INTERNATIONAL ORGANIZATION FOR STANDARDIZATION

ORGANISATION INTERNATIONALE DE NORMALISATION

ISO/IEC JTC1/SC29/WG11

CODING OF MOVING PICTURES AND AUDIO

ISO/IEC JTC1/SC29/WG11

MPEG/N13293

Geneva, Switzerland, January 2013

Title	**Text of ISO/IEC 2nd CD 23008-1 MPEG Media Transport**
Status	**CD**
Source	**MPEG-H Systems**
Editors	**Kyungmo Park, Youngkwon Lim (Samsung Electronics Co., Ltd), Shuichi Aoki (NHK), Gerard Fernando (ZTE), Jin Young Lee (ETRI)**

Document type: International StandardInternational Standard
Document subtype:
Document stage: (30) Committee(30) Committee
Document language: EE

STD Version 2.1c2STD Version 2.1c2

ISO/IEC JTC 1/SC 29 N13293

Date: 2013-02-07

ISO/IEC CD 23008-1~~ISO/IEC CD 23008-1~~

ISO/IEC J~~ISO/IEC J~~TC 1~~1~~/SC 29~~29~~/WG 11~~11~~

Secretariat:

Information technology — High efficiency coding and media delivery in heterogeneous environments — Part 1: MPEG media transport (MMT)~~Information technology — High efficiency coding and media delivery in heterogeneous environments — Part 1: MPEG media transport (MMT)~~

Élément introductif — Élément central — Partie 1: Titre de la partie~~Élément introductif — Élément central — Partie 1: Titre de la partie~~

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.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

The main task of the joint technical committee is to prepare International Standards. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

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.

ISO/IEC 23008-1 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 29, *Coding of audio, picture, multimedia and hypermedia information*.

ISO/IEC 23008 consists of the following parts, under the general title *Information technology — High efficiency coding and media delivery in heterogeneous environments*:

— *Part 1: MPEG media transport (MMT)*

— *Part 2: High efficiency video coding (HEVC)*

— *Part 3: 3D Audio*

Introduction

This part of ISO/IEC 23008 specifies technologies for the delivery of coded media data for multimedia services over concatenation of heterogeneous packet based network segments including bidirectional IP networks and unidirectional digital broadcasting networks.. In this specification, “coded media data” includes both timed audiovisual media data requiring synchronized decoding and presentation of each specific unit of data at a designated time, and non-timed data that could be decoded and presented at an arbitrary time based on the context of the service or the user interaction.

MMT is designed under the assumption that the coded media data will be delivered through a packet based delivery network. Several characteristics of such delivery environments have been taken into consideration such as the non-constant end-to-end delivery delay of each packet from the sending entity to the receiving entity and the means to distinguish signaling messages from the media data provided by the underlying network.

For efficient and effective delivery of coded media data over heterogeneous packet based delivery networks, this specification provides the following elements:

— the logical model to construct content composed of components from various sources, for example for mash-up applications;

— the structure of data conveying information about the coded media data for the delivery layer processing such as packetization and adaptation;

— the packetization method and the structure of the packet to deliver media content over packet-based delivery network supporting hybrid multichannel delivery agnostic to the specific type of media or coding method;

— the format of signaling messages to manage presentation and delivery of media content;

— the format of information to be exchanged across the delivery network layers to facilitate cross layer communication.

Information technology — High efficiency coding and media delivery in heterogeneous environments — Part 1: MPEG media transport (MMT)Information technology — High efficiency coding and media delivery in heterogeneous environments — Part 1: MPEG media transport (MMT)

1 Scope

This part of ISO/IEC 23008 specifies MPEG Media Transport (MMT), a single encapsulation format, delivery protocol, signaling messages and a method describing composition information for delivery of multimedia services over packet-based heterogeneous networks. Types of packet-based networks supported by this specification include bidirectional networks such as IP networks and unidirectional networks such as digital broadcast networks regardless of their use of IP. It is required for entities to be synchronized with UTC, although the means by which this synchronization is achieved is outside the scope of this specification.

The technologies for encapsulation function specify the logical model and its associated encapsulation format for timed and non-timed media content. The encapsution format includes a self-contained encapsulation structure enabling independent consumption of media data, which hides codec specific details from the delivery function.

The technologies for delivery function specify the payload format agnostic to media and codec types, which allows fragmentation and aggregation of content encapsulated with this specification for delivery using packet oriented streaming protocols; it also provides an application layer transport protocol and associated QoS enabling mechanisms, which allows multiplexing, reliable delivery of media content and cross layer communication.

The technologies for signaling function specify the format of signaling messages carrying information for media content consumption such as specific location and delivery condition of media content.

The technologies for composition function specify the method associating content encapsulated with the format defined in this specification to the presentation and the method representing synchronization between timed and non-timed content.

2 Normative references

The following documents, in whole or in part, are normatively referenced in this document and are indispensable for its application. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO/IEC 14496-12, *Information technology – Coding of audio-visual objects – Part 12: ISO base media file format (technically identical to ISO/IEC 15444-12)*

IETF RFC 1738, *Uniform Resource Locators (URL)*, December 1994.

IETF RFC 2141, *URN Syntax*, May 1997.

IETF RFC 3550, *RTP: A Transport Protocol for Real-Time Applications*, July 2003.

IETF RFC 2327, *SDP: Session Description Protocol*, April 1998.

IETF RFC 5905, *NTPv4: Network Time Protocol Version 4*, June 2010.

IETF RFC 3406, *Uniform Resource Names (URN) Namespace Definition Mechanisms*, October 2002.

IETF RFC 3261, *SIP: Session Initiation Protocol*, June 2002.

IETF RFC 3986, *Uniform Resource Identifier (URI): Generic Syntax*, January 2005.

IETF RFC 4122, *A Universally Unique Identifier (UUID) URN Namespace*, July 2005.

W3C HTML *HyperText Markup Language (HTML) Version 4.1*, W3C Recommendation 24, Dec 1999.

W3C HTML5 *HyperText Markup Language (HTML) Version 5*, W3C Working Draft, 25 October 2012.

W3C Media Queries, W3C Recommendation 19 June 2012.

W3C XLINK *XML Linking Language (XLink) Version 1.1*, W3C Recommendation 06, May 2010.

W3C XML *Extensible Markup Language (XML) Version 1.0*, W3C Recommendation 26, Nov 2008.

3 Terms, definitions, symbols and abbreviated terms

For the purposes of this document, the following terms and definitions apply.

3.1 Terms and definitions

3.1.1

access unit
the smallest data entity to which timing information can be attributed, access unit (AU) is not defined for coded media data with any designated timing information for decoding and presentation.

3.1.2

ARQ feedback message
a message sent by the receiving entity when it detects packet loss.

3.1.3

ARQ packet section
original MMT packets that are retransmitted by the sending entity in response to an ARQ feedback message from the receiving entity.

3.1.4

ARQ signalling message
a signalling message is sent at the beginning of a session, and from this the receiving entity is able to determine the profile for the transmission mechanism, the feedback mechanism, and the timeout window.

3.1.5

asset

a data entity containing data with the same transport characteristics and that is composed of one or more MPUs with same Asset ID.

3.1.6

asset delivery characteristics
description about required Quality of Service (QoS) for delivery of Assets. ADC is represented by the parameters agnostic to specific delivery environment.

3.1.7

code rate
the ratio between the number of source symbols and the number of encoding symbols.

3.1.8

composition information
description of spatial and temporal relationship among Assets.

3.1.9

encoding symbol
unit of data generated by the encoding process. Where, source symbols are part of the encoding symbols.

3.1.10

encoding symbol block
set of encoding symbols from the encoding process of a source symbol block.

3.1.11

FEC code
an algorithm for encoding data such that the encoded data flow is resilient to data loss.

3.1.12

FEC encoded flow
a logical set of flows that consists of an FEC source flow and its associated one or more FEC repair flows.

3.1.13

FEC payload ID
an identifier that identifies the contents of a MMT packet with respect to the MMT FEC scheme.

3.1.14

FEC repair flow
a data flow carrying repair symbols to protect source flow.

3.1.15

FEC repair packet
MMT packet along with repair FEC payload ID to deliver one or more repair symbols of a repair symbol block.

3.1.16

FEC source flow
a flow of MMT packets protected by a single instance of MMT FEC scheme.

3.1.17

FEC source packet
MMT packet along with source FEC payload ID.

3.1.18

media fragment unit
a generic structure that is media codec agnostic and that contains independently decodable media data such as a complete or partial AU for timed media data by a media decoder or one file for non-timed media data.

3.1.19

media processing unit
a generic container for independently decodable timed or non-timed data, that is media codec agnostic and that contains one or more AUs for timed data or a portion of non-timed data as well as additional delivery and consumption related information. In this context, processing means encapsulation into a Package or packetization for delivery.

3.1.20

MMT entity
a software or hardware implementation compliant to a profile of MMT

3.1.21

MMT FEC scheme
a specification that defines the additional protocol aspects required to use FEC codes in MMT.

3.1.22

MMT packet
a formatted unit of the data generated or consumed according to the MMT protocol.

3.1.23

MMT payload
formatted unit of data to carry the Package or signaling message either using MMT protocol or Internet application layer transport protocols (e.g. RTP).

3.1.24

MMT protocol
an application layer transport protocol for delivering MMT payload over IP network.

3.1.25

non-timed data
any data that does not have inherent synchronization information for the decoding and/or presentation of its media units.

3.1.26

package
a logical collection of data, which is composed of one or more Assets and their related Asset Delivery Characteristics, and an Composition Information.

3.1.27

repair FEC payload ID
FEC payload ID specifically for use with repair packets.

3.1.28

repair symbol
encoding symbol that is not a source symbol.

3.1.29

repair symbol block
set of repair symbols which can be used to recover lost source symbols.

3.1.30

service data unit
a unit of data that has been conveyed from the transmitter to the receiver of the underlying network layer.

3.1.31

source FEC payload ID
FEC payload ID specifically for use with source packets.

3.1.32

source packet block
segmented set of FEC source flow that are to be protected as a single block.

3.1.33

source symbol
unit of data used during the encoding process.

3.1.34

source symbol block
set of source symbols generated from a single source packet block.

3.1.35

timed data
any data that has inherent synchronization information for the decoding and/or presentation of its media units.

3.2 Symbols and abbreviated terms

For the purpose of this document, the symbols and abbreviated terms given in the following apply:

AU	access unit
ADC	asset delivery characteristics
AL-FEC	application layer forward error correction
ARQ	automatic repeat request
AVC	advanced video coding
CAT	conditional access table
CI	composition information
CRI	clock relation information
DCI	device capability information
DSCP	differentiated services code point
ECM	entitlement control message
HEVC	high efficiency video coding
HRBM	hypothetical receiver buffer model
HTTP	hypertext transfer protocol
IDR	instantaneous decoding refresh
IBG	information block generation
ISOBMFF	ISO base media file format
LA-FEC	layer aware forward error correction
MCI	mmt composition information
MFU	media fragment unit
MMT	MPEG media transport
MMTP	MMT protocol
MPT	mmt package table
MPU	media processing unit
MTU	Maximum Transmission Unit
MVC	multi-view video coding

NAM	network abstraction for media
NTP	network time protocol
PA	package access
PCR	program clock reference
PES	packetized elementary stream
PID	packet identifier
PMT	program map table
PSI	program specific information
PTS	presentation time stamp
RAP	random access point
RTP	real-time protocol
RTCP	real-time control protocol
SIP	session initiation protocol
SDP	session description protocol
SVC	scalable video coding
TCP	transmission control protocol
TS	transport stream
UDP	user datagram protocol
URI	uniform resource identifier
URL	uniform resource locator
URN	uniform resource name
UUID	universally unique identifier
UTC	coordinated universal time
XML	extensible mark-up language

3.3 Conventions

The following conventions apply in this document:

— Big endian is used.

4 Overview

This specification defines a set of tools to enable the building of multimedia delivery services. The tools are spread over four different functional areas: composition, encapsulation, delivery and signalling. Even though the tools defined in each functional area are designed to be efficiently used together, the tools defined in one functional area may also be used independently regardless of the use of tools from the other functional areas.

The encapsulation functional area defines the logical structure of media content, the Package, and the format of the data units to be processed by the MMT entity and their instantiation with ISOBMFF. The Package specifies the components comprising the media content and the relationship among them to provide the necessary information for adaptive delivery. The format of the data units in this specification is defined to encapsulate the coded media either for storage or delivery, and to be able to easily converted between these two formats

The delivery functional area defines the application layer transport protocol and the payload format. The application layer transport protocol defined in this specification provides enhanced features for delivery of the Package compared to conventional application layer transport protocols for the delivery of multimedia, including multiplexing and cross layer communiation. The payload format is defined to enable the carriage of coded media data agnostic to the specific media type or encoding method.

The signaling functional area defines the format of messages to manage delivery and consumption of the Package. Messages for consumption management are used to signal the structure of the Package and messages for delivery management are used to signal the structure of the payload format and protocol configuration.

The composition functional area defines the presentation aspects of MMT encapsulated content. The composition function in this specification addresses delivery of synchronization information of the contents composed of various contents components where multiple device screens are used to present the contents, some parts of the component are replaced by others based on the context of the user such as location or personal preferences and presentation time of some components are not designated when the content is encapsulated but decided when the content is acutally consumped. To efficiently deliver flexible synchronization information supporting such contents, the unified method to synchronize timed and non-timed components by using widely adopted presentation language, HTML5 with some extensions, is introduced.

A multimedia service may use any subset of the tools defined in this specification according to their specific needs. Interfaces to other protocols and standards are either defined by this specification or may be defined. Figure 1 shows the different functions as well as their relationships to existing protocols and standards.

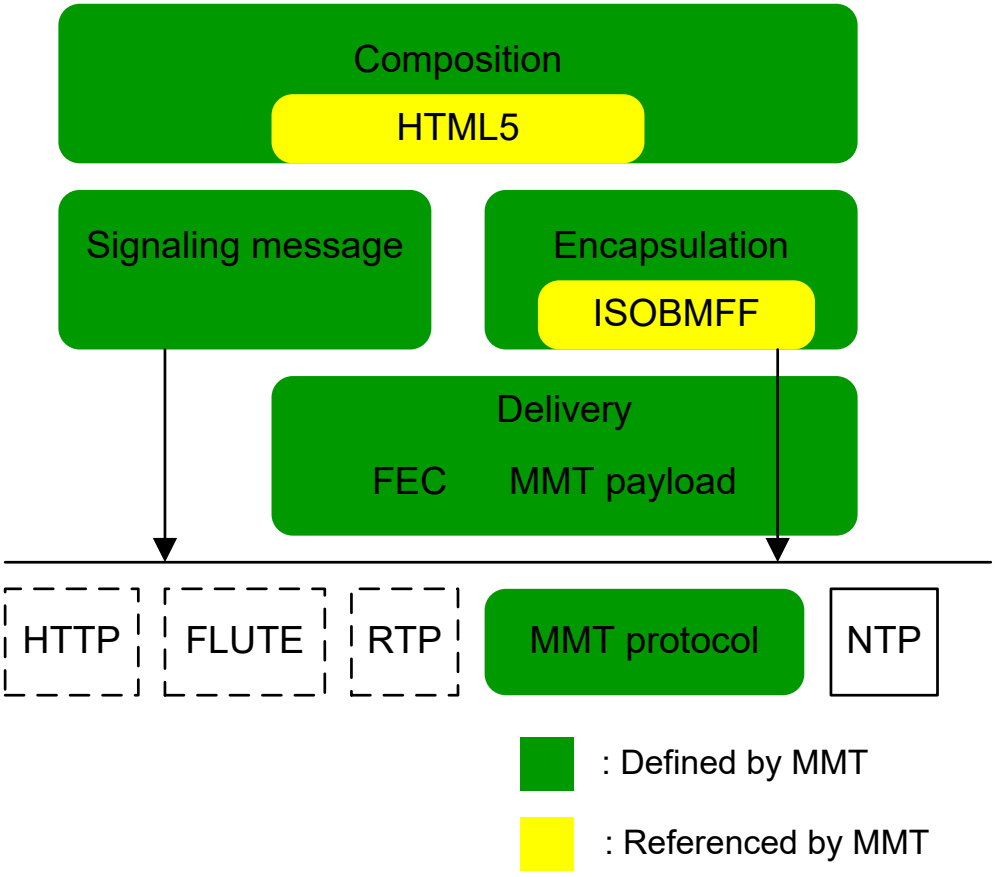


Figure 1 — MMT protocol Stack

Figure 2 depicts the end-to-end architecture for this specification. The sending entity is responsible for delivering Package to the receiving entity as packet flows. The sending entity may be required to gather content from the Content provider based on the Composition Information of the Package provided by the Package provider. The Package provider and the Content provider may be co-located as a single entity. The content is provided as an Asset that is encapsulated into a series of MMT Processing Units. The packet flow of such content is generated by using the associated Delivery Characteristics information. Signalling messages may be used to control the delivery session and the presentation of the Package between the MMT sending entity and the MMT receiving entities.

It should be noted that this specification only defines the interfaces between the sending entity and the receiving entity for the different functionalities.

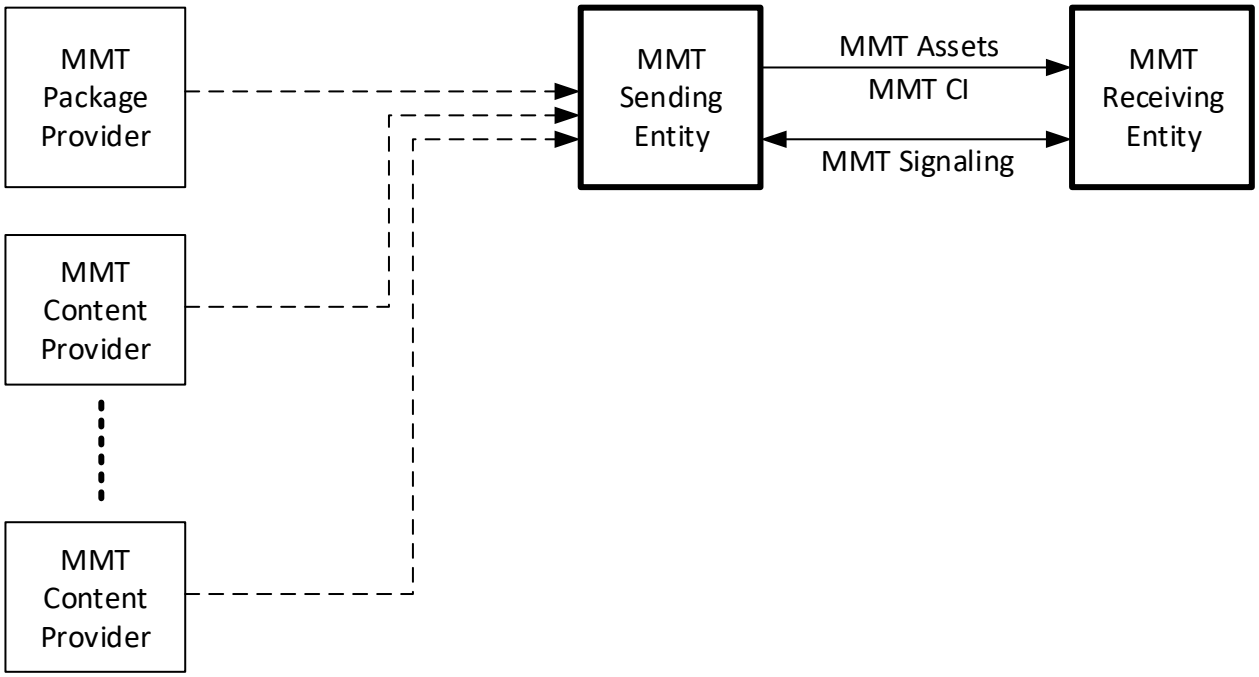


Figure 2 — End-to-end Architecture of MMT

5 MMT Content Model

5.1 Introduction

This clause defines the logical structure of a Package, as a collection of coded media data and associated information, to be processed by the MMT entity.

