TECHNICAL REPORT

# ISO/IEC TR 23008-13

Second edition
2017-08

Corrected version
2017-10

# Information technology — High efficiency coding and media delivery in heterogeneous environments —

## Part 13: MPEG media transport implementation guidelines

*Technologies de l'information — Codage à haute efficacité et livraison des medias dans des environnements hétérogènes —*

*Partie 13: Lignes directrices de mise en oeuvre du transport des medias MPEG*



Reference number
ISO/IEC TR 23008-13:2017(E)



ISO copyright office
Ch. de Blandonnet 8 • CP 401
CH-1214 Vernier, Geneva, Switzerland
Tel. +41 22 749 01 11
Fax +41 22 749 09 47
copyright@iso.org
www.iso.org

# Contents

Page

# Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular the different approval criteria needed for the different types of document should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see www.iso.org/directives).

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights. Details of any patent rights identified during the development of the document will be in the Introduction and/or on the ISO list of patent declarations received (see www.iso.org/patents).

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation on the voluntary nature of standards, the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT) see the following URL: www.iso.org/iso/foreword.html.

This document was prepared by Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 29, *Coding of audio, picture, multimedia and hypermedia information*.

This second edition cancels and replaces the first edition (ISO/IEC/TR 23008-13:2015), which has been technically revised.

The main changes compared to the previous edition are as follows:

— areas on MMT deployment case in Japan and conversion between MMTP and MPEG-2 TS stream have been improved by adding new guidelines.

A list of all parts in the ISO/IEC 23008 series can be found on the ISO website.

This corrected version of ISO/IEC TR 23008-13:2017 incorporates the following corrections:

— headers have been corrected and now read “ISO/IEC TR” instead of “ISO/TR”.

# Introduction

This document provides guidelines for implementation and deployment of multimedia systems based on the ISO/IEC 23008 series. These guidelines include the following:

— guidelines on usage of MMT functions;

— guidelines on deployment use cases designed based on ISO/IEC 23008-1.

# Information technology — High efficiency coding and media delivery in heterogeneous environments —

# Part 13:
# MPEG media transport implementation guidelines



## 1 Scope

This document provides technical guidelines for implementing and deploying systems based on ISO/IEC 23008-1.

## 2 Normative references

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO/IEC 23008-1:2014, *Information technology — High efficiency coding and media delivery in heterogeneous environments — Part 1: MPEG media transport (MMT)*

## 3 Terms, definitions, symbols and abbreviated terms

For the purposes of this document, the terms, definitions, symbols and abbreviated terms given in ISO/IEC 23008-1 apply.

ISO and IEC maintain terminological databases for use in standardization at the following addresses:

— IEC Electropedia: available at http://www.electropedia.org/

— ISO Online browsing platform: available at http://www.iso.org/obp

## 4 Overview

### 4.1 System overview

This clause describes the exemplary but typical system overview of MPEG Media Transport (MMT) as shown in Figure 1.

The media origin provides A/V media or generic files to MMT sending entity in the form of Packages or Assets which are defined in ISO/IEC 23008-1. A Package is comprised of Assets, Presentation Information and Transparent Characteristics, etc. Physically, an Asset is a group of MPUs or generic files.

The MMT sending entity fragments MPU/generic files and generates MMTP packets to deliver A/V media data itself. Concurrently, it also generates signalling message for the successful delivery and presentation of A/V media included on that MMTP packet flow.

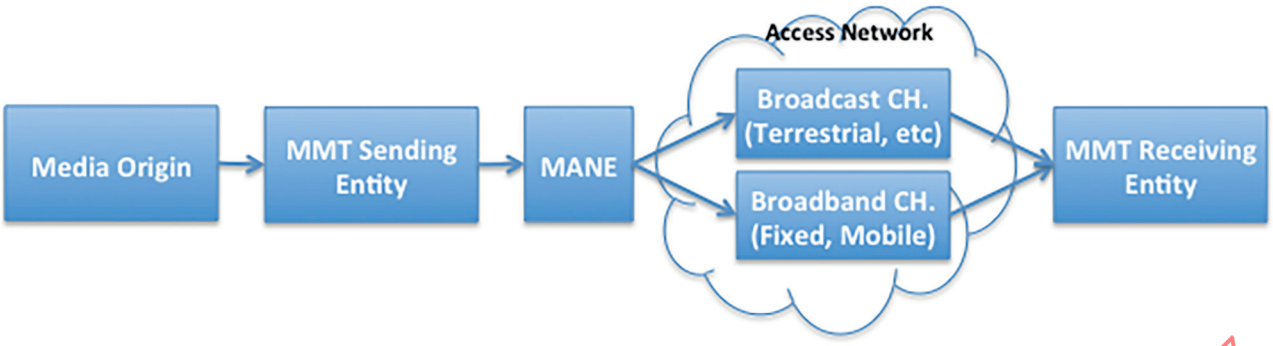


**Figure 1 — Example of MMT-based media distribution chain**

The MMT Aware Network Element (MANE) may be any network element, such as media caches and routers, that are aware of MMTP and has augmented functions for its own purposes to utilize tools from MMT.

Then, MMTP packets can be transmitted through either or both of broadcast channel and broadband channel at its own environment and scenarios.

The CI information provides presentation information, such as the location of media objects, as well as the timing and relation of the media objects that MMT receiving entity has to follow. This information is provided by MMT sending entity and also pushes related MMTP packet flow to the MMT receiving entity. It means It fully controls the media streaming session, i.e. it manages the on-time delivery, playback and temporal/spatial presentation information of the media.

## 4.2 Normative parts

ISO/IEC 23008-1 specifies a set of tools to enable advanced media transport and delivery services. Figure 2 depicts the end-to-end architecture and illustrates the different functional tools and their relationships. Moreover, it shows interfaces between existing protocols and standards defined by ISO/IEC 23008-1 and those defined in other specifications. The tools spread over three different functional areas, Media Processing Unit (MPU) format, delivery and signalling, defined in ISO/IEC 23008-1 as follows.

— The Media Processing Unit (MPU) defines the logical structure of media content format of the data units to be processed by an MMT entity and their instantiation with ISO Base Media File Format as specified in ISO/IEC 14496-12.

— The delivery function defines an application layer transport protocol and a payload format. The MMTP transport protocol provides enhanced features for delivery of multimedia data, e.g. multiplexing and support of mixed use of streaming and download delivery in a single packet flow. The payload format is defined to enable the carriage of encoded media data which is agnostic to media types and encoding methods.

— The signalling function defines formats of signalling messages to manage delivery and consumption of media data.

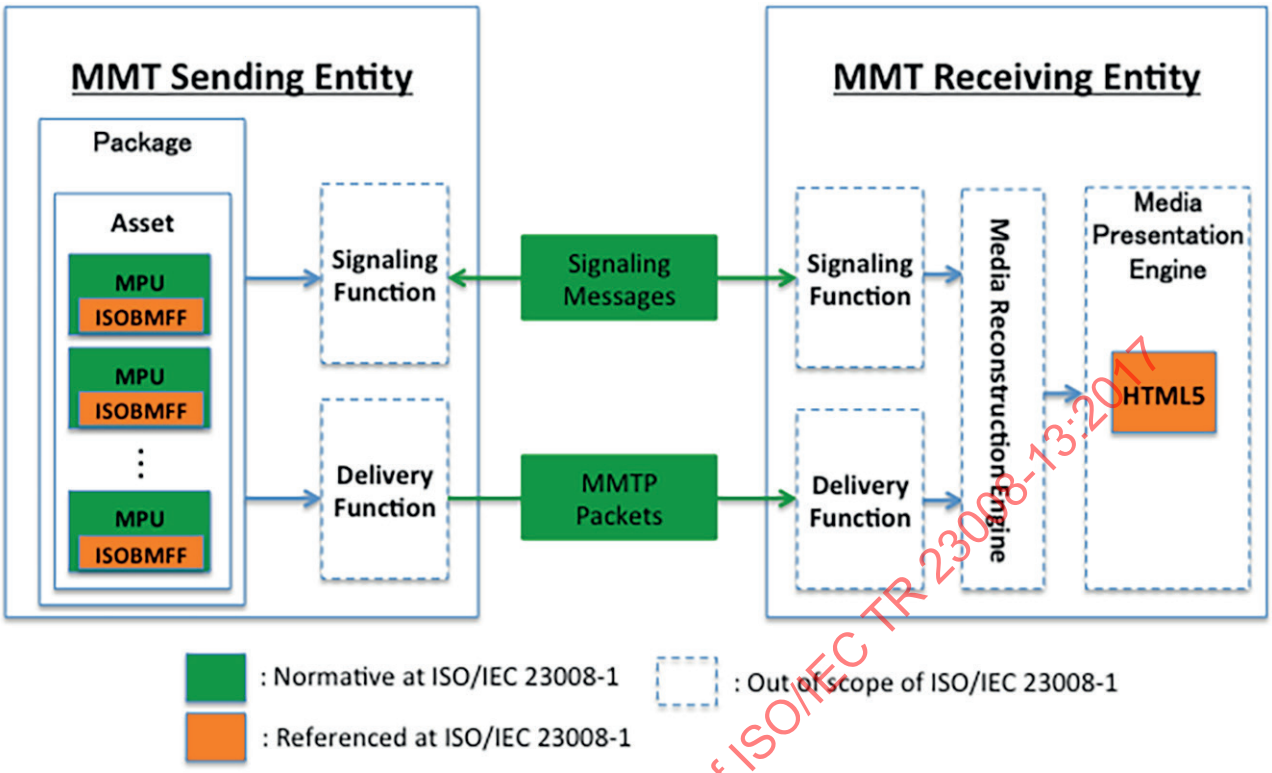


**Figure 2 — MMT functions deployment**

Other aspects, such as client implementations for media reconstruction and presentation itself, are not defined as normative parts of the ISO/IEC 23008-1.

# 5 MMT function deployments

## 5.1 General

This clause gives implementation guidelines on general MMT deployment based on basic functionalities provided by MMT standard itself. This subclause intends, in particular, to guide implementers to make best use of the specification for the basic topics such as, but not limited to the following:

— low delay media consumption;

— media adaptation;

— hybrid delivery;

— error recovery.

## 5.2 Object reconstruction

### 5.2.1 General

MMTP is designed to deliver object flows that may be multiplexed together in the same MMTP flow. The objects of an object flow are usually related to each other, meaning that the application is likely to consume all objects of an object flow, if the flow or one of its objects is of interest to that application.

Depending on the delivery mode, the recovery of the object may differ. The GFD mode usually requires that the full object is recovered prior to its delivery to the application. However, the application may request that correctly received contiguous byte ranges of the object are forwarded to the application.

The MPU mode is used to deliver MPUs and usually operates on movie fragments. Alternatively, the application may request that each received MFU is forwarded to the application without additional delay. It may also require that the complete MPU be reconstructed prior to forwarding it to the application.

### 5.2.2 Recovery in MPU mode

When operating in the MPU mode, the object flow consists of MPUs of the same asset. Each MPU is a single object in the object flow and shares the same packet_id as all MPUs of the same asset.

The MMT receiving entity performs the following steps.

a) Receive MMTP packet.

b) Check if packet_id is equal to the packet_id of the object flow of interest, discard packet and go to step a) if it does not belong to an object flow of interest.

c) Assert that type of the MMTP packet is MPU.

d) If fragmentation flags are set (different than "00")

   1) if fragmentation flag is equal to "11", attempt to recover packet and if successful go to step f), or

   2) add packet to the list of packet fragments based on the MMTP sequence number and go to step a).

e) If Aggregation flag A is set, extract all aggregated data units and proceed to step g) for each extracted data unit.

f) If object map with the same MPU_sequence_number does not exist, create a new object map for the MPU with that sequence number.

g) Check fragment type (FT) of the MPU payload header.

   1) If FT is MPU metadata

      i) Check if MPU metadata is already received.

         I) If yes, discard the MPU metadata as being a duplicate, or

         II) insert MPU metadata at the beginning of the object map

         — optionally, forward MPU metadata to application.

      ii) Go to step a).

   2) If FT is Fragment metadata

      i) Check if movie fragment with the same movie_fragment_sequence_number already exists.

         I) If no, create a placeholder for the movie fragment in the object map, or

         II) check if Fragment metadata has already been received.

         — If yes, discard fragment metadata as being a duplicate,

         — otherwise, insert fragment metadata at the beginning of the fragment placeholder.

         III) Go to step a).

   3) If FT is MFU

      i) If fragment placeholder with sequence number movie_fragment_sequence_number does not exist in the object map of the MPU with sequence number MPU_sequence_number, then create movie fragment placeholder in the object map of the MPU.



 

ii) If timed metadata flag is set

I) insert payload in the fragment placeholder in the correct order based on the sample_number and offset values;

II) check if movie fragment is complete, and

— if yes, forward fragment to the application.

III) Go to step a).

iii) If timed metadata flag is not set

I) insert payload in the item in the object map based on the item item_ID,

II) recover item information from MPU metadata for the recovered item and forward the item to the application, and

III) go to step a).

The sender may send the movie fragment out of order, i.e. sending the movie fragment header after sending all the media units that are contained in that movie fragment. At the receiver side, step g.3.i ensures that the movie fragment is recovered appropriately by reordering the received data using the MPU_sequence_number and the movie_fragment_sequence_number. This is necessary if the receiver is operating in the Fragment mode or MPU mode, where only complete movie fragments or complete MPUs are forwarded to the application. When operating in the very low delay mode, the receiver will forward every single MFU to the application. In this case, it has to make sure that the content supports this operation, so that MFUs will be self-describing and self-contained. In particular, the receiver should be able to recover the presentation timestamp of that MFU payload using the sample number, fragment_sequence_number and MPU_sequence_number,

For fragments and items that cannot be recovered correctly by the time the fixed end to end delivery delay passes, error concealment is performed on the movie fragment or the partially recovered item.

### 5.2.3 Recovery in GFD mode

When operating in the GFD mode, the object flow consists of a set of related files. The files of the same object flow share the same packet_id. The application forwards each recovered file or contiguous byte range of a file to the application. The receiver creates an object map to recover each file separately.

The operation of the MMTP receiver is as follows.

a) Receive MMTP packet.

b) Check if packet_id is equal to the packet_id of the object flow of interest, discard packet and go to step a) if it does not belong to an object flow of interest

c) Assert that type of the MMTP packet is GFD.

d) If object map with same TOI does not exist, create new object map for the file with that TOI.

e) Insert payload in the correct place in the object map using the start_offset information.

f) It is recommended that chunks of contiguous byte ranges that lie between two MMTP packets with the RAP flag R set to 1 be forwarded to the application. Applications may choose to forward sufficiently large contiguous byte ranges whenever they are recovered correctly.

g) If complete TOI is recovered

1) extract metadata from the transport object or from the GFD table, and

2) forward file to the application.

## 5.3 Default assets

In order to cater for basic receivers with limited processing capabilities and also to facilitate fast channel tune-in, an alternative and simple way for service consumption has been devised that can function without the need for more advanced and highly demanding presentation information solutions (such as HTML 5). It is of course not possible to achieve the same level of service complexity and richness with the basic solution, but it enables receivers to quickly tune in to the channel and, if needed, to completely avoid processing the complex presentation information.

MMT provides the tools to identify default service components and to enable the receiver to consume them in a synchronized manner by processing the MMT signalling information. Default service components are usually the main video stream together with the default audio stream.

An MMT receiver that wants to achieve fast tune in or wants to bypass processing the presentation information checks the MP table for the MMT package of interest and identifies the assets of that MMT package that are marked as default assets. It then starts receiving and reconstructing the default assets by first locating the asset using the MMT_general_location_info and looking for the MPU metadata information as a starting point for the reconstruction. The MPU header is necessary as it delivers the information about the used media codecs and any applied encryption.

Each MPU of a default asset provides its presentation time, which can be used for synchronized playback of the media components. This is done using the MPU timestamp descriptor, which assigns an NTP playback timestamp for the MPU with sequence number mpu_sequence_number.

If the MMT receiver decides later on to consume the presentation information, it might stop relying on timing information provided in the MP table and use the presentation information instead.

## 5.4 Low-delay live streaming

MMT streaming is based on the MPU concept, which in turn is an ISO-based media file format (ISOBMFF) with certain restrictions. However, the usage of the ISOBMFF may give the impression of high end-to-end delay, which may seem not suitable for live broadcast. This however, is not true. This subclause shows how low-delay live streaming may be performed using MMT.

Live streaming requires real-time media encoding and transmission of the encoded media to a set of receivers. In such scenarios, end-to-end delay has a significant impact on the perceived quality by the end user.



 

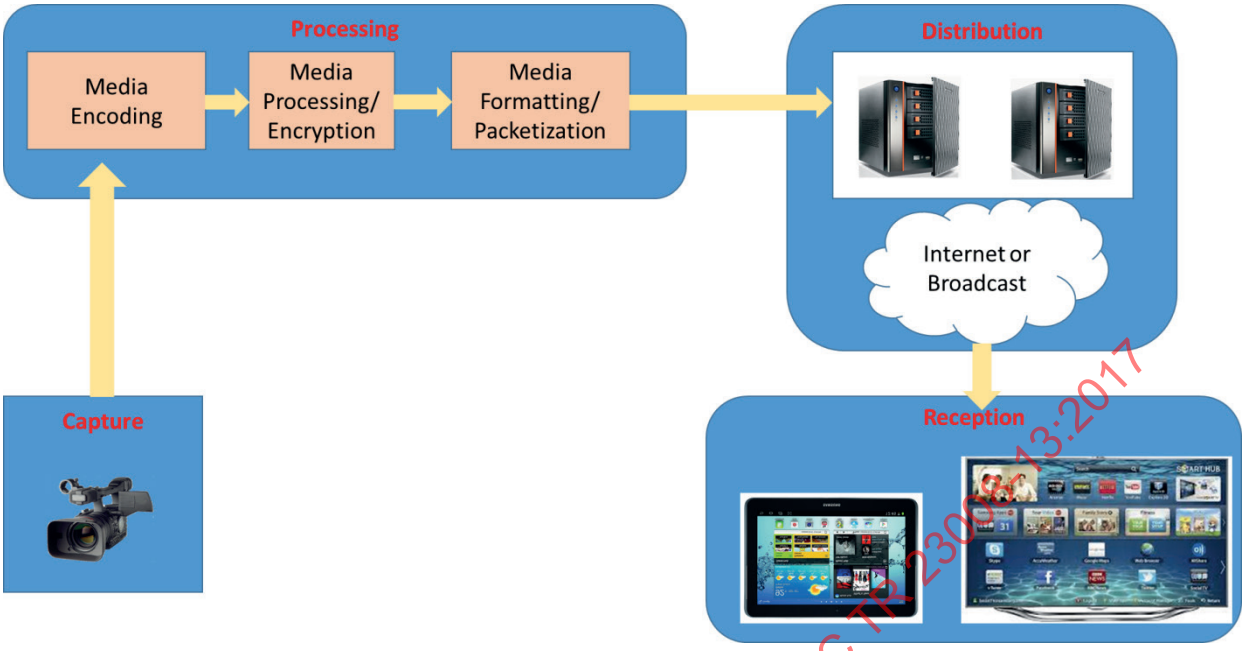


**Figure 3 — Example of a broadcast scenario**

After media encoding and any other related processing (such as encryption), the media data is formatted according to the transmission protocol in use and the packets are then sent down to the receivers as shown in Figure 3. In the case of MMT, MMTP is the transport protocol used for media streaming. MMTP operates on MPUs in the MPU mode, which is the most appropriate mode for streaming. Theoretically, MPU should be completed to start packetizing it and send to the client. However, in real implementation, there are ways to further optimize generation of MPU and packetization of it to minimize delay by starting packetization and delivery of MPU before completion of generation of MPU.

The MPU mode of MMTP is designed to operate in a very low-delay mode and without any restrictions on the MPU size. An MPU is streamed progressively as soon as media data becomes available in a way similar to RTP streaming. Each media unit, such as an AVC NAL unit or an AAC audio frame, is encapsulated into an MMTP packet that also contains the MPU payload header. The MMTP packet looks as shown in Figure 4.

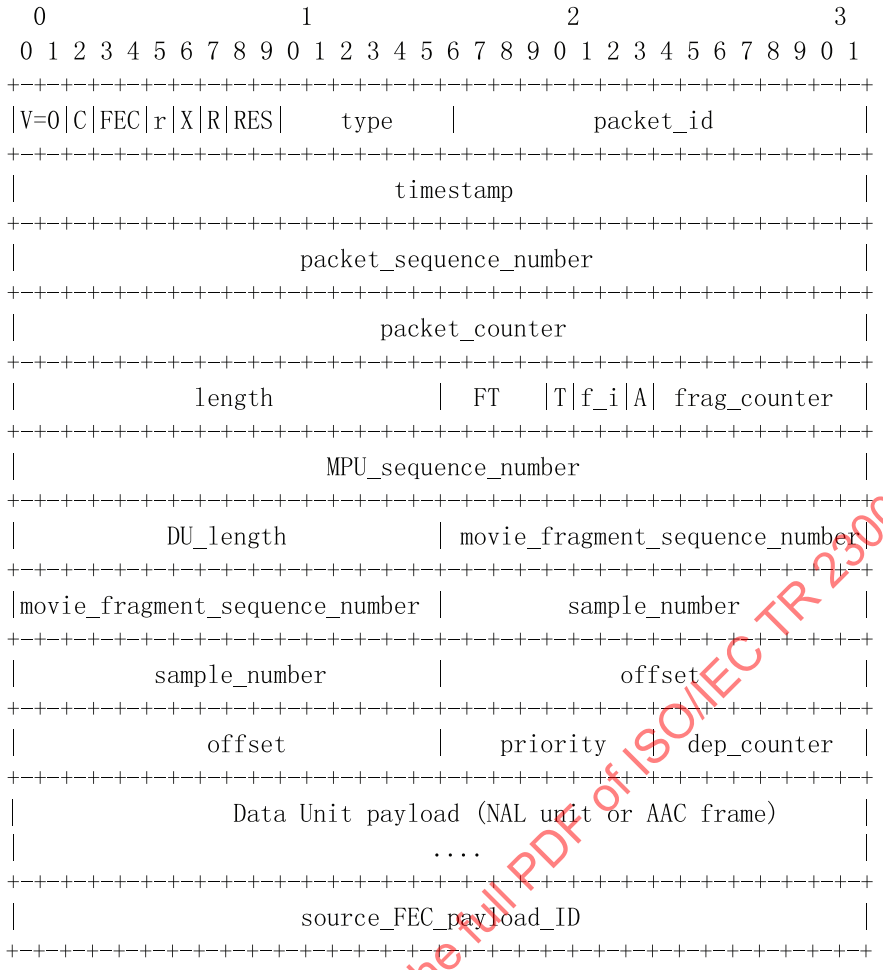


**Figure 4 — MMTP packet**

As can be seen from the packet header, all fields of the packet can be generated immediately at the streaming server, i.e. no need to delay the transmission of the media unit. In particular, the fields *movie_fragment_sequence_number*, the *sample_number* and the *offset* of the media unit in that sample are all know a priori for each media unit, even before generation of the complete movie fragment. The *priority* and *dep_counter* fields may be decided based on the encoder configuration parameters. Most of the case, encoder predefines encoding structure, how many P-frames will be coded between I-frames and how many B-frames will be coded between I- and P-frames, before encoding starts. Therefore, high priority can be assigned for the NAL Units containing intra-coded macroblocks. dep_counter can be set based on the coding structure. If multi-path encoding is performed at the encoder, such configuration could be altered during actual encoding process to improve encoding quality. However, in that case, actual coding structure is known before the last path which generates final compressed data. Therefore, the precise dependency structure can also be known before encoding of each video frame is completed.

The MPU metadata is a data which would need to be sent before transmission of compressed media data for low-delay processing at the client. It consists of the *ftyp* and *moov* boxes, where the latter does not contain any media sample tables and serves as the initialization data, which could be known before encoding is started. Consequently, the MPU metadata may be generated a priori with the knowledge of the media encoder configuration.

The movie fragment metadata contains the *moof* box which provides the timing information for the samples in the movie fragment, as well as their offset. The fragment metadata is constructed progressively and will be ready at the end of the movie fragment. This information is not required for the generation and delivery of the media units and will be sent out of order after all the media data of that movie fragment is transmitted.



 

At the receiver side, the receiver may either consume the media data immediately upon reception of a media unit or it may reconstruct the movie fragment first. In both cases, the overall delay is reduced down to the duration of a movie fragment or less than that.

The reconstruction of the movie fragment at the receiver side is straightforward. All media data that belongs to that particular movie fragment (based on the MPU_sequence_number and the movie_fragment_sequence_number) is first collected progressively to build the mdat box. Finally, after reception of the movie fragment metadata, the fragment can be recovered fully. Any missing media data will be corrected by either marking it as lost or fixing the movie fragment metadata appropriately.

This operation mode is very close to that of RTP streaming and ensures minimal end-to-end delay. It still maintains the characteristics of streaming an ISOBMFF file in a generic and media-independent way.

Another important aspect needs to be considered for low-delay streaming, in particular broadcast application is using very short duration for MPU. In a broadcast application, the client needs to quickly find a starting point of decoding. To support it, conventional broadcasting service repeatedly transmits initialization information for decoder and use very short duration for GOP. In MMT, as each MPU is self-contained, by defining the length of MPU as the period of repetition of decoder initialization parameter in conventional broadcast application, delay can be maintained same as that of conventional broadcast application.

MMTP allows for operation at very low end-to-end delay, in a way suitable to and required by several applications such as live broadcast applications.

## 5.5 Parallel processing in MMT sending and receiving entities

### 5.5.1 Processing in MMT sending entity

MMT protocol performs parallel generation of MMTP flows and signalling message generation/processing. The data packet processing part of MMT protocol performs the packetization of the media data using either the MPU mode for MPUs or the GFD mode for generic files. The generated source data is encapsulated into MMTP packets and transmitted to Transport layer. The signalling message generation part of MMTP processes CI, ADC and other data from the MMT Package and encapsulates them into signalling messages and packetizing and passing them to the Transport layer as shown in Figure 5.

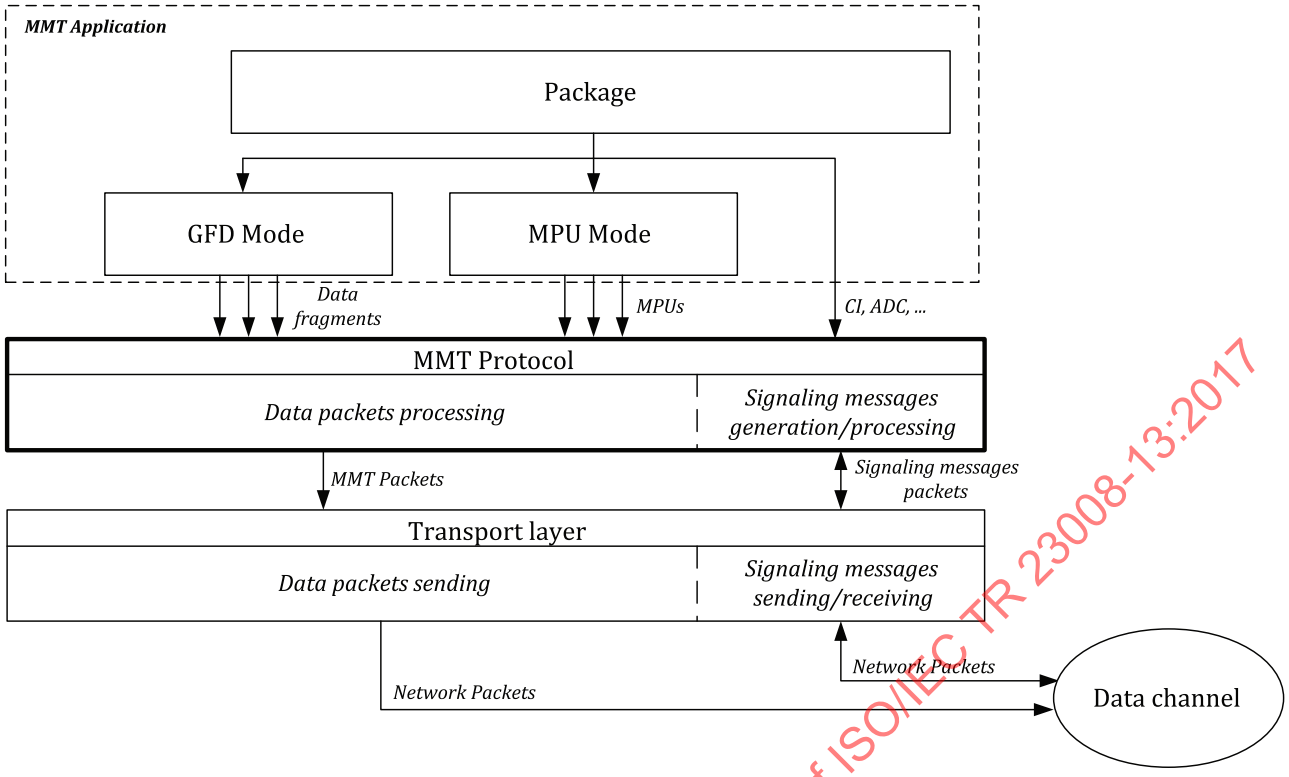


**Figure 5 — MMT sending entity structure**

More detailed architecture of data packet processing part of MMTP is presented in Figure 6. MMTP packets are stored in separate buffers for each data flow after their processing with the help of data flow controller. After that, these MMT packets are passed to the corresponding MMT FEC scheme for protection. Each MMT FEC scheme returns repair symbols with repair FEC payload IDs and source FEC payload IDs. After that, the repair symbols are packetized into FEC repair packets and passed to the Transport layer. The identification of each FEC encoded flow and specifying of FEC coding structure and FEC code are provided by the FEC configuration information.

