

Internet-Draft  
Intended Status: Informational  
Expires: October 29, 2023

S. Bale  
R. Brebion  
G. Bichot  
Broadpeak  
April 27, 2023

**MSYNC**  
**draft-bichot-msync-13**

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

Copyright (c) 2023 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to [BCP 78](#) and the IETF Trust's Legal

Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

<a href="#">1</a>	Introduction . . . . .	<a href="#">3</a>
<a href="#">1.1</a>	Terminology . . . . .	<a href="#">3</a>
<a href="#">1.2</a>	Definitions . . . . .	<a href="#">3</a>
<a href="#">2.</a>	Overview . . . . .	<a href="#">5</a>
<a href="#">2.1.</a>	A typical MSYNC deployment . . . . .	<a href="#">5</a>
<a href="#">2.2.</a>	Unicast Networks . . . . .	<a href="#">8</a>
<a href="#">2.3.</a>	Multicast Network and congestion avoidance . . . . .	<a href="#">8</a>
<a href="#">2.4.</a>	Handling third party content . . . . .	<a href="#">10</a>
<a href="#">3.</a>	MSYNC Protocol . . . . .	<a href="#">10</a>
<a href="#">3.1.</a>	MSYNC Packet Format . . . . .	<a href="#">10</a>
<a href="#">3.2.</a>	Object Info Packet . . . . .	<a href="#">12</a>
<a href="#">3.3.</a>	Object Data Packet . . . . .	<a href="#">14</a>
<a href="#">3.4.</a>	Object HTTP Header Packet . . . . .	<a href="#">15</a>
<a href="#">3.5.</a>	Object Data-part Packet . . . . .	<a href="#">16</a>
<a href="#">3.6.</a>	Maximum Size of an MSYNC Packet . . . . .	<a href="#">17</a>
<a href="#">3.7.</a>	Sending and Receiving MSYNC Objects . . . . .	<a href="#">18</a>
<a href="#">3.7.1.</a>	Mapping over Transport Multicast Sessions . . . . .	<a href="#">18</a>
<a href="#">3.7.2.</a>	Detecting the End of an Object Reception . . . . .	<a href="#">19</a>
<a href="#">3.7.3.</a>	Congestion Control . . . . .	<a href="#">20</a>
<a href="#">3.8.</a>	HAS Protocol Dependency . . . . .	<a href="#">21</a>
<a href="#">3.8.1.</a>	Object Info Packet . . . . .	<a href="#">21</a>
<a href="#">3.8.1.1.</a>	Media Sequence . . . . .	<a href="#">21</a>
<a href="#">3.8.1.2.</a>	Object URI . . . . .	<a href="#">22</a>
<a href="#">3.8.2.</a>	Sending Rules . . . . .	<a href="#">23</a>
<a href="#">3.9.</a>	RTP as the Transport Multicast Session Protocol . . . . .	<a href="#">23</a>
<a href="#">4.</a>	IANA Considerations . . . . .	<a href="#">26</a>
<a href="#">5.</a>	Security Considerations . . . . .	<a href="#">26</a>
<a href="#">6.</a>	References . . . . .	<a href="#">26</a>
<a href="#">6.1.</a>	Normative References . . . . .	<a href="#">26</a>
<a href="#">6.2.</a>	Informative References . . . . .	<a href="#">27</a>
<a href="#">7.</a>	Acknowledgments . . . . .	<a href="#">28</a>
<a href="#">8.</a>	Change Log . . . . .	<a href="#">28</a>
	Authors' Addresses . . . . .	<a href="#">29</a>



## **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 [[MPEGCMF](#)].

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.

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

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

media chunk: A part of a media segment of a fixed duration as specified in [[MPEGCMF](#)].

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](#)].



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 information. As part of the control channel, an MSYNC object may transport some control information (as e.g., the MSYNC receiver configuration).

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 over 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 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 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 MSYNC receiver joins the multicast IP group associated with the HAS representation requested by the HS 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.