5.2 Logical structure of MMT Content

5.2.1 Package

As shown in Figure 3Figure 3, a Package is a logical entity. A Package shall contain one Composition Information (CI), one or more Assets and for each Asset an associated Asset Delivery Characteristics (ADC). In other words, as processing of a Package is applied per MPU basis and an Asset is a collection of one or more MPUs whose Asset ID is the same, it can be also considered that one Package is composed of one Composition Information, one or more MPUs and associated Asset Delivery Characteristics per each Assets.

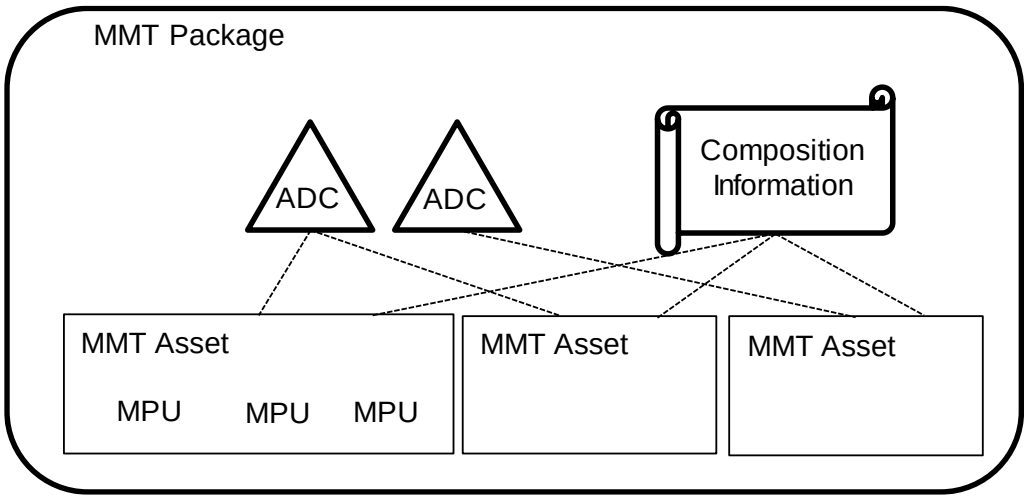


Figure 3 — Overview of Package

An Asset shall be a component of a Package that encapsulates coded media data such as audio or video or a web page data of timed or non-timed nature. An Asset collectively references a group of MPUs having the same Asset ID.

Composition Information (CI) shall specify the spatial and temporal relationship among the Assets for consumption and presentation by using HTML5 with extensions. It may also be used to determine the delivery order of Assets. CI shall be delivered either as one or more messages defined in this specification or as a complete document by some means that is not defined in this specification. Service providers may decide to carousel CI and determine the frequency at which carouseling is to be performed.

Asset Delivery Characteristics (ADC) shall provide the required QoS information for transmission of Assets. Multiple Assets can be associated with a single ADC. However a single Asset shall not be associated with multiple ADCs. This information can be used by the entity packetizing the Package to configure the parameters of the MMT payload and MMT protocol for efficient delivery of the Assets. An ADC may be associated with multiple Assets.

5.2.2 Asset

Asset is the logical data entity containing coded media data. An Asset contains coded media data. Such coded media data can collectively reference a group of MPUs with the same Asset ID. Any type of data which can be individually consumed by the entity directly connected to the MMT receiving entity is considered as a separate Asset. Examples of data types which can be considered as an individual Asset are MPEG-2 TS, PES, MP4 file, MPEG-U Widget Package, JPEG file, etc.

Coded media of an Asset can be either timed data or non-timed data. Timed data is coded media data requiring synchronized decoding and presentation of a specific unit of data at a designated time. Non-timed data is other types of data that can be decoded and presented at an arbitrary time based on the context of a service or interaction by the user.

5.2.3 Media Processing Unit (MPU)

A Media Processing Unit (MPU) is media data which may be independently and completely processed by an MMT entity and consumed by the media codec layer. Processing of an MPU by an MMT entity includes encapsulation/decapsulation and (de)packetization. Consumption of an MPU by the media codec layer includes decoding and presentation. An MPU may have data structure indicating the boundaries of

independently decodable data smaller than AU for packetization. An MPU may contain a contingent portion of data formatted according to other standards, e.g. MPEG-2 TS.

NOTE For scalable video coding extension and multiview video coding extension of ISO/IEC 14496-10, some MPU may not be independently and completely consumable by the media codec layer.

A single MPU shall contain either one AU or an integer number of AUs, or non-timed data. For timed data, a single AU can be carried by one or more MFUs: however a single AU shall not be fragmented into multiple MPUs. For non-timed data, a single MPU contains one or more non-timed data items, which can be independently and completely processed by an MMT entity and consumed by the media codec layer.

An MPU shall be uniquely identified by an associated Asset ID and a sequence number.

An MPU that contains timed media shall have at least one Random Access Point. The first byte of an MPU shall be a Random Access Point for processing by an MMT entity as well as decryption. The first byte of an MPU payload shall start with the Random Access Point of the media data. For timed media, this implies that the decoding order of the first AU in the MPU payload is always '0'. For the MPU containing the data formatted according to other standards, the MPU payload starts with the information necessary for the processing of such a format. For example, if an MPU contains MPEG-2 TS packets, the MPU payload starts with TS packets containing PAT and PMT for the remaining or subsequent TS packets. For timed media data, the presentation duration and the decoding order, and the presentation order of each AU are signalled as part of the MPU structure. The MPU does not have its initial presentation time. The presentation time of the first AU in an MPU is described by the Composition Information. The CI specifies the initial presentation time of each MPU. Figure 4 depicts an example of the timing of the presentation of the MPUs from different Assets that is provided by the CI. In this example, the CI requests the MMT receiving entity to present MPU #1 of asset #1 and of Asset #2 simultaneously. At a later point, MPU #1 from Asset #3 is scheduled to be presented. Finally, MPU #2 of Asset #1 and Asset #2 are requested to be presented in sync. The specified presentation time for an MPU defines the presentation of the first AU of that MPU.

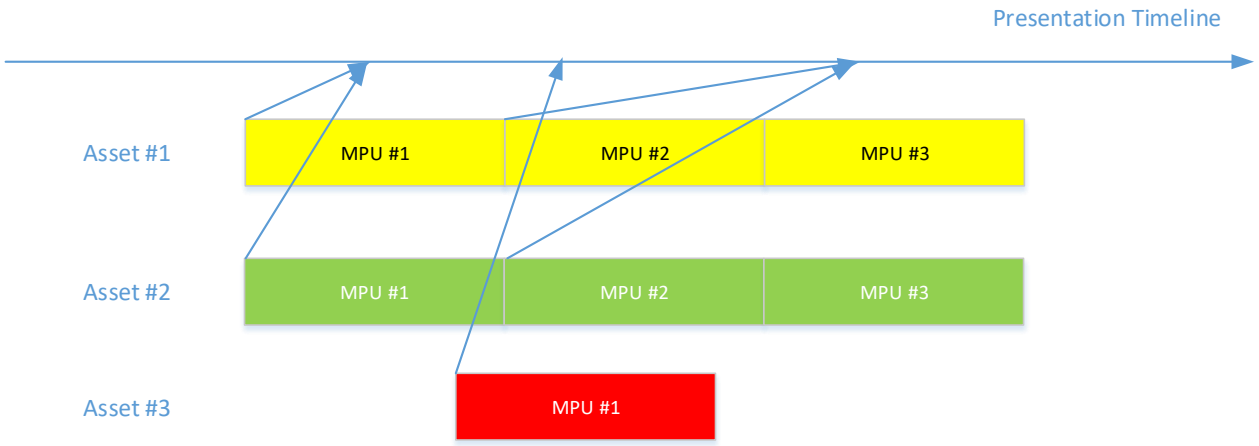


Figure 4 — Example of mapping MPUs to the presentation timeline

An MPU that contains non-timed media data may designate one data item as the entry point or the main data item, if necessary.

5.2.4 Media Fragment Unit (MFU)

The Media Fragment Unit (MFU) defines the format identifying fragmentation boundaries of MPU payload to allow to MMT sending entity to perform fragmentation of MPU considering consumption by media codec layer. An MFU may provide boundaries of data, smaller than AU for timed media data, where the data within the boundary can be independently decoded.

NOTE If the size of an MFU is bigger than the size of an MTU, an MFU could be fragmented into multiple packets. However, it is not recommended to place two or more incomplete MFUs in the same packet.

An MFU has an identifier to distinguish one MFU from another and generalized relationship information agnostic to coded media format among MFUs within a single AU. The dependency relationship among MFUs in a single AU is described, as are the relative priorities of MFUs as as a part of such information. The information can be used by underlying delivery layers to manipulate delivery. For example, the delivery layer can skip delivery of discardable MFUs to support QoS under certain circumstances such as when there is insufficient bandwidth in the network.

5.3 Composition Information

5.3.1 Introduction

Composition Information in MMT provides information on both spatial and temporal relationships among Assets in a Package. In addition, CI provides information for delivery optimization and presentation of a Package in the multi-screen environment. CI is a descriptive language extending the HTML5. To support descriptive representation of temporal relationship among Assets and mapping of Assets to particular screens for the multiple screen environments, several extensions to HTML5 are defined as follows:

— Association of Assets in a Package as a resource.

— Temporal information to decide delivery and consumption orders of Assets.

— Mapping of the Asset to a particular screen for the multi-screen environment.

5.3.2 Structure

CI defines the scene to be displayed and consumed in a Package as a “View” shown in Figure 5. The “View” consists of several Areas, and each Area has one or more Assets. Definitions of “View” and “Area” are as follows:

— View: represents a whole display region, which is composed of Areas.

— Area: represents a part of View, which is composed of Assets.

Figure 5 shows a relationship example of the Assets, the Area, and the View in a Package, where a “View” can be considered as a page of HTML5 and an Area as a region obtained by the **div** element of HTML5.

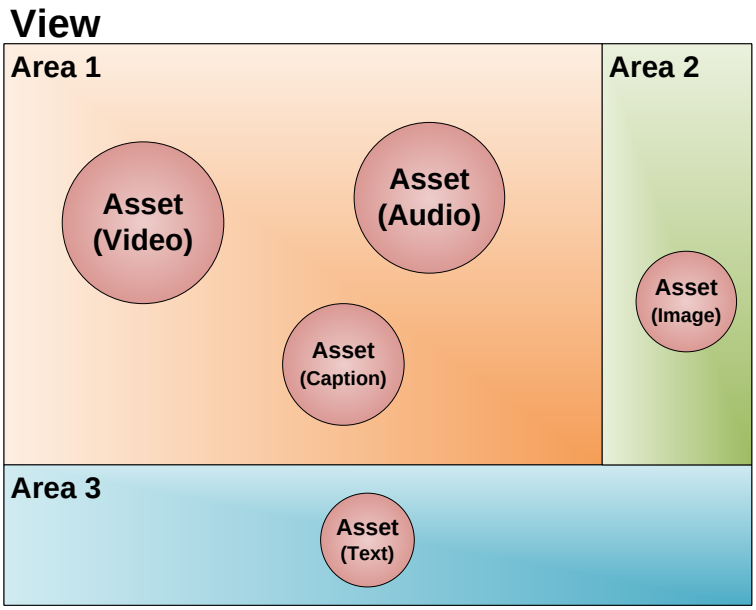


Figure 5 — Example of Asset/Area/View

CI has the basic structure shown in Figure 6 including the extended and added elements to HTML5.

The basic structure of CI is derived from the structure of the HTML5 document, and thus, the root element of CI is the **html** element, which contains two elements such as **head** and **body** elements. Some elements in the **head** and **body** elements are extended or added in order to meet the requirements of CI.

The **head** element should contain **view**, and **divLocation** elements. The roles of these elements are as follows:

— **view**: provides spatial and temporal information about a View.

— **divLocation**: provides temporally changed spatial information about an Area.

The **body** element should include **div** elements. The **div** element contains media elements corresponding to Asset such as **video**, **audio**, **img**, and **track**. The roles of these elements are as follows:

— **div**: provides spatial information about an Area.

— **video**, **audio** ... (Asset): provides spatial and temporal information of an Asset.

The identification of Asset which is defined in 6.3 in this specification, shall be described in the **source** element defined in 4.8.8 of [W3C HTML5] or *src* attribute defined in 4.8.10.2 of [W3C HTML5].

The initial Areas in a View are generated by the **div** element, and the temporal change of the initial Areas in the View can be obtained by **divLocation** element**.** These two elements allow the spatial location of Areas to be dynamically changed and updated without reloading a View page.

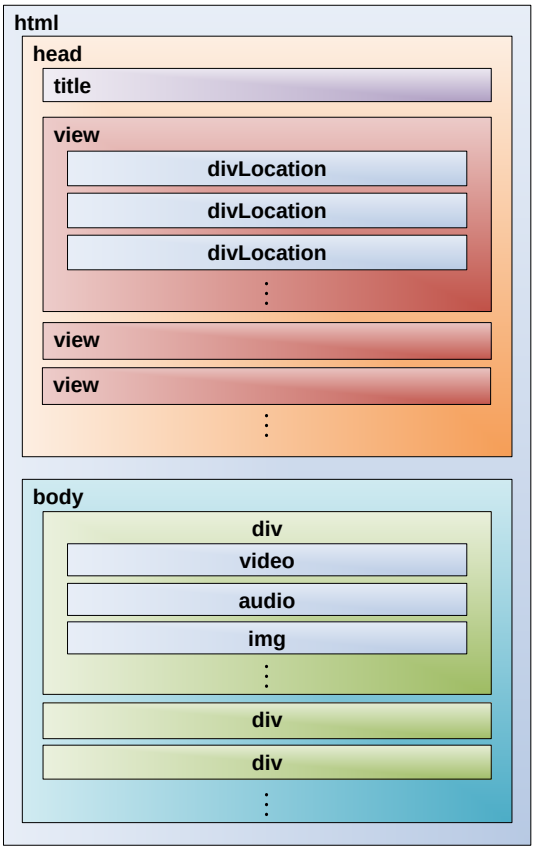


Figure 6 — Basic Structure of CI

CI can be partitioned into multiple subsets and each subset of CI can be delivered separately by using a signaling message. (see sub-clause 5.3.3.7 and 5.3.3.8)

5.3.3 The added attributes for CI

5.3.3.1 The *begin* attribute

The *begin* attribute represents when media elements corresponding to an Asset or an Area or a View becomes active. This attribute is used for the **video**, **audio**, **img**, **track**, **object** element of [W3C HTML5]. It can have an offset-value or synchronization-value, event-value, or wall clock-value which are described as follows. The default value of this attribute is “0s”. In order to specify more than one begin time, a white space separated list of begin values is provided. Valid values for this attribute can be categorized as follows:

— *Offset-value*: describes the beginning time as an offset from an implicit syncbase. The definition of the implicit syncbase of Area and media element depends upon their parents. The definition of the implicit syncbase of View depends on the service and previous View.

— *Synchronization-value*: describe the beginning time relative to the begin, active, and end of another media element or Area or View. The specific synchronization-value can be expressed by two identifiers which are relevant, such as `"View1.Video1.begin"`, `"Area2.Video3.end"`.

— *Event-value*: describes an event that determines the beginning time. The Event-value is defined in HTML5. The specific event value can be expressed by two identifiers which are relevant, such as `"View1.Video1.click"`, `"Area2.Video3.click"`.

— *Wallclock-value*: describes the beginning time as UTC.

5.3.3.2 The *end* attribute

The *end* attribute represents when a media element or an Area or a View becomes inactive. This attribute is used for the **video**, **audio**, **img**, **track**, **object** element of [W3C HTML5]. It can have an offset-value or synchronization-value, event-value, wall clock-value which are described as follows. The default value is "indefinite". In order to describe a number of ending time, a white space separated list of end values is provided.

— *Offset-value*: describes the ending time as an offset from an implicit syncbase. The definition of the implicit syncbase of Area and media element depends upon their parents. The definition of the implicit syncbase of View depends on the service and previous View.

— *Synchronization-value*: describes the ending time relative to the begin or active end of another media element or Area or View. The specific synchronization-value can be expressed by two identifiers which are relevant, such as "View1.Video1.begin", "Area2.Video3.end".

— *Event-value*: describes an event that determines ending time. The Event-value is defined in HTML5. The specific event value can be expressed by two identifiers which are relevant, such as "View1.Video1.click", "Area2.Video3.click". Additionally, CI defines the events in**Error! Reference source not found.** for multi-screen presentation.

— *Wallclock-value*: describes the ending time as UTC.

5.3.3.3 The *dur* attribute

The *dur* attribute represents the length of the simple duration, measured as active time of a media element, or an Area, or a View. It is described by *Offset-value*. This attribute is used for the **video**, **audio**, **img**, **track**, **object** element of [W3C HTML5].

5.3.3.4 The *clipBegin* attribute

The *clipBegin* attribute represents the beginning of a time interval of a continuous media element by offset from the start of the media. In addition, it is able to describe the sequence number of an MPU in order to play from a certain MPU in an Asset. This attribute is used for the **video**, **audio** element of [W3C HTML5].

5.3.3.5 The *clipEnd* attribute

The *clipEnd* attribute represents the end of a time interval of a continuous media element by offset from the start of the media. In addition, it is able to describe the sequence number of an MPU in order to play to a certain MPU in an Asset. This attribute is used for the **video**, **audio** element of [W3C HTML5].

5.3.3.6 The *refDiv* attribute

The *refDiv* attribute describes an ID of **div** element, which is used for linking its **divLocation** element.

5.3.3.7 The *xlink:href* attributes

The *xlink:href* attribute provides the data that allows XLink application to find a subset of CI. The value of the href attribute shall be a URI reference as defined in IETF RFC 2396 and shall be provided. This attribute is used for the **div** element of [W3C HTML5].

5.3.3.8 The *xlink:actuate* attributes

The *xlink:actuate* attribute defines the desired timing of traversal linked resources. If a value is supplied for an actuate attribute, it must be one of following values. This attribute is used for **div** element of [W3C HTML5].

— *onLoad*: liking a subset of CI immediately on loading of the current CI.

— *onRequest*: linking a subset of CI when available or on user request.

5.3.3.9 The *viewtype* attributes

The *viewtype* attribute defines the role of the **view** element. Several **view** elements can be described in one CI, so this attribute can be used to distinguish each view. The following is the meaning of the value of *viewtype* attribute.

Table 1 — the *viewtype* for multi-screen presentation

Value	Description
default	There is no multi-screen in the network. Or the screen works standalone without the interaction with another screen.
multiple	There are multi-screens in the network.
receptible	There are blank Areas for the receiving Area in a multi-screen presentation.

5.3.3.10 The *isDependent* attributes

The *isDependent* attribute indicates whether the media element depends on another media element or not, in order to be properly decoded and presented. If the *isDependent* attribute is “TRUE”, the Asset should refer to other Assets listed in *depAssetID*. This attribute is used for the **source** element of [W3C HTML5].