FEC source packets and their FEC configuration information for each data flow are passed to the corresponding MMT FEC scheme for protection. The MMT FEC scheme uses FEC code(s) for the repair symbol generation. Then FEC source and repair packets are delivered to the MMT receiving entity.

The data flow controllers of MMT sending entity may perform both encapsulation and packetization functions.

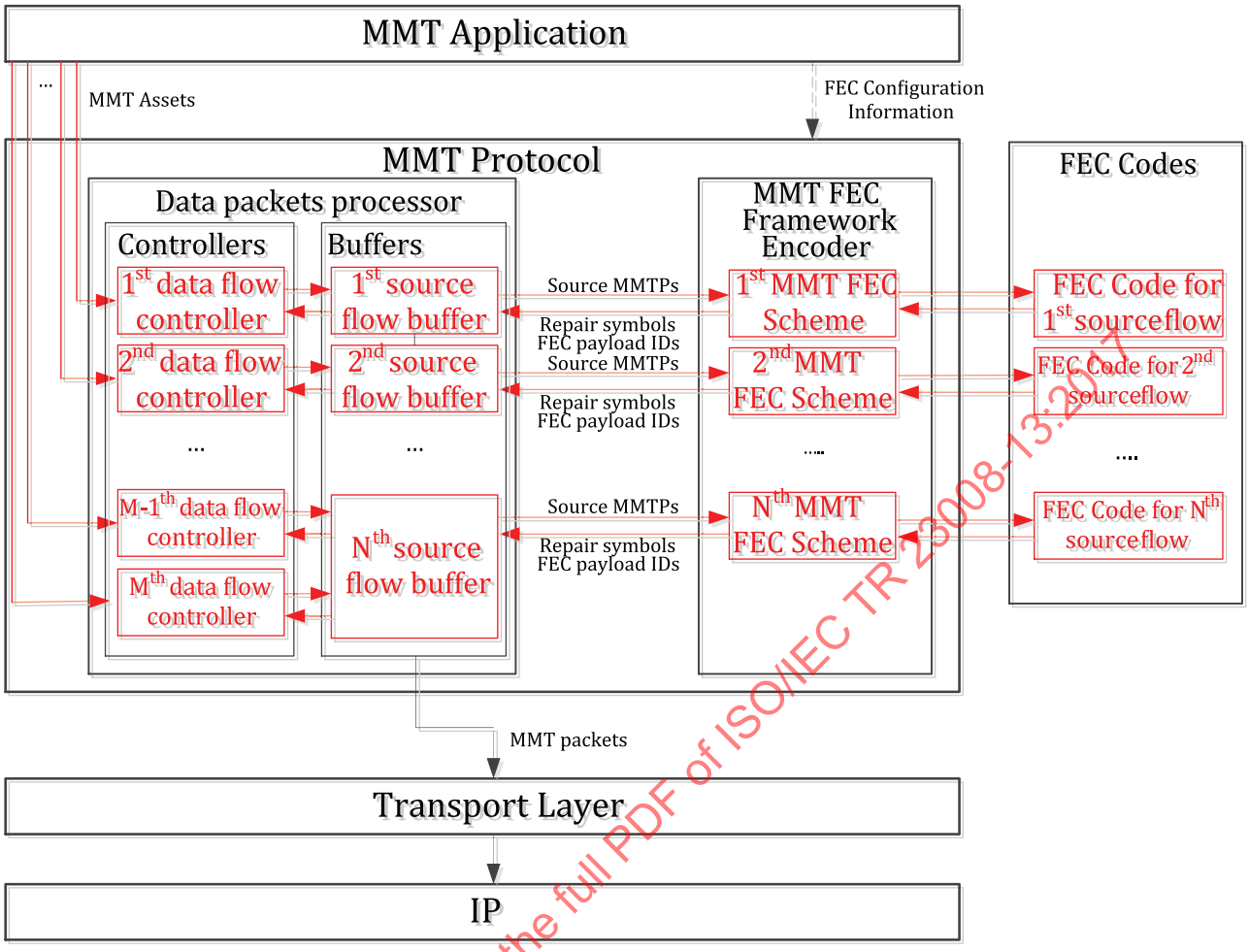


**Figure 6 — Architecture for AL-FEC (MMT sending entity)**

### 5.5.2 Processing in MMT receiving entity

MMT protocol performs parallel processing of MMT packet data flows and generation/processing of signalling messages. The data packet processing part of MMT protocol receives MMTP packets from the transport layer and transfers them into the corresponding data flow processor. Each data flow performs recovering of lost FEC source packets and then passes them for generic object reconstruction and/or MPU reconstruction, which happen in parallel. The signalling message receiving part of MMTP processes incoming signalling message packets and passes them for signalling message reconstruction. The reconstruction of generic objects and MPUs from different assets and of signalling messages is also performed in parallel as shown in Figure 7.

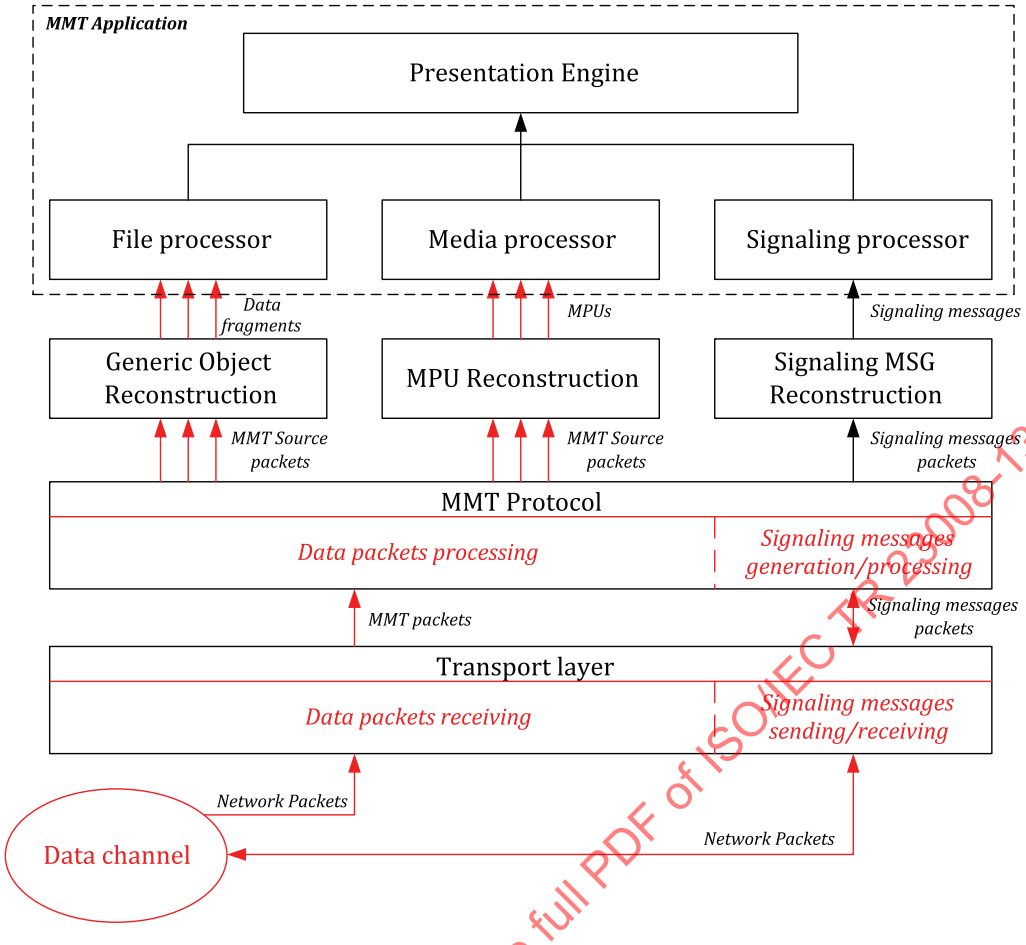


**Figure 7 — MMT receiving entity structure**

At MMT receiving entity, MMT protocol passes each FEC source flow and its associated FEC repair flow(s) to the corresponding MMT FEC scheme. After passing of flows to the MMT FEC schemes, each of the given schemes returns recovered source MMT packets. MMT packets are stored in separate buffers for each data flow and processed by separate data flow controllers. The outlined architecture is depicted in Figure 8.

Data flow controllers of MMT receiving entity unite functions of de-capsulator, de-packetizer and de-jitter.

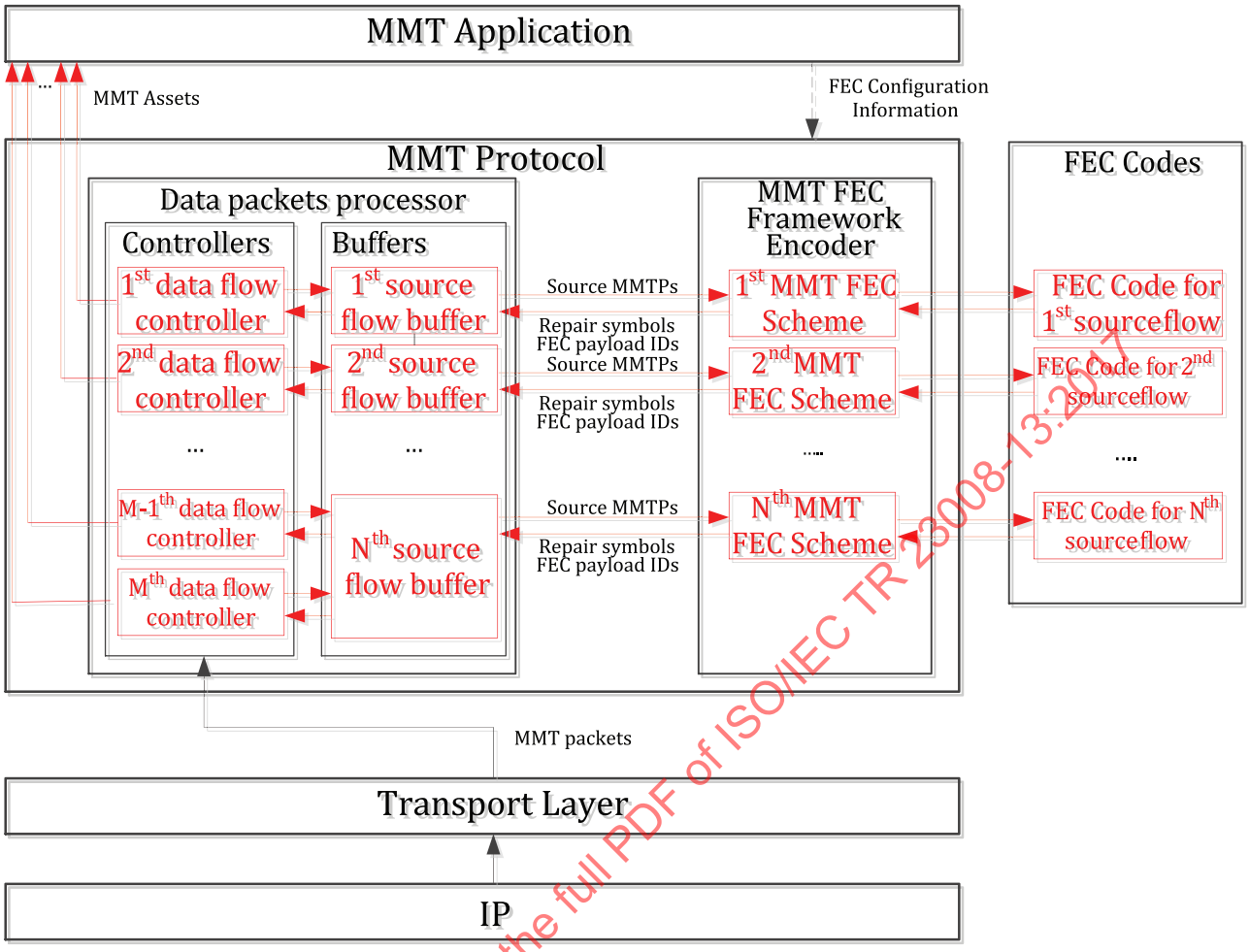


**Figure 8 — Architecture for AL-FEC (MMT receiving entity)**



## 5.6 MPU streaming for live services

### 5.6.1 MPU packetization

#### 5.6.1.1 MPU fragment building block

In MMT, there are two kinds of MPU structures: the timed data and the non-timed data. MMTP supports streaming modes, where the streaming mode is optimized for packetized streaming of ISO Base Media File formatted files. The MPU mode supports the packetized streaming of an MPU.

Figure 9 depicts an MPU fragment building block for packetized streaming of an MPU. This MPU fragment building block helps to prepare for the packetization of an MPU into MMTP packets. The process of creating the MMTP packet passes by two steps: generating the MMTP payload and generating the MMTP packet.

According to the MMT specification, the format of the MMTP payload takes into account the boundaries of data in the MPU to generate packets using the MPU mode. The MPU metadata delivery data unit consists of the “ftyp” box, the “mmpu” box, the “moov” box and any other boxes that are applicable to the whole MPU. The FT field of the MMTP payload carrying a delivery data unit from an MPU metadata is set to “0x00”. The fragment metadata delivery data unit consists of the “moof” box and the “mdat” box header (excluding any media data). The FT field of the MMTP payload carrying a delivery data unit of movie fragment metadata is set to 0x01. The media data, MFUs stored in the “mdat” box of an MPU, is then split into multiple delivery data units in a media-aware way. This may, for example, be performed with the help of the MMT hint track. The FT field of the MMTP payload carrying a delivery data unit

from an MFU is set to “0x02”. Each MFU is prepended with an MFU header. It is followed by the media data of the MFU.

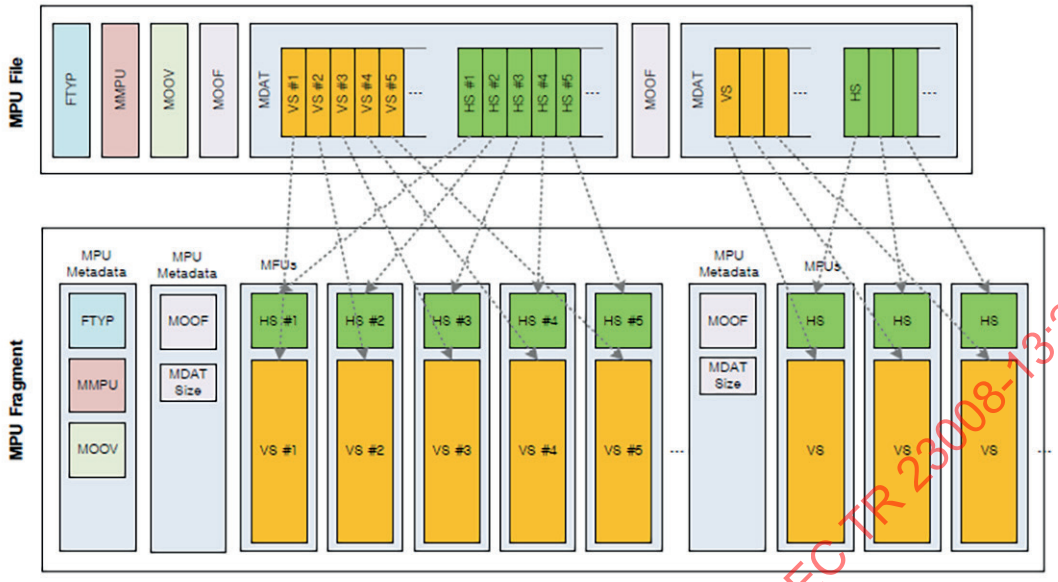


**Figure 9 — MPU fragment building block**



### 5.6.1.2 MMT broadcast sending procedure for low latency

#### 5.6.1.2.1 General

When considering the low latency requirement of broadcasting services, the broadcasting system may consider reducing the delay resulting from both the transmission and the reception procedure, while also considering the requirements on random access to the broadcast channel. In order to support this use case, the MMT receiving entity has to also receive the related presentation information. The MMT sending entity may also consider sending the signalling information periodically during the MPU delivery as shown in Figure 10.

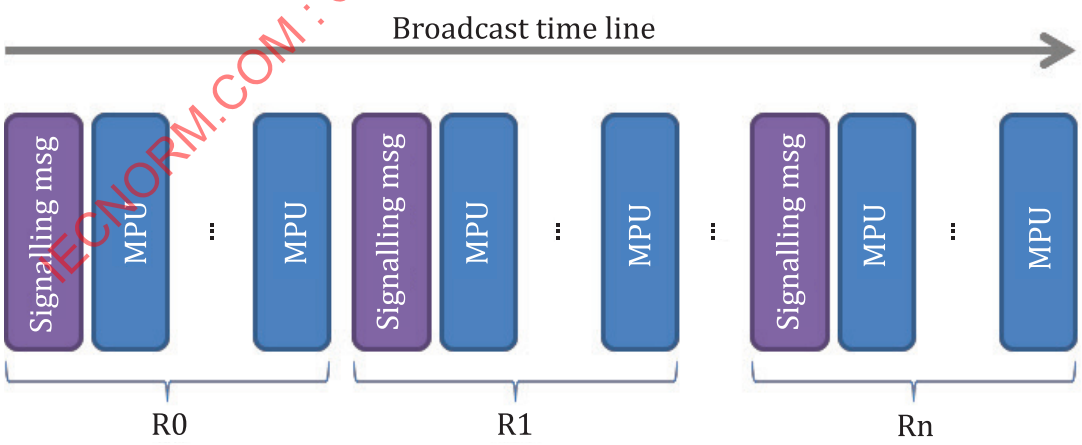


**Figure 10 — Periodical signalling information transmission**

#### 5.6.1.2.2 MPU sending procedure

In order to reduce the end-to-end latency up to the reconstruction of the MPU, the MMT sending entity may change the transmission order of fragments comprising one MPU, which means, for example, transmission of MFUs precede that of fragment metadata. Additionally, for reliable transmission, the MMT sending entity may send the MPU metadata frequently during the transmission of the related MFUs.

When building the MPU fragment building block, the delivery data unit for fragment can be considered as the following two cases.

The first case is that the delivery data unit is MPU. Fragmentation indicator contains information about fragmentation of data unit in the payload, especially boundary information of delivery data unit. When the MPU is delivery data unit, the MMT receiving entity prepares the reconstruction of MPU in MMT delivery layer. The fragmentation indicator indicates that MPU metadata is the first of delivery data unit. The MMT receiving entity can wait to reconstruct the MPU until the fragment indicator which indicating the last of it is detected.

The second case is that the delivery data unit is either of MPU metadata, fragment metadata or MFU. In this case, the MMT receiving entity can process the reconstruction of each type of delivery data unit in MMT delivery layer for fast processing of delivery data and managed buffer status.

#### 5.6.1.2.3 Signalling message sending procedure

Whenever an MMT receiving entity wants to present media resources in MPUs, it needs to receive the related signalling messages as soon as possible after joining a broadcast channel.

The channel reception can only be initiated after getting the related Presentation Information (PI). In order to minimize the channel switch time as short as possible, the frequency of sending the CI can be per-MPU basis.

However, there can be the time interval between consecutive MPUs in MMT broadcast, which effectively corresponds to the length of MPU playing time. Usually, signalling messages are much smaller than MPUs. Therefore, the sending time of signalling messages is trivial compared with that of the MPUs. It means when an MMT receiving entity randomly accesses the service, it is very highly probable that it will meet the MPU directly rather than signalling message. In that case, MMT sending entity can send the signalling message behind the MPU to make use of those received MPUs as shown in Figure 11.

In this case, the channel process instance is initialized as soon as possible.

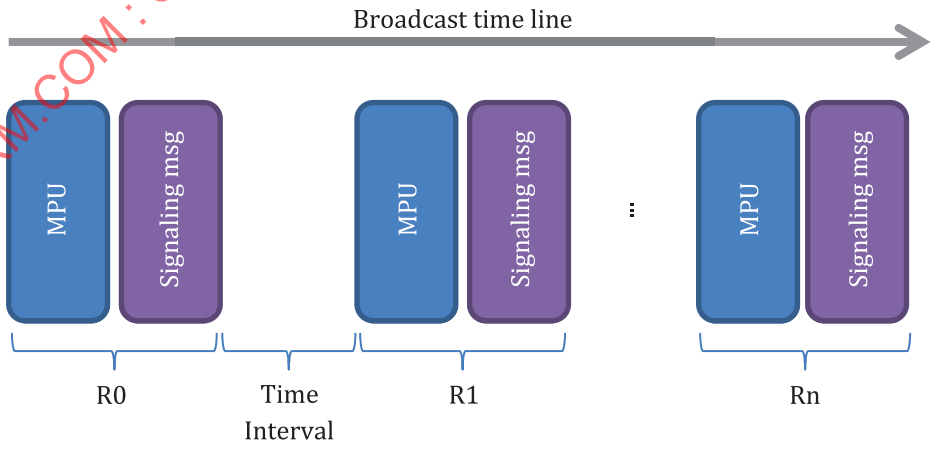


**Figure 11 — Signalling message behind the MPU**

### 5.6.2 Sending of MPU and signalling message

After channel process instance initialization, the media player should be initialized by MPU. Normally, the first completed MPU can initialize media player because the media player initialization only needs the head information of MPU.

When the MMT receiving entity randomly access to the broadcast channel, there are two cases.

The first case is that MMT receiving entity access to the channel while a certain MPU transmission is ongoing and the other case is that the MMT receiving entity access to the channel during time interval between consecutive MPUs as shown in Figure 12.

**Case 1**

In this case, normally the first MPU received is not complete MPU because the MMT receiving entity could not receive packets transmitted before it joins the channel. In order to initialize the media player, it needs a complete MPU, therefore it needs to wait for the next one whole MPU data to arrive. In that case, the expected wait time before media player initialization, ΔT, can be calculated using Formula (1):

$$\Delta T = T1 + T2 + T3 \quad (1)$$

where

T1 is the residual time to finish receiving first MPU since MMT receiving entity joins to the channel;

T2 is the time interval between two consecutive MPUs which is brought by the length of MPU playing time;

T3 is the receiving time of the next first MPU (because the signalling message sending time is very short compared with that of MPU, it may not be considered).

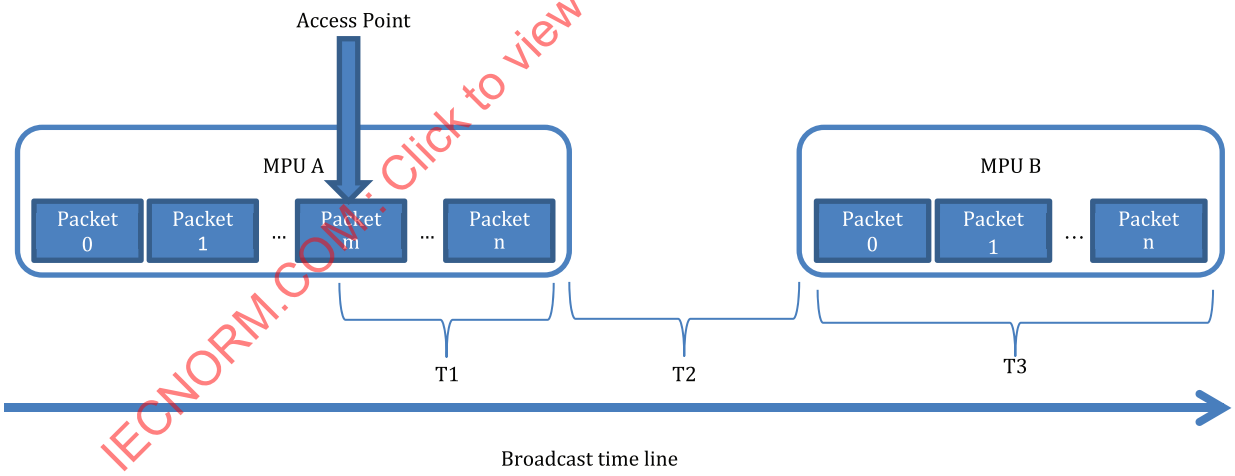


**Figure 12 — Random access in the middle of MPU (case 1)**

In this case, the MMT receiving entity can easily find a way to play the first MPU, by using the signalling message sent right after the MPU, then the wait time of T2 and T3 can be saved.

In MMT protocol, the payloads are generated according to the following description (see Figure 13):



 

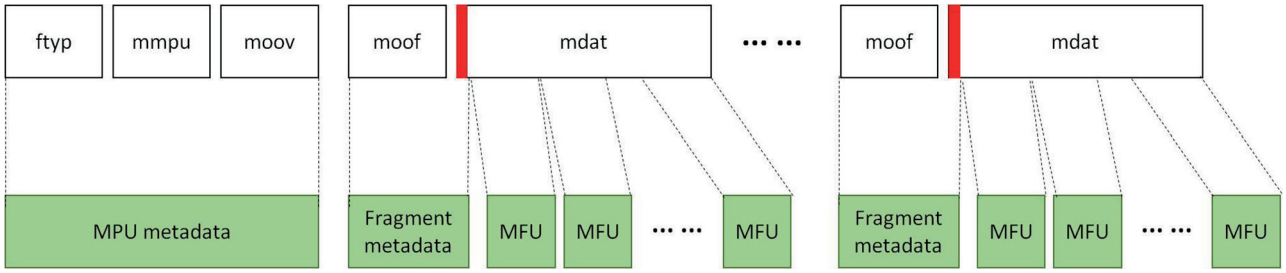


**Figure 13 — MMTP payload generation**

In this case, only one moof is included in one MPU. So MMT receiving entity can classify all packets as three categories. The first category belongs to MPU metadata. The second category belongs to fragment metadata and the third one belongs to MFU. MFU is considered as a (sub)sample of media.

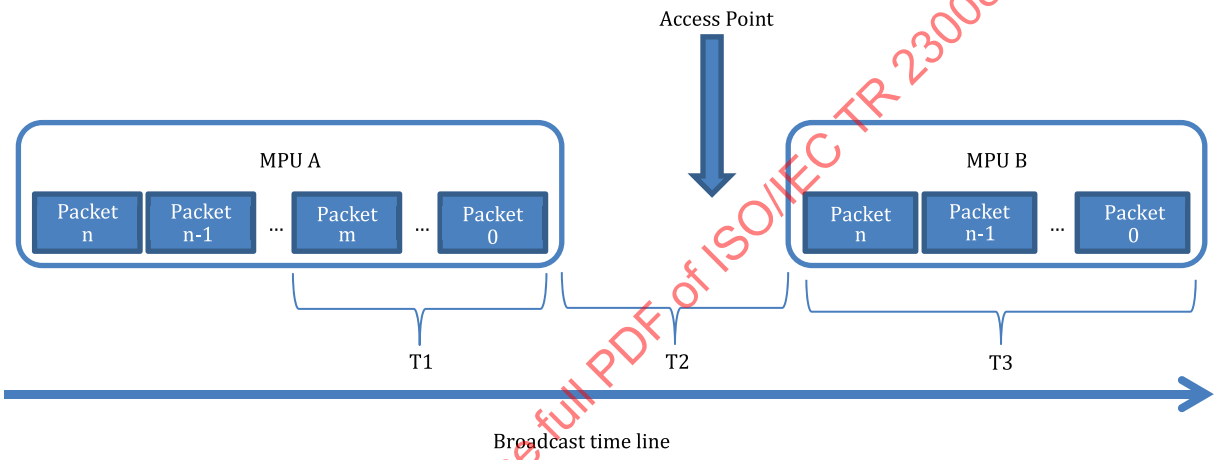


**Figure 14 — Random access in the middle of interval (case 2)**

Moreover, in this case, it can be ensured that the MPU metadata and the fragment metadata can be received by terminals with the highest probability. Then the waiting time should be as shown in Formula (2):

$$\Delta T = T1 \tag{2}$$

### Case 2

In this case, the first MPU received should be a complete MPU as shown in Figure 14. The total waiting time should be part of T2 and whole T3. It should be as shown in Formula (3):

$$\Delta T < T2 + T3 \tag{3}$$

## 5.7 Fast MMT session acquisition

In multimedia streaming environment, user switches between media which are very huge size when they are consuming the contents. In such scenarios, low-delay session acquisition has a significant impact on the perceived quality by the end user.

MMT streaming is based on the MPU concept as well as other kinds of data (e.g. media data itself and signalling messages) being fragmented and multiplexed into a sequence of MMTP packets for transmission.