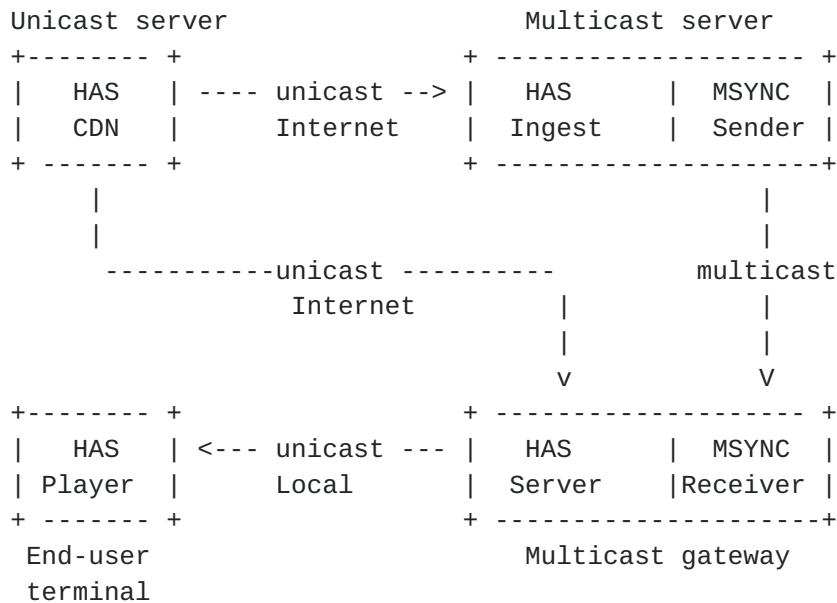


Figure 1: example of MSYNC deployment

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.

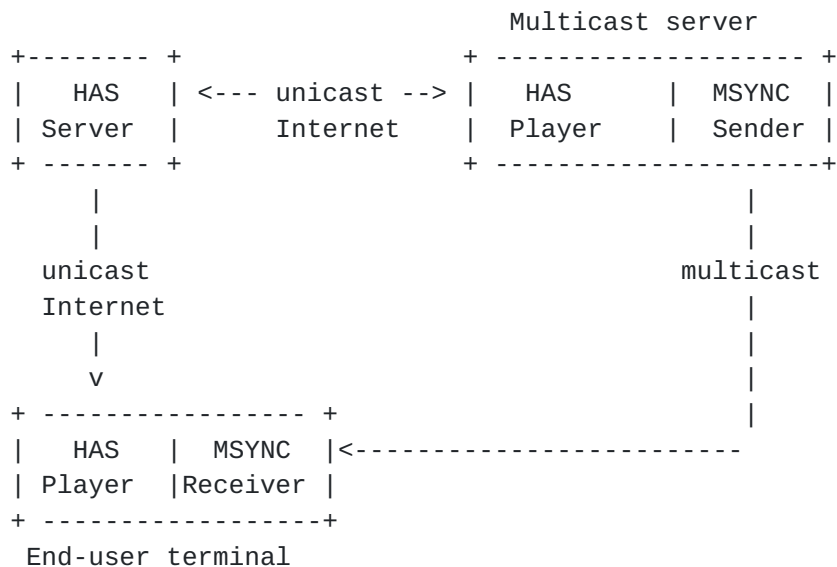


Figure 2: MSYNC receiver in the terminal



## **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 by [[RFC3550](#)] ([Section 10](#)).

The multicast network must support pre-provisioning bandwidth



resources. This assumption permits to configure the MSYNC sender 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 minimum 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 file) 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  $\min(\text{MAX\_BW\_RECEIVE\_1}, \text{MAX\_BW\_RECEIVE\_2}, \text{MAX\_BW\_RECEIVE\_3}, \dots, \text{MAX\_BW\_RECEIVE\_N})$ . There is one `MAX_BW_RECEIVER` 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 configured 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 [[RFC3376](#)].