5.3.3.11 The *depAssetID* attributes

The *depAssetID* attribute indicates a whitespace-separated list of IDs of the Assets that the current Asset depends on. The *depAssetID* attribute exists if and only if the isDependent attribute is “TRUE”. This attribute is used for the **source** element of [W3C HTML5].

```
<!-- MMT-CI:begin Type -->
<xsd:simpleType name="MMT-CI:beginType">
    <xsd:list itemType="MMT-CI:beginValueType"/>
</xsd:simpleType>

<!-- MMT-CI:beginValue Type -->
<xsd:simpleType name="MMT-CI:beginValueType">
    <xsd:union memberTypes="MMT-CI:offsetType MMT-CI:syncType MMT-CI:eventType MMT-
CI:wallclockType"/>
</xsd:simpleType>

<!-- MMT-CI:end Type -->
<xsd:simpleType name="MMT-CI:endType">
    <xsd:list itemType="MMT-CI:endValueType"/>
</xsd:simpleType>

<!-- MMT-CI:endValue Type -->
<xsd:simpleType name="MMT-CI:endValueType">
    <xsd:union memberTypes="MMT-CI:offsetType MMT-CI:syncType MMT-CI:eventType MMT-CI:wallclockType
MMT-CI:indefiniteType"/>
</xsd:simpleType>
```

```
<!-- MMT-CI:dur Type -->
<xsd:simpleType name="MMT-CI:durType">
    <xsd:restriction base="MMT-CI:offsetType"/>
</xsd:simpleType>

<!-- MMT-CI:offset Type -->
<xsd:simpleType name="MMT-CI:offsetType">
    <xsd:restriction base="xsd:token">
        <xsd:pattern value="[0-9]+?(\.[0-9]+)?(h|min|s|ms)"/>
    </xsd:restriction>
</xsd:simpleType>

<!-- MMT-CI:sync Type -->
<xsd:simpleType name="MMT-CI:syncType">
    <xsd:restriction base="xsd:token">
        <xsd:pattern value="([\i-[:]][\c-[:]]*\.)?[\i-[:]][\c-[:]]*\.(begin|end)"/>
    </xsd:restriction>
</xsd:simpleType>

<!-- MMT-CI:event Type -->
<xsd:simpleType name="MMT-CI:eventType">
    <xsd:restriction base="xsd:token">
        <xsd:pattern value="([\i-[:]][\c-[:]]*\.)?[\i-[:]][\c-[:]]*\.\c+"/>
    </xsd:restriction>
</xsd:simpleType>

<!-- MMT-CI:wallclock Type -->
<xsd:simpleType name="MMT-CI:wallclockType">
    <xsd:restriction base="xsd:token">
        <xsd:pattern value="[0-2][0-9]:[0-5][0-9](:[0-5][0-9](\.[0-9]+)?)?"/>
    </xsd:restriction>
</xsd:simpleType>

<!-- MMT-CI:indefinite Type -->
<xsd:simpleType name="MMT-CI:indefiniteType">
    <xsd:restriction base="xsd:token">
        <xsd:enumeration value="indefinite"/>
    </xsd:restriction>
</xsd:simpleType>

<!-- MMT-CI:clipBegin Type -->
<xsd:simpleType name="MMT-CI:clipBeginType">
    <xsd:union memberTypes="MMT-CI:offsetType MMT-CI:wallclockType MMT-CI:MPUidType"/>
</xsd:simpleType>

<!-- MMT-CI:clipEnd Type -->
<xsd:simpleType name="MMT-CI:clipEndType">
    <xsd:union memberTypes="MMT-CI:offsetType MMT-CI:wallclockType MMT-CI:MPUidType"/>
</xsd:simpleType>

<!-- MMT-CI:MPUid Type -->
<xsd:simpleType name="MMT-CI:MPUidType">
    <xsd:restriction base="xsd:token">
        <xsd:length value="256"/>
    </xsd:restriction>
</xsd:simpleType>

<!-- MMT-CI:refDiv Type -->
<xsd:simpleType name="MMT-CI:refDivType">
    <xsd:extension base="xsd:ID"/>
</xsd:simpleType>

<!-- xlink:href Type -->
<xsd:simpleType name="xlink:hrefType">
    <xsd:restriction base="xsd:anyURI"/>
</xsd:simpleType>

<!-- xlink:actuate Type -->
<xsd:simpleType name="xlink:acturateType">
    <xsd:restriction base="xsd:token">
        <xsd:enumeration value="onLoad"/>
```

```
        <xsd:enumeration value="OnRequest"/>
    </xsd:restriction>
</xsd:simpleType>

<!-- MMT-CI:viewtype Type -->
<xsd:simpleType name="MMT-CI:viewtype">
    <xsd:restriction base="xsd:token">
        <xsd:enumeration value="default"/>
        <xsd:enumeration value="multiple"/>
        <xsd:enumeration value="receptible"/>
    </xsd:restriction>
</xsd:simpleType>

<!-- MMT-CI:StringVector Type -->
<xsd:simpleType name="MMT-CI:StringVectorType">
    <xsd:list itemType="xsd:string"/>
</xsd:simpleType>
```

5.3.4 The added element for CI

5.3.4.1 The view element

5.3.4.1.1 Semantics

The **view** element is a child element of the **head** element, and provides spatial and temporal information of a View. The child elements of the **view** element describe Areas of which a View is composed of. This element may occur several times depending on the Views to be presented.

5.3.4.1.2 Syntax

```
<!-- MMT-CI:view Type -->
<xsd:complexType name="MMT-CI:viewType">
    <xsd:complexContent>
        <xsd:extension base="HTML5:HTMLElementType">
        <!-- The view element inherits HTMLElement interface which is specified in 3.3.2 of[W3C
HTML5]-->
            <xsd:sequence>
                <xsd:element name="divLocation" type="MMT-CI:divLocationType" minOccurs="1"
maxOccurs="unbounded"/>
            </xsd:sequence>
            <xsd:attribute name="begin" type="MMT-CI:beginType" default="0s"/>
            <xsd:attribute name="end" type="MMT-CI:endType" default="indefinite"/>
            <xsd:attribute name="dur" type="MMT-CI:durType"/>
            <xsd:attribute name="type" type="xsd:token" fixed="simple"/>
            <xsd:attribute name="href" type="xlink:hrefType"/>
            <xsd:attribute name="show" type="xsd:token" fixed="embed"/>
            <xsd:attribute name="actuate" type="xlink:actuateType" default="onRequest"/>
            <xsd:attribute name="viewtype" type="MMT-CI:viewtype" default="default"/>
        </xsd:extension>
    </xsd:complexContent>
 </xsd:complexType>
```

5.3.4.1.3 Attributes

The **view** element inherits *HTMLElement* interface which is specified in sub-clause 3.3.2 of [W3C HTML5]. Thus, it also inherits all Global attributes of *HTMLElement* interface.

In addition, the **view** element includes following attributes:

— *begin*: is defined in sub-clause 5.3.3.1 of this specification.

— *end*: is defined in sub-clause 5.3.3.2 of this specification.

— *dur*: is defined in sub-clause 5.3.3.3 of this specification.

— *xlink:href*: is defined in sub-clause 5.3.3.7 of this specification.

— *xlink:actuate*: is defined in sub-clause 5.3.3.8 of this specification.

— *viewtype*: is defined in sub-clause 5.3.3.9 of this specification.

5.3.4.2 The divLocation element

5.3.4.2.1 Semantics

The **divLocation** element is child element of the **view** element, and provides spatial and temporal information of an Area. This spatial information consists of mutable values that can be changed in accordance with a View such as position and size of the Area within the View. The other spatial information is inherent values that will not be changed regardless of a View such as width and height. It is described by the **div** element. The size of an Area can be increased or decreased from the inherent values to the mutable values in order to display the Area suitable for regions in a View.

5.3.4.2.2 Syntax

```
<!-- MMT-CI:divLocation Type -->
<xsd:complexType name="MMT-CI:divLocationType">
    <xsd:complexContent>
        <xsd:extension base="HTML5:HTMLElementType">
        <!-- The view element inherits HTMLElement interface which is specified in 3.3.2 of[W3C
HTML5]-->
            <xsd:attribute name="begin" type="MMT-CI:beginType" default="0s"/>
            <xsd:attribute name="end" type="MMT-CI:endType" default="indefinite"/>
            <xsd:attribute name="dur" type="MMT-CI:durType"/>
            <xsd:attribute name="refDiv" type="MMT-CI:refDivType"/>
            <xsd:attribute name="plungeIn" type="xsd:positiveInteger"/>
            <xsd:attribute name="plungeOut" type="MMT-CI: plungeOutType" default="disable"/>
        </xsd:extension>
    </xsd:complexContent>
 </xsd:complexType>

<!-- MMT-CI:plungeOut Type -->
<xsd:simpleType name="MMT-CI: plungeOutType">
    <xsd:restriction base="xsd:token">
        <xsd:enumeration value="disable"/>
        <xsd:enumeration value="complementary"/>
        <xsd:enumeration value="sharable"/>
        <xsd:enumeration value="dynamic"/>
        <xsd:enumeration value="userselect"/>
    </xsd:restriction>
</xsd:simpleType>
```

5.3.4.2.3 Attributes

The **divLocation** element inherits *HTMLElement* interface which is specified in sub-clause 3.3.2 of [W3C HTML5]. Thus, it also inherits all Global attributes of *HTMLElement* interface.

In addition, the **divLocation** element includes following attributes:

— *begin*: is defined in sub-clause 5.3.3.1 of this specification.

— *end*: is defined in sub-clause 5.3.3.2 of this specification.

— *dur*: is defined in sub-clause 5.3.3.3 of this specification.

— *refDiv*: is defined in sub-clause 5.3.3.6 of this specification.

— *media*: is defined in sub-clause **Error! Reference source not found.**~~1.1.1.1~~ of this specification.

— *plungeIn*: makes its own Area as a blank for the receiving Area in multi-screen presentation. The *plungeIn* attributes in a View should have different positive integer value and the received Areas are composed sequentially at blank Areas depending on the value of order.

— *plungeOut*: provides the property of its own Area in terms of multi-screen presentation. Table 2 shows the meaning of each value of the *plungeOut* attributes.

Table 2 — The value for plungeOut attribute

Value	Description
disable	The Area is not allowed to be used for multi-screen presentation. It is a default value.
complementary	The Area is allowed to be used for multi-screen presentation, and shown at the secondary screen only. Before the beginning of a multi-screen presentation, the Area is invisible at the primary screen.
sharable	The Area is allowed to be used for a multi-screen presentation, and shown at both the primary and secondary screen.
dynamic	The Area is allowed to be used for a multi-screen presentation, and shown at the secondary screen only.
userselect	The Area is allowed to be used for a multi-screen presentation. User can select whether the Area is shared or moved.

5.4 Asset Delivery Characteristics

5.4.1 Introduction

The Asset Delivery Characteristics (ADC) describes the QoS requirements and statistics of Assets for delivery. Each Asset in a Package shall be associated with one set of ADC. The ADC for each Asset is used by an MMT sending entity to derive the appropriate QoS parameters and the transmission parameters to which a resource reservation and a delivery policy may apply. The ADC is represented in a protocol agnostic format to be generally used by QoS control service entity defined by other standard development organizations, such as IETF, 3GPP, IEEE, etc. It consists of a `QoS_descriptor` and a `bitstream_descriptor`.

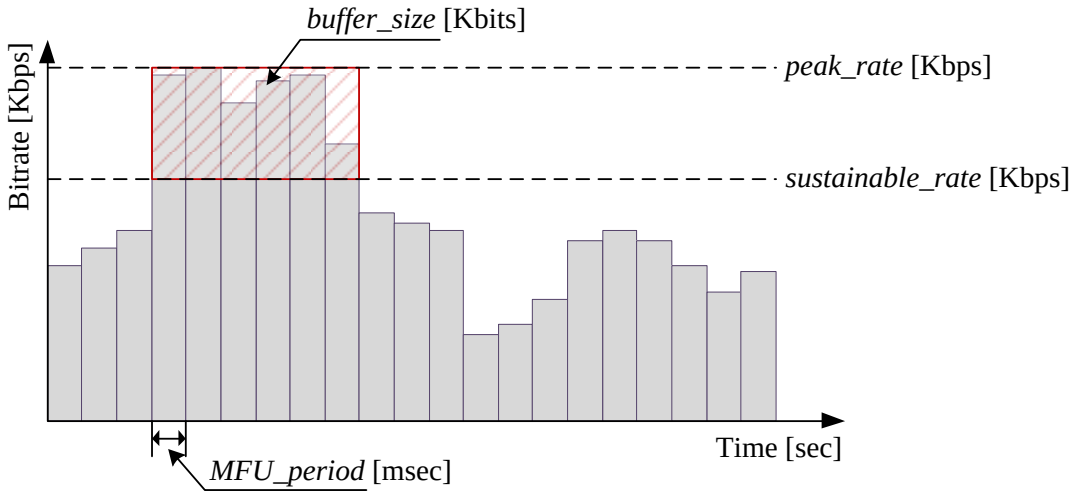
5.4.2 ADC Descriptors

5.4.2.1 QoS descriptor

The `QoS_descriptor` defines required QoS levels on delay and loss for Asset delivery. It consists of `transmission_priority`, `delay_priority`, and `class_of_serivce`.

5.4.2.2 Bitstream descriptor

The `bitstream_descriptor` provides the statistics of Asset. It provides the parameters to implement token bucket traffic shaping such as sustainable rate and buffer size. In addition, peak rate and maximum MFU size represent burst of Asset as shown in Figure 7.



※ *max_MFU_size* [Kbits] = *MFU_period* x *peak_rate*

Figure 7 — The `bitstream_descriptor` depicted for a variable bit-rate of Asset

5.4.3 Syntax

```
<complexType name="AssetDeliveryCharacteristic">
    <sequence>
        <element name="QoS_descriptor" type="mmt:QoS_descriptorType" />
        <element name="Bitstream_descriptor" type="mmt:Bitstream_descriptorType"/>
    </sequence>
</complexType>

<complexType name="QoS_descriptorType">
    <attribute name="transmission_priority" type="integer"/>
    <attribute name="delay_priority" type="integer"/>
    <attribute name="class_of_service" type="string"/>
    <attribute name="bidirection_indicator" type="boolean"/>
</complexType>

<complexType name="Bitstream_descriptorType">
    <choice>
        <complexType name="Bitstream_descriptorVBRType">
            <attribute name="sustainable_rate" type="float"/>
            <attribute name="buffer_size" type="float"/>
            <attribute name="peak_rate" type="float"/>
            <attribute name="max_MFU_size" type="integer"/>
            <attribute name="mfu_period" type="integer"/>
```

```
        </complexType>

        <complexType name="Bitstream_descriptorCBRUBRType">
            <attribute name="peak_rate" type="float"/>
            <attribute name="max_MFU_size" type="integer"/>
            <attribute name="mfu_period" type="integer"/>
        </complexType>

        <complexType name="Bitstream_descriptorABRType">
            <attribute name="mfu_period" type="integer"/>
        </complexType>
    </choice>
</complexType>
```

5.4.4 Semantics

`transmission_priority` – the `transmission_prioirty` indicates loss tolerance of the Asset. The value of `transmission_priority` is listed in Table 3.

Table 3 — value of `transmission_priority`

Value	Description
4	transmission priority 0 (Lossless)
3	transmission priority 1 (Lossy, High priority)
2	transmission priority 2 (Lossy, Medium priority)
1	transmission priority 3 (Lossy, Low priority)

`delay_priority` – the `delay_prioiry` indicates delay sensitivity of the Asset. The value of `delay_priority` is listed in Table 4 .

Table 4 — value of `delay_priority`

Value	Description
3	high sensitivity: end-to-end delay << 1sec (e.g., VoIP, video-conference)
2	medium sensitivity: end-to-end delay approx. 1 sec (e.g., live-streaming)
1	low sensitivity: end-to-end delay < 5~10 sec (e.g., VoD)
0	don’t care (e.g., FTP, file download)

`class_of_service` – the `class_of_service` classifies the services in different classes and manage each type of bitsteram with a particular way. This field indicates the type of bitstream as listed in Table 5.

Table 5 — value of `class_of_service`

Value	Description
0	The Constant Bit Rate (CBR) service class shall guarantee peak bitrate at any time to be dedicated for transmission of the Asset. This class is appropriate for realtime services which require fixed bitrate such as VoIP without silence suppression.
1	The Real-Time Variable Bit Rate (rt-VBR) service class shall guarantee sustainable birate and allow peak bitrate for the Asset with delay constraints over shared channel. This class is

	appropriate for most realtime services such as video telephony, videoconferencing, streaming service, etc.

`Bidirection_indicator` – If set to '1', bidirectional delivery is required. If set to '0', bidirectional delivery is not required.

`Bitstream_descriptorVBRType` – when `class_of_service` is '1', "`Bitstream_descriptorVBRType`" shall be used for "`Bitstream_descriptorType`".

`Bitstream_descriptorCBRUBRType` – when *`class_of_service`* is '0', "`Bitstream_descriptorCBRUBRType`" shall be used for "`Bitstream_descriptorType`".

`sustainable_rate` – The *`sustainable_rate`* defines the minimum bitrate that shall be guaranteed for continuous delivery of the Asset. The `sustainable_rate` corresponds to drain rate in token bucket model. The `sustainable_rate` is expressed in bytes per second.

`buffer_size` – The `buffer_size` defines the maximum buffer size for delivery of the Asset. The buffer absorbs excess instantaneous bitrate higher than the `sustainable_rate` and the `buffer_size` shall be large enough to avoid overflow. The `buffer_size` corresponds to bucket depth in token bucket model. `Buffer_size` of a CBR (constant bit rate) Asset shall be zero. The `buffer_size` is expressed in bytes

`peak_rate` – The `peak_rate` defines peak bitrate during continuous delivery of the Asset. The `peak_rate` is the highest average bit rate during every `MFU_period`. The `peak_rate` is expressed in bytes per second.

`MFU_period` – The `MFU_period` defines minimum period of MFUs during continuous delivery of the Asset. The `MFU_period` is expressed in millisecond.

`max_MFU_size` – The `max_MFU_size` is the maximum size of MFU, which is `MFU_period*peak_rate`. The `max_MFU_size` is expressed in byte.

6 Encapsulation of MPU

6.1 Introduction

An MPU shall be encapsulated as a single ISOBMFF file. The sequence number and Asset ID of the MPU is provided in the `'mmpu'` box to uniquely indentify the MPU encapsulated in the file. Additionally, in case of timed media, a `'sidx'` box may be present to index movie fragments comprising MPU. As each MPU is self-contained, each file encapsulating MPU shall include the initialization data by using the `'ftyp'`, `'styp'`, and `'moov'` boxes. The `'moov'` box shall contain all codec configuration information for decoding and presentation of media data in MPU.

Timed media data is stored as a track of the ISOBMFF (a single media track is allowed). Non-timed media is stored as part of metadata in an ISOBMFF. Figure 4 depicts two example MMT encapsulations, one for timed and one for non-timed media. For packetized delivery of MPU, MMT hint track provides the information to convert encapsulated MPU to MMT payloads and MMT packets.

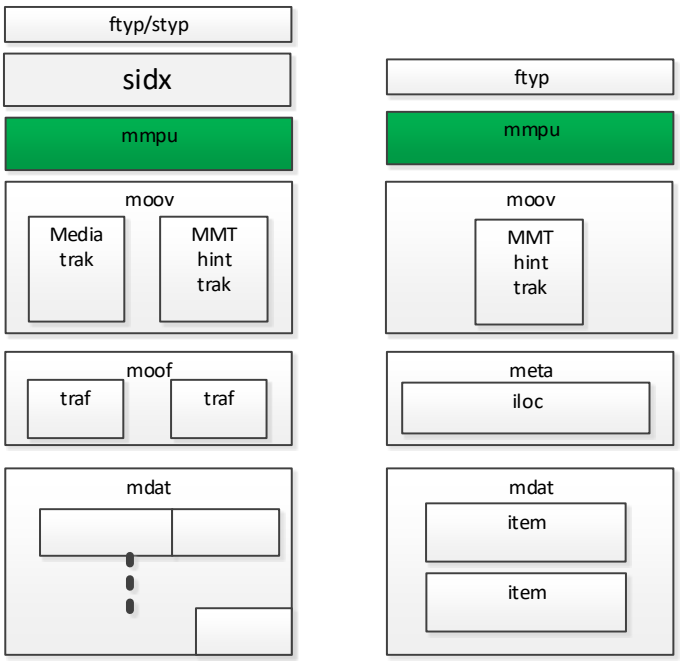


Figure 8 — Encapsulation of MPU

6.2 Encapsulation Rules

The type 'mpuf' (MPU file) defined in this specification identifies files that conform to the encapsulation rules for MPU . The 'mpuf' brand requires support of the 'isom' brand.