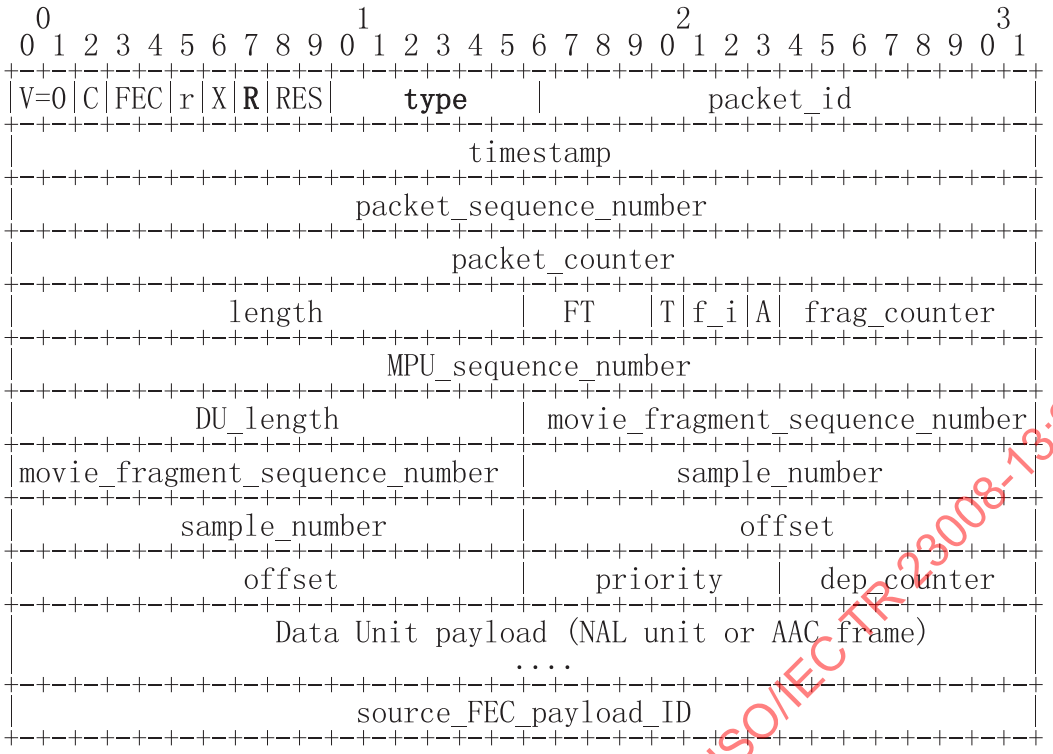


**Figure 15 — Structure of MMTP packet**

In the current MMTP packet header, the fields of *RAP_flag* (R: 1 bit) and *type* (6 bits) can be used for fast session acquisition as shown in Figure 15.

The *RAP_flag* indicates that the payload contains a random access point (RAP) to the data stream of that data type. The exact semantics of this flag are defined by the data type itself. The *RAP_flag* should be set to mark data units of MPU Fragment Type value 0 and 1 and for MFUs that contain a sync sample or a fragment thereof, in the case of timed media, and for the primary item of non-timed MPUs. RAP can be MPU metadata, signalling message or frames which contain I-frames.

The *type* indicates the type of payload data, i.e. MPU, generic object, signalling message. As one more further step when MMTP packet includes MPU data (*type*=0x00), MMT receiving entity can also find out whether it has MPU metadata by checking MPU Fragment Type (FT: 4 bits) which is in the payload header indicating the fragment type as defined in ISO/IEC 23008-1:2014, Table 4.

For fast accessing of the IDR frames after constructing MPU, MMT receiving entity can be processed to parse “moov” box in the MPU metadata which indicated the presentation time for sample and also processed to get starting time from MPU_timestamp descriptor or begin time specified in MPEG-CI.

If *RAP_flag* is set to “0”, it means the corresponding MMT packet contains no high priority random access data. Then MMT receiving entity can discard the MMTP packets without RAP_flag set to “1” for efficient buffer management before meeting the RAP_flag set to “1” or processing it depending on its policy or MMT receiving entity situation.

## 5.8 Referencing and processing non-timed data

### 5.8.1 General

MMTP provides a mode for the grouped transmission of non-timed media resources, such as resources of a website. This mode facilitates the referencing, reception and consumption of media resources that are closely related. For instance, a receiver that is consuming a website would only need to locate a single resource and receive it instead of receiving the root resource, wait for the application to parse it and detect the referenced resources and then request the MMTP receiver to receive them and make



 

them available. On the sender side, this removes the necessity of describing every single resource and its location in the signalling.

### 5.8.2 Resource grouping and referencing

The MMTP sender is instructed to send a set of related non-timed media resources with a designated primary resource/entry point. The primary resource is usually the resource that connects all other resources or that constitutes the entry point for the application. Based on this information, the MMTP sender decides to encapsulate these related resources and send them together.

The MMTP sender sets the MP table according to the following rules.

— The identifier_type of the identifier_mapping syntax element is set to 0x01.

— The URL_count field is at least equal to 1.

— The first URL in the list should be the URL of the primary resource/entry point.

— It is recommended that only the URL of the primary item is provided in the MP table for the corresponding asset.

— The asset_type is set to “mmpu”.

It then creates the MPU by setting the primary resource as the primary item of the MPU and the remaining resources as additional items. The sender should set all resource metadata, such as URL and MIME type of the resource as part of the item information box.

The presentation information is authored to reference the primary resource.

### 5.8.3 Receiver handling

Based on the information received as part of the presentation information, the receiver detects a referenced resource. It builds the URL for that resource and requests the MMTP receiver to make that resource available. The MMTP receiver checks the MP table to locate the asset that carries that resource. It finds out that the asset is of type “mmpu” and lists the URL of the requested resource. It then uses the packet_id of that asset to receive and reconstruct the MPU.

Once the MPU that contains the primary resource is reconstructed, the receiver extracts all embedded resources together with their metadata and makes them available to the application, e.g. through a web cache. The application requesting the primary resource will later find all related resources available and quickly accessible from the cache.

## 5.9 Media adaptation for quality control in MMTP

### 5.9.1 General

This subclause shows how MMT protocol can transmit media streams adaptively to environment changes, such as network congestions, while also minimizing the service quality degradation.

### 5.9.2 Parameters for media adaptation

MMT streaming is to deliver MPUs which is a media file based on ISO-based media file format (ISOBMFF). To transmit the MPU, it is fragmented and packetized into MMTP packets which have three different payload fragments, MPU metadata, fragment metadata and MFU.

Each MPU has “ftyp”, “sidx”, “mmpu”, “moov” and “moof” box in MPU metadata. Among them, “mmpu” box includes asset identifier, MPU information and *is_complete* parameter indicating whether this MPU has all MFUs described by the MFU structure or not. The “moov” box contains all codec configuration information for decoding and presentation of media data and especially MMT hint track providing the information to convert encapsulated MPU to MMTP payloads and MMTP packets.

Those MMTP hint track provides two important parameters: *priority* and *dependency_counter*. *priority* indicates the priority of the MFU relative to other MFUs within a MPU and *dependency_counter* indicates the number of MFUs whose decoding is dependent on this MFU.

These two parameters can be utilized to judge the importance of the MFUs in an MPU.

### 5.9.3 Adaptation operation of MMT entity

When MMT sending entity transmits media streams to MMT receiving entity, there may be cases that MMT sending entity can intentionally skip some portion of it. For example, MMT sending entity can know poor network condition or lack of resources in the MMT receiving entity side through some feedback. Then MMTP packets will be dropped anyway according to network condition by MANE or other network entities if there is no adaptation operation. In streaming service case, omission of some media may not affect critically to user experience.

MMT sending entity can skip transmission of some MFUs to accommodate to network condition. The MMT sending entity needs information to decide which MFUs they need to omit. In that case, *priority* and *dependency_counter* can provide the priority information between MFUs. MMT sending entity should drop MFUs with the lowest priority and highly independent from other MFUs first by comparing *priority* and *dependency_counter*. For example, MFUs including RAP information will have high priority value and MFUs corresponding I-frame will have *dependency_counter* value.

MMT sending entity also has to announce to MMT receiving entity side that some of MFUs are intentionally dropped. If that information is not provided to MMT receiving entity side, they will wait for MFUs which are intentionally omitted or try to request again. When *is_complete* is set as "0", MMT receiving entity should recognize there are more than one missing MFUs and should reconstruct MPU through current MFUs without requesting MFUs in that MPU. For that reason, MMT sending entity has to set *is_complete* as "0" to acknowledge that current MPU has some missing MFUs to MMT receiving entity side when some MFUs are omitted intentionally.

## 5.10 Hybrid delivery in MMT

### 5.10.1 General

This subclause provides MMT implementation information when MMT Assets are delivered on hybrid networks.

Hybrid delivery in this subclause is defined as simultaneous delivery of one or more content components over more than one different types of network. One example is that one media component is delivered on broadcast channels and the other media component is delivered on broadband networks. The other example is that one media component is delivered on broadband networks and the other media component is delivered on another broadband networks.

### 5.10.2 Classification of hybrid delivery

The basic concept of hybrid delivery is to combine media components on different channels. However, in practice, there are several scenarios for hybrid delivery. The classification can be classified as follows.

— Live and non-live:

  — combination of streaming components (see Figure 16);

  — combination of streaming component with pre-stored component (see Figure 17);

— presentation and decoding:

  — combination of components for synchronized presentation (see Figure 16);



 

— combination of components for synchronized decoding (see Figure 18);

— same transport schemes and different transport schemes:

— combination of MMT components;

— combination of MMT component with another-format component such as MPEG-2 TS.

NOTE The case in which another-format component is encapsulated into MMT-format is excluded.

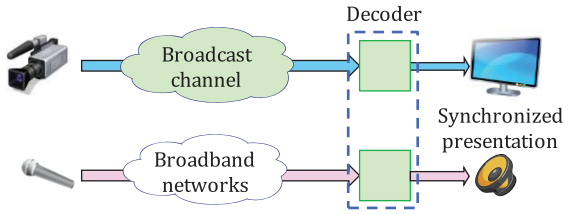


**Figure 16 — Combination of streaming components for presentation**

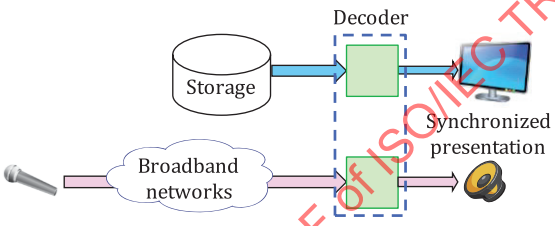


**Figure 17 — Combination of streaming component with pre-stored component for presentation**

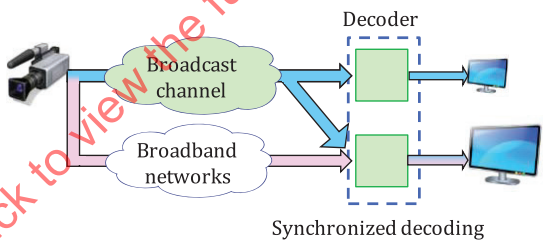


**Figure 18 — Combination of components for decoding**

The so-called seamless switching can be categorized into hybrid delivery of streaming components for presentation since switching components is possible when both components are synchronized for presentation.

### 5.10.3 Technical elements for hybrid delivery

While many types of information are specified in MMT signalling messages and MMT-PI, three types of information are mainly required for hybrid delivery:

— MMT asset information;

NOTE When an MMT component is combined with another-format component, the latter component can be neither an asset nor an MPU.

— information on spatial relationships among media components;

— signalling message for media consumption.

5.11 provides detailed information on the implementation of MMT for individual cases categorized in 5.10.2.

## 5.11 Example of detailed implementation of MMT

### 5.11.1 Use case: Combination of MMT and MPEG-2 TS for synchronized presentation

#### 5.11.1.1 General

This subclause describes the case in which MMT and MPEG-2 TS are used as a transport scheme in broadband networks and broadcast channels, respectively.

#### 5.11.1.2 MMT asset information

In order to identify the type and location of media components of MPEG-2 TS in broadcast channels, the following signalling messages are required in MMT in broadband networks:

— `MMT_general_location_information` syntax in MP table.

This syntax specifies the address of a media component of MPEG-2 TS in broadcast channels. In the case of combination with MPEG-2 TS in broadcast channels, MPEG-2 TS location is used.

A typical example is to identify `network_id` (16 bits assigned by SDO), MPEG-2 `transport_stream_id` (16 bits assigned by the operator) and MPEG-2 `PID` (13 bits assigned by the operator). By using these IDs, a media component in broadcast channels can be identified.

#### 5.11.1.3 Information on temporal relationships among media components

MPEG-2 TS components have timestamps based on STC. MMT components have timestamps based on UTC. To synchronize these different types of timestamps in MMT and MPEG-2 TS, Clock Relation Information messages are required.

— Clock Relation Information message can carry a set of `STC_sample` and `NTP_timestamp_sample` that are identical timing.

At an MMT compliant receiving entity, the STC based clock in MPEG-2 TS can be converted to the wall clock based on UTC by processing the Clock Relation Information. An MMT component in broadband networks and an MPEG-2 TS component in broadcast channels are presented in synchronized manner since both components can share the same time domain as wall clock.

### 5.11.2 Use case: True real-time video streaming over lossy channel

#### 5.11.2.1 General

This subclause describes the case in which the client starts replaying video within 150 ms after requesting a video to the server[7]. It is assumed that de-jittering delay is zero.

#### 5.11.2.2 Main features

**MFU-by-MFU processing**: In the server and client, video data is processed MFU by MFU rather than MPU by MPU. An MPU is not reconstructed in the client. An MFU is replayed before receiving its MPU fully. In the server, video data is packetized and sent as soon as being parsed as part of MPU. At the same time, the sent packets are stored in the FEC encoder buffer. There is no bi-directional frame, i.e. only intra- and predictive frames.

**Short FEC period**: FEC period is independent of borders of MPU and MFU. It is much shorter than an MPU period. Number of data symbols, k, is fixed (k = 49 in the implementation) whenever the number is reached, repair packets are calculated and sent [10 repair packets, (n, k) = (59, 49) in the



 

implementation]. When symbol size T is 800 bytes and packet size is 1 500 bytes, repair packets are sent in every 24 to 25 packets, which is about 100 ms long on average for 3 Mbps video [(49 symbols/FEC) × (1 500 × 8 bits/packet)/(2 symbols/packet)/3 Mbps].

**Centralized timing control**: In the server and client, timing is controlled by the session object.

In every MFU period, the client session object orders to retrieve an MFU from FEC buffer and decode and replay it. The client session object orders to flush the receiver socket buffer to the FEC ring buffer from time to time [in every MFU period (= 33 ms) in the implementation].

#### 5.11.2.3 Ring-buffer in the client

The ring buffer is composed of N FEC associate groups. The number N is determined in terms of de-jittering delay plus some margin in order to prevent buffer overflow. Here, "buffer overflow" means over-writing lately arrived packets on the packets which are not read by the FEC decoder yet [N = 50, in the implementation; N = 50 allows about 5 s de-jittering delay and total buffer size becomes 2,2 Mbytes = 3 Mbps/(8 bits/byte) × 5 s × (59/49)].

#### 5.11.2.4 FEC/deFEC performance and delay

Code rate is 49/59, which is enough to recover up to 15 % packet loss. The code rate could be adaptive to channel condition. Mostly, deFEC is successful at most, k+2 symbols. Theoretical failure rate at k+2 is $10^{-4}$.

Since data packets are sent immediately after being stored in the FEC encoder buffer, total delay FEC encoding and decoding is one associate group period (about 100 ms in the implementation).

Complexity of FEC encoding is very low. Because unlikely as Raptor Q, all the coefficients are stored in the coefficient table and calculation in the server is just to calculate weighted sum of symbols [that is, a repair packet is calculated by just GF(256) multiplication of a T by k array and k by 1 column vector]. Complexity of FEC decoding is also very low since systematic FEC has been used. If L symbols are lost, then, total calculation includes one L by L inverse matrix calculation and GF(256) multiplication of an L by L array and an L by T array (T = 800, k = 49, L < 9, in the implementation).

### 5.12 HRBM signalling for hybrid delivery

#### 5.12.1 Hybrid delivery from the single MMT sending entity

As shown in Figure 19, the main station is composed of the broadcast tower and at least one of the broadcast services are provided to any broadcasting service using the hybrid delivery.

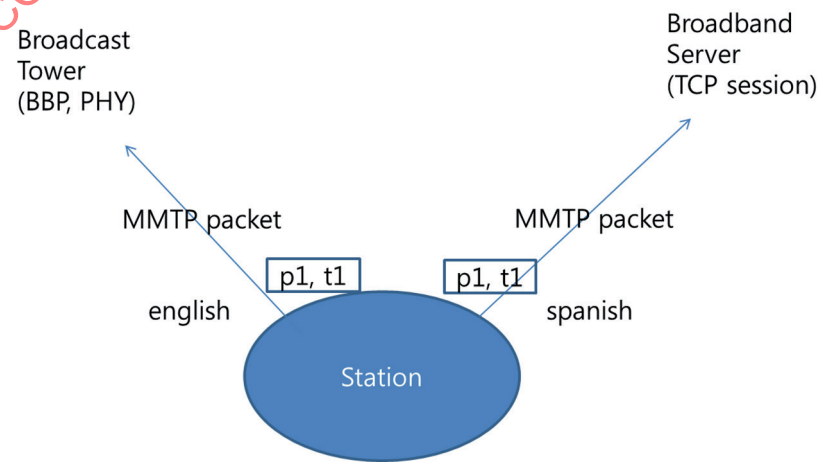


**Figure 19 — Single station for hybrid delivery**

The consideration of broadcast tower and broadband server is transmitter of MMTP packets to distribute to the multiple MMT receivers.

The station can send the MMTP packets through the broadcast tower that contain the English audio streams for major broadcasting channel and can also send the MMTP packets with Spanish audio streams for secondary audio for the main channel via broadband channel. In this case, the MMT receiver, the broadcasting tower and the broadcast server, through the news program about the English and Spanish audio by selecting the alternative, may be received.

In the case above, those MMTP packets are composed of each MPU that has same presentation time for rendering and sending the same time to leave from the station with same timestamp of MMTP packet header. Those MMT sending entities are synchronized based on UTC.

In MMT, the MMT receiving entity consists of the multiple HRBM buffers as shown in Figure 20.

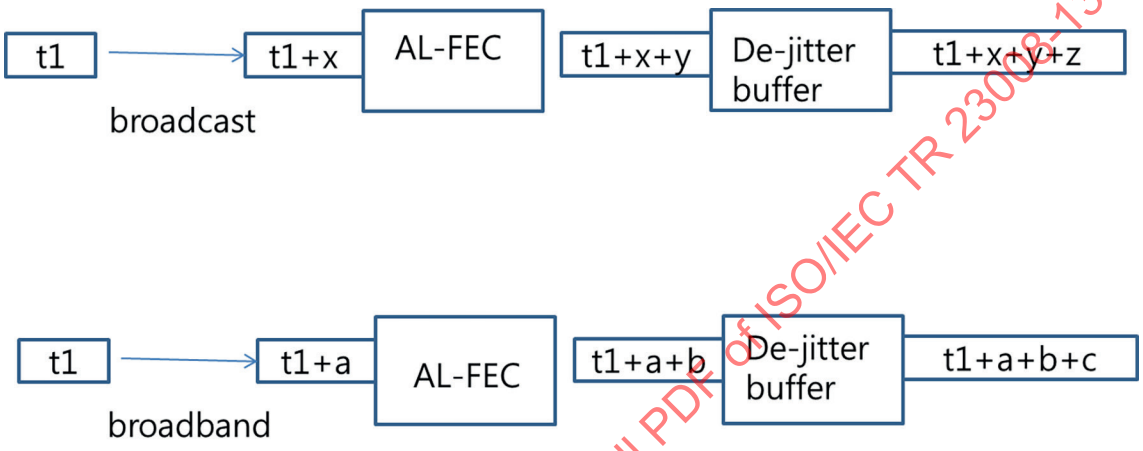


**Figure 20 — HRBM buffers**

Each HRBM has AL-FEC decoding buffer and de-jitter buffer. The AL-FEC decoding buffer is used for recovering from the loss of MMTP packets in the network and de-jittering buffer removes the jitter of MMTP packets from the network. The goal of HRMB structure is to provide the constant delay of the delivery system of the MMT receiving entity for synchronized play-out of the media data.

However, the MMTP packets lose and jitter of the MMTP packets depends on their delivery channel conditions. In this example, x is the delay of the network, y is the time for AL-FEC processing and z is the time that the MMTP packet remains in the de-jittering buffer in broadcasting channel. The broadband channel also has lost their MMTP packet, different arrival time of MMTP packets and corresponding to de-jittering buffer time in broadband channel.

For synchronized play-out of the media in hybrid delivery, the value of t1 + x + y + z need to be equal to t1 + a + b + c.

The MMT Asset corresponding to the content set by each component, the play out delay (D), is determined by the value of the MMT sending entity via a signalling message to the MMT receiver. For example, the value of play out delay is transmission network delay incurred by the characteristics of the maximum value and AL-FEC protection window time and may be calculated using Formula (4):

$$D = \max(x1, x2, x3, \ldots) + \text{AL-FEC protection window time} \quad (4)$$

Here, x1, x2, x3 are the nature of the transmission network delay including the propagation delay specified from max_transmission_delay in HRBM message. AL-FEC protection window time has occured when the AL-FEC decoding processing is performed on the window interval (i.e. AL-FEC protection window time of AL-FEC encoding and decoding unit performs the FEC packets corresponding to the block is the first of the packet including the transmission time of the packet being sent and the FEC



 

packets of a packet block containing). Finally, a transmitted MMTP packet including the AL-FEC and transmission delay time is defined as the maximum value.

The "fixed end-to-end delay" is specified the maximum value in HRBM signalling message for the hybrid delivery network via multiple delivery paths to the MMT receiver for MMT receiving packets from the MMT sending entity within the same service corresponds to the MMT receiver's output timing.

### 5.12.2 Hybrid delivery from the multiple MMT sending entities

Figure 21 illustrates the multiple stations used for the MMT services. Station 1 is composed of the broadcast tower and station 2 has at least one of the broadcast servers provided to any broadcasting service using the hybrid delivery.

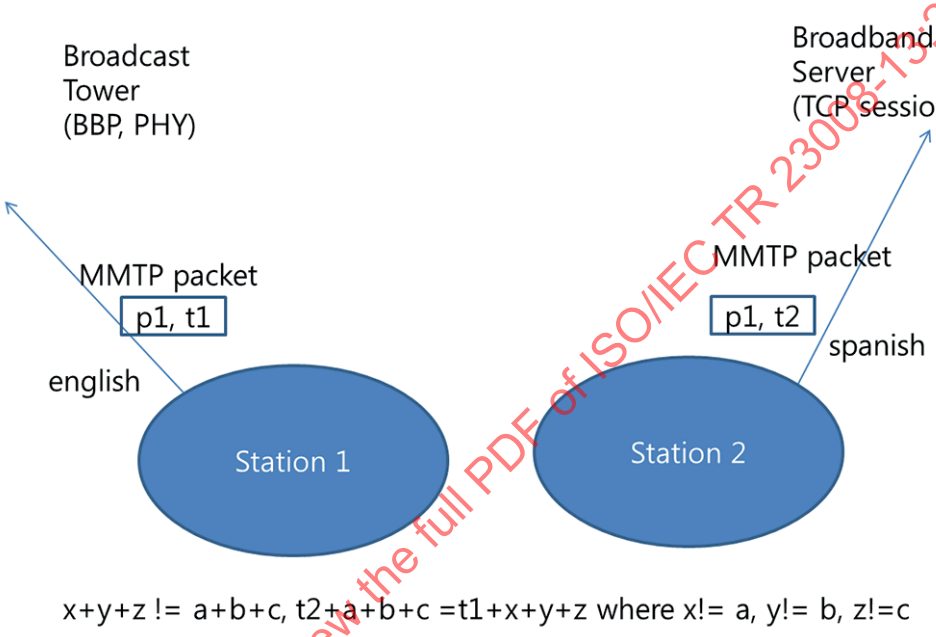


**Figure 21 — Multiple stations for hybrid delivery**

This is also considered the broadcast tower. The broadband server is the transmitter of MMTP packets to distribute to the multiple MMT receivers.

In this case, the presentation time of the MMTP packet is the same, but the sending of the MMTP packet is different.

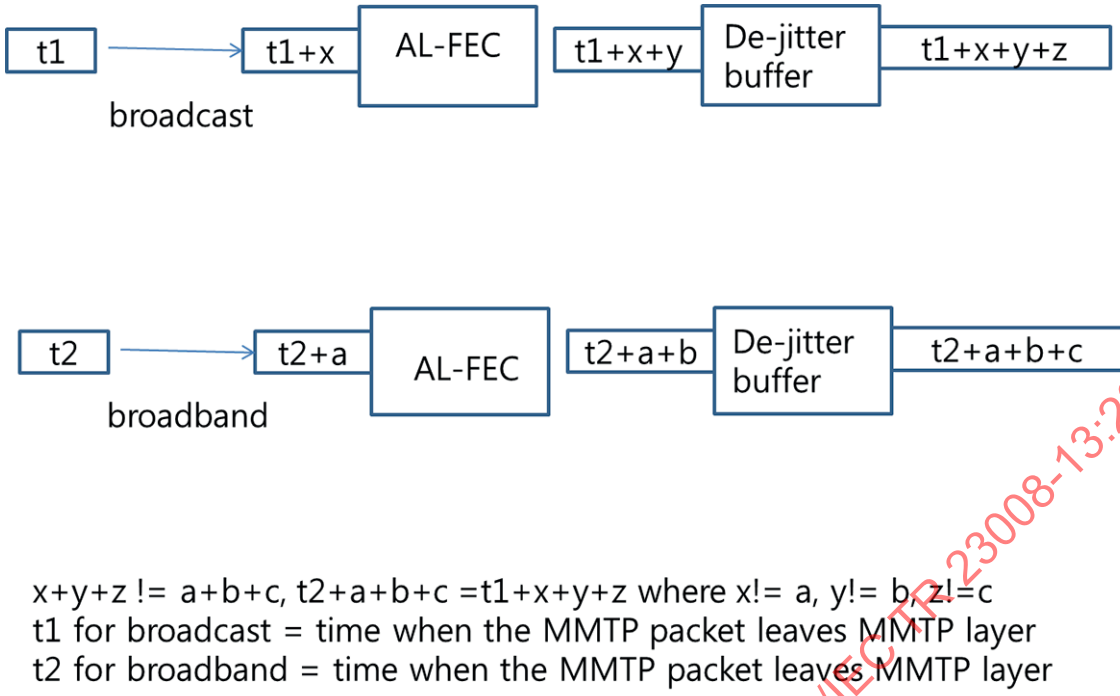


**Figure 22 — Multiple HRBM buffers**

The "fixed_end_to_end_delay" is predefined the time to consider in delay including propagation delay time between the MMT sending entity and the MMT receiving entity. The "fixed_end_to_end_delay" is calculated by the summation of the "max_transmission_delay" and "FEC_protection_window_time".

The HRBM message as described above is based on the structure of the first use case in this contribution according to the maximum transmission delay which is a defined value for each of the MMT receiving entity to de-jitter buffer. The play-out of the time can be adjusted. See Formula (5):

T_de_jitter_out_time = ts + alpha (5)

Here, ts is the timestamp of the received MMTP packet in MMT receiver. Wherein the timestamp can be calculated by the MMTP packet sending timing for transmission delay. Then, "fixed_end_to_end_delay" can be defined as a value in HRBM message. Based on ts and alpha, which is calculated by max_transmission_delay corresponding to network B, in this case, alpha is corresponding to the fixed_end_to_end_delay. MMT receiving entity can control their playout time for synchronization.

In the above example, the delay of transmission A is relatively small than B. The MMT sending entity sends to the initial setting in broadcasting channel and MMT receiving entity can analyse that the network transmission delays and jitter in the transmission network B is larger than A. Here, the "max_transmission_delay" as specified in HRBM message by the service provider corresponds to this transmission characteristic of B in the hybrid delivery network. The HRBM message may be sent periodically or if sent by component content, may be transmitted by a specific event.

Consequently, de-jitter buffer of A and the de-jitter buffer of B, respectively, according to the fixed end-to-end delay specified in HRBM signalling message need to specified with max_transmission_delay can be controlled the play out time which has the same value between the final output (P: 't1 + x + y + z) of de-jitter buffer of A, and the final output (Q: 't2 + a + b + c) of de-jitter buffer of B. See Figure 22.

## 5.13 Error resilience in MMT protocol

MMTP is optimized for the delivery of MPUs, which are ISOBMFF files. The delivery of the MPU is performed movie fragment by movie fragment, thus enabling fast start-up delays and fast access to the content. MMTP defines three different payload fragments for error resilience purpose.

— The MPU metadata: This information contains the metadata of the ISOBMFF file and the MPU. The MPU metadata thus contains all codec configuration information and is crucial for the consumption of the whole MPU. The MPU mode allows to mark the packets of the MPU metadata (usually only one or a few packets), so that the client can clearly identify them and recognize if it has received it correctly. To provide for random access and enhance the probability of receiving the MPU metadata, the sender should send the metadata repeatedly and periodically throughout the transmission time of that MPU.

— The fragment metadata: This information contains the "moof" box and the skeleton of the "mdat" box. This metadata provides information about the sample sizes and their timing and duration. This information is important for all media samples of the current fragment. However, it may be possible to recover from loss of fragment metadata and it is also possible to send it out of order. The sender may deliver the fragment metadata repeatedly and interleaved with the packets that contain the media samples of that fragment to increase the probability of correct reception and to enable random access inside a movie fragment.

