R. Hakenberg, "RTP Retransmission Payload Format", RFC 4588, July 2006, <https://www.rfc-editor.org/info/rfc4588>.

[RFC5280] : D. Cooper, S. Santesson, S. Farrel, S. Boeyen, R. Housley, W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", IETF RFC 5280, May 2008, <https://www.rfc-editor.org/rfc/rfc5280.html>

[RFC5374] B. Weis, G. Gross, D. Ignjatic. "Multicast Extensions to the Security Architecture for the Internet Protocol", RFC 5374, November 2008, <https://www.rfc-editor.org/info/rfc5374>.

[RFC6770] G. Bertrand, E. Stephan, T. Burbridge, P. Eardley, K. Ma, G. Watson, "Use Cases for Content Delivery Network Interconnection", RFC 6770, November 2012

[RFC7336] L. Peterson, B. Davie, R. van Brandenburg, "Framework for Content Distribution Network Interconnection (CDNI)", RFC 7336, August 2014

[RFC8084] G. Fairhurst, "Network Transport Circuit Breakers", RFC 8084, March 2017

[RFC8085] L. Eggert, G. Fairhurst, G. Shepherd, "UDP Usage Guidelines", RFC 8085, March 2017

[RFC9530] R. Polli, L. Pardue, "Digest Fields", RFC 9530, February 2024

[RFC9421] A. Backman J. Richer, M. Sporny, "HTTP Message Signatures", RFC 9421, February 2024

[DVB-MABR] "Adaptive media streaming over IP multicast", ETSI TS103769 V1.2.1, November 2024

7. 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).

8. Change Log

- 18: Updated the Object URI and Object URI size descriptions (Section 3.2) in order to clarify the usage of padding bytes.

- 17: Clarify the role and operations strictly attached to the MSYNC sender and receiver (sections 2.1 and 3.11). Harmonize some editorial choices.

- 16: "Digest Fields" and "HTTP Messages signatures" Internet drafts became RFC 9530 and RFC 9421 respectively. Authors address updated.

- 15: Section 3.9 detailed including a figure about the RTP repair workflow. Two new sections dedicated to MSYNC configuration (3.10) and a complete MSYNC transport session workflow (3.11). Various editorial updates including some more details in the Security

section .

- 14: A set of editorial changes after a review from Markus .

- 13: A minor edit in Section 3.7.3.

- 12: An extensive review of grammatical and orthographical bugs. Adding clarification regarding congestion control.

- 11: Another round of grammatical/orthographical errors correction. Clarified the Figures 1 and 2 regarding the directional media flows, adding a statement in the introduction about multicast and capacity planning

- 10: Introduced sub-sections in Section 2 allowing to describe the multicast network assumptions and in particular related to congestion avoidance (pre-provisioning the bandwidth resources) . Similarly introduced new sub-sections in Section 3.7 for describing congestion control. Performed several minor editorial corrections. Corrected the new mtype value associated with the media HAS playlist.

- 09: New set of editorial/clarification changes. Added a new mtype value (Section 3.2) for differentiating master and media HLS playlist backward compatible.

- 08: Another round of editorial changes

- 07: Lots of editorial changes

- 06: Example in Section 3.8.1.2. update the example for using the "#" character as the bloc number prefix instead of the "_" character.

- 05: Updated Section 3.9 adding reference (RFC4588) and details for RTP retransmission. Updated/normalized references in Section 5.1 and Section 5.2.

- 04: Added detection of super object transmission (Section 3.2 and Section 3.8.1.2); several adjustments regarding RFC style; Section numbering correction.(Sections 3.9 and 3.10 are now Sections 3.8 and 3.9 respectively).

Authors' Addresses

Sophie Bale
Broadpeak
3771 Bd des Allies,

35510 Cesson-Sevigne
France

Email: sophie.bale@broadpeak.tv

Remy Brebion
Broadpeak
3771 Bd des Allies,
35510 Cesson-Sevigne
France

Email: remy.brebion@broadpeak.tv

Guillaume Bichot (Editor)
Broadpeak
3771 Bd des Allies,
35510 Cesson-Sevigne
France

Email: guillaume.bichot@broadpeak.tv