The MPU file is composed of a set of metadata boxes that enable the MPU to be self-contained. The MPU shall contain an 'ftyp' or 'styp', an 'mmpu', and a 'moov', and optionally an 'sidx' box, which all build the MPU metadata. Other boxes are allowed and will be ignored if the parser does not recognize them.

The 'moov' box shall contain at most one media track and may contain MMT hint tracks for MFU. The tracks in the 'moov' box shall contain no samples to ensure small overhead (i.e. the entry_count in the 'stts', 'stsc', and 'stco' boxes shall be set to '0'). The 'mvex' box shall be contained in the 'moov' box for the file storing an MPU with timed media data to indicate that the movie fragment structure is used. The 'mvex' box also sets default values for the tracks and samples of the following movie fragments.

In addition, an 'mmpu' box shall occur at the file level and the following rules including orders of boxes shall be applied:

1) The 'mmpu' box shall be placed right after 'ftyp' box or 'styp' box, if 'sidx' box is not used.

2) For timed media, zero or more 'sidx' boxes may be present in the file and if present, they shall index the movie fragments that build the current MPU.

In addition to the box orders, following restrictions shall be applied to the 'mpuf' brand:

1) The maximum number of media tracks in this file shall be one. Additionally, a hint track may be available.

2) For timed media data, the file shall have at least one movie fragment.

3) For non-timed media, one 'meta' box shall be present at the file level and shall contain the non-timed media items of the MPU.

In order to correctly position the media contained in the MPU on the Asset timeline, one of the following shall be present:

1) A 'tfdt' box inside the traf of the media track of the first moof box in the MPU

2) An 'elst' box inside the 'edts' of the media track

3) In case of the MPU with a sequence number 0 (i.e. the first MPU of an Asset), no indication is necessary

Timed media is stored as a track of the ISOBMFF and indexed by the 'moov' and 'moof' boxes in a fully backward-compatible manner. An MMT hint track guides the MMT sending entity in converting the file encapsulating MPU into a packetized media stream to be delivered using a packet streaming protocol such as MMT protocol.

Non-timed media is stored as metadata items inside a 'meta' box. The 'meta' box shall appear at the file level. Each file of the non-timed media shall be stored as a separate item of the 'meta' box. The entry point to the non-timed media shall be marked as the primary item of the 'meta' box.

6.3 MMT Processing Unit Box

6.3.1 Definition

Box Type: 'mmpu'
Container: File
Mandatory: Yes
Quantity: One or more

The MMT Processing Unit Box provides the identifier of the Asset to which the current MPU belongs as well as other information of the current MPU. The Asset identifier provides globally unique identification. The MPU information includes the sequence number of the MPU in the corresponding Asset. When it is required to store the ADC together with MPU, it shall be stored in the 'meta' box at the file level and its presence shall be indicated through the 'adc_present' flag and the mime type of the item that stores the ADC.

6.3.2 Syntax

```
aligned(8) class MPUBox
   extends FullBox('mmpu', version, 0){
   unsigned int(1) asset_type;
   unsigned int(1) is_adc_present;
   unsigned int(6) reserved;


   unsigned int(32) mpu_sequence_number;


   AssetIdentifierBox();
}
```

```
aligned(8) class AssetIdentifierBox {
   unsigned int(32) asset_ID_scheme;
   unsigned int(32) asset_ID_length;
   unsigned int(8)  asset_ID_value[asset_ID_length];
}
```

6.3.3 Semantics

`asset_type` – indicates the type of data in this Asset. `asset_type` as listed in Table 6**Error! Reference source not found.**.

Table 6 — value of `asset_type`

Value	Description
0	Type of data in this Asset is timed media.
1	Type of data in this Asset is non-timed media.

`is_adc_present` – indicates whether the ADC is present as an XML box in a `'meta'` box. The MIME type of the ADC file shall be indicated in a item info box `'iinf'`.

`mpu_sequence_number` – specifies the sequence number of the current MPU. For the first MPU in an asset, the sequence number shall be 0 and it is incremented by '1' for each following MPU and will be unique within an Asset.

`asset_ID_scheme` – identifies the scheme of Asset ID used in `asset_id_value` as listed in Table 7 **Error! Reference source not found.**. There are many schemes to express identification of content. It is recommended to use scheme-length-value, not to define a new identification scheme in this specification.

Table 7 — value of `asset_ID_scheme`

Value	Description
0	UUID
1	URI
2~0xFFFFFFFF	reserved

`asset_ID_length` – is length of `asset_id_value`.

`asset_ID_value` – is identifier of Asset. The format of value is specific to the `asset_id_scheme`.

6.4 MMT Hint Track

The MMT hint track hints the sending entity about the fragmentation of an MPU. One or more MFUs may then be used to build the MMT payload. The media data are extracted and conveyed in MMT payload at transport time by the sending entity. Consequently, the format at storage differs from the delivery format. This requires an active sending entity that is able to extract media data and generate MMT payaload on the fly.

The MMT hint track hints the extraction and building of MFUs for encapsulation using the MMT payload format. The MMT payload may contain the MPU metadata or one or more MFUs. The MPU metadata includes `'ftyp'`, `'styp'`, `'sidx'`, `'mmpu'`, and `'moov'` boxes. Each MFU is composed of a header and the associated media data. The MFU header shall be an exact copy of the MFU hint sample and the media data is a copy of the media data that is referenced by that MFU hint sample.

6.4.1 MMT Hint Track Format

The MMT hint track hints the sending entity on how to extract MFU data. If fragmentation is not used, the hint track may be omitted completely.

6.4.2 Sample Description Format

6.4.2.1 Definition

MMT hint tracks are hint tracks with an entry format in the sample description of `'mmth'` box:

6.4.2.2 Syntax

```
aligned(8) class MMTHintSampleEntry() extends SampleEntry('mmth') {
   unsigned int(16) hinttrackversion = 1;
   unsigned int(16) highestcompatibleversion = 1;
   unsigned int(16) packet_id;
   unsigned int(1) has_mfus_flag;
   unsigned int(1) is_timed;
   unsigned int(6) reserved;

   AssetIdentifierBox();
}
```

6.4.2.3 Semantics

`packet_id` – `packet_id` is a unique identifier for the Asset for which this hint track applies.
`has_mfus_flag` – indicates whether the MPUs are fragmented into MFUs or not. In case not, the hint track hints complete MPUs, i.e. each track fragment will have a single sample. Otherwise, each hint sample will refer to an MFU.
`is_timed` –indicates whether the media hinted by this track is timed or not.

6.4.3 Sample Format

6.4.3.1 Definition

Each media sample will be assigned to one or more MFUs. Each sample of the MMT hint track will generate one or more MFUs.

6.4.3.2 Syntax

```
aligned(8) class MMTHSample {
```

```
    unsigned int(32) sequence_number;
    if (is_timed) {
        signed int(8)    trackrefindex;
        unsigned int(32) samplenumber;
        unsigned int(8)  priority;
        unsigned int(8)  dependency_counter;
        unsigned int(16) offset;
        unsigned int(32) length;
        multiLayerInfo();
    } else {
        unsigned int(16) item_ID;
    }
}


aligned(8) class multiLayerInfo extends Box('muli') {
    bit(1)  multilayer_flag;
    bit(7)  reserved0;

    if (multilayer_flag==1) {
        bit(3)  dependency_id;
        bit(1)  depth_flag;
        bit(4)  reserved1;
        bit(3)  temporal_id;
        bit(1)  reserved2;
        bit(4)  quality_id;
        bit(6)  priority_id;
        bit(10) view_id;
    }
    else{
        bit(6) layer_id;
        bit(3) temporal_id;
        bit(7) reserved3;
    }
}
```

6.4.3.3 Semantics

`sequence_number` – the sequence number of this MFU within the MPU. Discontinuity of sequence number of MFU in an MPU indicates certain MFU having such sequence number was not processed after packetization of MPU. Examples of the processing are delivery and caching by underlying network entity.

`Trackrefindex` – the id of the media track from which the MFU data is extracted.

`Samplenumber` – the number of the sample from which this MFU is extracted. Samplenumber n points the n^{th} sample from accumulated samples of all the `'moof'` boxes in an MPU. The samplenumber of the first sample in an MPU is '0'.

`item_ID` – for non-timed media data, this refers to the item that builds this MFU.

`priority` – is priority of MFU among MFUs within an MPU.

`dependency_counter` – indicates that the number of MFUs whose decoding is dependent to this MFU. The value of this field is equal to the number of subsequent MFUs in the order of `sequence_number` which may not be correctly decoded without this MFU. For example, if the value of this field is equal to n, then n subsequent MFUs may not be correctly decoded.

`offset` – gives the offset of the media data. The offset base is the beginning of the containing `'mdat'` box. If giving backward compatibility to entity that does not support the boxes defined in this standard is required, MFU may be placed at the position that offset indicates with length that length field of this MFU indicates.

`length` – gives the length of the data corresponding to this MFU in bytes.

`multilayer_flag` – equal to '1' indicates that detailed multi-layer information is present in this box.
`dependency_id` – is dependency ID of this MFU. If it is non-zero value then it enhances the video by one or more scalability levels in at least one direction (temporal, quality or spatial resolution).
`depth_id` – equal to '1' indicates that the given MFU conveys depth data of video.
`quality_id` – is quality ID of this MFU. If it is non-zero value then it enhances the video by one or more scalability levels in at least one direction (temporal, quality or spatial resolution).
`temporal_id` – is temporal ID of this MFU. If it is non-zero value then it enhances the video by one or more scalability levels in at least one direction (temporal, quality or spatial resolution).
`view_id` – is view ID of this MFU. If it is non-zero value then it enhances the video by one or more scalability levels in at least one direction (temporal, quality or spatial resolution).
`layer_id` – indicates the id of a scalable layer whose information about scalability dimensions is provided in initialization information.

7 Packetized delivery of Package

7.1 Introduction

This sub-clause defines the payload format and an application layer transport protocol for packetized delivery of a Package.

NOTE For non-packetized delivery of Packages, e.g., using other file delivery protocols, the MMT payload is not required.

The MMT payload format is defined as a generic payload format for the packetization of the contents components of a Package. It is agnostic to specific media codecs used for coded media data, so that any type of media that is encapsulated as an MPU can be packetized into payload for an application layer transport protocol supporting streaming delivery of media content. MMT payload can be used as a payload format for RTP or MMT and other packet transport protocols. The MMT payload is also used to packetize signaling messages.

The MMT protocol defines an application layer transport protocol supporting streaming delivery of Package through packet-based heterogeneous delivery network including IP network environments. The MMT protocol provides essential features for delivery of Package such as protocol level multiplexing that enables various Assets to be delivered over a single MMT packet flow, delivery timing model independent of presentation time to adapt to a wide range of network jitter, and information to support Quality of Service (QoS).

7.2 MMT payload

7.2.1 Introduction

MMT payload is a generic payload to packetize and carry Assets and other information for consumption of Package using MMT protocol or other existing application layer transport protocol such as RTP. The MMT payload shall be used to packetize MPU and signaling messages described in sub-clause 9.2.

The MMT payload carries MPUs, signaling messages, FEC repair symbols, etc. For each data type, a single data unit for delivery is defined. For example, a single complete MPU is considered as a single data unit when MMT payload carries MPUs. The MMT payload can aggregate multiple data units with the same data type into a single payload. It can also fragment a single data unit into multiple payloads. The MMT payload shall only aggregate MPUs with the same Asset ID in a single payload.

The size of MMT payload header is fixed excluding header_extension field when single data unit is carried. The size of MMT payload header is variable when aggregation and fragmentation is performed. When aggregation or fragmentation is performed, the offset to the first byte of data implicitly indicates the size of payload header. The MMT payload can also include the padding bytes at the end of the payload. The size of padding byte is calculated by using the underlying transport layer protocol packet size.

7.2.2 Syntax

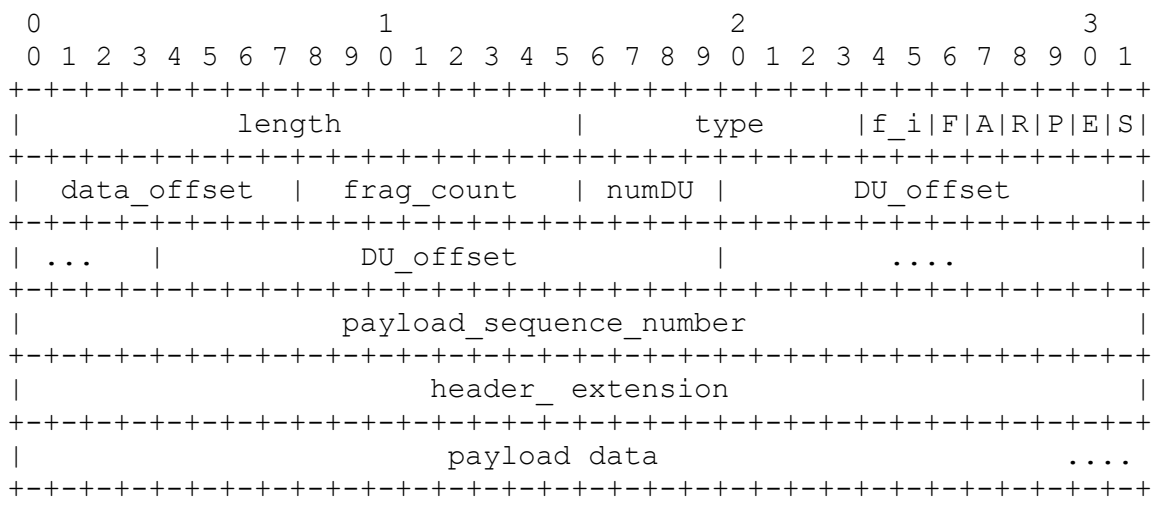


Figure 9 — structure of MMT payload header

7.2.3 Semantics

`length (16bits)` - This field indicates the length of the payload in bytes excluding the header. This doesn't include the size of padding data.

`type (8bits)` − This field indicates the type of payload data. Payload type values are defined in Table 8. For the fragmentation and aggregation indication data unit of each type of payload data is defined as follows:

Table 8 — Data type and Definition of Data unit

Value	Data type	Definition of Data unit
0x00	MPU	a single complete MPU (see subclause 5.2.3)
0x01	signaling message	message single complte signaling message (see subclause 9.2)
0x02	FEC repair symbol	a single complete FEC repair symbol (see subclause 7.5.4)
0x03 ~ 0x9F	ISO reserved for future use	
0xA0 ~ 0xFF	Reserved for private use	

`f_i (2bits)` − This field indicates the fragmentation indicator contains information about fragmentation of data unit in the payload. The four values are listed in Table 9:

Table 9 — value of fragmentation indicator

Value	Description
00	Payload contains one or more complete data units.
01	Payload contains the first fragment of data unit

10	Payload contains a fragment of data unit that is neither the first nor the last part.
11	Payload contains the last fragment of data unit.

The following flags indicate the presence of the following information carried in the MMT payload. Multiple bits can be set simultaneously.

`fragmentation_flag (F: 1bit)` – Set to '1', if `fragment_counter` is present.

`aggregation_flag (A: 1bit)` – Set to '1', if `aggregation_info` is present.

`RAP_flag (R: 1bit)` – Set to '1', if payload contains random access point (or part thereof).

`payload_sequence_flag (P: 1bit)` – Set to '1', if `payload_sequence_number` is present.

`extension_flag (E: 1bit)` – Set to '1', if `header_extension` is present.

`reserved (S: 1bit)` – reserved for future usage.`data_offset (8 bits)` − This field indicates location of the 1st byte of the payload data from the beginning of payload.

`fragment_counter (frag_count: 8 bits)` − This field specifies the number of payload containing fragments of same data unit succeeding this MMT payload. This field shall be '0', if `aggregation_flag` is set to '1'.

`number_data_unit (numDU: 4 bits)` − This field specifies the number of data unit within this MMT payload. This field shall be '0', if `fragmentation_flag` is set to '1'.

`DU_offset (16bits)` − This field specifies location of each data unit from the byte indicated by `data_offset`. This field shall be used, when `aggregation_flag` is set to '1'.

`payload_sequence_number (32 bits)` − This field specifies the sequence number of payload associated with the same Asset.

`header_extension` − This field contains user-defined information. The header extension mechanism is provided to allow for proprietary extensions to the payload format to enable applications and media types that require additional information to be carried in the payload format header. The header extension mechanism is designed in a way that it may be discarded without impacting the correct processing of the MMT payload. The header extension shall have the following format as shown in Figure 10.

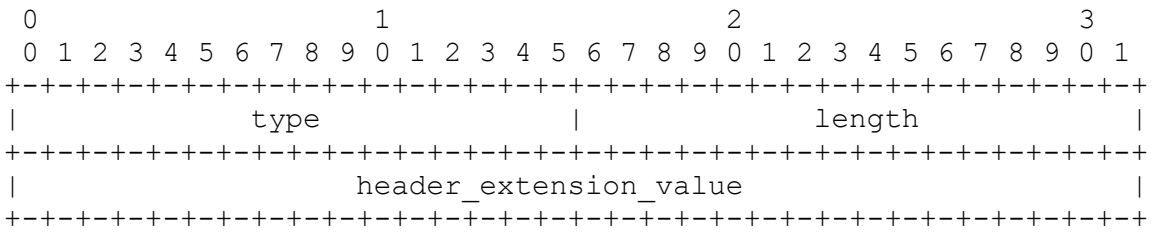


Figure 10 — structure of header extension

7.3 MMT protocol

7.3.1 Introduction

The MMT protocol is an application layer transport protocol designed to deliver Packages efficiently and reliably. MMT protocol is for the unified delivery of timed and non-timed media data. The MMT protocol supports several features, such as multiplexing, network jitter calculation, and QoS indication, which are essential to deliver content composed of various types of coded media data. The MMT protocol may run on top of the UDP and IP Protocols. A single MMT packet shall carry only one MMT payload.

In MMT protocol, the sending entity is responsible for the congestion control. As MMT protocol runs on top of UDP/IP and due to the wide variety of applications, congestion control is not specified in this specification but is rather left up to the implementation of sending entities.

The MMT protocol supports multiplexing of different Assets over a single MMT packet flow. It delivers multiple types of data in the order of consumption at the receiving entity to help synchronization between different types of media data without introducing a large delay or buffer requirement. MMT protocol also supports multiplexing of media data and signaling messages within a single packet flow of MMT protocol. A single MMT payload shall be carried in only one MMT packet.

MMT protocol provides the means to calculate and remove jitter introduced by the underlying delivery network, resulting in a constant delay of data stream. By using the timestamp field in the packet header, jitter can be precisely calculated without any additional signaling protocols.

MMT protocol provides priority related information to enable underlying network layers or the intermediate network entities to map the media transport layer priority information to the network protocol according to predetermined priority mapping policy. When DiffServ [RFC2474] is used, this priority information may be used to set the 6-bit DSCP value of the DS field in the IP header. The underlying network entity supporting Diffserv shall then process the media packets according to the mapping defined by the QoS fields.