— The MFU: An MFU contains a media unit from a sample of a particular movie fragment. The MFU also provides enough information such as the sample number, the fragment sequence number and the position inside the media sample to position the media unit on the timeline and inside the "mdat" box. It may also contain information about the importance of that media unit for the decoding process. Based on that information, the sender, as well as intermediate MMT entities, may undertake appropriate steps to enhance error resilience respective to the priority and importance of the media unit. A media unit from a SAP is, for instance, more important than a media unit for which there are no dependencies.

One of MMTP's advantages is its ability to enable error robustness at the receiver side by enabling the client to recover from packet losses and still generate a compliant MPU when needed.

When the MPU metadata is lost, the client should keep any correctly received data from that MPU until a new copy of the MPU metadata is correctly received.

When a fragment metadata is lost, the client should use the information from previous fragments about the sample durations to correctly reconstruct the lost "moof" box. It uses information from the received MFUs to recover the movie fragment segment number. The offsets of the media data may be recovered later using the start of a fragment as the baseline and the sample number and MFU sizes to reconstruct the "mdat" box as well.

When an MFU is lost, the loss can be discovered at the receiver based on the gap in the sequence numbers. The missing MFU is replaced by a null value array in the "mdat" box, if at least one MFU of the same media sample has been received correctly. If the complete sample is lost, the space occupied by that media sample may be removed completely and the information in the containing chunk, the "trun", should be edited appropriately to adjust the sample duration of the previous sample and the sample offsets of the following samples. Figure 23 shows an example of such processing.

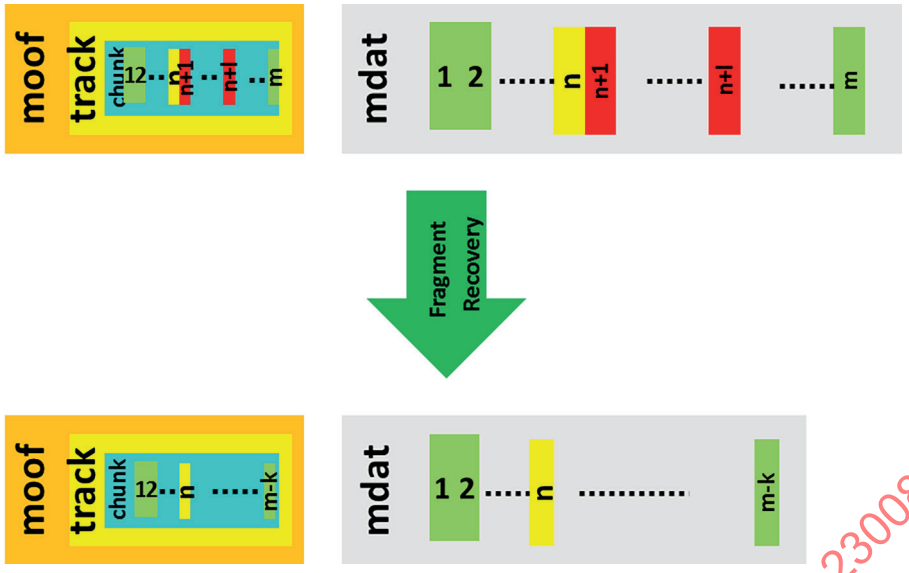


**Figure 23 — Fragment recovery after packet loss**

## 5.14 Delay-constrained ARQ

### 5.14.1 General

This subclause provides methods on how to recover lost packets under delay-constrained environment through ARQ. Delay constraints can be considered both in delivery-time basis and arrival-deadline basis.

### 5.14.2 Delivery-time constrained ARQ

MMT employs ARQ mechanism as an error control technique for recovery of packets which is lost during MMTP packet delivery. When MMT receiving entity detects lost MMTP packets, it can request retransmission of those packets.

In timed media application cases, retransmitted packets which cannot arrive before play-out of media will be useless because they cannot be utilized for timely media recovery. The *arrival_deadline* is suggested as the maximum tolerable latency for the requested packets in AF message. However, *fixed_end_to_end_delay* in a HRBM message which the MMT sending entity already knows can be used by the server to decide whether they will retransmit requested packets to MMT receiving entity or not.

HRBM is used to ensure effective MMT operation under a fixed end-to-end delay and limit memory requirements for buffering of incoming MMTP packets. Especially, in de-jitter buffering phase of HRBM, MMT receiving entity should remove MMTP packets that experience a transmission delay larger than the *fixed_end_to_end_delay* or *max_transmission_delay*. It means MMTP packets will be discarded before it is passed to decapsulation phase into MPU or MFU, if it experiences more than that delay.

When MMT sending entity receives AF message requesting the lost packets, it should decide whether it will retransmit the requested packets or not based on *fixed_end_to_end_delay* in HRBM. To do that, it can use the most updated *propagation_delay* value included in the AF message and calculate the expected delivery time of the MMT packets from MMT receiving entity to MMT sending entity by subtracting `timestamp` in packet header from the NTP time at arrival instant of the AF message.

When retransmitted packets are expected to arrive at MMT receiving entity eventually within *fixed_end_to_end_delay* since it is the very first original transmission, it will decide to transmit the requested packet, otherwise, it will give up to retransmit it.

### 5.14.3 Arrival-deadline constrained ARQ

#### 5.14.3.1 Basic operation of the arrival-deadline constrained ARQ

Figure 24 shows the basic operation of the arrival-deadline constrained ARQ. In Figure 24, arrival deadline denotes the maximum tolerable latency for the requested retransmission packet to arrive at the receiver. If the retransmitted packet arrives later than arrival deadline, even if the packet arrives in tact, it is regarded as useless and discarded.

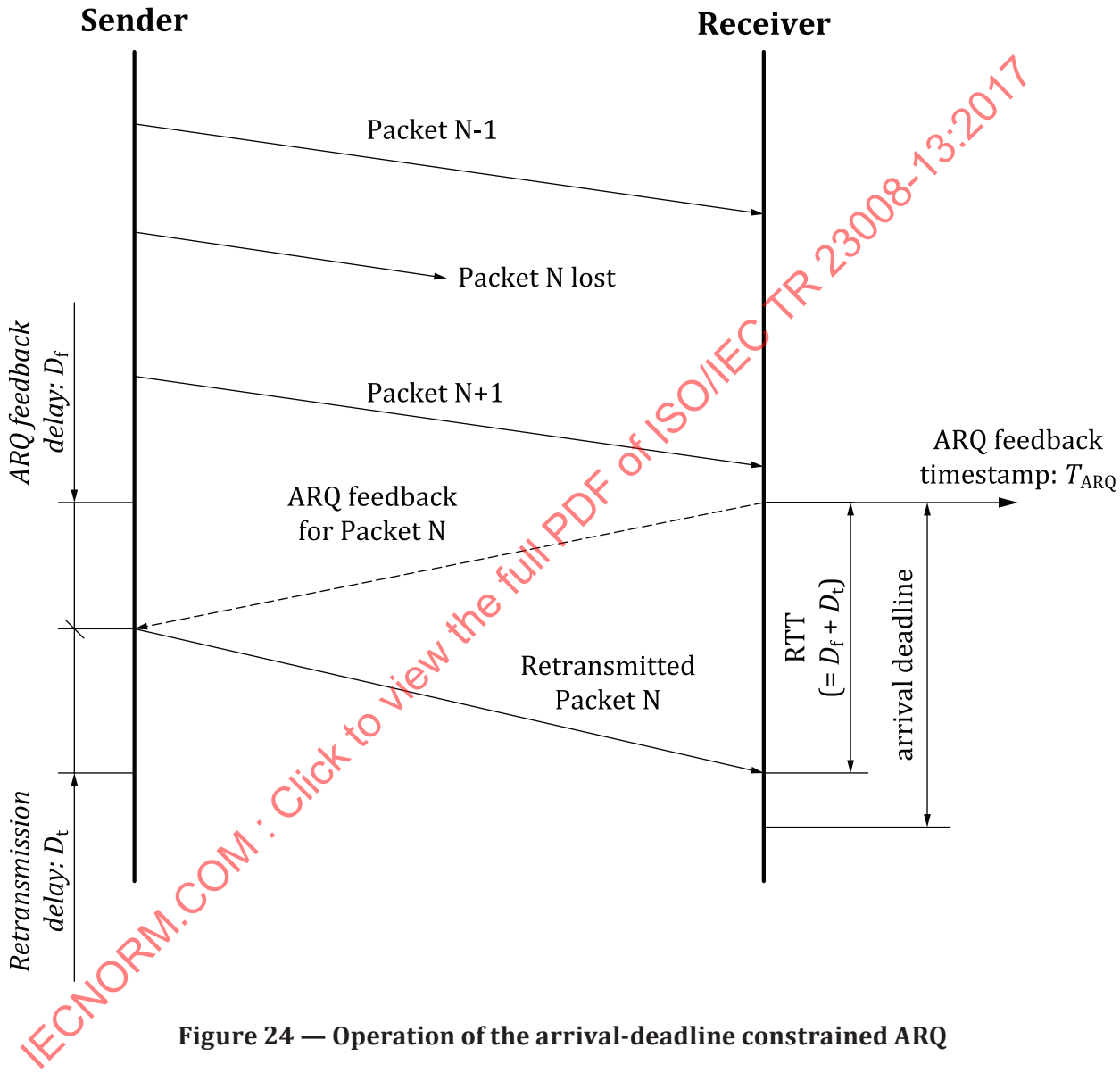




**Figure 24 — Operation of the arrival-deadline constrained ARQ**

Based on the above analysis, when the receiver detects the loss of packet, the receiver sends ARQ feedback to the sender and then the sender performs the following arrival-deadline constrained retransmission decision:

**If (RTT < arrival_deadline)**
**Sender retransmits the requested lost packet to the receiver;**
**Else**
**Sender decides not to transmit the requested lost packet to the receiver.**

In Figure 24, the round-trip time (RTT) can be obtained a priori at the sender side by using the reception quality feedback (RQF) message. For more accurate estimation of the RTT, up-to-date ARQ feedback

delay ($D_f$) value can be informed to the sender by enclosing ARQ feedback timestamp ($T_{ARQ}$) into the ARQ feedback message. Using the ARQ feedback timestamp, $T_{ARQ}$, the sender can obtain up-to-date ARQ feedback delay ($D_f$). And this updated $D_f$ value can be used to compute up-to-date RTT.

#### 5.14.3.2 Example of calculating arrival-deadline

There can be various ways to calculate arrival deadline depending on the service scenarios and preference of the service provider. This subclause shows one way to do that. The *arrival_deadline* can be obtained at the receiver side by considering the remaining amount of safely arrived packets in the receiver buffer when the ARQ feedback message is prepared for sending. Figure 25 shows an example of calculating the *arrival_deadline* value at the receiver.

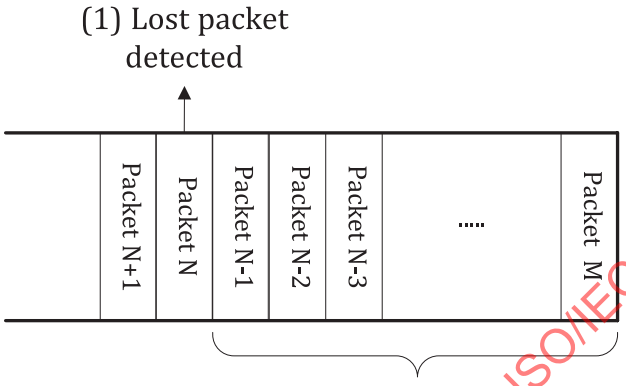


**Figure 25 — Example of calculating *arrival_deadline* at the receiver**

After detecting packet loss, the amount of data safely stored in the receiver buffer, $D_S$, is obtained. If some packets before the current lost packet N were also lost, $D_S$ can be calculated considering the successive packets that are safely received prior to and following the previous lost packet from an affirmative point of view. Then, by using the average bitrate of the MMTP packet stream, arrival deadline for the lost packet N, $T_a$, is calculated using Formula (6):

$$T_a = D_S/B_{avg}(s) \tag{6}$$

In Figure 25, with the available amount of packets in the buffer, the client can continue the presentation roughly for the duration of $T_a$ (arrival-deadline). The requested retransmission packet should arrive until the arrival-deadline, $T_a$.

## 5.15 Application layer forward error correction (AL-FEC)

### 5.15.1 FEC decoding method for ssbg_mode2

#### 5.15.1.1 General

This subclause provides recommendations for the FEC decoding method when MMT employs ssbg_mode2 as source symbol block format. Depending on FEC encoding scheme, FEC decoding algorithm can be decided. However, this implementation guideline does not cover a specific FEC decoding algorithm, but deals with only the method to choose a proper unit of data for FEC decoding. This subclause provides some guidelines of FEC decoding unit and a method to choose a proper unit.

#### 5.15.1.2 Source symbol block format for ssbg_mode2

In ssbg_mode2, source symbol block (SSB) usually consists of MMTP packets of variable sizes. Figure 26 presents an example of SSB for ssbg_mode2 which is built of six MMTP packets having distinct sizes. More precisely, the six MMTP packets and some padding data (e.g. all 00h) have been placed into the SSB. Note that any MMT packet should be started at the first byte of a symbol element in SSB. The role of padding data may be regarded as adjusting the start point of MMTP packets.

The columns of SSB in Figure 26 correspond to the source symbols of size $T$ (bytes) which is composed of $N$(=4) symbol elements of size $T/N$. In other words, the SSB consists of $K$(=13) source symbols of size $T$, i.e. $K \times N$(=52) symbol elements of size $T/N$(=$T$/4). Furthermore, an SSB can be divided into $N$ regions which consist of $K$ symbol elements, respectively, such as Regions-1, -2, -3 and -4 in Figure 26. The concept of regions in an SSB will be used for recommended FEC decoding method later.

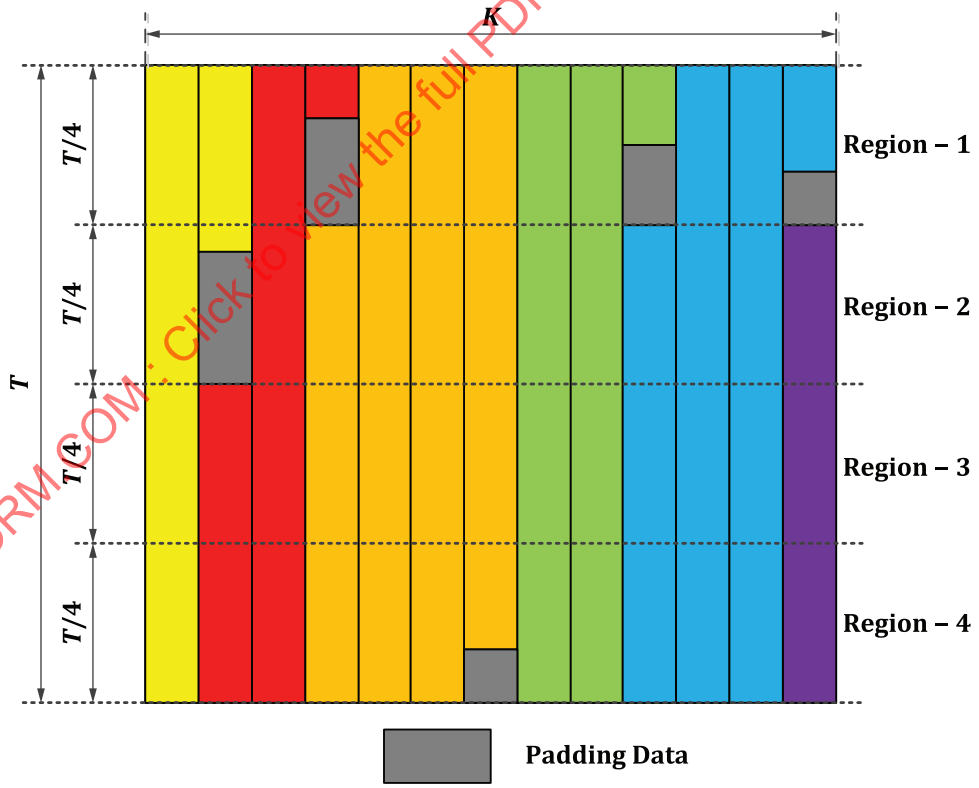


**Figure 26 — Example of source symbol block**

#### 5.15.1.3 Regionalization of source symbol block for FEC decoding

First, assume that the second and fifth MMTP packets in Figure 26 are lost, i.e. two MMTP packets are not received in the MMT receiving entity side. When an MMTP packet is lost, the MMT receiving entity

cannot acquire its boundary information in the SSB since its source FEC payload ID and size information are also lost. In other words, the MMT receiving entity cannot acquire the information on the start and end positions of MMTP packet and the amount of padding data, and so on. Therefore, the MMT receiving entity can rebuild SSB as depicted in Figure 27. Note that source FEC payload ID provides information related to the start position of the MMTP packet in SSB in terms of the symbol element for ssbg_mode2. For example, the start position of the second and fifth MMTP packets in Figure 26 in terms of symbol elements may be 6 and 37. After rebuilding the SSB from received MMTP packets, the MMT receiving entity carries out the FEC decoding process to recover the lost MMTP packets.

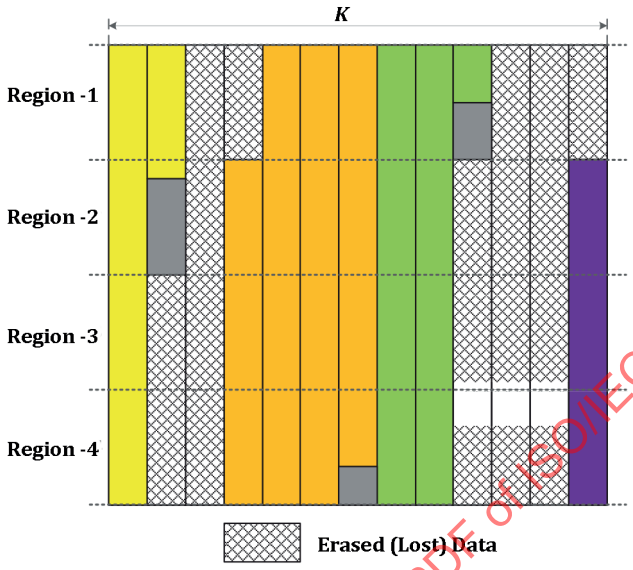


**Figure 27 — SSB rebuilt when two MMTP packets are lost**

The unit of data used during the decoding process can be changed according to decoding requirements, e.g. the decoding complexity, latency and the performance of erasure recovery, etc.

For the first example, Figure 28 presents the FEC decoding method based on source symbol unit. To carry out the decoding process based on source symbol, SSB in Figure 28 should be interpreted into the SSB in Figure 28. It is easily checked that any source symbol including lost MMTP packet is regarded as a lost source symbol. Consequently, the rebuilt SSB has seven lost source symbols, and therefore, at least seven repair symbols are required to recover the lost MMTP packets perfectly.

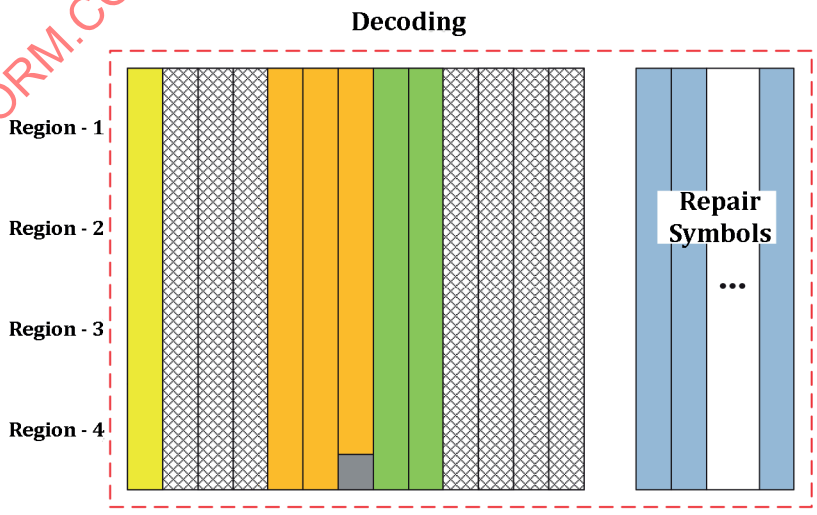


**Figure 28 — FEC decoding based on source symbol unit**

The advantage of FEC decoding based on source symbol unit is low-complexity decoding due to one erasure pattern during the decoding for the given SSB. More precisely, the preprocessing (e.g. Gaussian elimination to form a decoding schedule) is required only once and the subsequent process is related to simple repeated computations.

For the second example, Figure 29 presents the FEC decoding method based on two-symbol element unit. Here, two-symbol element means a virtual unit for two-symbol elements bonded in each divided region. To carry out the FEC decoding based on multiple symbol elements, a regionalization step is required. After rebuilding the SSB from received MMTP packets in Figure 26, the SSB should be divided into two regions as depicted in Figure 29. One region consists of Regions-1 and -2, and the other consists of Regions-3 and -4. Next, any two-symbol element including lost MMTP packet is regarded as a lost two-symbol element.

Consequently, one region and the other of SSB in Figure 29 have six and five lost two-symbol elements, respectively. Therefore, at least six repair symbols are required to recover the lost MMTP packets perfectly. Finally, FEC decoding is carried out with a proper amount of repair symbols for each region. Note that repair symbols also should be transformed into repair two-symbol elements for FEC decoding.

In general, the erasure patterns of two regions are different in case of FEC decoding based on multiple symbol elements unit. Therefore, the preprocessing for FEC decoding (e.g. Gaussian elimination) should be applied to each divided region in SSB, i.e. two distinct FEC decoding processes are carried out. This causes an increase of decoding complexity compared with FEC decoding based on source symbol, while the performance of erasure recovery can be improved for the given repair symbols since the number of erasures is reduced. Note the number of erasures and their positions for the first and second examples are different.

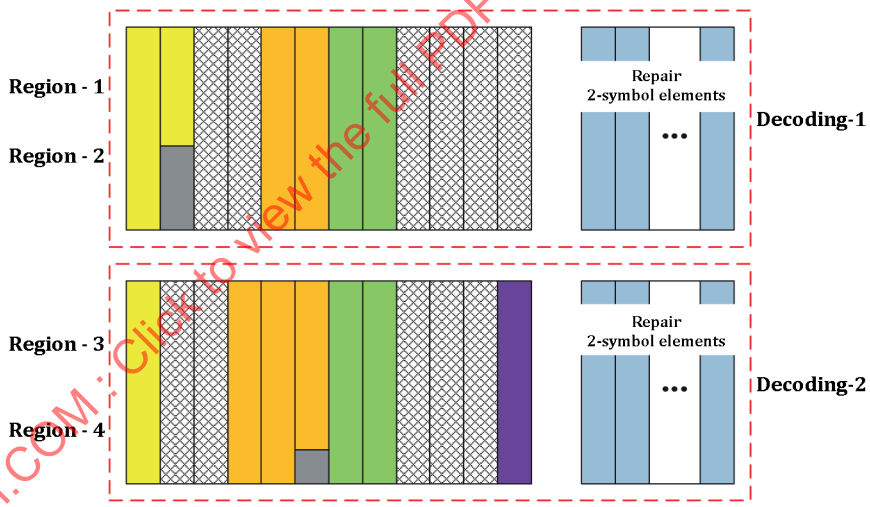


**Figure 29 — FEC decoding based on multiple symbol elements unit**



For the third example, Figure 30 presents the FEC decoding based on symbol element unit. In this case, the SSB should be divided into four regions as depicted in Figure 30. The four regions are the same as Regions-1, -2, -3 and -4 in Figure 30. Furthermore, each region has five, four, five and five lost symbol elements, respectively. Therefore, at least five repair symbols are required to recover the lost MMTP packets perfectly. Finally, FEC decoding is carried out with a proper amount of repair symbols for each region. Note that repair symbols also should be transformed into repair symbol elements for FEC decoding.

In general, the erasure pattern of each region is different in case of FEC decoding based on symbol elements unit. Therefore, the preprocessing for FEC decoding (e.g. Gaussian elimination) should be applied to each region in the SSB, i.e. four distinct FEC decoding processes are carried out. This causes an increase of decoding complexity compared with FEC decoding based on two-symbol element while the performance of erasure recovery can be improved for the given repair symbols since the number

of erasures is reduced. Note the number of erasures and their positions for the first, second and third examples are different.

The unit of data used during the decoding process is related to the decoding complexity and the performance, i.e. there is a trade-off between them. The smaller FEC decoding unit induces the larger decoding complexity, while its performance of erasure recovery becomes better. Therefore, it is important to choose a proper unit of data for FEC decoding according to system requirements.

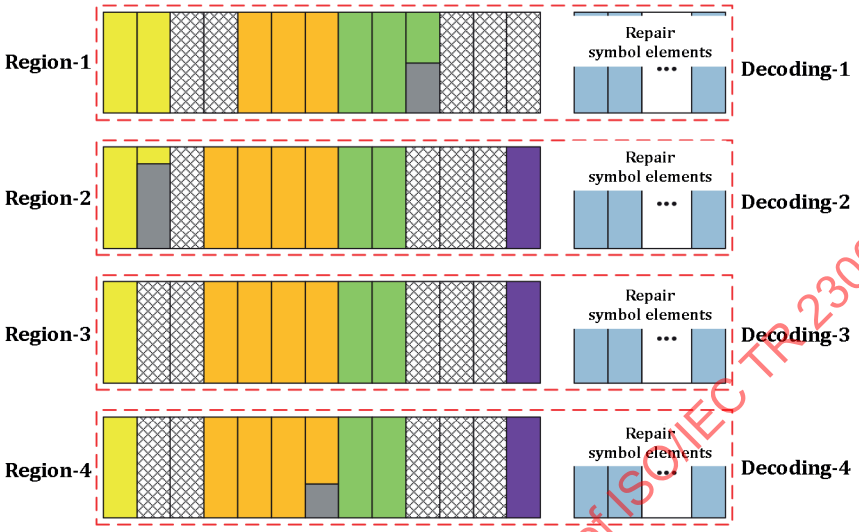


**Figure 30 — FEC decoding based on symbol element unit**

### 5.15.1.4 How to choose a proper unit of data for FEC decoding

As previously described, the number of erasures depends on the unit of data for FEC decoding as depicted in Figure 31. It is clear that the smaller the unit is, the less erasures are induced. On the other hand, the larger unit is, the smaller decoding complexity is induced. Therefore, it is recommended to choose the symbol element for the best FEC performance and choose the source symbol for the smallest decoding complexity as the FEC decoding unit.

However, if there is not much difference for the number of erasures among the FEC decoding units, it is better for FEC decoder to choose the large unit as possible since the effect of decreasing complexity is more dominant than that of degrading the performance, i.e. the performance degradation may not be critical. At this point, the number of erasures for each unit can be a measure to choose a proper unit of data for FEC decoding. More precisely, after counting erasures for each FEC decoding unit by several counters, compare their values and determine a proper FEC decoding unit based on a predetermined selection rule. For example, if the difference between the numbers of erasures for two decoding units is larger than a predetermined threshold value, the FEC decoder chooses a smaller unit, otherwise, a larger unit.

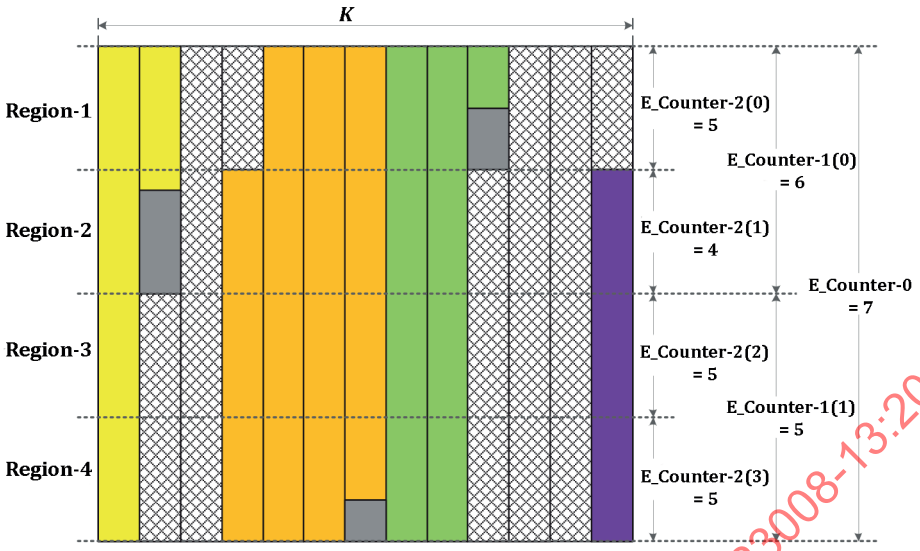


**Figure 31 — Example of counting erasures for each FEC decoding unit**

### 5.15.2 Usage of two-stage FEC coding structure

#### 5.15.2.1 General

For error resilience timed and non-timed data delivery service, MMT AL-FEC scheme based on block (N, K) code is applied. For given code rate CR (= K/N), FEC recovery performance on application layer is mainly dependent on loss rate, loss model and source packet block length K. For given packet loss rate and on given loss model, the greater K it is, the lower overhead it requires for target FEC recovery performance while the longer delay it introduces and the more buffer memory it requires. However, the smaller K it is, the higher FEC overhead it requires while the less delay (low-delay service) it is achievable and the less buffer memory it requires.