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](#)**

#### **[3.1. MSYNC Packet Format](#)**

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



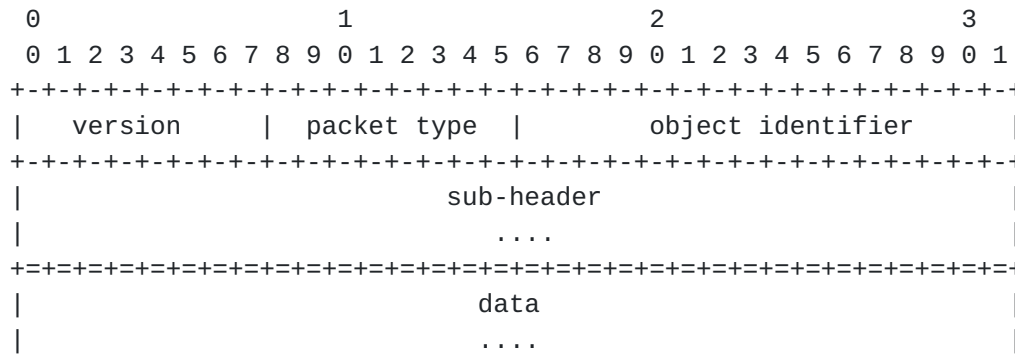


Figure 3: MSYNC Packet

version: 8 bits

version of the MSYNC protocol = 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 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.

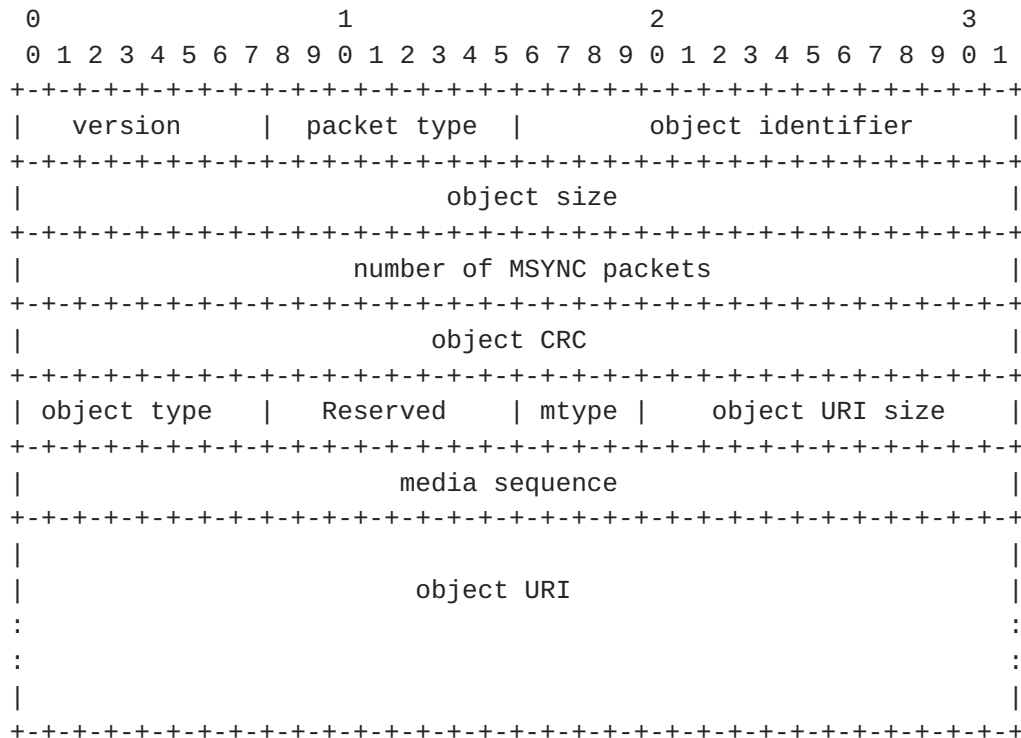


Figure 4: Object Info packet

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 payload transported with a MSYNC object data packet ([Section 3.3](#)) or MSYNC object



data-part packet ([Section 3.5](#)).