7.3.2 Syntax

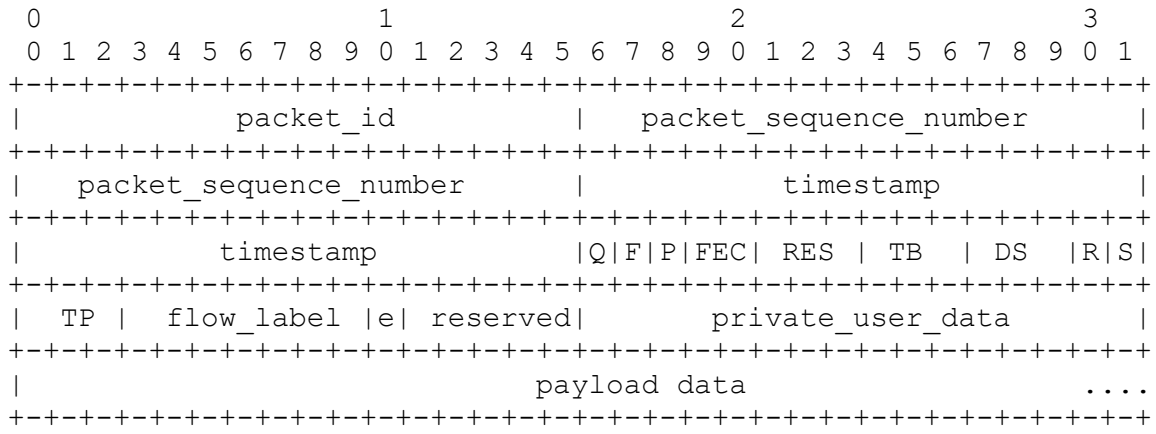


Figure 11 — MMT packet header

7.3.3 Semantics

`packet_id (16bits)` − This field is an integer value assigned to each Asset to distinguish packets of one Asset from another. Separate value will be assigned to signaling messages and FEC repair flows.The `packet_id` is unique throughout the lifetime of the delivery session and for all MMT flows delivered by the same MMT sending entity. The mapping between the `packet_id` and the `asset_id` is signalled as part of a signalling message. For AL-FEC, the mapping between `packet_id` and the FEC repair flow is provided in the AL-FEC message. The `packet_id` is unique for all MMT packet flows delivered by the same MMT sending entity.

`packet_sequence_number (32bits)` − This field is an integer value that is scoped by the `packet_id` and that starts from arbitrary value incremented by one for each MMT packet. It wraps around to '0' after its maximum value.

`timestamp (32bits)` − This field specifies the time instance of MMT packet delivery. The NTP time is used in timestamp as specified as the "short-format" in clause 6 of IETF RFC5905, NTP version 4. This timestamp is measured at the first bit of MMT packet.

`QoS_classifier_flag (Q:1bit)` − When set to '1', it indicates that QoS classifier information is used. QoS classifier contains `delay_sensitivity` field, `reliability_flag` field, and `tranmission_priority` field. It indicates the QoS class property. The application can perform per-class QoS operations according to the particular value of one property. The class values are universal to all independent sessions.

`flow_identifier_flag (F:1bit)` – When set to '1', it indicate that Flow identifier information is used. Flow identifier contains `flow_label` field, and `extension_flag` field. It indicates the flow identifier. The application can perform per-flow QoS operations in which network resources are temporarily reserved during the session. A flow is defined to be a bitsream or a group of bitstreams whose network resources are reserved according to transport characteristics or ADC in Package.

`private_user_data_flag (P:1bit)` – When set to '1', it indicates that `private_user_data` information is used.

`FEC_type (FEC: 2bits)` – The field indicates a FEC related type information of MMT packet. The value of protection type is listed in Table 10.

Table 10 — value of FEC_type

Value	Description
0	MMT packet without AL-FEC protection
1	MMT packet with AL-FEC protection (FEC source packet)
2	MMT packet for repair symbol(s) (FEC repair packet)
3	Reserved for future use

`reserved (RES: 3bits)` – reserved for future use

`type_of_bitrate (TB: 3bits)` – This field indicates the type of bitrate as listed in Table 11.

Table 11 — value of `type_of_bitrate`

Value	Description
000	Constant Bit Rate (CBR)
001	Non-Constant Bit Rate (nCBR)
010 ~ 111	reserved

`delay_sensitivity (DS: 3bits)` – This field indicates the delay sensitivity of the data for the given service as listed in Table 12.

Table 12 — value of `delay_sensitivity`

Value	Description
111	conversational service (~100ms), (e.g., Expedite Forwarding of DSCP)
110	live-streaming service (~1sec), (e.g., Assured Forwarding 1 of DSCP)
101	delay-sensitive interactive service (~2sec), (e.g., Assured Forwarding 2 of DSCP)
100	interactive service (~5sec), (e.g., Assured Forwarding 3 of DSCP)
011	streaming service (~10sec), (e.g., Assured Forwarding 4 of DSCP)
010	non-realtime, (e.g., Best Effort)
001	reserved
000	reserved

`reliability_flag (R: 1bit)` – When "`reliability_flag`" is set to '0', it shall indicate that the data is loss tolerant (e.g. media data), and that the following 3-bits shall be used to indicate relative priority of loss. When "`reliability_flag`" is set to '1', the "`transmission_priority`" field will be ignored, and shall indicate that the data is not loss tolerant (e.g., signaling data, service data, or program data).

`reserved (S: 1bits)` – reserved.

`transmission_priority (TP: 3bits)` – This field provides the `transmission_priority` for the media packet, and it may be mapped to the NRI of NAL, DSCP of IETF, or other loss priority field in another network protocol. This field shall take values from '7' ('1112') to '0' ('0002'), where 7 is the highest priority, and '0' is the lowest priority.

`reserved (r: 1bits)` – reserved.

`flow_label (7bits)` – This field indicates the flow identifier. The application can perform per-flow QoS operations in which network resources are temporarily reserved during the session. A flow is defined to be a bitsream or a group of bitstreams whose network resources are reserved according to transport characteristics or ADC in Package. It is an implicit serial number from '0' to '127'. An arbitrary number is assigned temporarily during a session and refers to every individual flow for whom a decoder (processor) is assigned and network resource could be reserved.

`extension_flag (e: 1bit)` – If there are more than 127 individual flows, this bit is set to '1' and one more byte can be used.

`private_user_data (16bits)` – This field is used for the private user data

7.4 Delivery Timing Model for MMT protocol

7.4.1 Introduction

MMT specifies the timing model to be used for delivery of MMT packets. Delivery of timed media data, according to their temporal requirements, is an essential feature supported by MMT protocol. Preservation of timing relationships among packets in a single MMT protocol packet flow or between packets from different MMT protocol packet flows is also an essential feature in MMT. MMT specifies the timing model to be used for delivery of MMT packets. The delivery timing model provides the functionality to calculate jitter and the amount of delay introduced by the underlying delivery network during the delivery of a Package.

7.4.2 Clock synchronization for MMT packet

NTP, as specified in RFC 5905, is used to synchronize clocks between sending and receiving entities.

7.4.3 Network jitter calculation

Network jitter calculation is essential for the network jitter compensation that may be required by some services using MMT protocol, and this is adopted from RFC 3550. The following equation is used to calculate the difference in packet spacing, $D_{MMT}(i.j)$, for a pair of MMT packets *i* and *j* at the receiving entity:

$$D_{MMT}(i,j) = (T_{A,j} - T_{A,i}) - (T_{D,j} - T_{D,i}) = (T_{A,j} - T_{D,j}) - (T_{A,i} - T_{D,i})$$

where $T_{D,i}$ and $T_{D,j}$ denote the delivery time instantce of two MMT packets *i* and *j*, respectively carried in MMT packet header. The $T_{A,i}$ and $T_{A,j}$ are the time instances at which MMT packets *i* and *j* have arrived at the MMT receiving entity respectively.

The inter-arrival jitter, $J_{MMT}(i)$ which is defined to be the mean deviation of the difference in packet spacing is calculated continuously as each MMT packet *i* is received according to the following formula:

$$J_{MMT}(i) = J_{MMT}(i-1) + (|D_{MMT}(i-1,i)| - J_{MMT}(i-1))/16$$

7.5 Application Layer Forward Error Correction Framework for MMT

7.5.1 Introduction

MMT provides the application level forward error correction (FEC) mechanism for reliable delivery in IP network environments that are prone to packet losses. The MMT FEC scheme is described as a building block of delivery function. After packetization, MMT packets are passed to the MMT FEC scheme for protection. The MMT FEC scheme returns repair symbols with repair FEC payload IDs and source FEC payload IDs. Then the repair symbols are delivered by MMT protocol with the MMT packets. The FEC configuration information provides the identification of FEC encoded flow, the information specified FEC coding structure and FEC code. It is delivered to MMT receiving entity for FEC operation. The outlined architecture is depicted in Figure 12.

The MMT sending entity determines the Assets within Packages which require protection and the number of FEC source flows. One or more of the Assets are protected as a single FEC source flow, which consists of MMT packets carrying the one or more Assets. The FEC source flow and its FEC configuration information are passed to the MMT FEC scheme for protection. The MMT FEC scheme uses FEC code(s) to generate repair symbols composing one or more FEC repair flows. They are passed to MMT protocol along with source FEC payload IDs and repair FEC payload IDs. Then MMT protocol delivers FEC source and repair packets to MMT receiving entity. At MMT receiving entity, MMT protocol passes the FEC source flow and its associated FEC repair flow(s) to MMT FEC scheme. Then the MMT FEC scheme returns recovered MMT packets.

According to the applied FEC coding structure the MMT FEC scheme divides the FEC source flow into source packet blocks and generates source symbol blocks. Then MMT FEC scheme passes source symbol blocks to the FEC code for FEC encoding. Here FEC encoding means the process to generate repairs symbols from the source symbol block. FEC code algorithms which may be used to generate repairs symbols from source symbol block are described in Annex E.

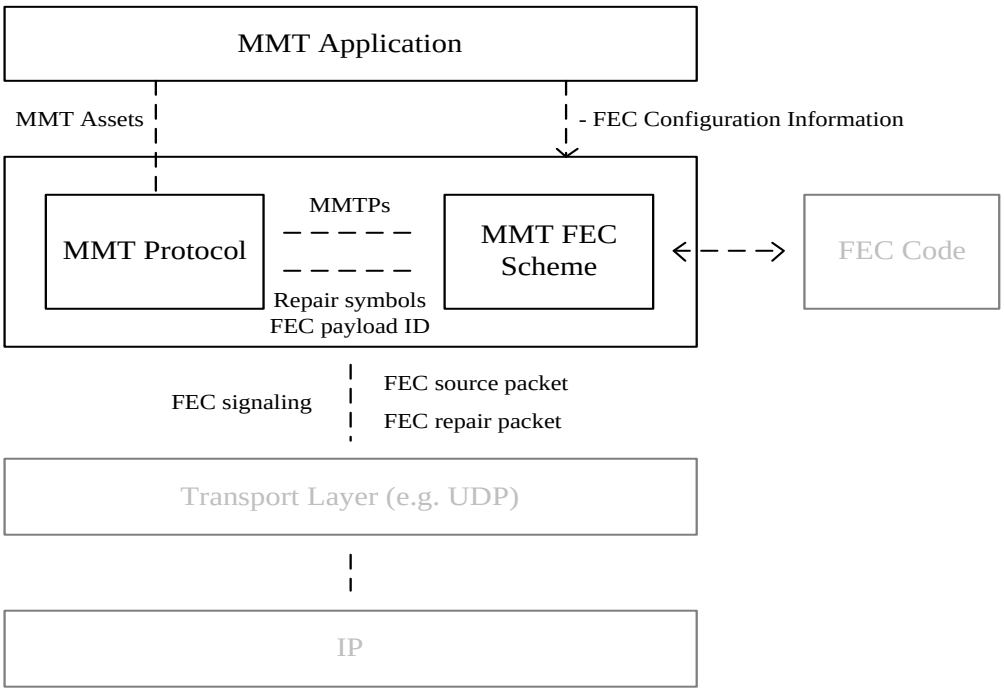


Figure 12 — Architecture for AL-FEC in MMT

7.5.2 FEC Coding Structure

7.5.2.1 Introduction

Compared to conventional AL-FEC scheme, MMT FEC scheme can provide multi-level construction of MMT packets for layered or non-layered media data for an appropriate level of protection each Asset in an FEC source flow. This sub-clause shows two FEC coding structures. One is two stage FEC coding structure and the other is Layer-Aware FEC (LA-FEC) coding structure.

7.5.2.2 Two stage FEC Coding Structure

This sub-clause specifies the two stage FEC coding structure as an FEC coding structure for AL-FEC to protect a source packet block which consists of a pre-determined number of MMT packets. The two stage FEC coding structure is depicted as shown in Figure 13.

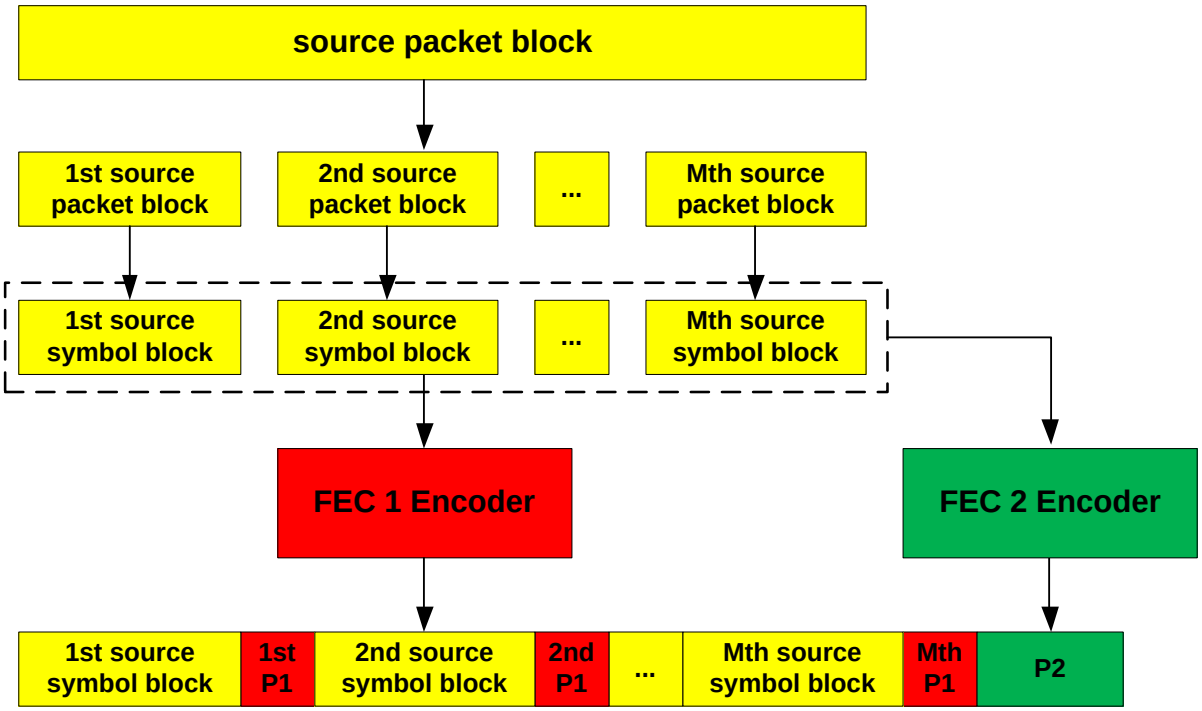


Figure 13 — Two Stage FEC Coding Structure

NOTE In Figure 13~~Figure 13~~, the ith P1 is repair symbol block for the ith source symbol block and P2 is repair symbol block for source symbol block (i=1, 2, …, M).

Source packet block shall be encoded in one of following FEC coding structures:

— Case 0: No FEC coding applied.

— Case 1: One stage FEC coding structure

— Case 2: Two stage FEC coding structure

For two stage FEC coding structure, one source packet block is split into M (>1) number of source packet blocks. The split ith source packet block is converted to the ith source symbol block with one of the SSBG modes defined in sub-clause 7.5.3.2. Then, the ith source symbol block is encoded by FEC 1 code (i = 1, 2, …,

M). Then, M source symbol blocks are concatenated to form a single source symbol block encoded by FEC 2 code. M number of repair symbol blocks are generated from M source symbol blocks by FEC 1 code, respectively and one repair symbol block is generated from the concatenated source symbol block by FEC 2 code.

For Case 0, both FEC 1 and FEC 2 encoding shall be skipped, i.e. no repair symbols are generated.

For Case 1, M shall be set to '1' and either FEC 1 encoding or FEC 2 encoding shall be skipped.

7.5.2.3 Layer-Aware FEC (LA-FEC) Coding Structure

Layer-Aware FEC (LA-FEC) is a FEC coding structure that can be applied with any FEC code and is specific for layered media data (e.g. SVC, MVC, or any other). The LA-FEC exploits the dependency across layers of the media for FEC construction and consist in the generation of several repair flows associated to each layer, where each repair flow protects the data of its corresponding layer and the data of all layers, this layer depends on (hereafter referred to as complementary layer), if any.

First, the MMT packets from the different layers are grouped into source symbol blocks independently. When LA-FEC structure is used the source symbol block generated for FEC encoding of a repair flow shall combine the source symbol block of the corresponding layer and the source symbol blocks from all its complementary layers, if any. The combination of source symbol blocks from the different layers shall be done following the dependency hierarchy from the media, i.e. with each source symbol block following the source symbol block of its complementary layer.

Figure 14Figure 14 shows an example of source symbol block generation for a layered media data with two layers for LA-FEC. The Base layer and the Enhancement layer, with the enhancement layer depending on the base layer of a layered media stream.

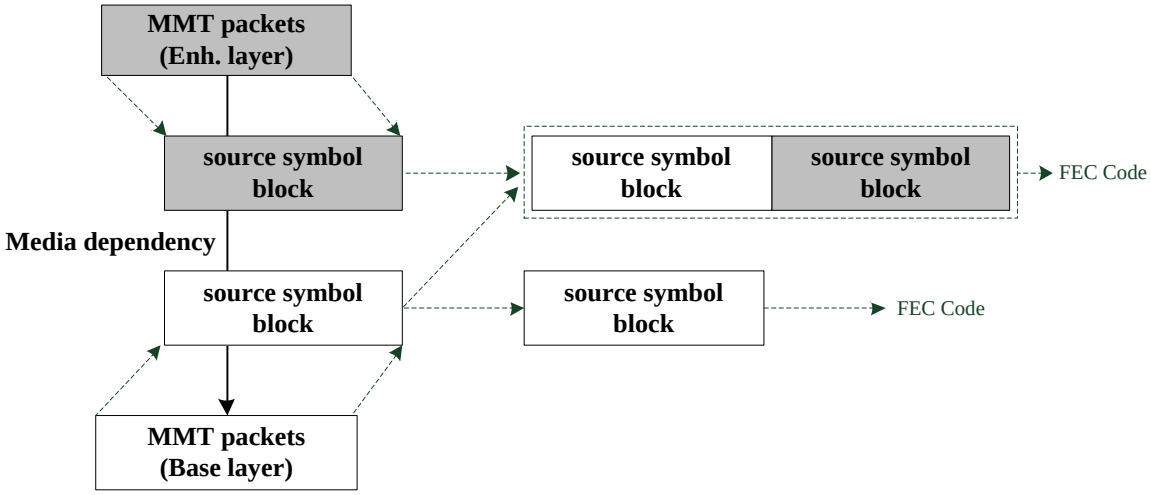


Figure 14 — LA-FEC FEC structure

7.5.3 Encoding Symbol Format

7.5.3.1 Introduction

This sub-clause specifies the format of encoding symbol block which consists of a source symbol block and a repair symbol block generated from the source symbol block. Source symbol block is generated from source packet block according to a given SSBG mode. A repair symbol block is generated from the associated source symbol block by FEC encoding method. Encoding symbol format is depicted as shown in Figure 15.