Usually, Asset for timed data requires low delay under reasonable FEC recovery performance and Asset for non-timed data does not allow any loss, and these delivery characteristics about required QoS for delivery of Assets are described in MMT-ADC. For this, Asset for timed data is delivered and protected with relatively smaller encoding symbol block to support low delay and Asset for non-timed data is delivered and protected with relatively greater encoding symbol block to get higher FEC recovery performance and lower FEC overhead. Therefore, Case 2 of two-stage FEC coding structure is used for delivery service of hybrid contents which requires two different QoSs such as AV and file data.

On the other hand, when Asset for timed data such as AV streaming delivery service is multicast (or broadcast), some end-users (User Group A) of the multicast (or broadcast) group can be under relatively good channel condition (e.g. 1 % packet loss or random packet loss) and the others (User Group B) can be under relatively bad channel condition (e.g. 10 % packet loss or burst packet loss). For this, it is preferable the Asset to be delivered and protected with relatively smaller encoding symbol block to provide low delay service to User Group A and with relatively greater encoding block to provide reasonable FEC recovery performance to User Group B. Therefore, Case 2 of two-stage FEC coding structure is used for streaming multicasting (or broadcasting) service of Asset for timed-data.

The MMT HRBM is applied for each Asset, i.e. the MMTP packets having the same packet_id. For two-stage coding structure, the MMT HRBM is extended to multiple Assets which are protected as an FEC source flow. The HRBM messages for all Assets in an FEC source flow should be determined by considering whether FEC 2 decoding is applied to each Asset or not.

Figure 32 depicts HRBM block diagram for two-stage FEC coding structure. MMTP packets (FEC source or repair packets) for both Asset A and Asset B are passed to both FEC 1 and FEC 2 decoding buffer. After FEC 1 decoding, MMTP packets for Asset A are passed to de-jitter buffer for Asset A and the recovered

source symbols are passed to FEC 2 decoding buffer. Then, if MMTP packets for Asset B are not still recovered yet, MMTP packets for Asset B after FEC 2 decoding are passed to de-jitter buffer for Asset B. Otherwise, without FEC 2, decoding MMTP packets for Asset B are passed to de-jitter buffer for Asset B.

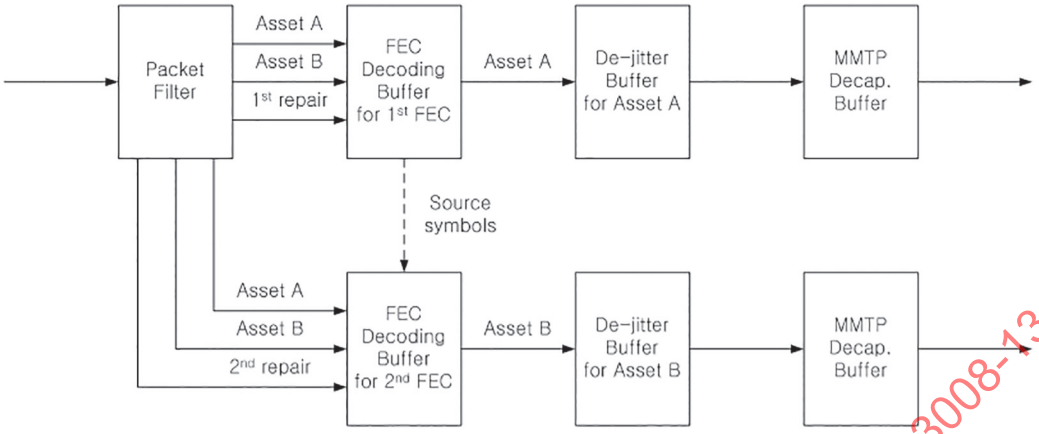


**Figure 32 — HRBM block diagram for two-stage FEC coding structure**

### 5.15.2.2 Use case: Hybrid contents delivery

When hybrid contents, which consists of Assets (video and audio for timed-data and file for non-timed data), are delivered, each Asset is packetized in MMT payloads and the packetized MMT payloads for the Assets are multiplexed on MMT packets to be a single FEC source flow (a sequence of MMT packets for the Assets). The single FEC source flow is segmented into one or more source packet blocks and each source packet block is protected by Case 2 of two-stage FEC coding structure.

During FEC decoding process, Assets for timed data is recovered by using FEC 1 decoder in the split source packet block units to provide low-delay service and Assets for non-timed data is recovered by using both FEC 1 decoder in the split source packet block units and FEC 2 decoder in source packet block units to provide higher FEC recovery performance.

In order to support this use case, the MMT sending entity should send HRBM messages as follows. For each Asset for video and audio, the value of `fixed_end_to_end_delay` field is calculated by summing of `max_transmission_delay` and `protecion_window_time` for FEC 1. On the other hand, the value of `fixed_end_to_end_delay` field for File Asset is calculated by summing of `max_transmission_delay` and `protecion_window_time` for FEC 2. Note that the MMTP packets delivering Assets for video and audio can be recovered by FEC 2 decoder, but those packets may not be used in MPU reconstruction process.

### 5.15.2.3 Use case: Streaming multicasting (or broadcasting) to two different end-user groups which is under two different channel conditions each other

When AV contents, which consists of Assets (video and audio for timed-data), are multicast (or broadcast) to two different end-user groups who are under two different channel conditions, each Asset is packetized in MMT payloads and the packetized MMT payloads for the Assets are multiplexed on MMT packets to be a single FEC source flow (a sequence of MMT packets for the Assets). The single FEC source flow is segmented into one or more source packet blocks and each source packet block is protected by Case 2 of two-stage FEC coding structure.

During FEC decoding process, User Group A, who is under relatively good channel condition, recovers the Assets by using FEC 1 decoder in the split source packet block units for low-delay service or reducing power consumption and User Group B, who is under relatively bad channel condition, recovers the Assets by using FEC 1 decoder in the split source packet block units and FEC 2 decoder in source packet block units to get reasonable FEC recovery performance.

### 5.15.3 MPU mapping to source packet block

#### 5.15.3.1 General

An MMT FEC scheme can be applied to protect MMT assets. An FEC source flow is a flow of MMTP packets delivering one or more MMT Assets. MPUs composing the MMT Assets are packetized into MMTP packets. The sequence of MMTP packets is segmented into source packet blocks and FEC encoding is applied to those blocks. The resulting encoding symbols are delivered by FEC source and repair packets.

In order to recover lost MMTP packets, the FEC decoding needs to collect sufficient number of encoding symbols from FEC source and parity packets. An MPU could be de-packetized only after the FEC decoding process ended for all FEC source packet blocks containing any MMTP packet from the MPU. As a result, there is a different delay dependent of how to map MPUs to source packet blocks. Therefore, MMT needs a strategy for mapping MPUs to source packet blocks to prevent unintended delay in MMT client.

#### 5.15.3.2 Aligned MPU mapping method to source packet block

Firstly, it will be assumed that an FEC source flow is a flow of MMTP packets delivering a single Asset. For AL-FEC encoding of the MPUs for the Asset, the MPUs are packetized in MMTP packets and these MMTP packets are mapped to source packet blocks. In this process, a source packet block contains one or more complete MPUs or part of a single MPU. More precisely, the MMTP packets from the Asset are mapped to source packet blocks in one of following three cases to minimize the decoding delay.

— Case 1: A source packet block only contains a complete set of MMTP packets packetized from a single MPU of the Asset.

— Case 2: The MMTP packets packetized from a MPU of the Asset are mapped to N (>1) source packet blocks. These source packet blocks contain only MMTP packets packetized from the MPU.

— Case 3: A source packet block only contains a complete set of MMTP packets packetized from M (>1) MPUs of the Asset.

Figure 33 shows examples for the three cases for “aligned MPU mapping to source packet block (SFB)” in case of that FEC source flow consists of MMTP packets for a single Asset. In Figure 33, “Case 1 Ex.” is an example for Case 1, “Case 2 Ex.” for Case 2 with N = 2 and “Case 3 Ex.” for Case 3 with M = 2.

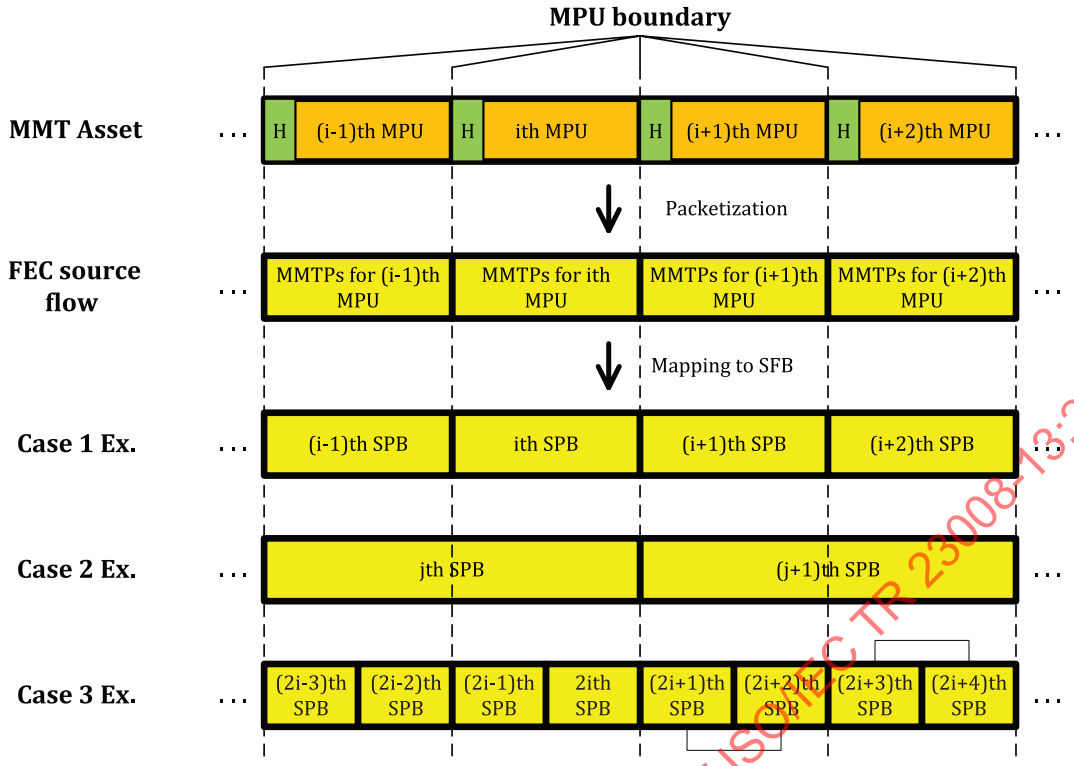


**Figure 33 — Examples for MPU mapping to source packet block (SFB)**

The aligned MPU mapping to source packet block can be easily extended to the cases where an FEC source flow is a flow of MMTP packets delivering more than one MMT assets. The design concept of "aligned MPU mapping to source packet block" is to minimize the unintended delay caused by AL-FEC protection. Some MPUs in different Assets can have close relationship and be multiplexed and then considered as a unit. This kind of set of MPUs is reffered to as group MPU (GMPU). Then, in the encoding process of the source flow, a source packet block contains one or more complete GMPUs or part of a single GMPU. More precisely, the MMTP packets from the Assets are mapped to source packet blocks in one of following three cases to minimize the decoding delay.

— Case 1: A source packet block only contains a complete set of MMTP packets packetized from a single GMPU of the Assets.

— Case 2: The MMTP packets packetized from a GMPU of the Assets are mapped to N (>1) source packet blocks. These source packet blocks contain only MMTP packets packetized from the GMPU.

— Case 3: A source packet block only contains a complete set of MMTP packets packetized from M (>1) GMPUs of the Assets.

### 5.15.4 FEC for hybrid service

For hybrid services (e.g. primary audio and video broadcasting as Figure 34 and secondary audio multicasting via broadband network as Figure 35) distributed by using the broadcast and broadband networks, a repair flow is created to protect a source flow by FEC encoding based on MMT AL-FEC framework. When the source flow is delivered via broadband network, the repair flow is delivered via broadcast network. As broadcast network provides relatively low delay while broadband network introduces relatively long delay, broadcasting delivery of repair flow enables to reduce the jitter on the MMTP packets for the source flow by recovering the lost MMTP packets which may arrive after fixed end-to-end delay (i.e. maximum transmission delay + FEC protection window time) which is provided by the HRBM message.

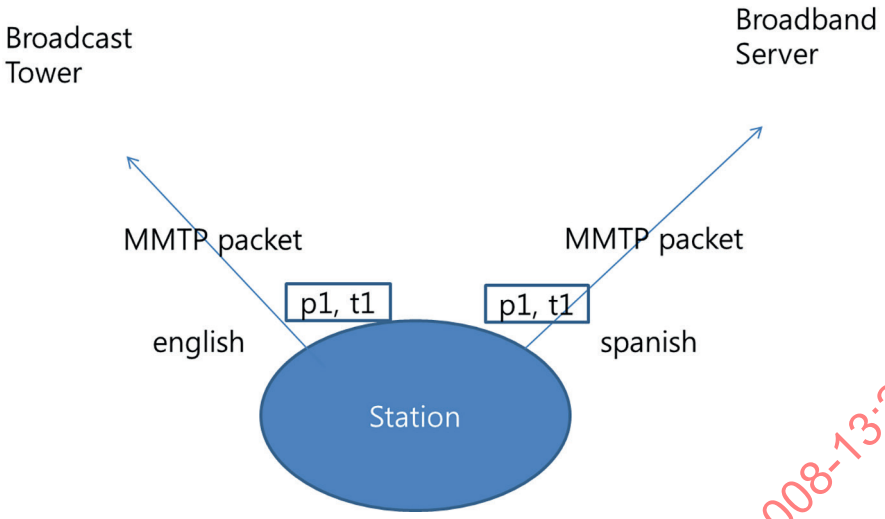


**Figure 34 — Hybrid service for primary and secondary audios**

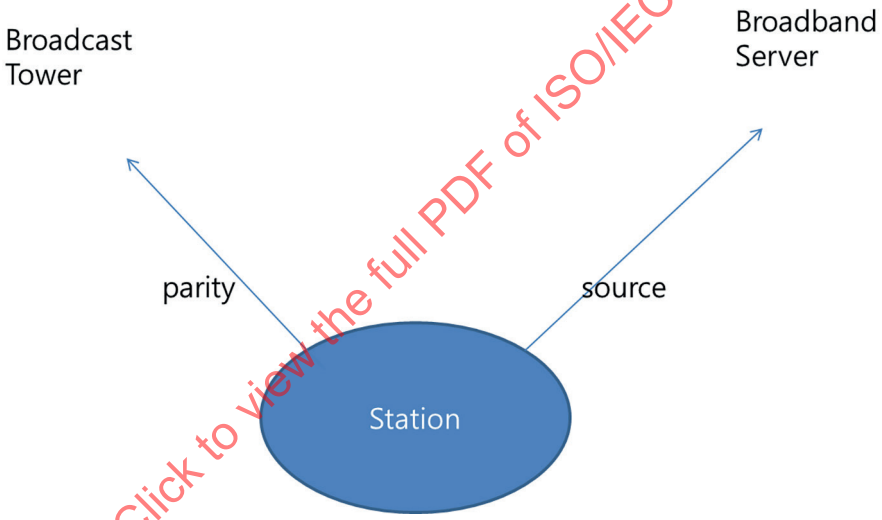


**Figure 35 — Broadcasting parity for broadband source**



When the MMT sending entity wants to provide the broadband contents with fixed end-to-end delay = D, the MMT sending entity creates HRBM message(s) with fixed end-to-end delay = D and maximum transmission delay = D – F (F is FEC protection window time) and generates a repair flow to protect the source flow of the broadband contents. In this case, repair symbols are generated as many as to be able to recover the MMTP packets of each source packet block which may arrive after fixed end-to-end delay.

The following are examples:

— maximum delay of broadband network is 5 s;

— broadcast network has 1 s constant delay;

— 80 % of MMTP packets for a source packet block arrive within 2 s at MMT receiving entity;

— 10 % of MMTP packets for the source packet block arrive within between 2 s and 3 s;

— the remaining 10 % of MMTP packets for the source packet block arrive between 3 s and 5 s.

Then,

— MMT sending entity creates HRBM message with fixed end-to-end delay = 3 s and maximum transmission delay = 2 s, and

— generates repair symbols as many as to be able to recover 20 % of MMT packets (for the source packet block) which are assumed to arrive between 2 s and 5 s based on FEC protection window time = 1 s and the recovery performance of FEC code algorithm to be used.

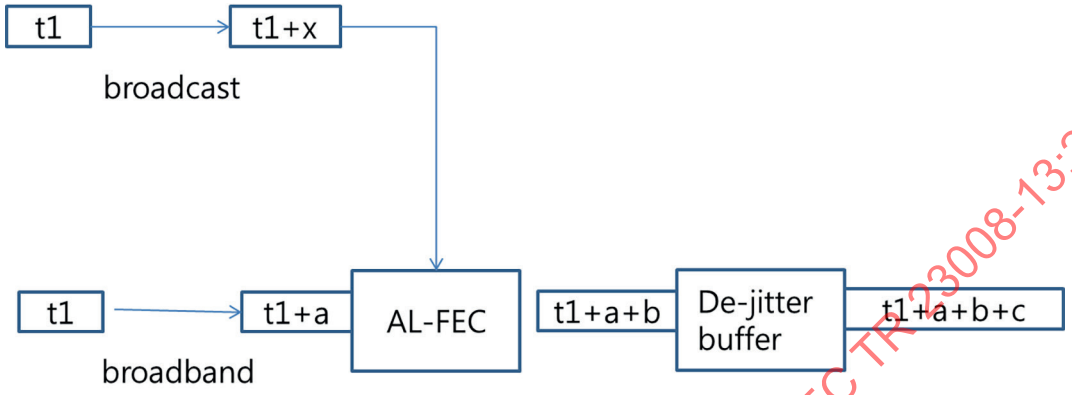


**Figure 36 — Jitter reducing mechanism by broadcasting repair**

In Figure 36, for the above example, $x \leq 1$, $a \leq 2$, $a + b \leq 3$ and $a + b + c = 3$.

Then, FEC source packets for the source packet block are delivered over UDP or TCP/IP via broadband network and the FEC repair packets for the repair symbols are delivered over UDP/IP via broadcast network.

## 5.16 Delivery of encrypted MPUs

Common encryption relies on different types of metadata for enabling decryption. The “pssh” box carries DRM-specific metadata that is opaque to common encryption. The “pssh” box may appear inside the “moov” box or any “moof” box. Default track encryption metadata is provided in the “tenc” box and provides default metadata for all samples of the track. This information may be overwritten by encryption metadata for sample groups, which are provided as part of sample descriptions of the corresponding track. Finally, sample-specific encryption metadata can be provided as part of sample auxiliary information, which is either stored as part of the “mdat” box and pointed at by “saiz” and “saio” boxes for size and location, respectively, or/and as of the second edition of common encryption, sample encryption metadata will be provided as part of the “senc” box in track fragments “traf” boxes of movie fragments.

When delivering a common encryption encrypted MPU, all sample auxiliary information, whether stored as part of the “tenc”, “senc” or the “mdat”, is considered as part of the Fragment Metadata. In the latter case, the auxiliary sample information should appear in the beginning of the mdat box prior to any media samples. Figure 37 depicts how packetization occurs.

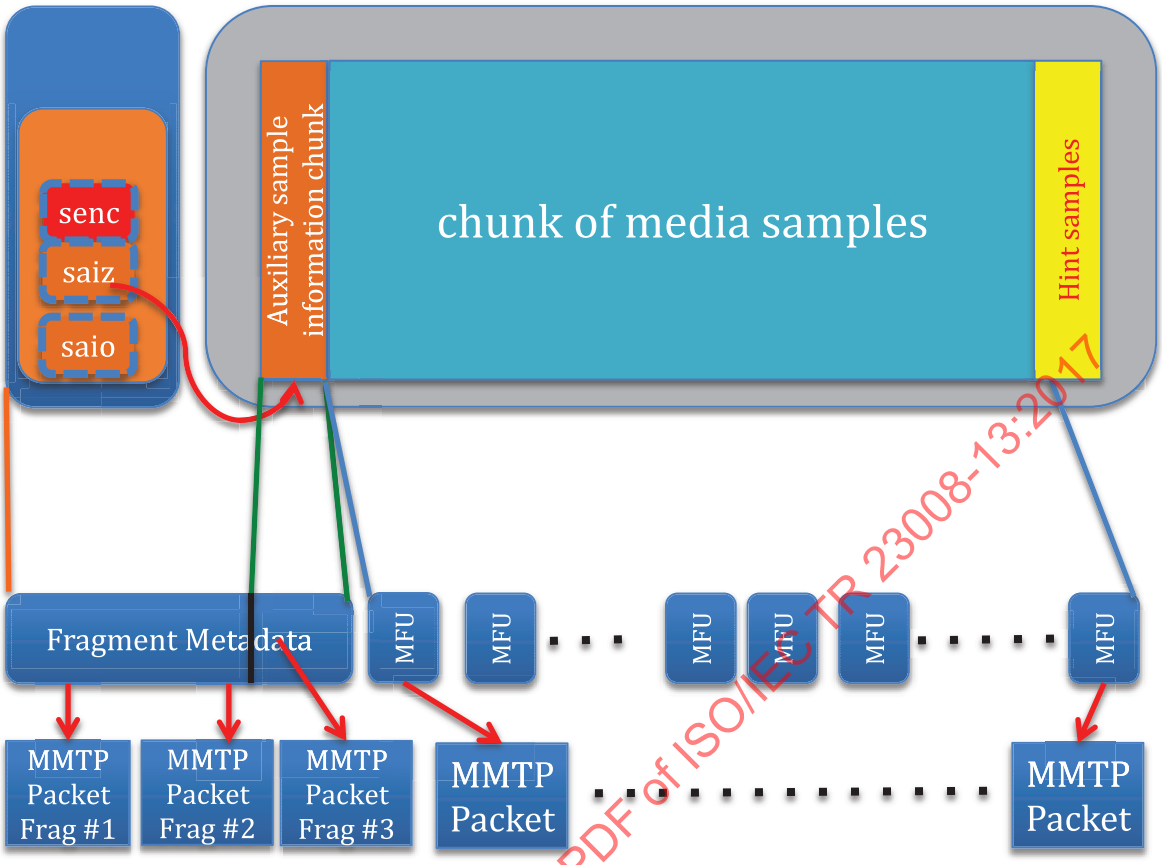


**Figure 37 — Packetization considering encryption**

The fragment metadata including the part in the "mdat" box is treated as a single payload. The receiver uses the offset and size information in the "moof" box to reposition the MPU metadata again in the "mdat" box when reconstructing the movie fragment.

## 5.17 HRBM message updating

### 5.17.1 General

The Hypothetical Receiver Buffer Model (HRBM) guarantees that each MMTP packet has a fixed end-to-end delay although it was delivered through a network with jitter. An MMT sending entity determines the required buffer size and the fixed end-to-end delay for each MMTP subflow and signals this information to an MMT receiving entity using the HRBM messages. Then MMT sending entity transmits MMTP packets with the schedule verified by the HRBM to ensure no buffer overflow in the MMT receiving entity.

The required buffer size is calculated by using the maximum duration of the buffer, which can be obtained by subtracting the minimum transmission delay from the fixed end-to-end delay, and the maximum bitrate of the MMTP packet stream. In some applications, the MMT sending entity can temporarily change the maximum bitrate of the MMTP packet stream. In that case, the HRBM message should be updated based on the new maximum bitrate of the MMTP packet stream.

### 5.17.2 HRBM message sending schedule

When updating the HRBM message, the MMT sending entity should assume that an MMT receiving entity will prepare another de-jitter buffer which operates according to the updated HRBM message and passes the MMTP packets transmitted after the MMTP packets containing the updated HRBM message to the new de-jitter buffer (it can be done by checking the value timestamp field in the MMTP

packet header). It is recommended that the updated HRBM message should be transmitted between consecutive MPUs in order to prevent unexpected delay in the MMTP packet decapsulation buffer.

### 5.17.3 Use case

Consider a service supporting a synchronized playout at multiple devices where each device is connected to the server using individual unicast connection. It can be assumed that the server has the same number of logical MMT sending entities as the number of devices connected to the server. In order to support a synchronized playout, the MMT sending entities can simultaneously transmit the same MPU to each MMT receiving entities using the same transmission schedule based on the same fixed end-to-end delay and the buffer size. When a new device has been joined to the service, the corresponding MMT sending entity is instantiated. If the device is joined in the middle of an MPU sending duration, the MPU cannot be delivered to the device completely. In order to reduce the service access time, the MMT sending entity can quickly transmit the whole MPU to the newly joined device if possible (burst mode) and then it transmits the next MPU using the same schedule with the other MMT sending entities in the server (normal mode).

For more detailed analysis, the following parameters can be defined for normal mode:

— MPU duration: D seconds;

— maximum bitrate of the MMTP packet stream: M bytes/s;

— maximum number of byte required to deliver a MPU: M × D bytes;

— fixed end-to-end delay: F seconds (the minimum transmission delay = d);

— required buffer size: B = (F – d) × M bytes.

Assuming that the device has been joined after J (<D) seconds from the start of the delivery of an MPU at other MMT sending entities in the server, then the maximum bitrate of the MMTP packet stream in the burst mode, M', is given by Formula (7):

$$M' = (M \times D)/(D - J) \tag{7}$$

As a result, the required buffer size for burst mode, B', is increased compared with the required buffer size for normal mode as follows:

$$B' = (F - d) \times M' = (F - d) \times M \times D/(D - J) = B \times D/(D - J)$$

$$= B + B \times J / (D - J).$$

Then the MMT sending entity can determine whether it can operate in burst mode or not considering the available bandwidth and the above parameters. If it is possible to operate in burst mode, the MMT sending entity signals the values of M' and B' by HRBM message and sends the first MPU in burst mode. After the MMT sending entity has finished the delivery of the first MPU in burst mode, it operates in normal mode. Before it transmits the MMTP packet containing the second MPU, the value of M and B should be signalled to the MMT receiving by updating HRBM message. If the HRBM message is not updated properly, then B × J/(D – J) bytes of the memory in the MMT receiving entity is wasted.

## 5.18 MMTP packet with padded data

MMTP packet could be delivered over TCP/IP and UDP/IP flows. When the MMT sending entity send the MMTP packet with padded data, it could be identified and calculated to remove the padded data at the MMT receiving entity. To identify the padded data to remove, the size of padded data is needed in the entire delivered IP packet. This document presents the way to calculate the size of padded data in MMTP packet over UDP packet as shown in Figure 38.


**Figure 38 — MMTP packet over UDP**

As shown in Figure 38 for MMTP packet over UDP, the length of UDP packet is specified as the length in bytes of the entire datagram which consists of UDP header and data in UDP header. The field size sets a theoretical limit of 65,536 bytes (e.g. the minimum length of 8 bytes header + 65,527 bytes of data) for a UDP datagram. In IPv4, the practical limit for data length is 65,507 bytes (65,536 bytes – 8 bytes of UDP header – 20 bytes of IP header). In MMTP, packet header indicates that the length of MMTP packet header (V = 1) contains the minimum length of MMTP packet header which is 18 bytes and the size of extension header. The length of padded data could be calculated by the provided information in MMTP packet header with extension header fields. See Figure 39.

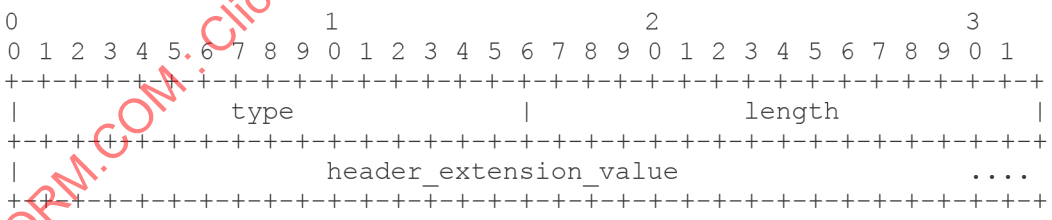


**Figure 39 — MMTP packet header with extension header field**

For example, the “type” indicates that this payload contains the padding data, the “length” presents the size of value and the “header_extension_value” provides the size of padding data from the end of UDP packet tail.

# 6 Use cases for MMT deployment

## 6.1 General

This clause gives implementation guidelines on specific deployment use cases and examples utilizing MMT standard for specific purposes. This clause intends in particular to guide implementers how MMT specification can be utilized or extended for the specific topics such as, but not limited to:

— DASH over MMT;

— network caching/MANE utilizing MMT;

— usage and extensions of ISO/IEC 23008-1 for Japanese broadcasting services.

## 6.2 Delivery of DASH Presentations using MMT

### 6.2.1 General

When streaming a DASH presentation, e.g. over a broadcast channel, the HTTP protocol can no longer be used. MMTP provides the necessary tools to support the delivery of a DASH presentation. A DASH presentation consists of the Presentation Information (PI) (which is the MPD) and the data segments (initialization and media segments).