The size may be 0 indicating that there is no corresponding object's 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 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.

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: reserved for future use.
- 0x01: media manifest (playlist)
- 0x02: unknown
- 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

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 0x00 (not applicable) otherwise. The field can take the following values.

- 0x00: Not Applicable
- 0x01: MPEG Dash as specified in [[MPEGDASH](#)].
- 0x02: Master HLS playlist as specified in [[RFC8216](#)].
- 0x03: Media HLS playlist as specified in [[RFC8216](#)].
- 0x04-0xF: Reserved





object URI size: 12 bits

The size in bytes of the object URI field. The object URI maximum size depends on the network MTU as discussed in [Section 3.7](#).

media sequence: 32 bits

A sequence number 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 to synchronize unicast and multicast receptions in the multicast gateway. 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: Quotient ((object URI size \* 8)/32) bits + 32 bits if remainder ((object URI size \* 8)/32) > 0

This is the path name associated with the object. 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. Remaining unused bytes of the last 32 bits field 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 prescribed by the associated object info packet. Object payloads are transported through a series of object data packets.

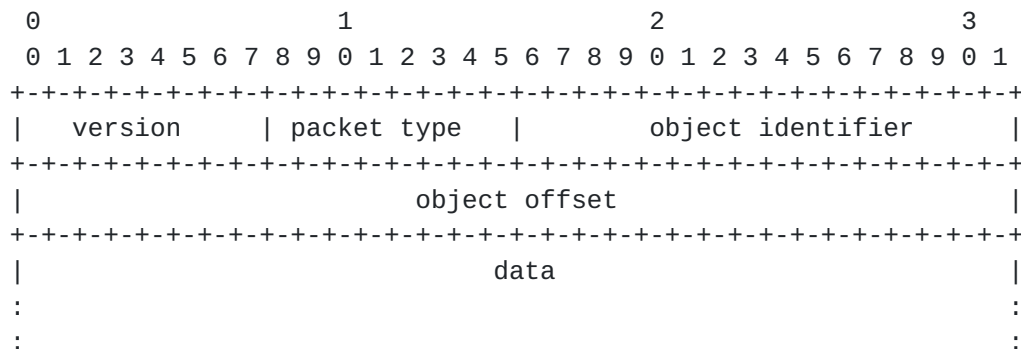


Figure 5: Object Data packet

packet type: 0x03



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 8 bits

The data 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](#). 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**

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 use 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 to be served to the HAS player.

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 data-part) that can be repeated.

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

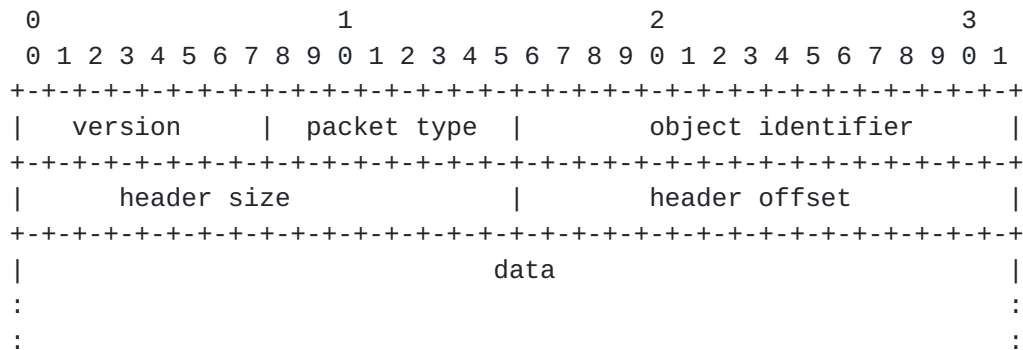


Figure 6: Object HTTP Header packet

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:  $N \times 8$  bits

The data related to the HTTP header ( e.g., the HTTP header associated with 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](#).

### **[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 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 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 data-part objects 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](#).