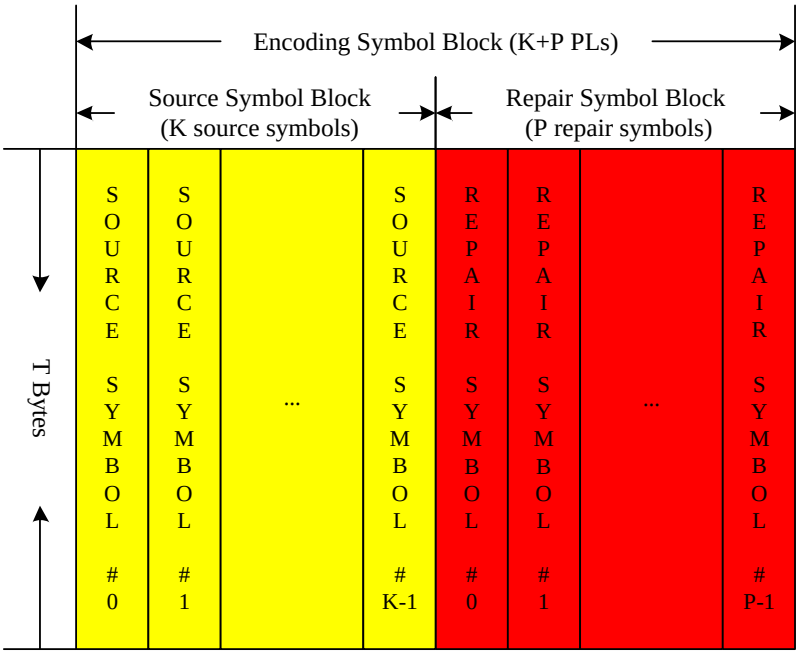


Figure 15 — Encoding Symbol Format

7.5.3.2 Source Packet Block

This sub-clause specifies the data entity protected by FEC. An FEC source flow associated with a designated FEC encoded flow identifier is protected. An FEC source flow is segmented into one or more source packet blocks. A source packet block consists of a pre-determined number of MMT packets to be protected whose sizes may be fixed or variable, and is converted to a source symbol block for FEC encoding.

7.5.3.3 Source Symbol Block Format

This sub-clause specifies source symbol block, which consists of pre-determined number of source symbols, which are generated from source packet block for FEC encoding. This sub-clause specifies three kinds of source symbol block generation (SSBG) modes, so called `ssbg_mode0`, `ssbg_mode1` and `ssbg_mode2`. In a mode with constant MMT packets size, `ssbg_mode0` is used and in a mode with variable MMT packets size, `ssbg_mode1` or `ssbg_mode2` is used.

In `ssbg_mode0`, the source symbol block is the exactly same as the source packet block as all MMT packets have the same size. This means that the number of MMT packets in the source packet block shall be the same as the number of source symbols in the source symbol block and each MMT packet #i is exactly the same as each source symbol #i (i=0, 1, …, K-1). In this mode, for one stage FEC coding structure (M=1) a source symbol block is generated from a source packet block without padding bytes and for two stage and LA-FEC coding structure (M>1) ith source symbol block is generated from ith source packet block in a source packet block (i=0,1,...,M-1) without padding bytes as shown in Figure 16.

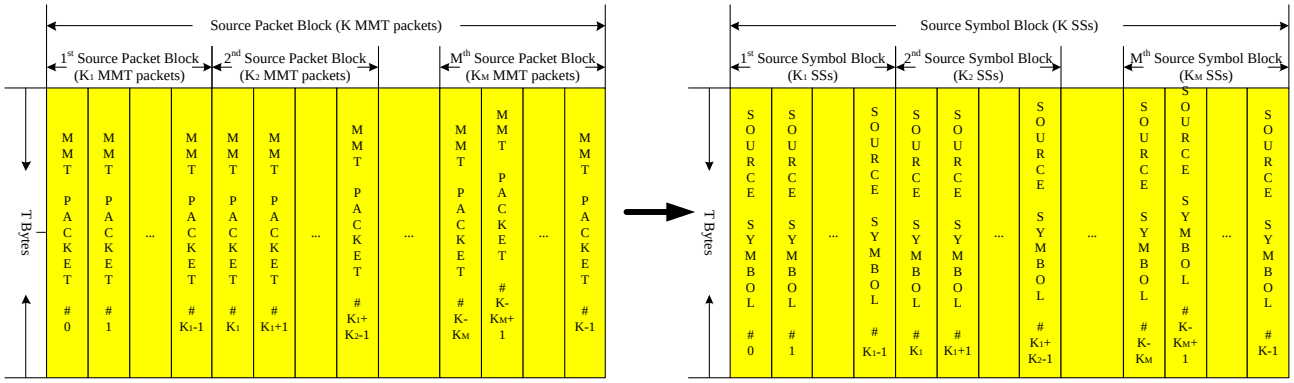


Figure 16 — Source Symbol Block Generation with no padding bytes (`ssbg_mode0`)

In `ssbg_mode1`, source symbol block is generated from source packet block in the same manner as `ssbg_mode0` except each MMT packet #i has possibly padding bytes to make its size be the same as T used as length of its associated parity symbols. This means that the number of MMT packets in the source packet block is the same as the number of source symbols in its associated source symbol block and each source symbol #i is generated by adding possible padding bytes (all 00h) to the corresponding MMT packet #i. In this mode, for one stage FEC coding structure (M=1) a source symbol block is generated from a source packet block with possibly padding bytes (all 00h) and for two stage and LA-FEC coding structure (M>1) ith source symbol block is generated from ith source packet block in a source packet block (i=0,1,...,M-1) with possibly padding bytes (all 00h) as shown in Figure 17.

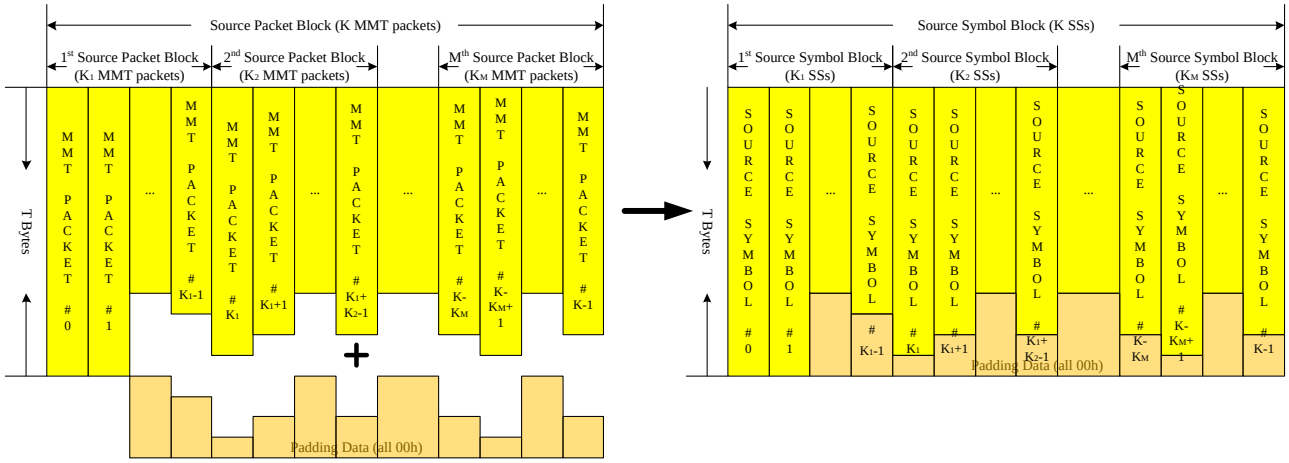


Figure 17 — Source Symbol Block Generation with padding bytes (`ssbg_mode1`)

In `ssbg_mode2`, for one stage FEC coding structure a source symbol block is generated from a source packet block with possibly padding bytes (all 00h). A single source symbol block consists of K_{SS} source symbols generated from a single source packet block with possibly padding bytes (all 00h) and each source symbol consists of the same N (>1) number of symbol elements. This means that the single source symbol block consists of $N*K_{SS}$ symbol elements. MMT packet #0 of the source packet block is placed into the first s_0 number of symbol elements in the source symbol block with possibly padding bytes up to the boundary of the last symbol element of the first s_0 number of symbol elements in the source symbol block. Then, MMT packet #1 of the source packet block is placed into the next s_1 number of symbol elements in the source symbol block in the same manner as that of the MMT packet #0. In this way, MMT packet #K_{SP}-1 of the source packet block is placed into the next s_{Ksp-1} number of symbol elements in the source symbol block in the same manner as that of the MMT packet #0. If $K_{SS}T$– sum { s_iT', i = 1, …, K_{SP}} is not zero, then P number of padding bytes (all 00h) are placed into the last symbol elements in the single source symbol block.

For two stage and LA-FEC coding structures ith source symbol block is generated from ith split source packet block in an source packet block with possibly padding bytes (all 00h) (i=0,1,...,M-1) in exactly the same manner as that of one stage FEC coding structure. All source symbol blocks generated from a single source packet block are concatenated to form a single source symbol block.

The detailed specification for this SSBG mode is as follows:

Let

K_{SP} be the number of MMT packets in an source packet block.

K_{SS} be the number source symbols in a source symbol block.

R_i denote the octets of the *i*-th MMT packet to be add to the source symbol block.

S_i be the length of R_i

T be the source symbol size in bytes.

N be the number of symbol elements composing an source symbol (N ≥ 2)

T' be the symbol element size in bytes. ($T' = T/N$)

s_i be the smallest integer such that $s_iT/N = s_iT' >= S_i$.

P_i denote $s_iT'–S_i$ zero octets.

P denote $K_{SS}T–$ sum { s_iT', i = 1, …, K_{SP}} zero octets.

NOTE Pi are padding octets to align the start of each MMT packet with the start of an symbol element.

Then, the source symbol block is constructed by concatenating R_i, P_i for i = 1, 2, …, K_{SP}, and P and dividing them into source symbols of size T sequentially.

An example of forming a source symbol block is depicted as shown in Figure 18. In this example, five MMT packets of lengths 34, 30, 56, 40 and 48 bytes have been placed into a source symbol block with 8 source symbols of size T = 32 bytes and N = 2. Note that each source symbol in Figure 18 consists of 2 symbol elements of size T/2 = 16 bytes.

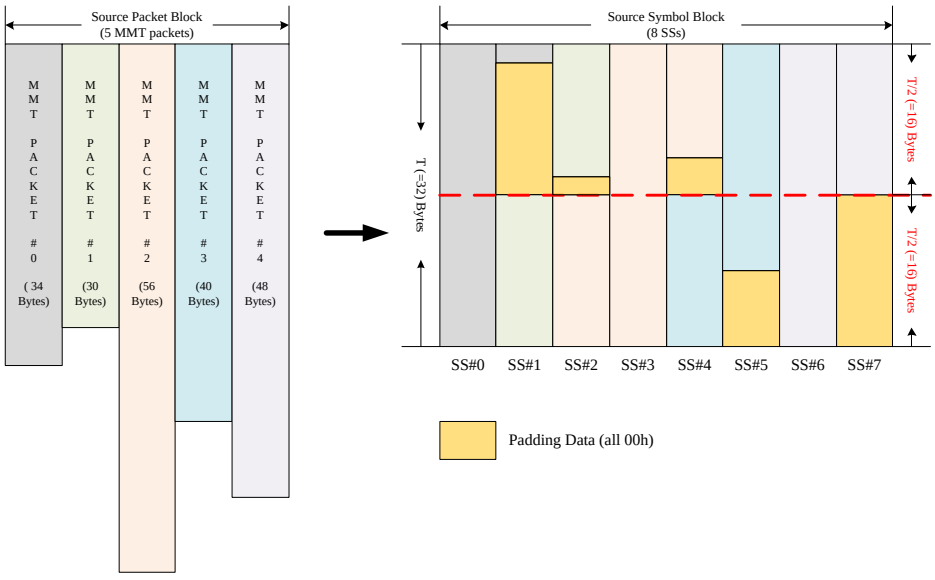


Figure 18 — An example of Source Symbol Block Generation with padding bytes (`ssbg_mode2`)

7.5.4 FEC Source or Repair Packet Format

7.5.4.1 Introduction

This sub-clause specifies the format of FEC source or repair packet, which includes FEC Payload ID, to deliver MMT packet or repair symbols. This sub-clause specifies two kinds of FEC source or repair packet formats. One is the format of FEC source packet and the other is the format of FEC repair packet.

For delivery of a source packet block, FEC source packets are generated from the source packet block and for delivery of a repair symbol block, which is generated to protect the source packet block by FEC encoding, FEC repair packets are generated from the repair symbol block. Source FEC payload ID is added to an MMT packet to form an FEC source packet and repair FEC payload ID is added to one or more repair symbols of repair symbol block to form an FEC repair packet.

FEC source packet is MMT packet with AL-FEC protection and FEC repair packet is MMT packet for repair symbol(s).

Source FEC Payload ID for an FEC source packet identifies the source symbol carried by the FEC source packet or the symbol element carried by the FEC source packet. Repair FEC Payload ID for an FEC repair packet identifies the repair symbol(s) carried by the FEC repair packet and its associated source packet block.

7.5.4.2 FEC Source Packet Format

An FEC source packet is a packet to deliver an MMT protocol which is protected by FEC and is composed of an MMT payload followed by an FEC payload ID.

FEC source packet format is depicted in Figure 19Figure 19. The MMT packet header, MMT payload header and payload data in Figure 19Figure 19 are protected by FEC encodingTo maintain consistency with the packet format and to avoid putting the MMT payload into separate parts of the packet, source FEC payload ID are placed at the end of the FEC source packet.

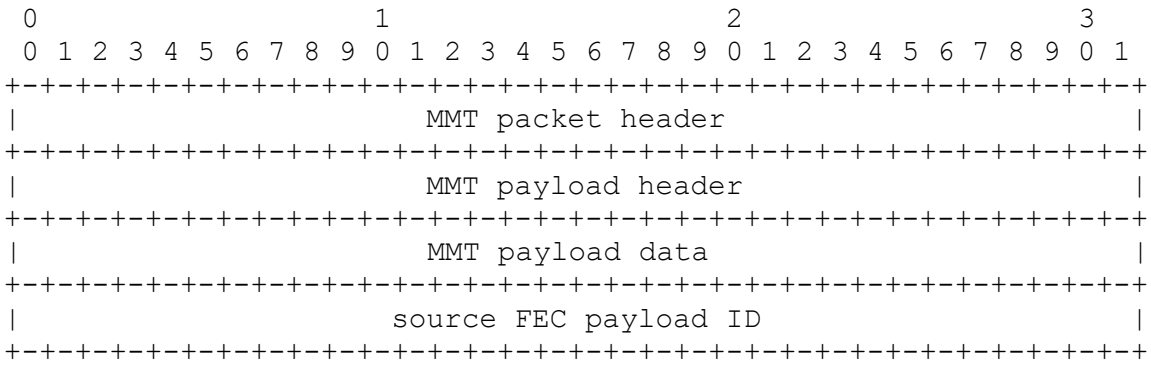


Figure 19 — FEC source packet format

7.5.4.3 FEC Repair Packet Format

An FEC repair packet is a packet to deliver one or more repair symbols to recover its associated source symbol block and it is composed of MMT packet header, MMT payload header, an repair FEC payload ID and one or more repair symbols.

FEC repair packet format is depicted in Figure 20Figure 20. The payload data in Figure 20Figure 20 represents repair data for recovering its associated source symbol block. For fast and easy acquisition of repair FEC payload ID, repair FEC payload ID is placed in front of repair symbol(s).

An FEC repair packet shall carry one repair symbol for `ssbg_mode0` and `ssbg_mode1`. In `ssbg_mode2`, an FEC repair packet can carry one or more repair symbols. Furthermore, all FEC repair packets in an FEC source or repair packet block shall carry the same number of consecutive repair symbols in the corresponding repair symbol block except the last FEC repair packet..

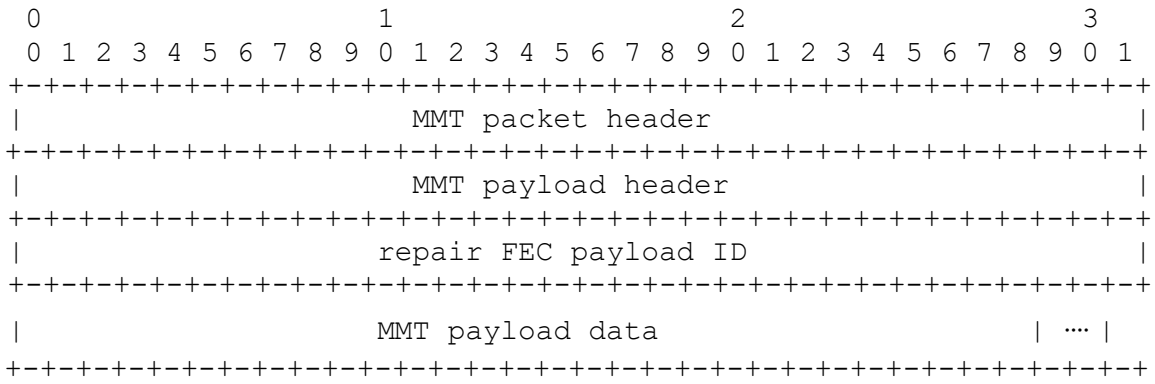


Figure 20 — FEC Repair Packet Format

7.5.5 FEC Signaling

7.5.5.1 Introduction

This sub-clause specifies signaling of FEC configuration information such as AL-FEC message and FEC payload ID. All FEC configuration information is communicated between sending entity and receiving entity by using MMT protocol.

AL-FEC message is specified in sub-clause 9.4.3 and FEC payload ID is specified following sub-clause..

7.5.5.2 Source FEC payload ID

Source FEC payload ID is defined as follows:

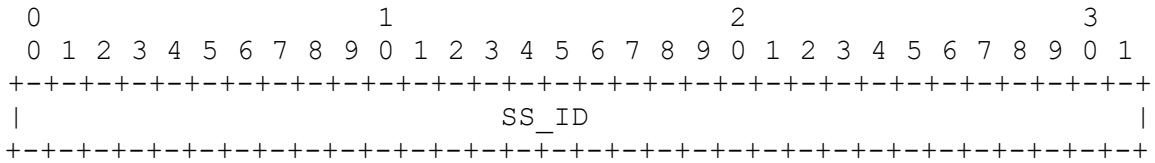


Figure 21 — Source FEC payload ID

`SS_ID (32bit)` – the sequence number identifying source symbols within FEC source packet. If `ssbg_mode == 00` or `ssbg_mode == 01`, the `SS_ID` shall be incremented per FEC source packet by one. If `ssbg_mode == 10`, the `SS_ID` shall be incremented per FEC source packet by number of symbol elements in it. If `fec_coding_structure =! 0011`, the `SS_ID` shall be incremented per source symbol, If `fec_coding_structure == 0011` is used, the lowest `SS_ID` of a source symbol block shall be the highest `SS_ID + 1` of all flows of the preceding source symbol block. The first `SS_ID` of the same source symbol block within each flow shall be used as synchronization point of all source symbol blocks of all flows as illustrated in Figure 22 with two media layers.

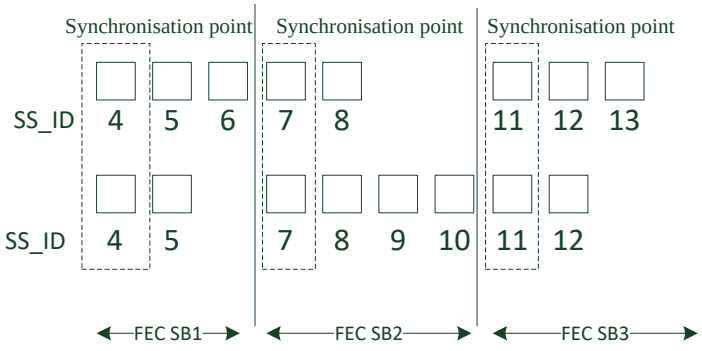


Figure 22 — Synchronization point for FEC SBs with synchronized starting SI per FEC SB

7.5.5.3 Repair FEC payload ID

The repair FEC payload ID is defined as follows:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                            SS_Start                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         RP_length             |           RP_ID               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         SB_length             |           ....
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 23 — Repair FEC payload ID

`SS_Start (32 bits)` – indicates the boundary of source block. It is set to the `SS_ID` of the first source symbol in the FEC source block.