### 6.2.2 Delivery of the MPD

The MPD makes the core part of the Presentation Information of the presentation. The MPD is assigned its own MIME type, “application/dash+xml”, which is used to identify the type of the presentation. The MPD is embedded in the MPI table, which in turn is delivered using the MPI message. The format of the message may either be binary or XML. In case of XML format, the MPD is embedded using the <![CDATA[ ]]> encapsulation. The MPD updates are delivered using the same mechanism. The updates are discovered by checking for the table identifier in signalling messages of that particular package.

The MPD is then used by the client (DASH client) to drive the presentation. It is required that all segments that are addressed by the MPD should be readily available at the receiver side, i.e. delivered and recovered by their segment availability start time. This may be ensured by setting delivery schedule and the HRBM parameters appropriately.

### 6.2.3 Delivery of the data segments

#### 6.2.3.1 Delivery using the MPU mode

When delivering DASH content using the MPU mode, each Representation is considered as an MPU. It is important to note that the concatenation of all segments of a Representation results in a fragmented ISOBMFF compatible file, which is equivalent to an MPU in MMT terms.

The initialization segment of the Representation is the MPU metadata and is marked appropriately during the delivery to ensure that it is recovered by the receiver. The MPU metadata should be delivered repetitively to ensure that all clients, independent of the time they join the session, will ultimately be able to recover it. The MPU mode provides the means for marking these important packets by setting the FT field to MPU metadata. Similarly, fragment metadata is important for every fragment and should also be delivered repetitively to increase the probability of recovery.

All segments of a Representation are then marked as belonging to the same MPU by using the same packet_id and MPU sequence number to mark them. The mapping between the Representation and the packet_id and MPU sequence number is provided by the Identifier_mapping syntax element that is provided in the MP table. The simplest usage of the identifier_mapping in this case is to provide the representation_id as provided in the MPD. Other approaches, e.g. using the URL template for all segments of that Representation, may be used instead.



 

At the sender side, the data is delivered on movie-fragment basis. However, at the receiver side, each media segment needs to be recovered appropriately and timely. It is then necessary for the receiver to be able to identify the segment boundaries and the segment information as shown in Figure 40. This may be done in one of the following ways.

— A segment is recovered by identifying its boundaries using the presence of the segment index box ("sidx") from the current and the following segment.

— A segment is recovered by checking the timestamp of the first sample in the movie fragment (e.g. extracted from the "tfdt" box or the "sidx" box) and using the MPD to identify the media segment that has the same starting time.

— The segment boundary may be deduced from the segment duration by dividing the fragment start time by the duration of the media segment for that Representation.

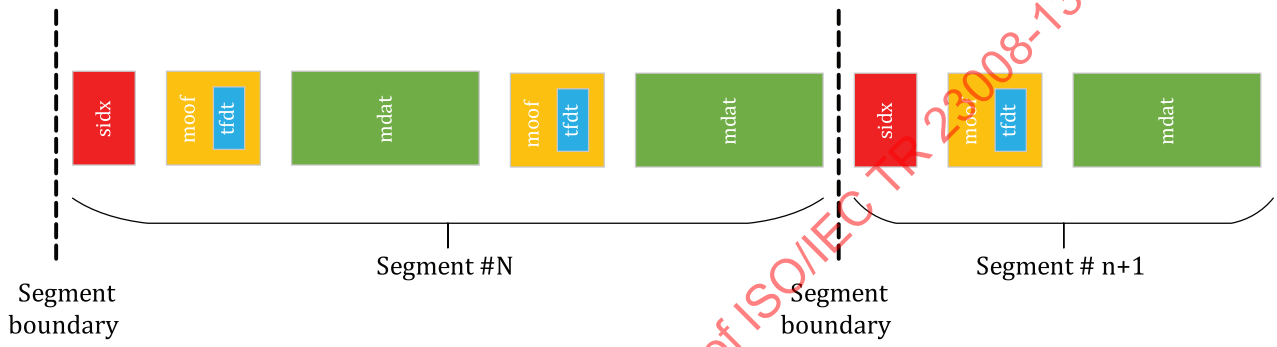


**Figure 40 — Segment boundary detection**

#### 6.2.3.2 Delivery using the GFD mode

When using the GFD mode, each segment of the DASH Representation is delivered as a regular file. The connection between those segments may be established using the packet_id. The connection between the file (transport object) and the segment information is established using the identifier_mapping information in the MP table. The most appropriate mapping in this case could be the URL template or the URL list. Based on that mapping, the TOI can be used to directly recover the segment number and the segment URL.

## 6.3 Client operation for DASH service delivered through MMT Protocol

### 6.3.1 Delivery of MPD with MMTP

For the application using MMTP for the delivery of DASH service, MPD can be received as an `MPI_table()` with MIME type set to `application/dash+xml`.

### 6.3.2 Delivery and consumption of DASH Segments with MMTP

#### 6.3.2.1 Reconstruction of DASH Segments

##### 6.3.2.1.1 Use of GFD mode

MMTP provides GFD mode for the delivery of any type of generic file. Any DASH Segment can be delivered as a generic file using the GFD mode of MMTP. Reconstruction procedure described in 5.2.3 is applied to reconstruct DASH Segments in this mode.

#### 6.3.2.1.2 Use of MPU mode

MPU mode of MMTP is used to deliver MPUs. Therefore, MPU mode can be used to deliver a DASH Segment when it also has "`mpuf`" as a compatible brand. Reconstruction procedure described in 5.2.2 is applied to reconstruct DASH Segments in this mode.

### 6.3.2.2 Consumption of DASH Segments

#### 6.3.2.2.1 Client with HTTP cache

If the client receiving DASH Segments over MMTP has an HTTP/1.1 cache, reconstructed DASH Segments are consumed by the DASH client through the HTTP cache as shown in Figure 41. As soon as complete DASH Segments are reconstructed, they are moved to the HTTP/1.1 cache, then the DASH client receives DASH Segments through normal HTTP/1.1 Request/Response operations.

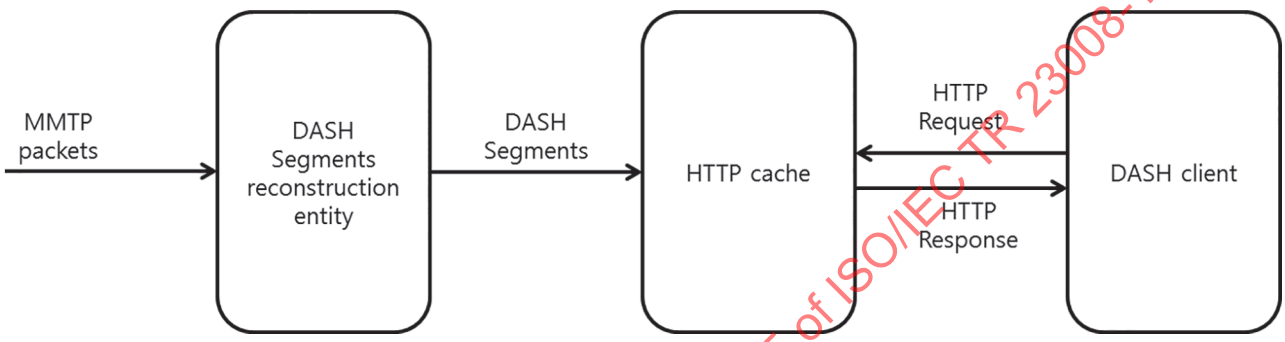


**Figure 41 — Client with HTTP cache**

#### 6.3.2.2.2 Client without HTTP cache

If the client receiving DASH Segments over MMTP does not have an HTTP/1.1 cache, reconstructed DASH Segments are consumed by the DASH client through special DASH Segments buffered as shown in Figure 42. As soon as complete DASH Segments are reconstructed, they are moved to a DASH Segments Buffer which stores DASH Segments with associated URLs as shown in Figure 43. Associated URLs of DASH Segments can be found from a GFDT when GFD mode is used or from an `mmpu` box when MPU mode is used. The DASH Client in this case looks in the DASH Segments Buffer to find an appropriate Segment by using a URL as an index for the search. There may be more than one representation delivered into the DASH Segments Buffer, each comprising different data rates, levels of video quality and robustness. If a DASH Segment at the desired quality level is not found, a DASH Segment at the next lower quality level can be found, if available.

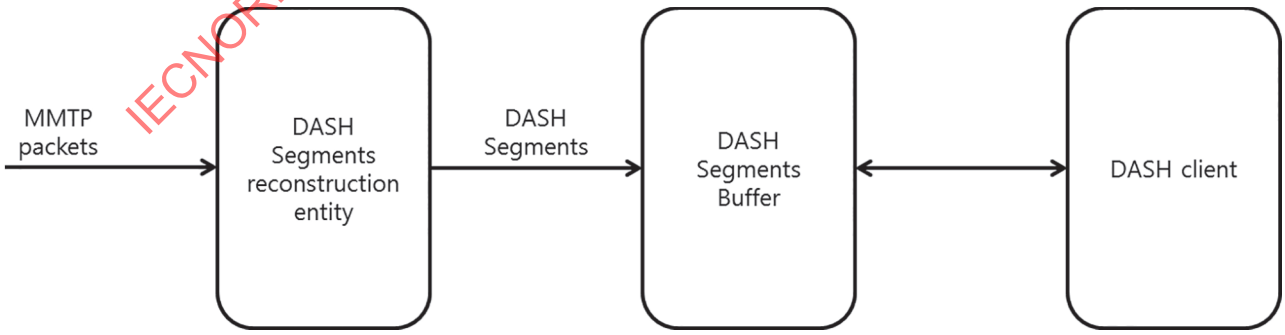


**Figure 42 — Client without HTTP cache**

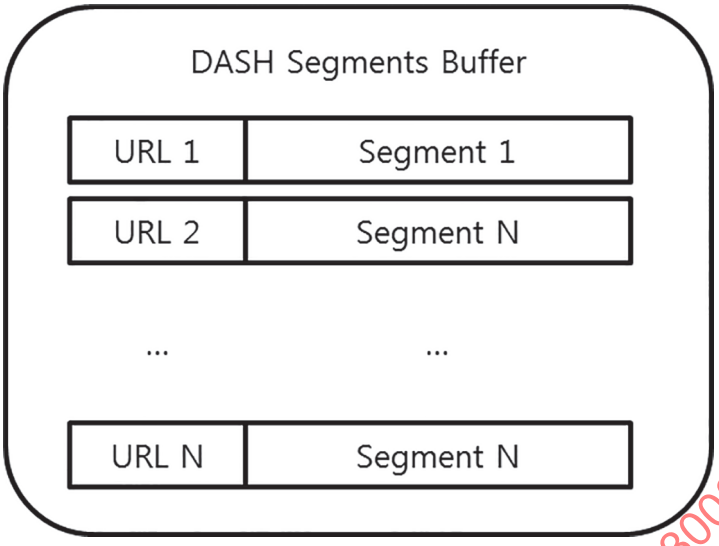


**Figure 43 — Storage of DASH Segments**

## 6.4 Hybrid of MMT and DASH over heterogeneous network

Figure 44 shows a simplified architecture of the client implementing both MMT and DASH. Figure 44 only shows the most important components of MMT and DASH.

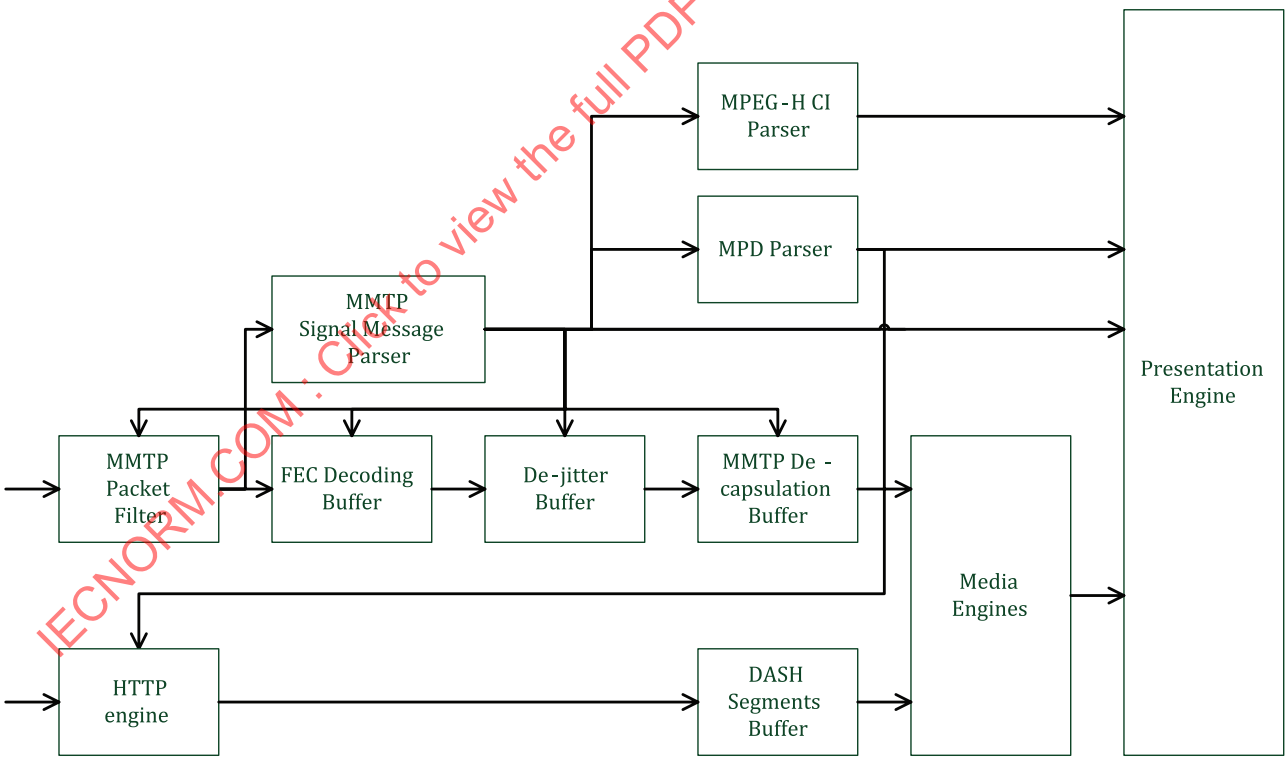


**Figure 44 — Conceptual architecture of hybrid service client**

The function of each block is described as follows.

a) The MMTP packet filter receives MMTP packets from the broadcast network and delivers them to the appropriate processing block, either the MMTP signalling message parser or the FEC decoding

buffer. The MMTP packet filter performs the following checks depending on the packet_id of each MMTP packets.

1) When the client is tuned into the specific service, then MMTP Packet Filter is only given the packet_id of MMTP signalling messages by the processing block which is not shown in Figure 44. After the MPT_message is processed by MMTP Signalling Message Parser, MMTP Packet Filter is given the packet_id of media component and start to filter MMTP packets with media component.

b) The MMTP signalling message parser receives MMTP Packets from the MMTP packet filter and parses the MMTP signalling message carried in the packet. Information retrieved from the MMTP signalling message is delivered to the appropriate processing blocks.

1) When the client is tuned into the specific service, the MMTP signalling message parser firstly looks for the MPT_message and processes it to get the list of MMT Assets and their packet_ids.

2) Whenever an MPI_message is received, it processes the PI_content_type_byte field of MPI_table() and then delivers the data carried as PI_content_byte of MPI_table() to either the MPEG-H CI parser, MPD parser or presentation engine. If the value of PI_content_type_byte is HTML 5 then the contained data is delivered to the presentation engine. If the value of the PI_content_type_byte is MPEG-H CI then the contained data is delivered to the MPEG-H CI parser. If the value of the PI_content_type_byte is DASH MPD then the contained data is delivered to the DASH MPD parser.

3) Whenever an AL_FEC message is received, the information in the message is signalled to the FEC Decoding Buffer.

4) Whenever an HRBM message is received, the information in the message is signalled to the de-jitter buffer.

c) MPEG-H CI parser receives MPEG-H CI from the MMTP signalling message parser and processes it. The presentation time of the first access unit of the MPU is known from the “MediaSync” element referencing such MPU and is delivered to the presentation engine. The presentation time of the first access unit of the first segment described by the DASH MPD is also known from the “MediaSync” elements referencing the relevant DASH MPD file and is delivered to the presentation engine.

d) The MPD parser receives the DASH MPD through the MMTP signalling message parser and processes it. The MPD parser provides relative presentation time of each DASH Segment to the Presentation Engine.

e) The FEC Decoding buffer receives MMTP packets with dedicated packet_id and immediately copies packets to de-jitter buffer. If there are any packets which are not received until protection window time of AL_FEC message signalled by MMTP Signal Message Parser, repair operation is applied to the received packets. If packets are recovered, they are also immediately copied to de-jitter buffer.

f) De-jitter buffer receives MMTP packets from FEC Decoding Buffer. De-jitter buffer stores those packets until the fixed_end_to_end_delay provided by the HRBM message has passed since the time specified in the time_stamp field of each MMTP packets and copies them to the MMTP decapsulation buffer.

g) The MMTP decapsulation buffer depacketizes MMTP packets received from the de-jitter buffer and reconstructs the MPUs. Reconstructed MPUs are delivered to media engines.

h) The HTTP engine makes GET requests with the URL received from the MPD parser through the broadband network and receives DASH Segments as responses. Received DASH Segments are delivered to the DASH Segment buffer.

i) DASH segment buffer receives DASH segments from the HTTP engine and delivers them to media engine as soon as the processing of previously delivered DASH Segments are finished.

j) Media engines receives MPUs and DASH Segments and decode them with appropriate decoders. It delivers decoded results to the presentation engine.

k) The presentation engine receives the HTML 5 document from the MMTP signalling message parser. It renders media data from the media engine in the location specified by the HTML 5 document and at the time provided by the MPEG-H CI parser and the MPD parser.

## 6.5 MMT caching for effective bandwidth utilization

### 6.5.1 Overview of MMT caching middlebox architecture

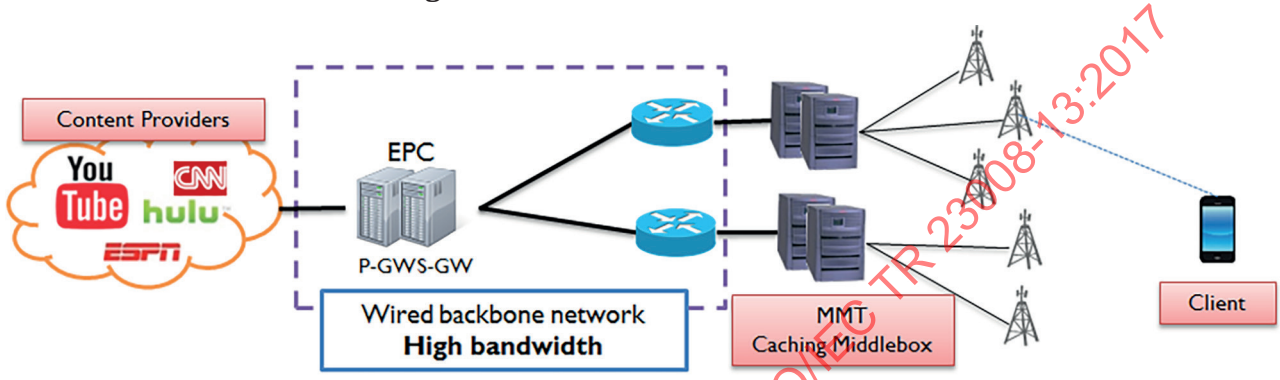


**Figure 45 — Overview of an MMT caching middlebox**

The network is composed of three entities: a server (or a content provider), a caching middlebox and a client as shown in Figure 45. From the perspective of a client, the caching middlebox works transparently such that the client cannot tell a middlebox from a server. It can be assumed that the server knows the existence of the caching middlebox.

The goal of the caching middlebox is a) to reduce the bandwidth usage at the path between the server and the middlebox and b) to adapt the video rate to available bandwidth per each client. For the former, the middlebox focuses on network redundancy elimination on the MPU delivery. It attempts to fetch all MPUs that belong to a requested video from the server for example by using HTTP, but actually downloads only those MPUs that do not exist in the middlebox cache (cache-missed MPUs). To identify an MPU, it uses a MPU ID which is a hash value of the content of the MPU.

For serving the media to clients, the middlebox adapts the MPUs to the available bandwidth per client and fairly distributes the bandwidth to each client. That is, the middlebox uses MMTP to deliver the MPUs to the clients.

Moreover, deploying MMT caching middleboxes along with an HTTP CDN is costly since the CDN should support MMT instead of HTTP. For easy deployment, it is desirable that the MMT caching middlebox cooperates with HTTP (caching) servers so that they communicate in HTTP.

### 6.5.2 Content-based caching of MMT media

#### 6.5.2.1 Content-based caching

Content-based caching caches a content by its chunks instead of by the whole content. It divides the content into smaller chunks and names each chunk using its content hash. Each chunk is a unit of caching, but its chunk name (or ID) is essentially a tightly-bound summary of the chunk content. So, using the chunk names, one can easily identify the duplicates across contents with different names (e.g. aliases) or even the set of partially-redundant chunks among different contents. Content-based caching was used to MMT media delivery.

#### 6.5.2.2 Caching unit

The MMT caching middlebox uses an MPU as a caching unit (chunk), so the chunking process is simple and fast.

#### 6.5.2.3 MPU ID (Chunk ID) generation

##### 6.5.2.3.1 General

An MPU ID is used for determining a cache hit or a miss. An MPU ID should be calculated by hashing over the content binary by carefully excluding the metadata such as an asset ID or an MPU sequence number that are not bound to the video or audio content itself. In addition, the MPU ID should be calculated differently according to the type of an MPU.

##### 6.5.2.3.2 MPU with timed media

Multiple mdat atoms could exist in a timed media. In that case, the MPU ID is conceptually a hash value over all mdat binaries stitched together in an MPU.

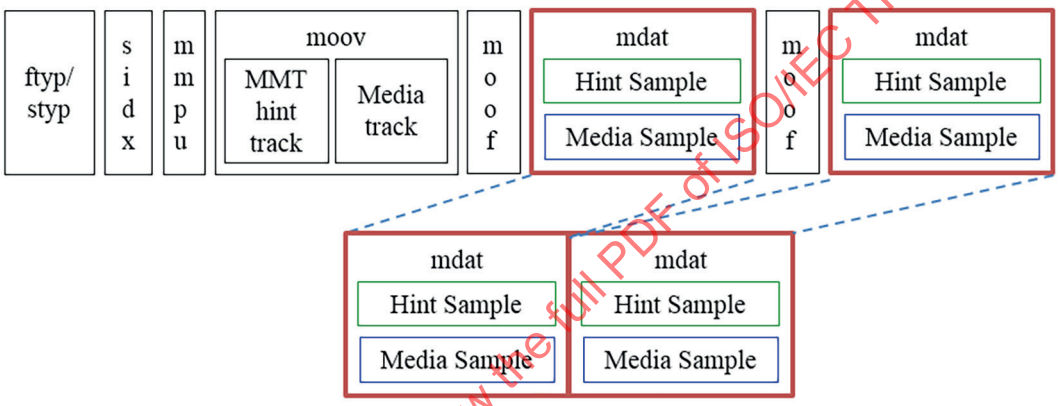


**Figure 46 — MPU ID generation for timed media**

The actual MPU ID is calculated over payloadized mdat sequences as in Figure 46. There are two reasons behind it. First, the middlebox can reduce undesirable delay if it fetches the MPU in the payloadized format as in Figure 47 since it can deliver each MFU right away without waiting for the full MPU. Second, payloadized mdat sequences would fix the location of the hint samples, which would produce the same ID (hash) even if those hint samples are stored in different locations at a storage.

*MPU ID = One-way hashing (payloadized mdat sequence)*

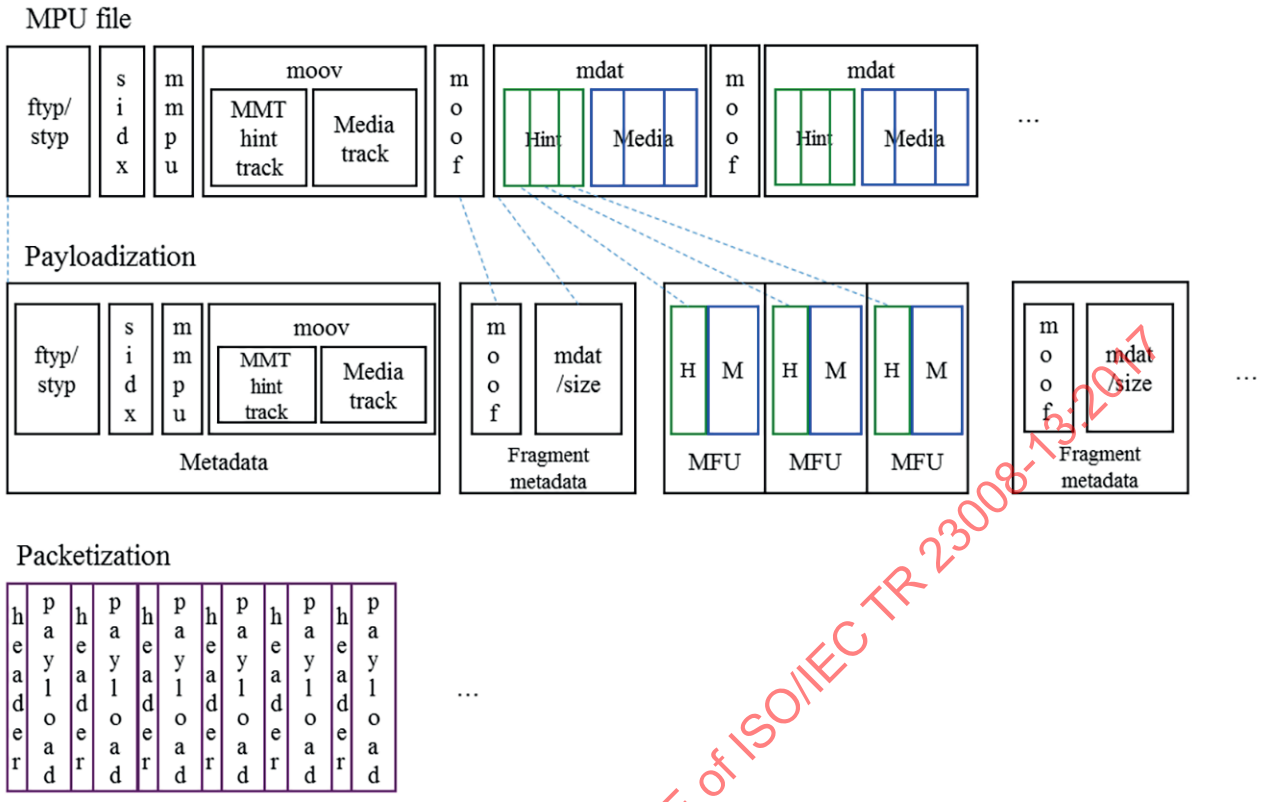


**Figure 47 — Payloadized format**

#### 6.5.2.3.3 MPU with non-timed media

For non-timed media, a MPU ID refers to a hash over all items stitched together in an MPU as in Figure 48.

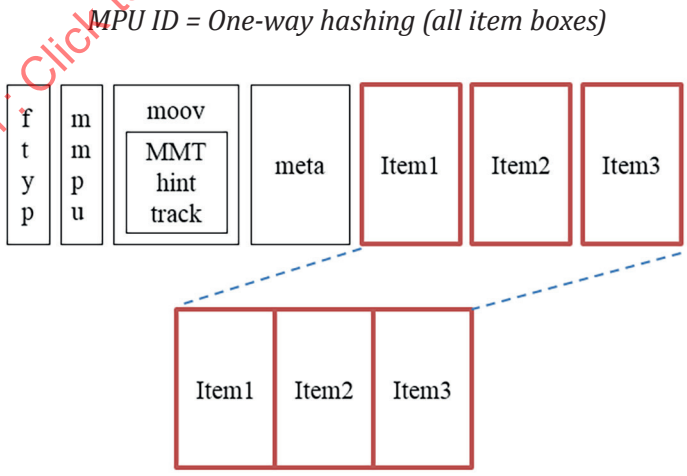


**Figure 48 — MPU ID generation for non-timed media**

### 6.5.3 MPU sync protocol between server and caching middlebox

#### 6.5.3.1 General

This subclause proposes a protocol based on a redundancy elimination technique for delivering MMT media from a server to a caching middlebox. It augments HTTP with a few custom headers for exchanging MPU requests, MPU IDs (chunk IDs) and metadata for delivering MMT packages.