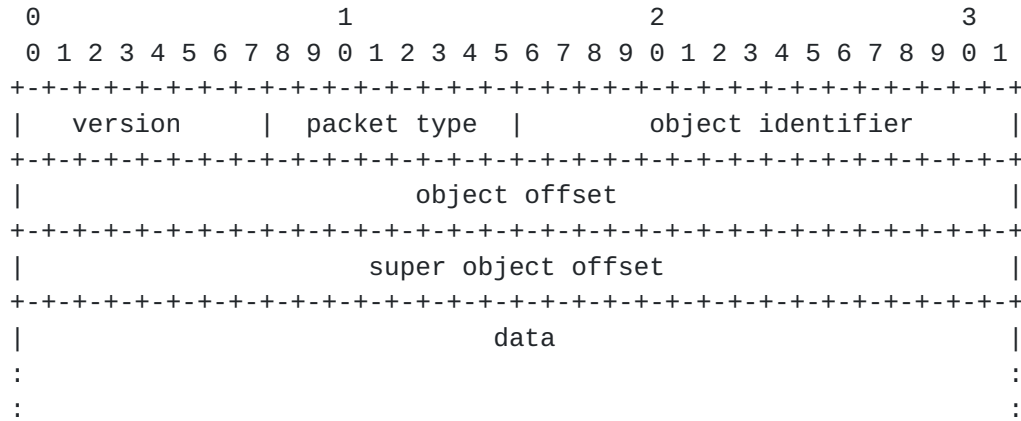


Figure 7: Object Data-part packet

packet type: 0x06

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.

data: N x 8 bits

The data 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 is indicated in the associated object info (field object size).

**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 sender and MSYNC receiver configuration. Note that the configuration procedure (protocol and format) is out of the scope of that document.

#### **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 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.

The MSYNC receiver is configured to join the IP transport multicast session corresponding to the sub-stream the application (the HAS server in figure 1) must receive depending on the incoming requests from the end user terminal/player. In general, the MSYNC receiver is configured to join the IP multicast stream associated with the content stream the application wants to listen/receive.

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 MSYNC control information 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 MSYNC control channel.

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

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 more HTTP header packets (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 a of size null (used to signal the end of the transmission of a super object) then only one object info packet is sent (see 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 a new object transmission.

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 may (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 (when the MSYNC receiver repairs the packets 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.

### **[3.8. 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 multicast gateway to synchronize the MSYNC (i.e., multicast) reception with unicast reception. The multicast gateway may operate jointly MSYNC/multicast and unicast for retrieving HAS elements as indicated in [Section 2](#) and illustrated in Figure 1. 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 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 (Info Packet header field presented in the [Section 3.2](#).) as listed below.

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  $\$time\$/(\text{divided by})@timescale$  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  $\$time\$/scale$  or  $\$Number\$\$$  value

### **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 4th 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 5th 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

### **[3.8.2](#). Sending Rules**

Whenever a manifest has to be sent over MSYNC, the following applies.

- The corresponding MSYNC object data packets MUST be sent over all the transport multicast sessions related to the transmission of the media segments the manifest refers to.
- The manifest MUST refer to addressable objects (segment or CMAF chunk) that have already been sent or for which the transmission has started.

### **[3.9](#). RTP as the Transport Multicast Session Protocol**

RTP [[RFC3550](#)] MAY be used as part of the transport multicast session protocol with the restrictions defined in [Section 2](#) of



[RFC3551] according to the following.

- The RTP header contains 0 (zero) contributing source identifier (CSRC) fields.
- The RTP header timestamp field is computed as indicated in [RFC3550]; it corresponds to the instant the MSYNC sender starts the MSYNC packet transmission.
- The RTP header payload type (PT) field MAY correspond to one of the values specified in [RFC3551]. Its value should be communicated to the MSYNC receiver as part of the MSYNC receiver configuration.
- Each RTP multicast session MUST operate a unique different SSRC number [RFC3550]. This allows packet retransmission (if used) on the RTP transport multicast session basis.
- RTCP usage is not required.