`RP_length (24 bits)` – the number of repair symbols generated from that source block.

`RP_ID (24 bits)` – an integer number for identifying the first repair symbol in the packet. It starts with 0 and is incremented by 1 with each repair symbol generated from the source block.

`SB_length[N] (N*24 bits)` – If `fec_coding_structure =! 0011`, N shall be equal to '1' and `SB_lenght` indicates the number of source symbols in the source block. If `fec_coding_structure = 0011`, N shall equal to the number of complementary representations + 1 and `SB_length[i]` indicates the number of source symbols from i-th flow of the source block.

NOTE A j-th flow is a complementary flow from the i-th flow if j<i.

8 Cross Layer Interface (CLI)

8.1 Introduction

Cross Layer Interface provides the means within single MMT entity to support QoS by exchanging QoS-related information between application layer and underlying layers including MAC/PHY layer. Application layer provides information about media characteristics as top-down QoS information while underlying layers provides bottom-up QoS information such as network channel condition.

CLI provides the unified interface between application layer and various network layers including IEE802.11 WiFi, IEEE 802.16 WiMAX, 3G, 4G LTE, etc. Common network parameters of popular network standards are abstracted as the NAM parameters for static and dynamic QoS control of real-time media application through any network.

8.2 Cross Layer Information

8.2.1 Introduction

MMT defines an interface for exchanging Cross Layer Information between the application layer and the underlying network layer. The interface allows for top-down as well as bottom-up flow of cross layer information. The Cross Layer Information provides QoS information that may be used by the involved functions to optimize the overall delivery of the media data. MMT entities may support this interface for Cross Layer Information.

8.2.2 Top-down QoS information

Application layer provides top-down QoS information about media characteristics to underlying layers. There are two kinds of top-down information such as Asset level information and packet level information. Asset information is used for capability exchange and/or (re)allocation of resources in underlying layers. Packet level top-down information is written in appropriate field of every packet for underlying layers to indentify QoS level to support.

8.2.3 Bottom-up QoS information

The underlying layers provide bottom-up QoS information to the application layer about time-varying network condition which enables faster and more accurate QoS control in the application layer. Bottom-up information is represented as an abstracted fashion to support heterogeneous network environments. These parameters are measured in the underlying layers and read by the application layer, periodically or on request of the MMT application.

8.2.4 Network Abstraction for Media (NAM) parameter

Network Abstraction for Media (NAM) parameter is used for an interface between application layers and underlying layers. NAM provides unified representation of network QoS parameters for assuring to communicate with legacy and future standards of underlying layers.

8.2.4.1 Absolute NAM

Absolute NAM information is raw QoS value measured in each appropriate unit. For example, bitrate is represented in the unit of bits per second while jitter is in the unit of second.

8.2.4.2 Relative NAM

Relative NAM information represents ratio of expected NAM value to current NAM value so that it is unit-less and inform tendency of change

8.2.5 Syntax

The CLI information is exchanged using a Network Abscration for Media (NAM) parameter or a relative NAM parameter.

The syntax of absolute parameters for NAM is shown in Table 13.

Table 13 — Absolute NAM parameter

Syntax	size (bits)	Mnemonic
Network Abstraction for Media information {		
CLI_id	8	unsigned int
available_bitrate	32	Float
buffer_fullness	32	float
peak_bitrate	32	Float
average_bitrate_period	16	unsigned int
current_delay	32	float
SDU_size	32	unsigned integer
SDU _loss_ratio	8	unsigned integer
generation_time	32	unsigned int
BER	32	float
}		

The syntax of relative parameters for NAM is shown in Table 14.

Table 14 — Relative NAM parameter

Syntax	size (bits)	Mnemonic
relative_difference Network Abstraction for Media information () {		
CLI_id	8	unsigned integer
relative_bitrate	8	float
relative_buffer_fullness	8	float
relative_peak_bitrate	8	Float
average_bitrate_period	16	unsigned int
current_delay	32	float

generation_time	32	float
BER	32	float
}		

8.2.6 Semantics

`CLI_id` – The `CLI_id` is an arbitrary integer number to identify this NAM among the underlying network.

`available_bitrate` – the `available_bitrate` is the instantaneous bitrate that the scheduler of the underlying network expects to be available for the MMT stream. The `available_bitrate` is expressed in kilobits per second. Overhead for the protocols of the underlying network is not included.

`buffer_fullness` – signal the buffer level of the generating function. The buffer is used to absorb any excess data that caused by the data rates above the `available_bitrate`. The `buffer_fullness` is expressed in bytes.

`peak_bitrate` – the `peak_bitrate` is maximum allowable bitrate that the underlying network is able to handle temporarily as input from to the MMT stream. The `peak_bitrate` is expressed in kilobits per second. Overhead for the protocols of the underlying network is not included. Note that the MMT input stream bitrate shall not exceed the `available_bitrate` over any period of `average_bitrate_period`.

`average_bitrage_period` – provides the period of time over which the average bitrate of the input stream shall be calculated. The `average_bitrate_period` is provided in units of milliseconds. If the `peak_bitrate_flag` is set to '1', then this field shall be set appropriately.

`current_delay` – the `current_delay` parameter indicates the last measured value of the last hop transport delay. The `current_delay` is expressed in milliseconds.

`SDU_size` – Service Data Unit (SDU) is data unit in which the underlying network delivers the MMT data. The `SDU_size` specifies the length of the SDU and is expressed in bits. Overhead for the protocols of the underlying network is not included.

`SDU_loss_rate` – The `SDU_loss_ratio` is fraction of SDUs lost or detected as errorneous. Loss ratio of MMT packets can be calculated as a function of `SDU_loss_ratio` and `SDU_size`. The `SDU_loss_ratio` is expressed in percentile.

`generation_time` – The timestamp of the generation of the current NAM. The `generation_time` is expressed in milliseconds and starting from an arbitrary value.

`relative_bitrate` – the `available_bitrate` change ratio(%) between the current NAM and the previous NAM parameter.

`relative_buffert_fullness` – the remaing `buffer_fullness` change ratio(%) between the current NAM and the previous NAM parameter.

`relative_peak_bitrate` – the `peak_bitrate` change ratio(%) between the current NAM and the previous NAM parameter.

`BER` – Bit Error Rate is the last measured BER at the PHY or MAC layer. For `BER` from PHY layer, the value shall be presented as a positive value. For `BER` from MAC layer, this value shall be presented as a negative valueof which the absolute value is to be used.

9 Signaling

9.1 Introduction

Signaling defines the set of message formats to be used to provide the information necessary for the delivery and consumption of the package. The delivery of signaling message format with MMT protocol is defined in this specification. This specification describes the message format for carrying tables, descriptors or the delivery related information. A table has a set of elements and attributes for specific information. A table may include descriptors for more detailed information.

Six messages are defined that relate to the consumption of the Package;

— Package Access (PA) message: It includes all tables required for Package access including MMT Package Table and MMT Composition Information Table (see subclause 9.3.2);

— MMT Composition Information (MCI) message: It includes MCI table encapsulating a complete CI or a subset of CI. It may also include MPT corresponding MCI table for fast Package consumption (see subclause 9.3.3);

— MMT Package Table (MPT) message : It includes MMT Package Table providing a whole information or a part of information required for a single Package consumption (see subclause 9.3.4);

— Clock Relation Information (CRI) message: it includes CRI Table providing the clock relation information used for the mapping between the NTP Clock and MPEG-2 System Time Clock (see subclause 9.3.5);

— Device Capability Information (DCI) message: it includes DCI table providing the required device capability information for a Package consumption (see subclause 9.3.6);

— Security Software Request (SSWR) message: it is used to request security software for comsuming MMT Package or Asset by a MMT receiving entity. It can also include PA table or MPT (see subclause 9.3.7).

Six messages are defined that relate to the delivery of the Package:

— Measurement Configuration(MC) message: it provides information to configure a measurement (see subclause 9.4.2);

— Application Layer Forward Error Correction (AL-FEC) message: it provides AL-FEC configuration information (see subclause 9.4.3);

— Hypothetical Receiver Buffer Model(HRBM) message: it provides information to configure HRBM operation (see subclause 9.4.4);

— Automatic Repeat-Request(ARQ) message: It provides information required for ARQ operation (see subclause 9.4.5);

— Reception Quality Feedback (RQF) message: It defines a format of measurement report from a receiving entity (see subclause 9.4.6);

— Network Aware Media Feedback (NAMF) message: It defines a format of NAM parameter report from a receiving entity (see subclause 9.4.7).

9.2 Signaling Message

9.2.1 Introduction

Signaling message uses general signaling message format consisting of 3 common fields, one specific field for each signaling message, and a message payload. Message payload carries MMT signaling table.,

The syntax and semantics of general signaling message format is given at 9.2.2 and 9.2.3 respectively.

9.2.2 Syntax

Syntax of general signaling message format is given at Table 15.

Table 15 — Syntax for general signaling message

Syntax	Value	No. of bits	Mnemonic
Signaling_Message () {			
message_id		16	uimsbf
version		8	uimsbf
if(message_id != PA_message && message_id != MCI_message) {			
length		16	uimsbf
} else {			
length		32	uimsbf
}			
extension }			
message_payload {			
}			
}			

9.2.3 Semantics

`message_id` – It indicates a type of signaling message. Each signaling message has the unique value given at Table 32~~Table 31~~.

`version` – It indicates the version of signaing message. MMT receiving entity can verify whether a received message has new information or not.

`length` – It indicates the length of signaling message. 2 bytes is the value for all signaing message except PA messages and MCI message. PA message and MCI have 4 bytes as the length because it is expected that occasionally an MCI table can't be carried by a message having 2 bytes length fields. Also note that a PA message includes at least one MCI table

`extension` – It provides specific information for each signaling message. The content and lengh of this field is specified per each signaling message.

`message_payload` – the payload of the signalling message is defined by each message type, identified by the `message_id`.

9.3 Message for Consumption

9.3.1 Introduction

Signaling messages are used to deliver information for Package consumption. Signaling messages are composed of signaling tables. Each signaling table carries information about a sepcifc aspect of the Package such as Package structure, CI or Clock. Signaling messages can aggregate multiple tables for efficient information provisioning. For example, MCI message delivers MCI only or MCI and a corresponding MPT. The relationship between a message and a table is shown at Figure 24.

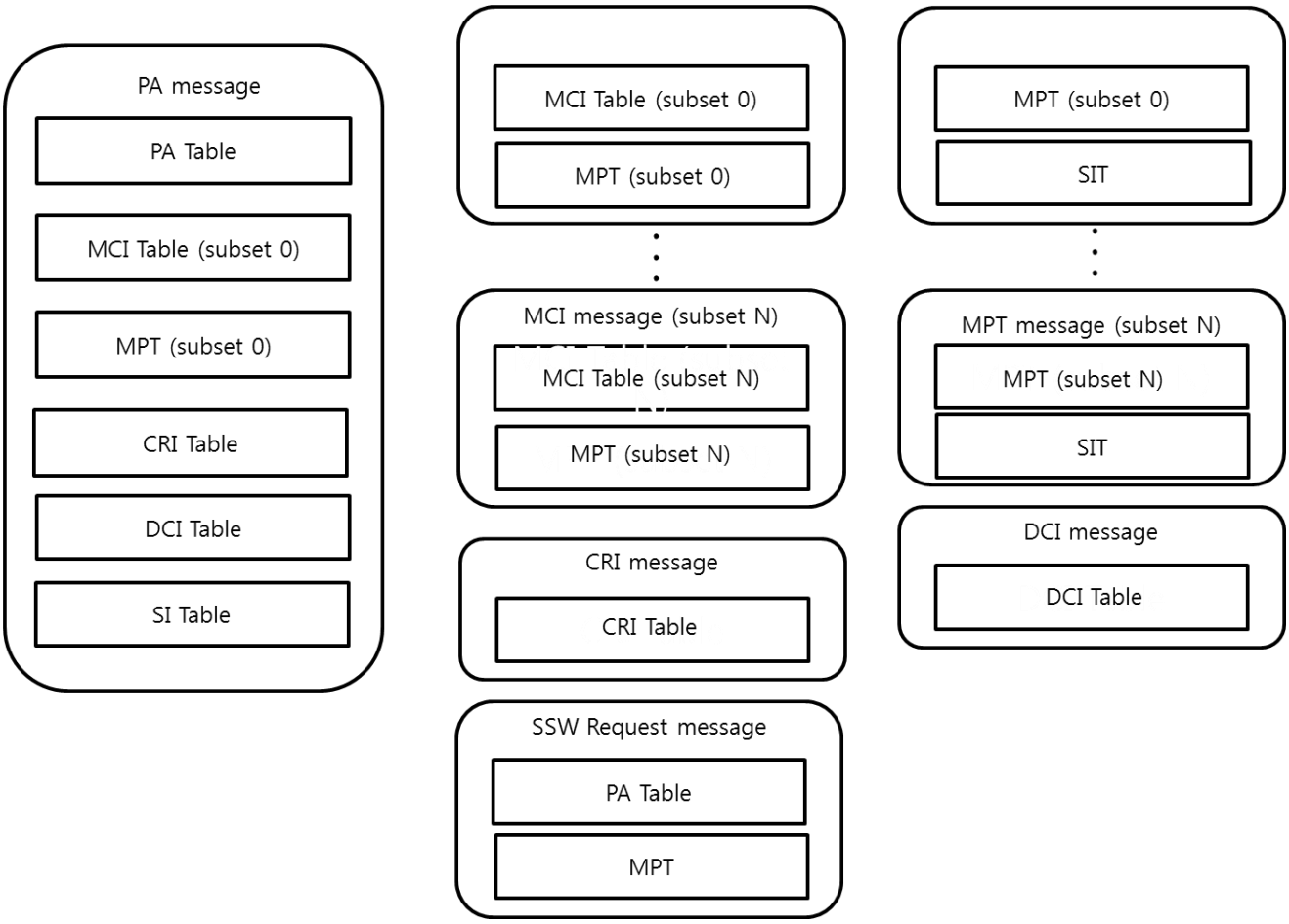


Figure 24 — Structure of the signaling messages and tables for Package consumption

PA messages shall carry a PA table, an MPT, MCI table and a DCI table. A PA message may carry a CRI table. An MCI message shall carry an MCI table and may carry an MPT. MPT message shall carry MPT. CRI message shall carry CRI table. DCI message shall carry DCI table. An MCI message and an MPT message can be split into multiple layers each of which providing a subset of the complete information, for efficient delivery and redundancy reduction.

Some MMT signaling tables shares the same structure of information. For the efficient provision of those information, two descriptors are defined. They are `Clock Relation Information` descriptor and `MMT_general_location_info` descriptor.

9.3.2 Package Access (PA) Message

9.3.2.1 Introduction

A Package Access (PA) message carries a PA table, which has the the information on all other signaling tables for a Package. PA message also carries an MMT Composition Information (MCI) table and an MMT Package Table (MPT) for the fast consumption of Package.

An MMT receving entity shall process the PA message before it processes any other messages.

9.3.2.2 Syntax

The syntax of the PA message is defined in Table 16 and the semantics of its syntax elements are provided in 9.3.2.3

Table 16 — PA Message Syntax

Syntax	Value	No. of bits	Mnemonic
PA_message () {			
message_id		**16**	**uimsbf**
version		**8**	**uimsbf**
length		**32**	**uimsbf**
extension {			
number_of_tables	N1	**8**	**uimsbf**
for (i=0; i<N1; i++) {			
table_id		**8**	**uimsbf**
table_version		**8**	**uimsbf**
table_length		**16**	**uimsbf**
}			
}			
message_payload {			
for (i=0; i<N1; i++) {			
table()			
}			
}			
}			

9.3.2.3 Semantics

`message_id` – It indicates the type of signaling messages.
`version` – It indicates the version of signaling messages.
`length` – It indicates the length of signaling messages. The length of this field is 32 bits. It indicates the length of the PA message counted in bytes starting from the next field to the last byte of the PA message. The value '0' is never used for this field.
`number_of_tables` – It indicates the number of tables included in this PA message.
`table_id` – It indicates the table identifier of the table included in this PA message. It is a copy of the `table_id field` in the table included in the payload of this PA message.
`table_version` – It indicates the version of the table included in this PA message. It is a copy of the version field in the table included in the payload of this PA message.
`table_length` – If the table is not an MCI table, it is a copy of the length field in the table included in the message_payload of this PA message. If the table is an MCI table, it is the length derived from the length and the extension lenth part. In this case the derived length is a value that equals "the actual lenth" – 5.
`table()` – It indicates an MMT signaling table instance. The tables in the payload appear in the same order as the `table_id`s in the extension field. An PA table shall be an instance for `table()`.

9.3.3 MMT Composition Information (MCI) Message

9.3.3.1 Introduction

MMT Composition Information (MCI) message delivers a complete CI or a subset of CI. MCI message uses MCI table for encapsulating CI.

When a subset of CI is used, CI is partitioned into multiple MCI tables. MCI tables for a subset of CI shall have different `table_id`s. The `table_id` values of MCI tables for subset of CI are allocated in a contiguous space in the same increasing order. The MCI table having the lowest `table_id` value provides the base CI amongst the subset of CI and other MCI tables for the remaining subset of CI have different `table_id` value. The maximum number of MCI Table for a subset of MMT-CI is 16.

Each MCI message carrying a subset of CI may have different transmission period and include the MMT Package Table (MPT) associated with the CI that the MCI message carries.

9.3.3.2 Syntax

The syntax of the MCI message is defined in Table 17 and the semantics of its syntax elements are provided below Table 17.

Table 17 — MCI Message Syntax

Syntax	Value	No. of bits	Mnemonic
MCI_message () {			
message_id		**16**	**uimsbf**
version		**8**	**uimsbf**
length	N1	**32**	**uimsbf**
extension {			
reserved	'111 1111'	**7**	**uimsbf**
associated_MPT_flag		**1**	**bslbf**
}			
message_payload {			
MCI_table()			
if (associated_MPT_flag) {			
MPT_table()			
}			
}			
}			

9.3.3.3 Semantics

`message_id`: It indicates MCI message ID. The length of this field is 16 bits.
`version`: It indicates the version of MCI Message. The length of this field is 8 bits.
`length`: It indicates the length of the MCI message counted in bytes starting from the next field to the last byte of the MCI message. The value '0' is never used for this field. The length of this field is 32 bits.
`associated_MPT_flag`: If this flag is set to '1', it indicates MCI message also carries an associated MPT. The simultaneous delivery of MCI table and MPT in a single MCI message helps an MMT receiving entity to reduce the signaling acquisition time for a Package consumption.
`MCI_table()`: an MCI table defined in 9.3.9.
`MPT_table()`: an MPT defined in 9.3.10.

9.3.4 MMT Package Table (MPT) Message

9.3.4.1 Introduction

The MPT signaling message carries an MPT. Subset of MPTs can be delivered by different MPT messages.

MMT Package Table (MPT) provides the information for a single Package. For layered delivery of a Package having a subset of CI, an MPT can be splitted into multiple subsets of MPT. The MPT Subset-0 is the base

MPT. MPTs at different subsets shall have different table_ids. 14 contiguous values are assigned for MPT table_id. The value of table_id '15' is assigned for the complete MPT.

An MPT may be included in a PA message with other tables for the efficient operation of signaling acquisition.

9.3.4.2 Syntax

The syntax of the MPT message is defined in Table 18~~Table 18~~ and the semantics of its syntax elements are provided below Table 18~~Table 18~~. An MPT message carries only one complete MPT or one Layer-N MPT.