#### 6.5.3.2 Protocol overview

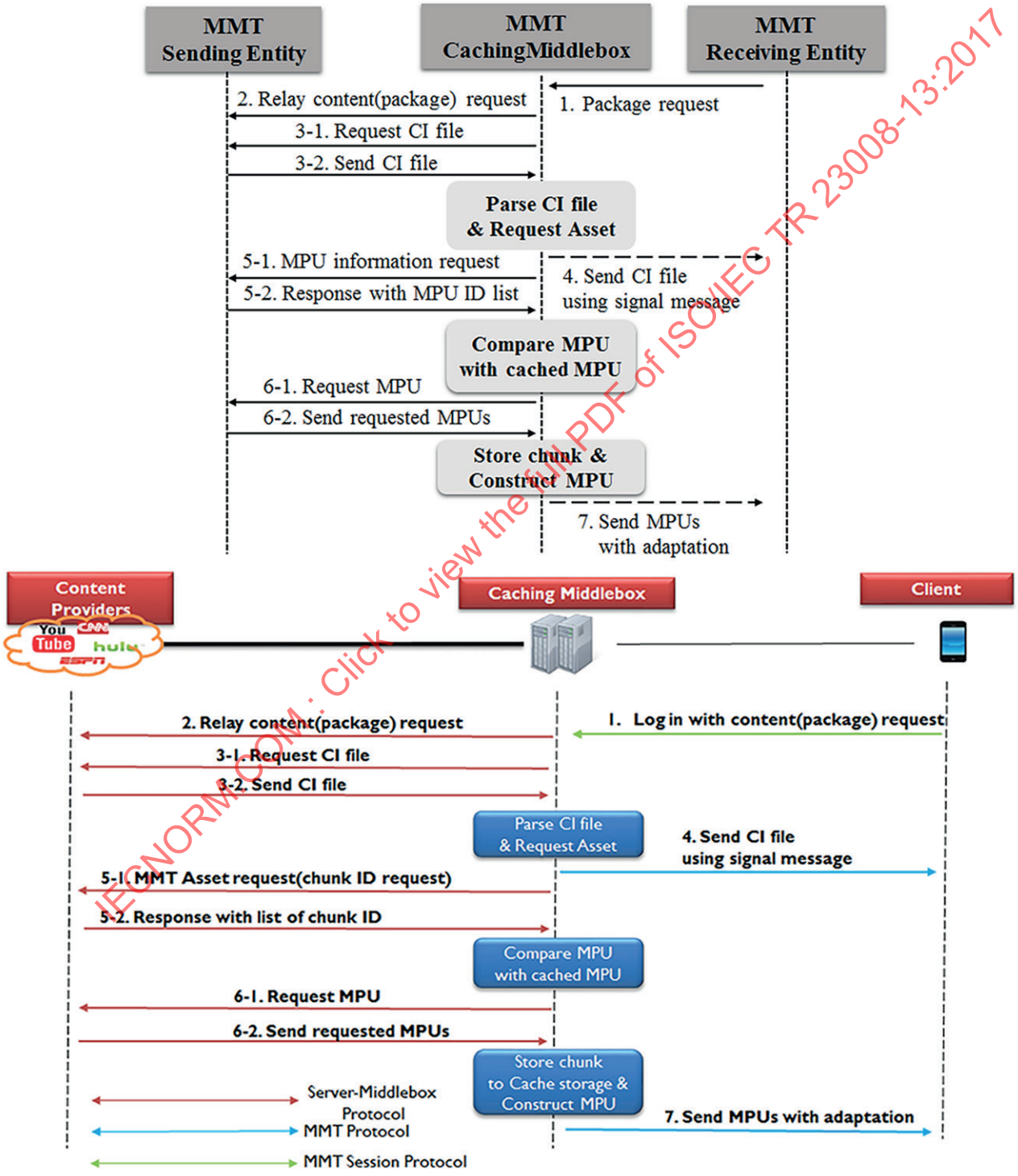


**Figure 49 — Operation sequence**

Figure 49 shows an overview of how the protocol works. Note that the following shows one implementation of the protocol using custom HTTP headers, but one can implement the same mechanism using custom web services based on XML-RPC, JSON-RPC, RESTful API, etc.

First, in order to work as server, the middlebox asks for a CI file and an HTML file at the server using "X-CIRequest" and "X-HTMLRequest" HTTP custom headers. After parsing the CI and HTML files, the middlebox retrieves the name of the asset and requests MPU IDs for the MPUs in the asset, using the "X-MPUIDRange" header. On receiving the MPU IDs from the server, the middlebox determines whether the chunk content is cached for each ID. In case of a cache hit, the middlebox requests the header of the MPU that consists of non-mdat boxes (e.g. moof and moov boxes) by using the "X-MPUHeaderRequest" header and reconstructs the MPU in real time. On a cache miss, the middlebox asks for the entire MPU by using the "X-MPURequest" header and store the MPU content (e.g. mdat part) in cache storage along with its MPU ID.

#### 6.5.3.3 Request messages

##### 6.5.3.3.1 General

In this subclause, custom HTTP headers is defined as one example for requesting a CI file, a MPU ID list, an MPU and an MPU header as in Table 1. The request and response messages follow the HTTP message format defined in RFC 2616, Section 4[10]. Again, these are just an example of a possible implementation and one can choose to use other web service mechanisms as well.

**Table 1 — Request header fields**

| | | |
|---|---|---|
| Request header = | X-CIRequest | ; 6.5.3.3.2 |
| | \| X-HTMLRequest | ; 6.5.3.3.3 |
| | \| X-MPUIDRange | ; 6.5.3.3.4 |
| | \| X-MPURequest | ; 6.5.3.3.5 |
| | \| X-MPUHeaderRequest | ; 6.5.3.3.6 |

##### 6.5.3.3.2 X-CIRequest

X-CIRequest is used with the HTTP GET method to request for the CI file of an MMT package. The path of this request represents the name of the package. When the "X-CIRequest" field is present in the request, the server should provide a CI file of requested package.

```
      Device_id = 1*DIGIT
      X-CIRequest = "X-CIRequest" ":" device_id

Example:
      GET /Package1 HTTP/1.1
      Host: test.mmt.com
      X-CIRequest: 1
```

##### 6.5.3.3.3 X-HTMLRequest

X-HTMLRequest is used with the HTTP GET method to request the HTML file of an MMT package. The path of this request represents the name of the package. When the "X-HTMLRequest" field is present in a request, the server should provide an HTML file of a requested package.

```
      Device_id = 1*DIGIT
      X-HTMLRequest = "X-HTMLRequest" ":" device_id

Example:
      GET /Package1 HTTP/1.1
      Host: test.mmt.com
      X-HTMLRequest: 1
```

#### 6.5.3.3.4 X-MPUIDRange

X-MPUIDRange is used with the GET method to request the list of MPU IDs. The X-MPUIDRange field in a GET request specifies two numbers that represent the start and end sequence numbers of MPUs and the response should contain the MPU IDs that correspond to those MPUs in the range. The path in the first line represents the asset name.

```
MPURange-specifier = MPU_SeqNumStart "-" MPU_SeqNumEnd
MPU_SeqNumStart = 1*DIGIT
MPU_SeqNumEnd = 1*DIGIT
X-MPUIDRange = "X-MPUIDRange" ":" MPURage-specifier
```

```
Example:
      GET /Package1/Asset1 HTTP/1.1
      Host: test.mmt.com
      X-MPUIDRange: 1-4
```

#### 6.5.3.3.5 X-MPURequest

X-MPURequest is used with the GET method to request a whole payloadized MPU from a server. The X-MPURequest field specifies the sequence number of an MPU that is being requested and the path in the first line represent the asset name.

```
MPU_SeqNumber = 1*DIGIT
X-MPURequest = "X-MPURequest" ":" MPU_SeqNumber
```

```
Example:
      GET /Package1/Asset1 HTTP/1.1
      Host: test.mmt.com
      X-MPURequest: 3
```

#### 6.5.3.3.6 X-MPUHeaderRequest

X-MPUHeaderRequest is used with the GET method to request the MPU header which consists of non-mdat boxes from a server. X-MPUHeaderRequest specifies the sequence number of an MPU whose header is being requested and the path in the first line represents the asset name.

```
MPU_SeqNumber = 1*DIGIT
X-MPUHeaderRequest = "X-MPUHeaderRequest" ":" MPU_SeqNumber
```

```
Example:
      GET /Package1/Asset1 HTTP/1.1
      Host: test.mmt.com
      X-MPUHeaderRequest: 3
```

### 6.5.3.4 Response messages

#### 6.5.3.4.1 General

Table 2 is a response header to the corresponding requests of MPU ID range, MPU header and MPU content.

**Table 2 — Response header fields**

| | |
|---|---|
| Response header = X-MPUIDRange | ; 6.5.3.4.2 |
| \| X-MPUHeader | ; 6.5.3.5 |
| \| X-MPUContent | ; 6.5.3.6 |

#### 6.5.3.4.2 X-MPUIDRange

If X-MPUIDRange is used in a response, it specifies the range of the requested MPU IDs in the response. Moreover, X-MPUIDRange could be used without the request as well. The length represents the total number of chunks in the requested asset. If the total number of chunks is unknown (e.g. live media), the length is set to –1. Each line in the response body consists of a sequence number of an MPU, name of the



 

hashing algorithm and its MPU ID. The number of lines in the response body should match the range in the X-MPUIDRange header.

```
Header format:
       MPURange-specifier = MPU_SeqNumStart "-" MPU_SeqNumEnd "/" Length
      MPU_SeqNumStart = 1*DIGIT
      MPU_SeqNumEnd = 1*DIGIT
      Length = 1*DIGIT
      X-MPUIDRange: "X-MPUIDRange" ":" MPURange-specifier

Body Format:
      MPU_SeqNumber Hash_scheme Hash_value(MPU ID)
      MPU_SeqNumber Hash scheme Hash_value(MPU ID)

Example:
      HTTP/1.1 200 OK
      X-MPUIDRange: 1-4/10
      Content-Length: 48

      1 SHA-1 2341242
      2 SHA-1 2421245
      3 SHA-1 2241244
SHA-1 1234124
```

#### 6.5.3.5 X-MPUHeader

X-MPUHeader is a response header that specifies the MPU sequence number, and payloadized MPU metadata is carried in the response body. This is used as a response to X-MPUHeaderRequest in the request.

```
Header format:
      MPU_SeqNumber = 1*DIGIT
      X-MPUHeader = "X-MPUHeader" ":" MPU_SeqNumber

Body format:
      Payloadized MPU metadata

Example:
      HTTP/1.1 200 OK
      X-MPUHeader: 3
      Content-Length: 2879097

      …ftyp…mp41…mmpu…moov…
```

#### 6.5.3.6 X-MPUContent

X-MPUContent is a response header that specifies the MPU sequence number whose payloadized full content is carried in the response body. This is used in response to X-MPURequest in the request.

```
Header format:
      MPU_SeqNumber = 1*DIGIT
      X-MPUContent = "X-MPUHeader" ":" MPU_SeqNumber

Body format:
      Payloadized full MPU content in the response

Example:
      HTTP/1.1 200 OK
      X-MPUContent: 2
      Content-Length: 2879097

      …ftyp…mp41…mmpu…
```

### 6.5.4 MMT cache manifest

The data path for cache in video broadcasting over the Internet is illustrated in Figure 50. Directly serving each end user from the originating CDN server is not feasible for several reasons:

— computing power: the server computing power is limited, can only support a limited number of concurrent connections;

— out-going bandwidth: the original content server should have n x B out-going bandwidth to support n users consuming video at bit rate B. For emerging high bandwidth applications like UHD, this can easily eat up the bandwidth, e.g. B = 16 Mbps, to serve 1 000 user will require 16 G out-going bandwidth.

— end-to-end uncertainty: once the packet left the originating server, the current state of Internet cannot guarantee throughput nor delay, in fact, this is the uncertainty that will stay for a long time.

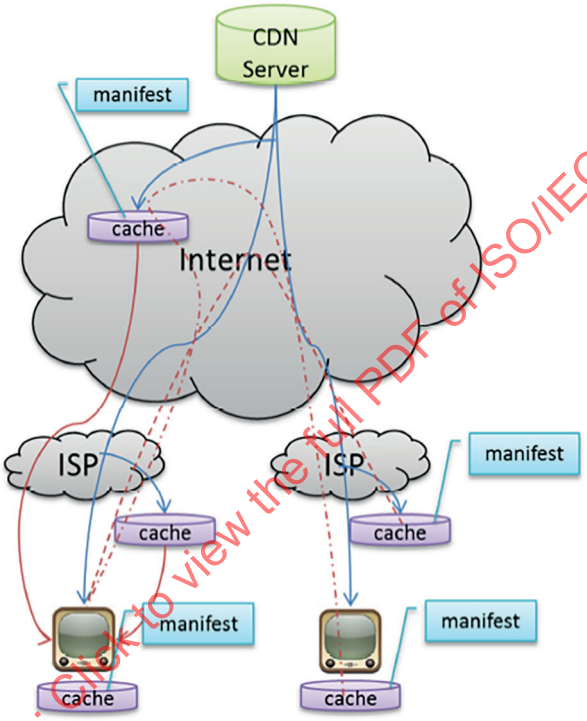


**Figure 50 — Cache in Internet video delivery**

By utilizing caches at various places in the network, instead of serving the end user from the originating server, cache hosts can do the job, with usually much shorter RTT and alternative delivery path that may have less congestion.

Content at the originating server is well behaving and predictable, as it is becoming available linearly over time. But for content at caches, such characteristics are not the case. Due to user behaviour, and different start time of streaming, different cache may contain different segments of content and with different sub-representations (in DASH terminology) with different quality and QoS requirement. A hypothetical simple case without fragmentation is illustrated in Figure 51.

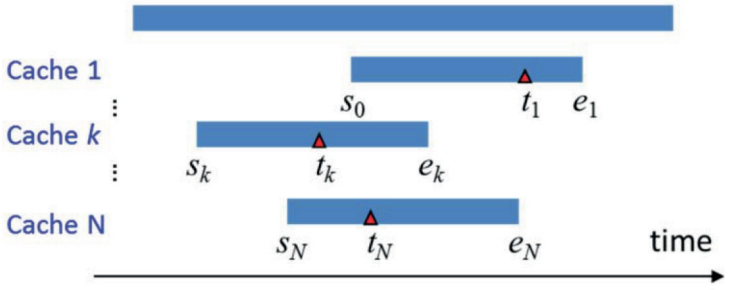


**Figure 51 — Cache status**

To facilitate cache-assisted streaming, at least the content start and end timestamp, {$s_k$, $e_k$} current playback timestamp, $t_k$, need to be signalled. For the same segment, when different sub-representations are available, their associated MFU index, resulting quality and QoS info need to be signalled, as similar with the multiple operating points ADC case in streaming. The MMT cache manifest is organized around asset ids, and then for each asset, those segments that are present in the cache is listed by timestamp/fingerprint. For each segment, a new data structure Segment Manifest or SegMf can be introduced, which in turn consists of multiple Rate-Distortion Operating Points, each is represented by MFU index, which references a flat mdat byte file and associated QoS parameters, like average rate, peak rate, and QoE parameters signalling the spatiotemporal quality of the segment conforming to the ISOBMFF quality metrics. Notice that the MFU index is internal to the cache operation and an API should be provided to retrieve associated MFUs from the cache host. This sequence is illustrated in Figure 52.

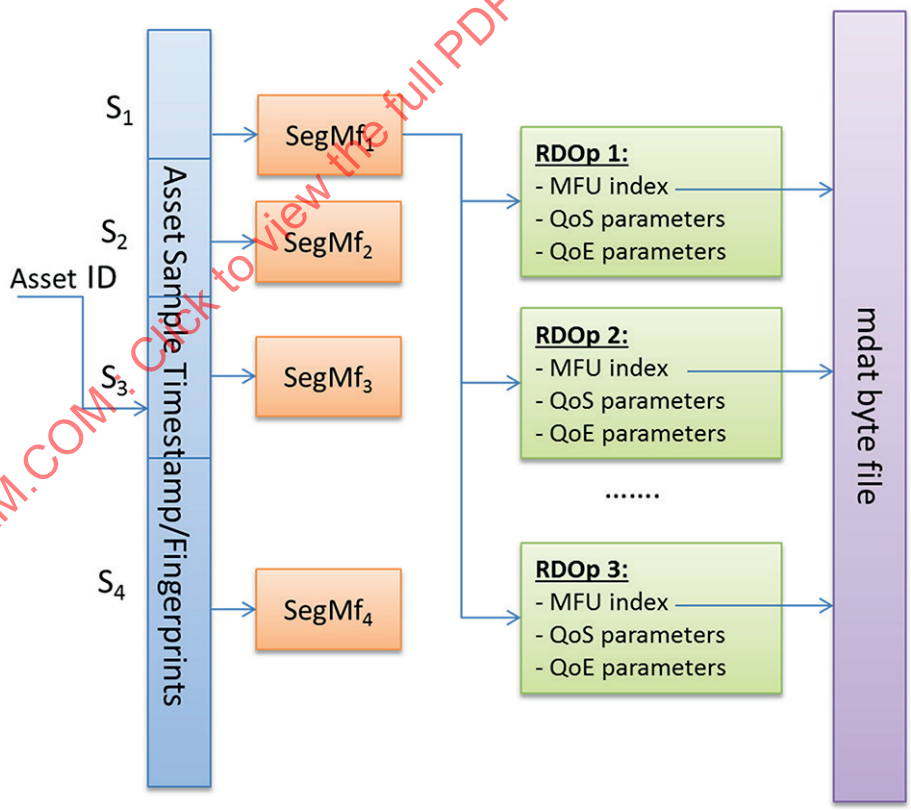


**Figure 52 — MMT cache illustration**

## 6.6 Usage of ADC signalling message

### 6.6.1 General

This subclause provides MMT implementation information, what MMT sending entity should do and MANE can do to gain a benefit from ADC message.

### 6.6.2 Operation in MMT sending entity

MMT sending entity can send the ADC information of an Asset to MANEs and receiving entities by ADC signalling messages. The ADC message is made from the ADC parameters by MMT sending entity for Assets within an MMT packet flow.

To provide more accurate information, MMT sending entity can update ADC message with parameter values that describes current Asset characteristics best and transmit it as tagged with newer *version* value. The update of values in ADC message can be done periodically or aperiodically and will be sent by MMT sending entity. Then, valid period of updated version of ADC should also be provided to help MANE router estimate how long those ADC message is valid and meaningful.

### 6.6.3 Operation in MANE router

MANE router can make more efficient utilization of network resources through ADC signalling messages. The MANE router identifies which ADC message is describing for which Asset by comparing *packet_id* within MMTP packet header and ADC message. By checking the change of *version* field in an ADC message, the MANE can acknowledge that there is an update of delivery characteristics of a corresponding Asset which will be expected to be valid until *validity_duration*.

The ADC describes QoS requirements and statistics of Assets for delivery. Even though the MMTP packet header has some QoS-related fields, it provides only packet level information on how to handle it. However, the ADC message can provide much more additional information than described in packet header, such as peak rate and required buffer sizes, etc., to MANE routers helping them expect delivery characteristics in media level, so it can provide a bigger picture of delivery characteristics about an Asset. By inspecting ADC messages, the MANE router can guess burstiness characteristics of each Asset from *peak_rate* and *sustainable_rate* fields in advance, which is hardly provided by packet header. Then, the MANE router can estimate how much buffers the MANE should secure to absorb burstiness of that Asset during *validity_duration*.

Moreover, the MANE router can characterize actually required network resources based on the corresponding updated ADC messages for an MMT flow which is comprised of flow of multiple Assets. In case of the delivery of Assets having delay constraints, network resources between media source and destination can be reserved based on its peak rate. However, it may result in inefficient bandwidth usage because fixed bandwidth resource is dedicated to certain traffic while it is time-varying. The MANE can estimate actually how much portion of those reserved resources will not be or is not being used from ADC message. Then, the MANE can make use of those tentatively idle resource for other traffics if available, by its own decision.

### 6.6.4 Example operation in MMT receiving entities

The received ADC can help MMT receiving entities to estimate the amount of buffers they should actually secure for each Asset (MMT subflow). As ADC information is provided with update, MMT receiving entity can also be updated with new buffer sizes to secure.

### 6.6.5 QoE multiplexing gain and bottleneck coordination

When multiple video sessions are sharing a congested link, the situation is called bottleneck. Dealing with bottleneck is the central theme of the multimedia transport, as when network is not congested, all traffic QoS requests can be fulfilled, then there is no need for traffic engineering.

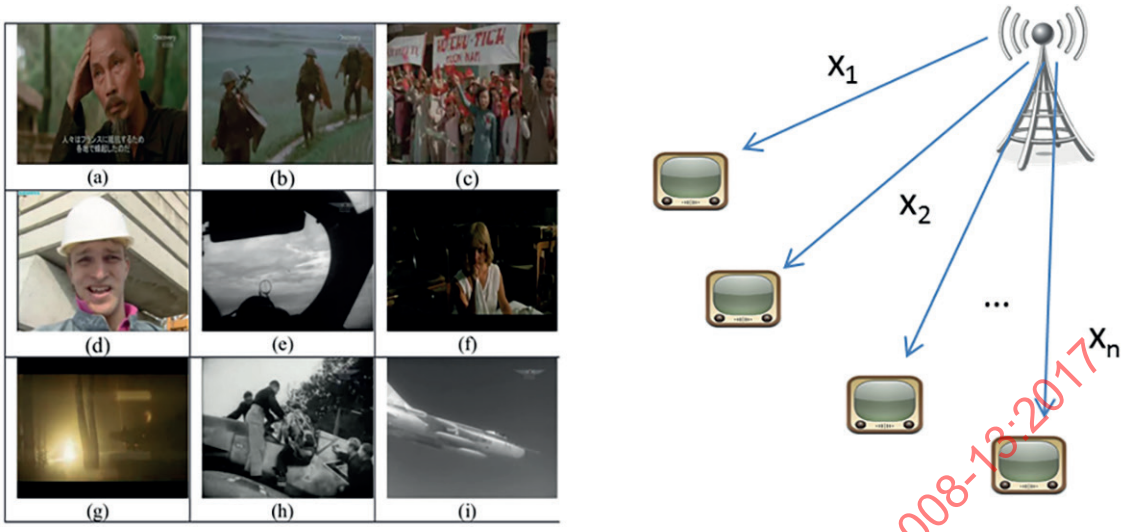


**Figure 53 — Bottleneck QoE multiplexing gain problem**

The problem is illustrated in Figure 53, multiple video traffics are sharing a capacity-constrained link, $\sum_k x_k \le C$, which is smaller than the total throughput required by the traffic, $\sum_k x_k > C$. The total resource constraint, $C$, can be estimated and provided from some sort of network nodes. For example, it can be an entity in the middle of the network such as router in wired network environment case and cellular base station in wireless case, etc. How to adapt each individual video stream, to meet this capacity constraint, while achieving the best QoE possible for all users is the objective. This problem is termed as QoE multiplexing gain problem.

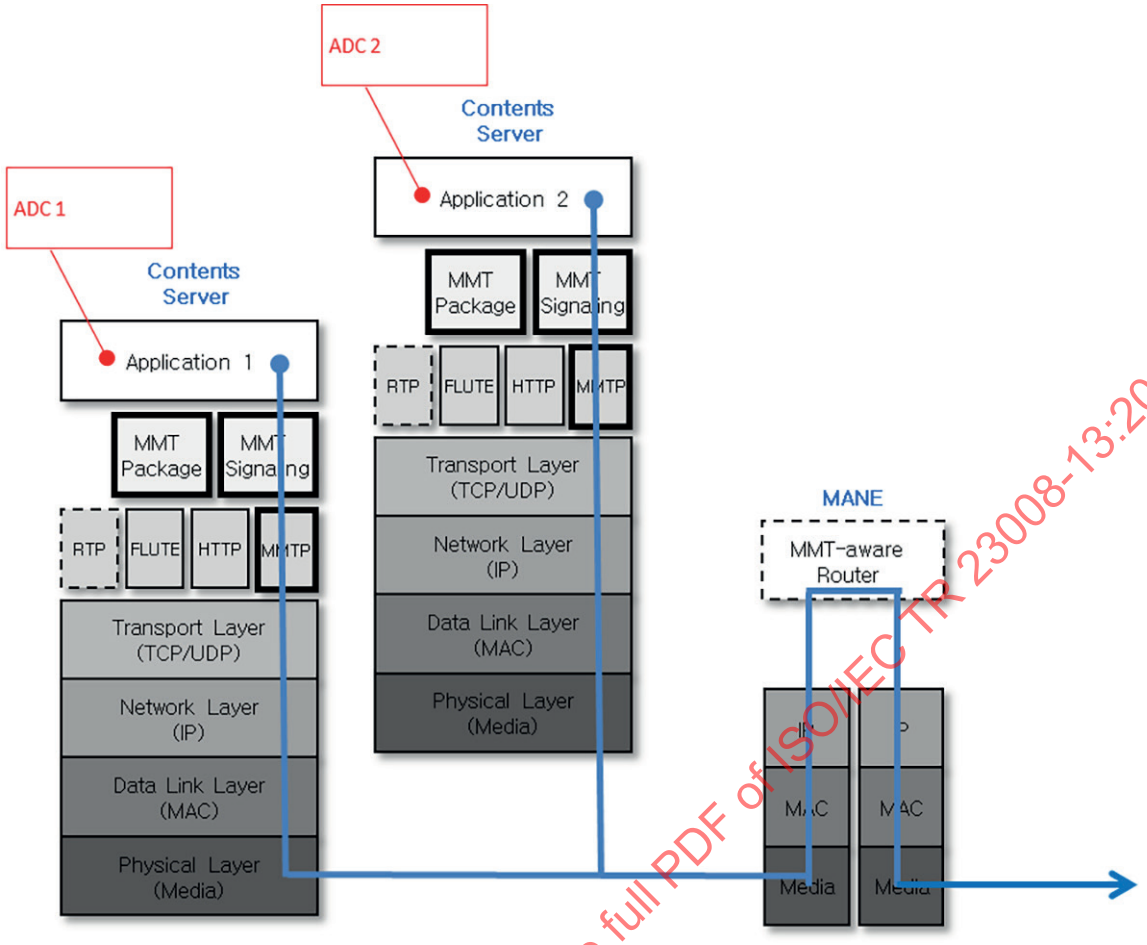


**Figure 54 — MANE-based bottleneck coordination**

To coordinate the transmission of video streams sharing, the bottleneck link, a new MMT function of Bottleneck Traffic Orchestrator (BTO) is introduced. The BTO reads the ADC information of each video bitstream at the bottleneck and construct the rate reduction-distortion incurred table and compute the pruning index for each video session. This is supported by the multiple operating point ADC supported in MMT. Example of the rate reduction-distortion (RD) table computed for the 4-s segments from the nine video sequences illustrated in Figure 53 are shown in Figure 55.

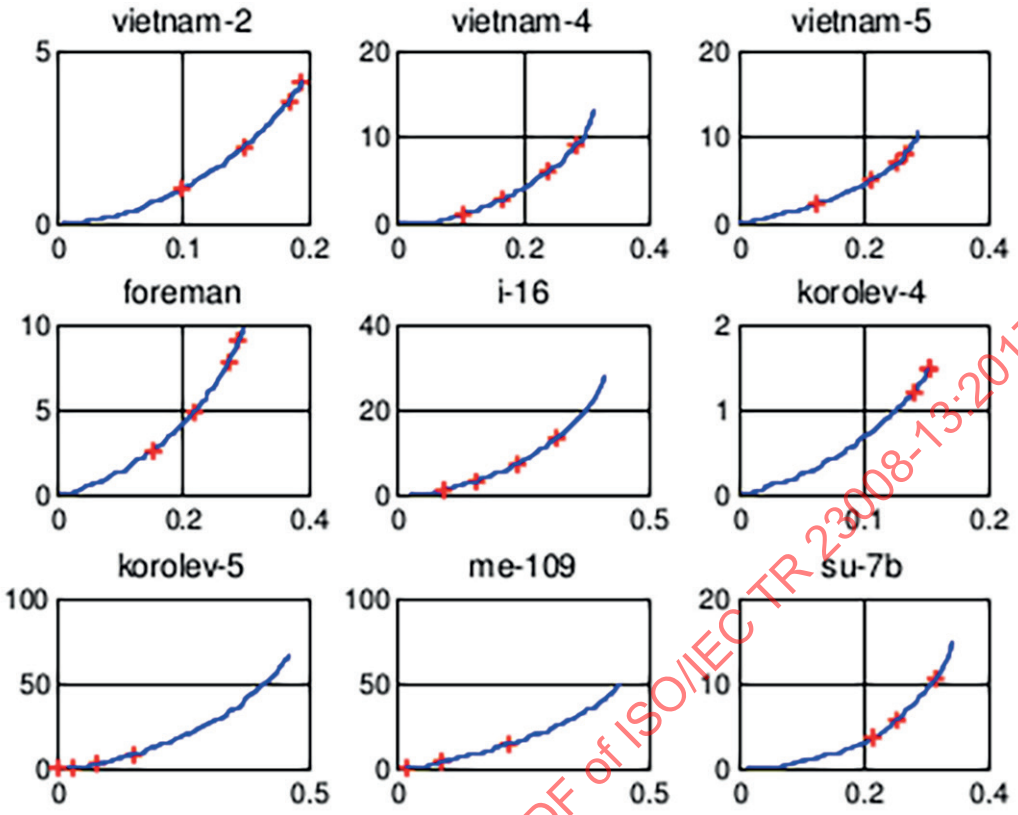


**Figure 55 — Temporal distortion from frame drops and rate reduction plots**

The syntax of signalling the rate-distortion trade-offs is specified in the MMT ADC; see ISO/IEC 23008-1.

The MMT aware routers, which can detect congestions and interpret the QoS/QoE operating points in the ADC messages, will act as the BTO. This is illustrated in Figure 55. The MANE node with MMT aware router first detects the congestion from the transport layer, gives an estimation of the approximate throughput reduction rate necessary to maintain QoS requirement and requests QoS operating points from applications via ADC messages. Upon receiving these operating points, a utility maximization via gradient search is conducted to compute the optimal MFU drop for each application at the bottleneck and to conduct the thinning at MANE node.