Packet retransmission (see Figure 8 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], [RFC5506] and [RFC5761] according to the followings:

- 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 [RFC5506] and [RFC4585].
- The RTP Repair server receives, processes and responds to the feedback NACK messages (FB) according to [RFC4588]. 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). In any case, the RTP Repair server and the RTP Repair client MUST operate a unicast interface.
- The Session-multiplexing scheme [RFC4588] MUST be applied: the RTP retransmission (repair) stream MUST be sent on a different RTP session than the original (multicast) RTP stream.
- The retransmission stream MUST support multiplexing the RTP and RTCP traffic on a single port according to [RFC5761].



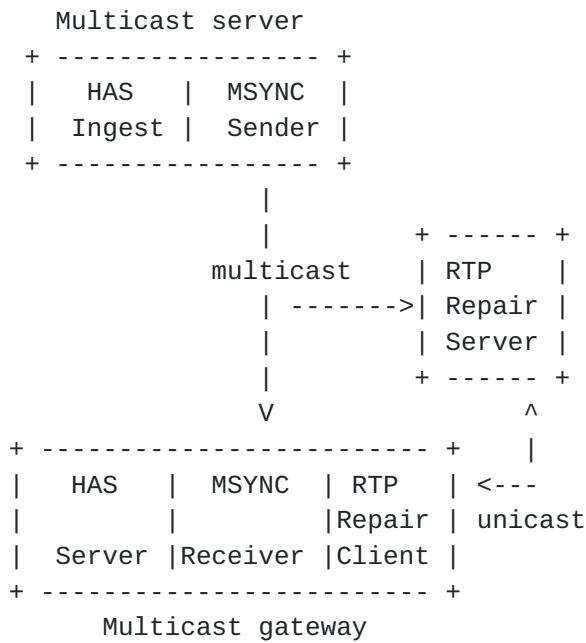


Figure 8: RTP repair

Note that instead of relying on "RTP retransmission", the MSYNC receiver (i.e., the multicast gateway) could attempt to recover/repair damaged HAS elements (e.g., segments, manifest) through HTTP (aka "HTTP repair") and byte-range requests. However the latter 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).





#### **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.

#### **6. References**

##### **6.1. Normative References**

- [RFC2119] S. Bradner, "Key words for use in RFCs to Indicate Requirement Levels", [RFC 2119](https://www.rfc-editor.org/info/rfc2119), March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC2236] W. Fenner, "Internet Group Management Protocol, Version 2", [RFC 2236](https://www.rfc-editor.org/info/rfc2236), 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](https://www.rfc-editor.org/info/rfc3550), July 2003, <<https://www.rfc-editor.org/info/rfc3550>>.
- [RFC3376] B. Cain, S. Deering, I. Kouvelas, B. Fenner, A. Thyagarajan, "Internet Group Management Protocol, Version 3", [RFC 3376](https://www.rfc-editor.org/info/rfc3376), October 2002, <<https://www.rfc-editor.org/info/rfc3376>>
- [RFC5506] I. Johansson, M. Westerlund. "Support for Reduced-Size Real-Time Transport Control Protocol(RTCP): Opportunities and Consequences", [RFC 5506](https://www.rfc-editor.org/info/rfc5506), April 2009, <<https://www.rfc-editor.org/info/rfc5506>>



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

## **6.2. Informative References**

- [BFTR145] "TR-145 Multi-service Broadband Network Functional Modules and Architecture, Issue: 1, Issue 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.
- [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>>.
- [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
- [RFC8216] R. Pantos, Ed., W. May, "HTTP Live Streaming", [RFC 8216](#), August 2017, <<https://www.rfc-editor.org/info/rfc8216>>.

## **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**

- 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 HS 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  
15 rue Claude Chappe  
Zone des Champs Blancs  
35510 Cesson-Sevigne  
France

Email: [sophie.bale@broadpeak.tv](mailto:sophie.bale@broadpeak.tv)

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

Email: [remy.brebion@broadpeak.tv](mailto: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](mailto:guillaume.bichot@broadpeak.tv)