There are a number of strategies the BTO can apply to achieve QoE multiplexing gains at the bottleneck. It can minimize the average distortion from pruning streams. See Formula (8).

$$\min_{x_1, x_2, \ldots x_n} \sum_k D_k(x_k), s.t., \sum_k R_k(x_k) \le C \tag{8}$$

where $x_k$ is the ADC operating point available for stream $k$, which is associated with a resulting frame loss induced distortion $D_k(x_k)$ and aggregated reduced bit rate of $R_k(x_k)$. The rate and distortion function for each stream, $\{D_k(), R_k()\}$, are carried in the ADC of each stream. The optimal solution to the formula above can be easily found by a search on the ADCs of the bottlenecked streams. Upon computing the optimal operating points $\{x_1^*, x_2^*, \ldots, x_n^*\}$, they are communicated to the bottleneck MANE node and the streams buffered are pruned to avoid congestion. Simulations on bottleneck coordination are performed on the nine sequences shown in Figure 53. The resulting playback indeed demonstrated the graceful degradation of the quality when the bottleneck deficit increases, and in particular, the "easy" sequences helping out "busy" sequences in sharing the bottleneck resources. Video clips are available for subjective evaluation.

## 6.7 MMT deployment in Japanese broadcasting systems

### 6.7.1 General

The Association of Radio Industries and Businesses (ARIB) has published its standard STD-B60 "MMT-based media transport scheme in digital broadcasting systems", which describes the media transport and multiplexing layers in broadcasting systems using MMT. It specifies the usage and extensions of ISO/IEC 23008-1 for broadcasting services. The extensions include the MMTP packet header and additional signalling information. Some of the signalling information used in the MPEG-2 TS-based broadcasting systems has been transplanted to the MMT-based broadcasting system.

This subclause gives implementation examples of Japanese broadcasting system based on MMT.

### 6.7.2 Broadcasting systems using MMT

#### 6.7.2.1 System structure

This subclause describes the general structure of MMT-based broadcasting systems. Figure 56 shows the protocol stack of MMT-based broadcasting systems.

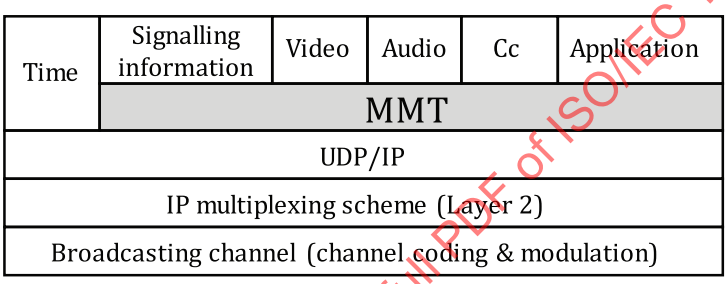


**Figure 56 — Protocol stack of MMT-based broadcasting systems**

In these systems, media components, such as video, audio and closed captions (cc) constituting a TV programme, are encapsulated into Media Fragment Units (MFUs)/Media Processing Units (MPUs). They are carried as MMT Protocol (MMTP) payloads of MMTP packets and delivered in IP packets. Data applications that are related to a TV programme are also encapsulated into MFUs/MPUs, carried in MMTP packets, and delivered in IP packets.

IP packets generated like this are multiplexed over broadcasting channels with an IP multiplexing scheme, also referred to as a layer 2 (L2) protocol, e.g. the TLV multiplexing scheme described in Recommendation ITU-R BT.1869.

The systems also have MMT signalling information (MMT-SI). MMT-SI is signalling information on the structure of a TV programme and associated information on TV services like the Electronic Programme Guide (EPG). MMT-SI is carried in MMTP packets and delivered in IP packets.

In order to provide Coordinated Universal Time (UTC) in broadcasting systems, time information is also delivered in IP packets.

#### 6.7.2.2 Service configuration in a broadcasting channel

ISO/IEC 23008-1 specifies the MMT Package as a logical structure of content. The MMT Package includes presentation information and associated Assets that constitute content.

A broadcasting service is generally a series of TV programmes. In MMT-based broadcasting systems, one MMT Package corresponds to one broadcasting service. The relationship between the broadcasting service and the MMT Package is shown in Figure 57. As shown in Figure 57, one TV programme is distinguished from the rest of the service by its start and end times and corresponds to one event.

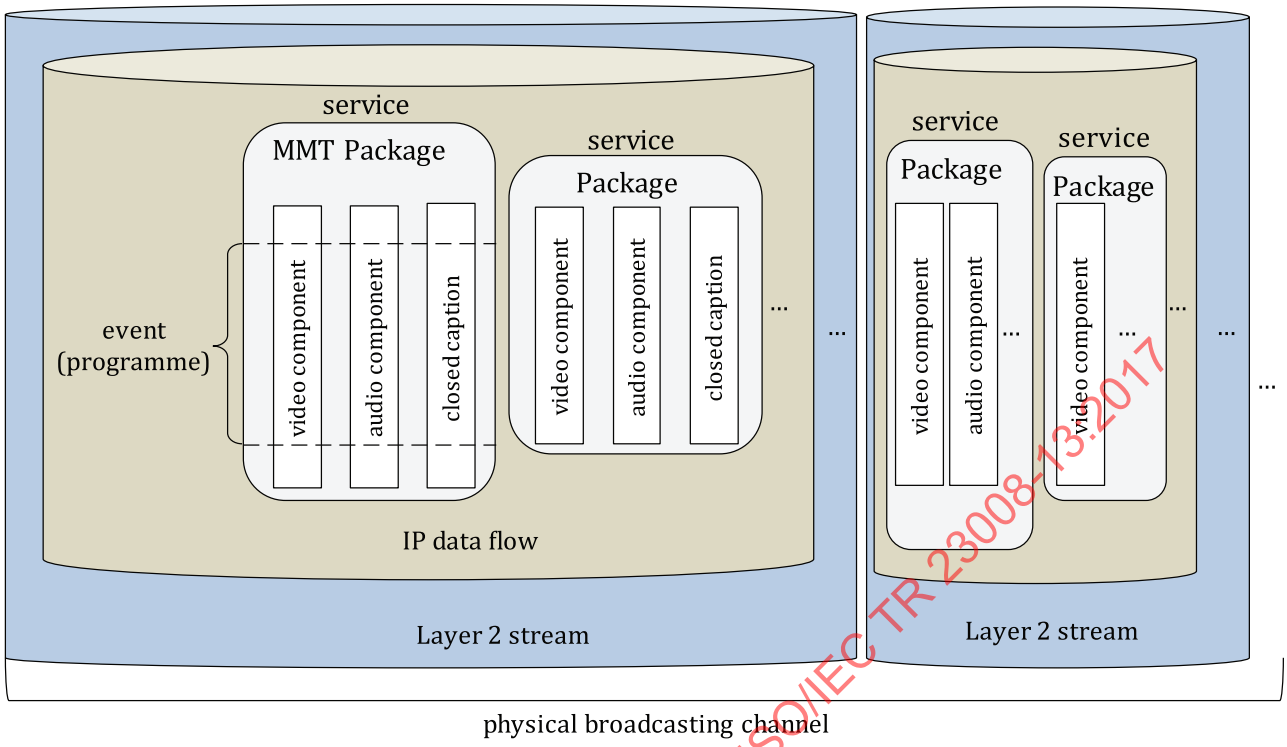


**Figure 57 — Relationship between a broadcasting service and MMT package**

In ISO/IEC 23008-1, an Asset is defined as a media component. An Asset is equivalent to a series of MPUs. In MMT-based broadcasting systems, one TV programme is an MMT Package including one or more Assets and signalling information. A Package Access (PA) Message is an MMT-SI and the MMT Package Table (MPT) carried in the PA Message identifies Assets constituting the TV programme.

Multiple MMT Packages can be delivered in one IP data flow as shown in Figure 57. Here, an IP data flow is defined as a sequence of IP packets of which the source IP address, destination IP address, protocol, source port number and destination port number are the same combination. There may be other IP data flows carrying content for download services or extended services in addition to IP data flows carrying MMT Packages.

Multiple IP data flows might be multiplexed into one layer 2 stream. The layer 2 stream includes signalling information for demultiplexing IP packets from broadcasting signals.

#### 6.7.2.3 Service configuration in broadcasting channels and broadband networks

ISO/IEC 23008-1 has been developed to support the delivery of media data over heterogeneous networks including broadcasting channels and broadband networks. In the MMT specifications, broadcasting channels and broadband networks can be treated in the same way for delivery of content. Figure 58 shows a service configuration using both broadcasting channels and broadband networks.

In Figure 58, video component 1, audio component 1 and closed caption 1 are delivered on broadcasting channels. In addition to these components, video component 2, audio component 2 and closed caption 2 are delivered on broadband networks.

In the broadcasting channels, the three components are multiplexed into one IP data flow and delivered in one layer 2 stream since all transmitted information are delivered to all receiver terminals. On the other hand, in the broadband networks, components are delivered as a separate IP data flow since each component is delivered to the receiver terminal requesting it.

In MMT-based broadcasting systems, media components delivered in different channels can easily be included in one MMT Package. MMT-based broadcasting systems support hybrid delivery of multimedia content.

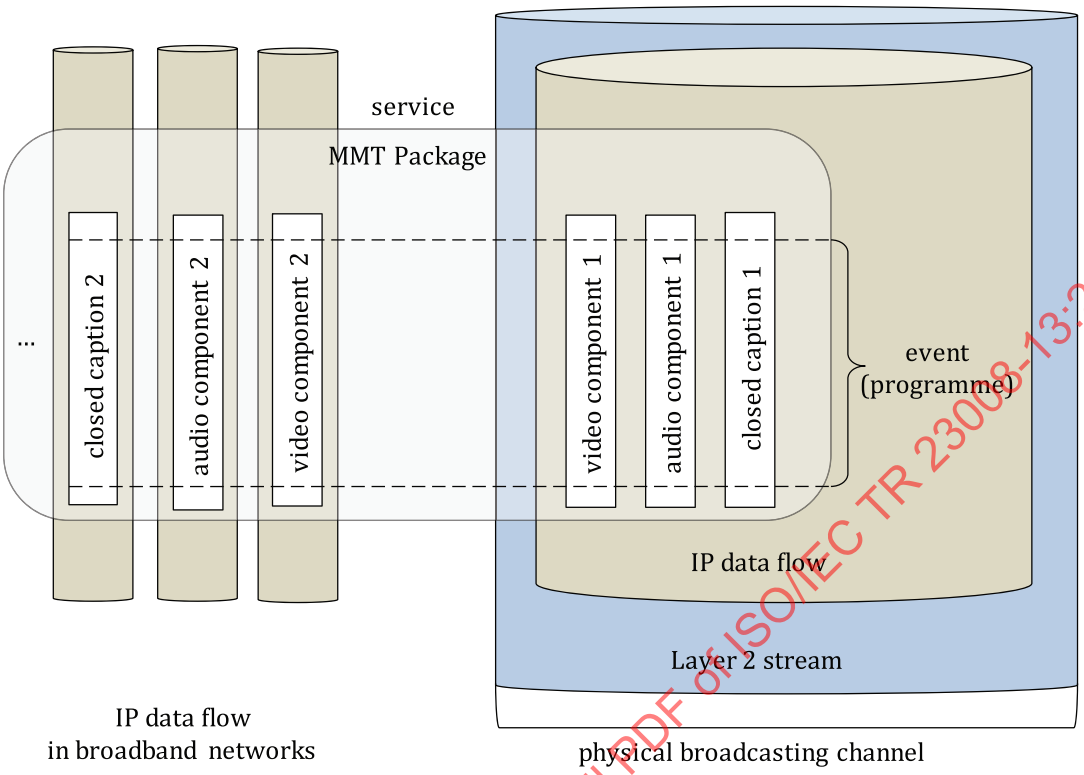


**Figure 58 — Service configuration over both broadcasting channels and broadband networks**

### 6.7.3 Media transport protocol

#### 6.7.3.1 General

MMT-based broadcasting systems use the syntax and semantics of the MMTP payload and MMTP packet specified in ISO/IEC 23008-1. The extensions described below are applied.

#### 6.7.3.2 Header extension of MMTP packets

ISO/IEC 23008-1 specifies a header extension in the MMTP packet. The header extension has three fields: extension_type, extension_length and header_extension_value. Although the header extension can be used for various purposes, it contains only one piece of information. The multi-type header extension described below enables it to contain multiple pieces of information.

**header_extension_value** – When the extension_type field is set to 0x0000, this field has the structure shown in Table 3.

**Table 3 — Structure of multi-type header extension**

| Syntax | Number of bits | Mnemonic |
|---|---|---|
| Header_extension_value { | | |
| for (i=0; i<N; i++) { | | |
| **hdr_ext_end_flag** | 1 | bslbf |
| **hdr_ext_type** | 15 | uimsbf |
| **hdr_ext_length** | 16 | uimsbf |
| for (j=0; j<M; j++) { | | |
| **hdr_ext_byte** | 8 | bslbf |
| } | | |
| } | | |
| } | | |

**hdr_ext_end_flag** – When this flag is set to "1", this multi-type header extension is the end of the header extension. When this flag is set to "0", this multi-type header extension is not the end of the header extension.

**hdr_ext_type** – This field specifies the type of multi-type header extension. The value of hdr_ext_type is specified in Table 4.

**Table 4 — Hdr_ext_type values**

| Value | Description |
|---|---|
| 0x0000 | reserved for future use |
| 0x0001 | reserved for ARIB STD-B61 (scrambling information) |
| 0x0002 | reserved for ARIB STD-B60 (download_id) |
| 0x0003 – 0x7FFF | reserved for future use |

**hdr_ext_length** – This field specifies the number of bytes of the following hdr_ext_byte field.

**hdr_ext_byte** – This field provides information on multi-type header extension.

#### 6.7.3.3 Encapsulation of multimedia data

##### 6.7.3.3.1 General

In order to improve the inter-operability of MMT-based broadcasting systems, the following constraints apply to carriage of multimedia data in MMTP packets.

##### 6.7.3.3.2 Encapsulation of video data

###### 6.7.3.3.2.1 MFU format for HEVC stream

When a High Efficiency Video Coding (HEVC) stream is carried in the MMT protocol, input to the MMT process is a sequence of Network Abstraction Layer (NAL) units. A NAL unit is encapsulated into an MFU when an HEVC stream is carried in the MMT protocol.

If an HEVC encoder generates the byte stream format specified in Rec. ITU-T H.265|ISO/IEC 23008-2:2015, Annex B, one start code prefix (0x000001) followed by one NAL unit is replaced with 32-bit length information of the NAL unit (unsigned integer format). Namely, the NAL unit together with the length information are encapsulated into one MFU.

Figure 59 shows an overview of generating MMTP packets and MFUs from a sequence of NAL units output from an HEVC encoder.

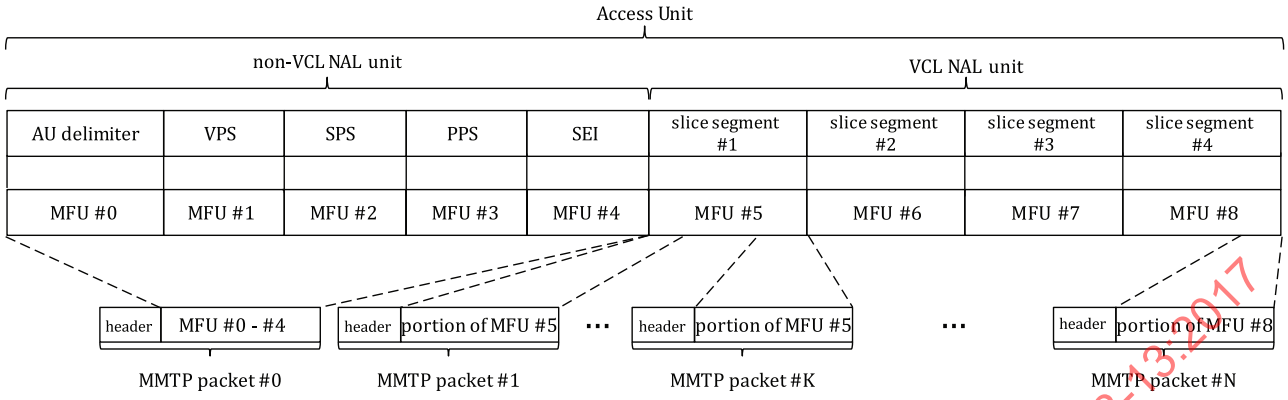


**Figure 59 — Overview of packetization of NAL units of HEVC streams**

The duration of video MPU greatly influences the channel change time at the receiver terminal, since the video stream is decoded and presented at the receiver terminal on a per-MPU basis. In order to reduce the channel change time, the MPU of an HEVC stream is constructed in Intra Random Access Point (IRAP) intervals.

#### 6.7.3.3.2.2 Encapsulation of HEVC bitstream subsets

HEVC supports temporal sublayer coding. One example is that when a 120-Hz video is encoded, two streams can be generated: one is a sub-bitsream for 60-Hz video, the other is a bitstream subset for 120-Hz video. At the receiver terminal, 60-Hz video can be decoded from the sub-bitstream and 120-Hz video can be decoded from both the sub-bitstream and the bitstream subset.

Figure 60 shows an overview of encapsulation of HEVC bitstream subsets. Note that this figure shows display order frame sequence. When an MMT Package is made up of various media components, the sub-bitstream and the bitstream subset are encapsulated into separate Assets. In Figure 60, the sub-bitstream is encapsulated into Asset 1 and the bitstream subset is encapsulated into Asset 2. Since they are separate Assets, the access units of Asset 1 and Asset 2 are carried in MMTP packets that have different packet IDs.

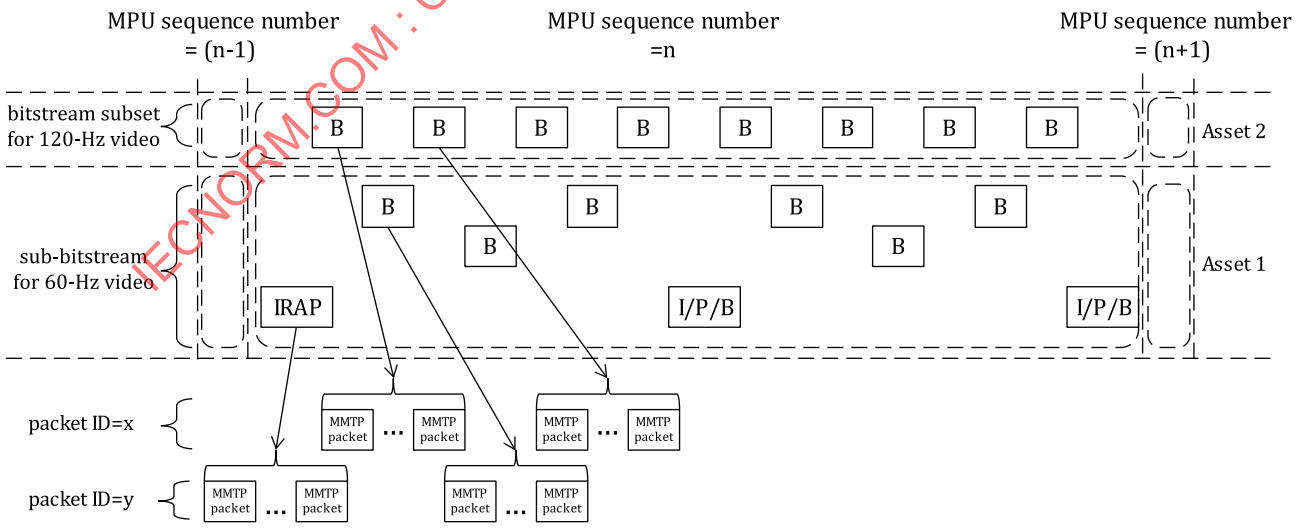


**Figure 60 — Overview of encapsulation of HEVC sub-bitstream and bitstream subsets for temporal sublayer coding**

The sequence number of an MPU that the access units of the bitstream subset belong to is identical to the sequence number of an MPU that the access units of the sub-bitstream belong to in the same time period. Assigning the same sequence number to both MPUs enables receiver terminals to easily identify the MPUs that includes corresponding access units in the same GOP.

In the example shown in Figure 60, the decoding of Asset 2 depends on Asset 1. A Dependency Descriptor stating that Asset 2 depends on Asset 1 is inserted in the asset_descriptors_byte field of the MP Table. In addition to the Dependency Descriptor, an MPU Timestamp Descriptor and MPU Extended Timestamp Descriptor are inserted in the asset_descriptors_byte fields of both Asset 1 and Asset 2.

#### 6.7.3.3.2.3 Clean random access of HEVC bitstream (for SAP Type 3)

For enabling a clean random access of SAP Type 3 in HEVC bitstream, the following operation in system-level is recommended.

When a random access occurred at the random access point with a Clean Random Access (CRA) or BLA (Broken Link Access) picture in an HEVC bitstream, the associated RASL pictures are discarded from the HEVC bitstream before being fed into an HEVC decoder. The NAL units of the CRA/BLA and the RASL pictures can be identified by NAL unit type (NUT), which is signalled in NAL unit header (NUH) or in HEVCDecoderConfigurationRecord of HEVCConfigurationBox defined in Reference [5]. When the NUTs of NAL Units at the random access point are CRA, the variable HandleCraAsBlaFlag of the HEVC decoder is set equal to 1 or the NUHs of the NAL Units of the CRA picture are rewritten by changing the values of NUTs from 21 to 17 in the HEVC bitstream. This is because the CRA picture should be handled as a BLA picture in the HEVC decoder, when a random access has occurred. The decoded picture buffer (DPB) of the HEVC decoder is refreshed before the CRA/BLA picture is decoded.

#### 6.7.3.3.2.4 Fast random access of HEVC bitstream (for SAP Types 2 and 3)

For enabling a fast random access of SAP Types 2 and 3 in HEVC bitstream, the following operation in system-level is recommended.

When a random access has occurred at a random access point with an IRAP picture, the presence of leading pictures is inspected by parsing the NUTs of NAL units associated with the IRAP pictures in the HEVC bitstream. The NUT is signalled in NAL unit header (NUH) or in the HEVCDecoderConfigurationRecord of HEVCConfigurationBox defined in Reference [5]. If the associated leading pictures are present in the HEVC bitstream, all the associated NAL units marked as RADL (NUT 6, 7) and RASL (NUT 8, 9) are discarded from the HEVC bitstream. After the IRAP picture is decoded, the decoded picture of the IRAP is immediately displayed by ignoring its original presentation time and repeatedly displayed until the following trail picture is presented. The decoded picture buffer (DPB) of the HEVC decoder is refreshed before the IRAP picture is decoded.

According to ISO/IEC 23008-2:2015, 6.6.2.3.2.4, to support the use of HEVC bitstream with SAP type 2 or 3 for MPU, the following operations should be done by MMT Server to create MMTP bitstream.

— The value of the `R` field of MMTP packets carrying MFUs of the pictures preceding the first IRAP picture of a movie fragment in decoding order, the MMTP packets whose value of the `movie_fragment_sequence_number` field is same with the value of the same field of the MMTP packets carrying MFUs of the IRAP picture but the value of the `sample_number` field is smaller than the value of the same field of the MMTP packets carrying MFUs of the IRAP picture, is set to 0.

— The value of the `R` field of MMTP packets carrying MFUs of the IRAP picture of a movie fragment is set to 1.

The following operation should be done by MMTP decapsulation buffer for each MMTP packets carrying MFUs of HEVC bitstream; the value of the `FT` field is equal to 2 to support use of HEVC bitstream with SAP type 2 or 3.

— Step 1: When the MMTP packets whose value of the `movie_fragment_sequence_number` field is different from the value of the same field of the preceding MMTP packets is firstly received, such MMTP packets and all succeeding MMTP packets whose value of the `R` field is equal to 0 are

immediately deleted from the MMTP decapsulation buffer and they are not delivered to the decoder buffer until the MMTP packets whose value of the movie_fragment_sequence_number field is same with the value of the same field of the previously received MMTP packet but the value of the R field is equal to 1 is received.

— Step 2: When the MMTP packets whose value of the R field is equal to 1 is firstly received as described in step 1, such MMTP packet and all succeeding MMTP packets carrying MFUs belong to same AU are processed by the MMTP decapsulation buffer and delivered to the decoder buffer.

— Step 3: After receiving MMTP packets, whose value of the R field is equal to 1 is processed as described in the step 2, all succeeding MMTP packets carrying MFUs, whose value of the movie_fragment_sequence_number field is same with the value of the same field of the MMTP packets described in the step 2, are immediately deleted from the MMTP decapsulation buffer and they are not delivered to the decoder buffer until the MMTP packet, whose value of the movie_fragment_sequence_number field is same with the value of the same field of the MMTP packets described in the step 2 but the value of the dep_counter field is 0, is received. Such MMTP packet is also immediately deleted from the MMTP decapsulation buffer and it is not delivered to the decoder buffer.

— Step 4: After receiving the MMTP packet whose value of the dep_counter field is 0 as described in the step 3, following operation to all succeeding MMTP packets carrying MFUs whose value of the movie_fragment_sequence_number field is same with the value of the same field of the MMTP packets described in the step 3 are applied.

  — If the value of the f_i field of the MMTP packet is equal to 00, all data units of such MMTP packet whose value of the dep_counter field is 0 is immediately deleted from the MMTP decapsulation buffer and they are not delivered to the decoder buffer until any data unit whose value of the dep_counter field is not equal to 0 is found.

  — If the value of the f_i field of the MMTP packet is not equal to 00, then this MMTP packet and all succeeding MMTP packets are stored in the MMTP decapsulation buffer and not process until the MMTP packet whose value of the f_i field is equal to 11 is received. If MMTP packets whose value of the f_i field is equal to 11 is received and whose value of the dep_counter field is equal to 0, then all packets stored in the MMTP decapsulation buffer whose value of movie_fragment_sequence_number field and the value of the sample_number field is the same with such packet are immediately deleted from the MMTP decapsulation buffer and they are not delivered to the decoder buffer.

  — If the value of the f_i field of the MMTP packet is not equal to 00, then this MMTP packet and all succeeding MMTP packets are stored in the MMTP decapsulation buffer and not process until the MMTP packet whose value of the f_i field is equal to 11 is received. If MMTP packets whose value of the f_i field is equal to 11 is received and whose value of the dep_counter field is not equal to 0, then all packets stored in the MMTP decapsulation buffer whose value of movie_fragment_sequence_number field and the value of the sample_number field is same with such packet are processed by the MMTP decapsulation buffer and delivered to the decoder buffer.



#### 6.7.3.3.3 Encapsulation of audio data: MFU format for MPEG-4 AAC and MPEG-4 ALS

When an MPEG-4 Advanced Audio Coding (AAC) stream or MPEG-4 Audio Lossless Coding (ALS) stream is carried in the MMT protocol, input to the MMT process is in the form of either LATM/LOAS stream or a data stream.

The Low Overhead Audio Transport Multiplex (LATM) includes an audio channel configuration and provides multiplexing functions for audio data. The Low Overhead Audio Stream (LOAS) provides synchronization for audio data. When an audio encoder generates a LATM/LOAS stream, one AudioMuxElement () specified in ISO/IEC 14496-3 is encapsulated into one MFU.

When an audio encoder generates a data stream, a Raw Data Stream is encapsulated into one MFU.

### 6.7.4 Signalling information

#### 6.7.4.1 General

There are three kinds of MMT signalling information: message, table and descriptor. Some of the signalling information specified in ISO/IEC 23008-1 is not used in broadcasting systems. This subclause summarizes the signalling information essential to broadcasting systems. Additional signalling information may be used in broadcasting systems.

#### 6.7.4.2 MMT signalling information messages

##### 6.7.4.2.1 List of MMT signalling information messages

Table 5 shows the list of messages.

**Table 5 — List of messages**

| Message name | Message_id assignment | Description | Specified in ISO/IEC 23008-1 | Use in broadcast-ing systems |
|---|---|---|---|---|
| Package Access (PA) Message | 0x0000 | Is the entry point of MMT-signalling information. Conveys one or more tables. | X | X |
| Media Presentation Information (MPI) Message | 0x0001 – 0x000F | Conveys a presentation information document. | X | |
| MMT Package Table (MPT) Message | 0x0010 – 0x001F | Conveys a whole or a subset of an MP table. | X | |
| Clock Relation Information (CRI) Message | 0x0200 | Conveys clock-related information to be used for mapping between the NTP timestamp and MPEG-2 STC. | X | |
| Device Capability Information (DCI) Message | 0x0201 | Conveys information on required device capabilities for the Package consumption. | X | |
| Application Layer-Forward Error Correction (AL-FEC) Message | 0x0202 | Conveys configuration information of an AL-FEC scheme to be used to protect Asset. | X | |
| Hypothetical Receiver Buffer Model (HRBM) Message | 0x0203 | Conveys information on end-to-end transmission delay and memory requirement to a receiving terminal. | X | |
| M2section Message | 0x8000 | Conveys the MPEG-2 Section-format table. Tables and Descriptors in MPEG-2 TS based conventional broadcasting systems can be reused by this message. | | X |



##### 6.7.4.2.2 Detailed specifications of messages

###### 6.7.4.2.2.1 PA Message

The syntax and semantics of the PA Message are specified in ISO/IEC 23008-1.