Table 18 — MPT Message Syntax

Syntax	Value	No. of bits	Mnemonic
MPT_message () {			
message_id		**16**	**uimsbf**
version		**8**	**uimsbf**
length		**16**	**uimsbf**
extension {			
}			
message_payload {			
MPT()			
}			
}			

9.3.4.3 Semantics

message_id – It indicates the ID of MPT Message. The length of this field is 16 bits.
version – It indicates the version of MPT message. MMT receiving entity can check whether the received message is new or not.
length – It indicates the length of the MPT message. The length of this field is 16 bits. It indicates the length of the MPT message counted in bytes starting from the next field to the last byte of the MPT message. The value '0' is never used for this field.
MPT() – MPT defined in 9.3.10

9.3.5 Clock Relation Information (CRI) Message

9.3.5.1 Introduction

This optional message carries clock relation information to be used for mapping between the NTP timestamp and MPEG-2 System Time Clock (STC).

To achieve synchronization between the Assets that uses NTP timestamps and the MPEG-2 ES that uses MPEG-2 Presentation Time Stamp (PTS), it is necessary to inform the relationship between the NTP timestamp and the MPEG-2 STC to an MMT receiving entity by periodically delivering values of the NTP_timestampe_sample and the STC_sample at the same time points. If more than one MPEG-2 ES with different MPEG-2 STCs are used, more than one CRI descriptors are delivered.

9.3.5.2 Syntax

The syntax of the CRI message is defined in Table 19~~Table 19~~ and the semantics of its syntax elements are provided below Table 19~~Table 19~~.

Table 19 — CRI Message Syntax

Syntax	Value	No. of bits	Mnemonic
CRI_message () {			
message_id		**16**	**uimsbf**
version		**8**	**uimsbf**
length		**16**	**uimsbf**
extension {			
}			
message_payload {			
CRI_table()			
}			
}			

9.3.5.3 Semantics

`message_id` – It indicates the type of CRI messages. The length of this field is 16 bits.
`version` – It indicates the version of CRI messages. MMT receiving entity can check whether the received message is new or not. The length of this field is 8 bits.
`length` – It indicates the length of the CRI message counted in bytes starting from the next field to the last byte of the CRI message. The value '0' is never used for this field. The length of this field is 16 bits.
`CRI_table()` – A CRI table defined in 9.3.11.

9.3.6 Device Capability Information (DCI) Message

9.3.6.1 Introduction

The DCI message delivers the Device Capability Information (DCI) table that provides the required device capabilities for the Package consumption.

9.3.6.2 Syntax

The syntax of the DCI message is defined in Table 20Table 20 and the semantics of its syntax elements are provided below Table 20Table 20.

Table 20 — DCI Message Syntax

Syntax	Value	No. of bits	Mnemonic
DCI_message () {			
message_id		**16**	**uimsbf**
version		**8**	**uimsbf**
length		**16**	**uimsbf**
extension {			
}			
message_payload {			
DCI_table()			
}			
}			

9.3.6.3 Semantics

`message_id` – It indicates DCI messages. The length of this field is 16 bits.
`version` – It indicates the version of DCI messages. MMT receiving entity can check whether the received message is new or not. The length of this field is 8 bits.
`length` – It indicates the length of the DCI message counted in bytes starting from the next field to the last byte of the DCI message. The value '0' is never used for this field.
`DCI_table()` – It provided the required device capabilities for the Package consumption. It is defined in 9.3.12.

9.3.7 Downloadable DRM and CAS SW Request(SSWR) Message

9.3.7.1 Introduction

The overall operation of downloadable DRM and CAS for MMT is described in Annex F. There are 5 steps in Annex F. Among them, the message for the DDRM/DCAS SW request is sent from a MMT client to Downloadable DRM/CAS server. The message for DRM and CAS SW request is defined below.

9.3.7.2 Syntax

The syntax of Security SW(SSW) Request message is defined in Table 21

Table 21 — SSW Request Message

Syntax	Value	No. of bits	Mnemonic
SSRW_message () {			
message_id		**16**	**uimsbf**
version		**8**	**uimsbf**
length	N1	**16**	**uimsbf**
extension {			
si_table_id		**16**	**uimsbf**
}			**bslbf**
message_payload {			
token_ID **{**			
token_ID_URI	**N1**	**8**	**uimsbf**
for (i=0; i<N1; i++) {			
uri_length		**8**	**uimsbf**
}			
Number_of_deviceID	**N2**	**8**	**uimsbf**
for (i=0; i<N2; i++) {			
device_ID {			
deviceID_length	**N3**	**8**	**uimsbf**
for (j=0; j<N3; j++) {			
length_byte		**8**	**uimsbf**
}			
}			
}			
tokenIssure_ID **{**			
tokenIssure_ID_URI	**N4**	**8**	**uimsbf**
for (i=0; i<N4; i++) {			
uri_length		**8**	**uimsbf**

}			
}		**64**	**uimsbf**
tokenIssueTime		**64**	**uimsbf**
tokenExfireTime			
information_table_info {			
number_of_tables			
for (i=0; i<N5; i++) {	**N5**	**8**	**uimsbf**
MMT_signaling_table_id		**8**	**uimsbf**
MMT_signaling_table_version		**8**	**umisbf**
}			
}			
}			
}			
}			

9.3.7.3 Semaintics

`message_id` – It indicates the type of MMT Signaling messages.
`version` – It indicates the version of MMT Signaling messages.
`length` – It indicates the length of MMT signaling messages.. The value '0' is never used for this field.
`extension` – It provide the additional information for SSW. In SSW, identification of Security Information table (`si_table_id`) corresponding to this SSW message is provided.
`tokenID` – Identification of Token and is provided by Token Provider. Token should be provided by the trustable entity. It has sub elements of Device ID, Token Issuer ID, Issue Time and Expire Time
`deviceID`– It provides identification of device(s) under Token. If MMT client want to consume MMT Asset/Package two different devices, then multiple Device ID should be provided.
`tokenIssureID`– Identification of trust entity that issues a token. This filed is to be used by D-DRM/D-CAS server to verify the validity of Token.
`tokenIssueTime` – a time at which the Token is issued. The unit of this field is second. NTC format will be used.
`tokenExfireTime` – a time at which the Token is expired. The unit of this field is second. NTC format will be used.
`MMT_signaling_table_info` – This field provides the information of MPT related to MMT Package/Asset to be descripted by downloaded DRM or CAS.

9.3.8 PA Table

9.3.8.1 Introduction

A PA table provides the information on all other signaling tables for the consumption of a Package.

9.3.8.2 Syntax

The syntax of the PA table is defined in Table 22 and the semantics of its syntax elements are provided below Table 22 .

Table 22 — PA Syntax

Syntax	Value	No. of bits	Mnemonic
PA () {			

table_id		**8**	**uimsbf**
version		**8**	**uimsbf**
length		**16**	**uimsbf**
information_table_info {			
number_of_tables	N1	**8**	**uimsbf**
for (i=0; i<N1; i++) {			
signaling_information_table_id		**8**	**uimsbf**
signaling_information_table_version		**8**	**uimsbf**
location {			
MMT_general_location_info()			
}			
reserved	'1111 11'	**6**	**bslbf**
alternative_location_flag		**1**	**bslbf**
if (alternative_location_flag == 1) {			
alternative_location {			
MMT_general_location_info()			
}			
}			
}			
}			
reserved	'1111 111'	**7**	**bslbf**
private_extension_flag		**1**	**bslbf**
if (private_extension_flag == 1)			
private_extension {			
}			
}			
}			

9.3.8.3 Semantics

`table_id` – indicates the identification of PA table.

`version` – It indicates Version of the PA table. The newer version obsoletes the information in any older version as soon as it has been received.

`length`: It indicates the length of the PA table counted in bytes starting from the next field to the last byte of the PA table. The value '0' is never used for this field.

`number_of_tables` – It indicates the number of signalling tables whose information is provided in this PA table.

`signaling_table_id` – It indicates the ID of the signaling table whose information is provided in this PA table.

`signaling_table_version` – It indicates the version of the signaling table whose information is provided in this PA table.

`MMT_general_location_info` – It provides the location of the signaling table whose information is provided in this PA table. MMT_general_location_info is defined in 9.3.14.2.

`alternative_location_flag` – If this flag is set '1', an alternative address where a MMT receiving entity gets the information table is provided.

`MMT_general_location_info_for_second_location` – It provides the information of an alternative address where a MMT receiving entity gets the signaling table. Only `location_type` from '0x07' to '0x0B' shall be used in `MMT_general_location_info` for second location.

`private_extension_flag` – If this flag is '1', the private extension is present.

`private_extension` – A syntax element group serving as a container for proprietary or application-specific extensions.

9.3.9 MCI Table

9.3.9.1 Introduction

An MCI table carries complete CI or subset of CI, which is a part of complete CI. In case of subset of CI, an MCI table for each subset is delivered in a separate message.

9.3.9.2 Syntax

The syntax of the MCI table is defined in Table 23Table 23 and the semantics of its syntax elements are provided below Table 23Table 23.

Table 23 — MCI Table Syntax

Syntax	Value	No. of bits	Mnemonic
MCI_table () {			
table_id		**8**	**uimsbf**
version		**8**	**uimsbf**
length	N1	**16**	**uimsbf**
reserved	'111 1111'	**7**	**bslbf**
CI_mode		**1**	**bslbf**
for (i=0; i<N1-1; i++) {			
CI_content		**8**	**uimsbf**
}			
}			

9.3.9.3 Semantics

`table_id` – It indicates the identifier of the MCI table. A complete CI and each subset of CI shall have distinct table identifiers. Thus order of subset of CI number can be implicitly represented by this field. Since the `table_id` values are assigned contiguously, the CI subset number can be deduced from this field, i.e., the CI subset number equals this field minus the `table_id` of the base MCI table. The number 0 indicates base CI and the numbers '1'~'14' indicate subset of CI. The number '15' has a special meaning since it indicates a complete CI.

`version` – It indicates version of the MCI table. The newer version overrides the older one as soon as it has been received if `table_id` indicates a complete MCI, if Subset-0 MCI has the same version value as this field (when `CI_mode` is '1'), or if all MCIs with lower-subset number have the same version value as this field (when `CI_mode` is '0'), or if processing of the MCIs are independent (when `CI_mode` is '2'). If Subset-0 MCI table has a newer version, all CIs with higher subset number up to 14 previously stored within an MMT receiving entity are treated as outdated. When the CI subset number is not 0 and `CI_mode` is '1', the contents of the MCI table with version different from that of subset-0 CI stored in a MMT receiving entity shall be ignored. Also when the CI subset number is not 0 and `CI_mode` is '0', the contents of the MCI table with version different from that of lower-subset CIs stored in a MMT receiving entity shall be ignored. It shall be modulo-256 incremented per version change.

`length` – It indicates the length of the MCI table counted in bytes starting from the next field to the last byte of the MCI table. The value '0' is never used for this field.

`CI_mode` – It indicates the mode of a CI subset processing. In "`sequential_order_processing_mode`" and with the subset number of this CI non-zero, a MMT receiving entity shall receive all CIs with lower subset number that have the same version as this CI before it processes this CI. In other words, a MMT receiving entity can't process subset-3 CI, if it doesn't have subset-2 CI with the same version. In "`order_irrelevant_processing_mode`" and with the layer number of this MMT-CI non-zero, a MMT receiving entity should process a CI right after it receives the CI as long as the subset-0 CI stored in a MMT receiving entity has the same version as

this CI. In "independent_processing_mode", versions of each subset of CIs are managed individually. Fragmented CI is adapted in this mode.

Table 2423 — value of CI_mode

Value	Description
00	"sequential_order_processing_mode"
01	"order_irrelevant_processing_mode"
10	"independent_processing_mode"
11	reserved

CI_content – A single byte at position of the CI.

9.3.10 MP Table

9.3.10.1 Introduction

A complete MPT has the information related to a Package including the list of all Assets. The subset MPT has a part of information from the complete MPT. In addition, MPT subset-0 has the minimum information required for Package comsumption.

9.3.10.2 Syntax

The syntax of the MPT is defined in Table 25Table 25 and the semantics of its syntax elements are provided below Table 25Table 25.

Table 25 — MPT Syntax

Syntax	Value	No. of bits	Mnemonic
MPT() {			
table_id		**8**	**uimsbf**
version		**8**	**uimsbf**
length		**16**	**uimsbf**
If (table_id == SUBSET_0_MPT_TABLE_ID) {			
MMT_package_id {			
MMT_package_id_length	N1	**8**	**uimsbf**
for (i=0; i<N1; i++) {			
MMT_package_id_byte		**8**	**uimsbf**
}			
}			
}			
MPT_descriptors {			
MPT_descriptors_length	N2	**16**	**uimsbf**
for (i=0; i<N2; i++) {			
MPT_descriptors_byte		**8**	**uimsbf**
}			
}			
}			
number_of_assets	N3	**8**	**uimsbf**
for (i=0; i<N3; i++) {			

asset_id {			
asset_id_length	N4	**8**	**uimsbf**
for (j=0; j<N4; j++) {			
asset_id_byte		**8**	**uimsbf**
}			
}			
reserved	'1111 111'	**7**	**bslbf**
asset_clock_relation_flag		**1**	**bslbf**
if (asset_clock_relation_flag == 1) {			
asset_clock_relation_id		**8**	**uimsbf**
reserved	'1111 111'	**7**	**bslbf**
asset_timescale_flag		**1**	**bslbf**
if (asset_time_scale_flag == 1) {			
asset_timescale		**32**	**uimsbf**
}			
}			
asset_location {			
MMT_general_location_info()			
}			
asset_descriptors {			
asset_descriptors_length	N5	**16**	**uimsbf**
for (j=0; j<N5; j++) {			
asset_descriptors_byte		**8**	**uimsbf**
}			
}			
}			
}			

9.3.10.3 Semantics

`table_id` – It indicates ID of the MPT. A complete MPT and each subset MPTs shall use the different table identifiers. Subset number of MPT is implicitly represented by this field. Since the `table_id` values are assigned contiguously, the MPT subset number can be deduced from this field, i.e., the MPT subset number equals this field minus the `table_id` of the base MPT. The MPT subset number provides the subset number of this MPT. The number '0' indicates base MPT and the numbers '1'~'14' indicate subset of MPT. The number '15' has a special meaning since it indicates a complete MPT

`version` – it is the version of the MPT. If MPT subset is used, this field indicates the version of the subset-N MPT. The newer version of a complete or a subset-0 MPT overrides the older one as soon as it has been received. If subset-0 MPT has a newer version, all rest of subset MPTs previously stored within a MMT receiving entity are treated as outdated. The newer version of subset-N MPT where N is '1'~'14' overrides the older one only if the version is the same as that of the current subset-0 MPT. Otherwise the received MPT is ignored

`length` – The length of the MPT counted in bytes starting from the next field to the last byte of the MPT table. The value '0' is never used for this field.

`MMT_package_id` – A globally unique identifier of the Package. The MMT package id shall use the UTF-8 character encoding.

`MMT_package_id_length` – The length in bytes of the `MMT_package_id` is string excluding the terminating null character

`MMT_package_id_byte` – A byte in the `MMT_package_id` is string. The terminating null character is not included in the string.

`MPT_descriptors` – It provides descriptors for MPT.

`MPT_descriptors_length` – The length of the descriptor syntax is loop. The length is counted from the next field to the end of the descriptor syntax loop. Several descriptors can be inserted in this syntax

loop. For example, additional_package_information_URL descriptor can be included here, which provides the URL of package information web page for this package.

MPT_descriptors_byte – one byte in the descriptors loop.

number_of_assets – It provides the number of Assets whose information is provided by this MPT.

asset_id – It provides Asset identifier. An asset_id is an ASCII string without the terminating null character that is equal to one of the id attributes of the AI elements in CI.

asset_id_length – It provides the length in bytes of the asset_id.

asset_id_byte – A byte in the asset_id.

asset_clock_relation_flag – It indicates whether an Asset uses NTP clock or other clock system as the clock reference. If this flag is '1', asset_clock_relation_id field is included. If this field is '0', the NTP clock is used for the Asset.

asset_clock_relation_id – It provides a clock relation identifier for the Asset. This field is used to reference the clock relation delivered by a CRI_descriptor() for the Asset. The value of this field is one of the clock_relation_id values provided by the CRI descriptors. (see sub-clause 9.3.14.1)

asset_timescale_flag – It indicates whether "asset_timescale" information is provided or not. If this flag is '1', asset_timescale field is included and if this flag is set to '0', asset_timescale is 90,000 (90kHz).

asset_timescale – It provides information of time unit for all timestamps used for the Asset expressed in the number of units in one second.

MMT_general_location_info_for_asset_location – It provides the location information of Asset. General location reference information for Asset defined in 9.3.14.2 is used. Only the value of location_type between '0x00' and '0x06' shall be used for an Asset location.

asset_descriptors_length – The number of bytes counted from the next field to the end of the Asset descriptors syntax loop.

asset_descriptors_byte – A byte in Asset descriptors.

9.3.11 CRI Table

9.3.11.1 Introduction

The CRI table defined in Table 26~~Table 26~~ is delivered by the CRI message. Also, it may be delivered by a PA message.

9.3.11.2 Syntax

The syntax of the CRI table is defined in Table 26~~Table 26~~ and the semantics of its syntax elements are provided below Table 26~~Table 26~~.

A CRI table may include multiple CRI descriptors. (see sub-clause 9.3.14.1).

Table 26 — CRI Table Syntax

Syntax	Value	No. of bits	Mnemonic
CRI_table () {			
table_id		**8**	**uimsbf**
version		**8**	**uimsbf**
length		**16**	**uimsbf**
for (i=0; i<N; i++) {			
CRI_descriptor()		**152**	**uimsbf**
}			
}			

9.3.11.3 Semantics

`table_id` – It indicates table identifier of the CRI table.
`version` – It indicates version of the CRI table. The newer version overrides the older one as soon as it has been received.
`length` – It indicates the length of the CRI table counted in bytes starting from the next field to the last byte of the CRI table. The value '0' is never used for this field. This value shall be multiple of 19bytes.
`CRI_descriptor()` – A clock relation information (CRI) descriptor. It is defined in 9.3.14.1.

9.3.12 DCI Table

9.3.12.1 Introduction

The DCI presents the required device capabilities for the consumption of the MMT Package. Depending on the mime type of the asset, a different set of information may be provided to support the delivery and consumption of the MMT Package. This specification differentiates between Video, Audio, and Download (applicable to non-timed assets) mime types

9.3.12.2 Syntax

The syntax and semantics of the DCI table is defined in Table 27Table 27.

Table 27 — DCI Table Syntax

Syntax	Value	No. of bits	Mnemonic
DCI_table() {			
table_id		**8**	**uimsbf**
version		**8**	**uimsbf**
length		**16**	**uimsbf**
number_of_assets	N1		
for (i=0; i<N1; i++) {			
asset_id {			
asset_id_length	N2	**8**	**uimsbf**
for (j=0; j<N2; j++) {			
asset_id_byte		**8**	**uimsbf**
}			
}			
mime_type_length		**8**	
mime_type		**mime_type_length*8**	**uimsbf**
reserved	'111 1111'	**7**	**bslbf**
codec_complexity_flag		**1**	**bslbf**
if (codec_complexity_flag == 1) {			
if (mime_type == VIDEO_MIME_TYPE) {			
video_codec_complexity {			
video_average_bitrate		**16**	**uimsbf**
video_maximum_bitrate		**16**	**uimsbf**
horizontal_resolution		**16**	**uimsbf**
vertical_resolution		**16**	**uimsbf**
temporal_resolution		**8**	**uimsbf**
video_minimum_buffer_size		**16**	**uimsbf**
}			
} else if (mime_type == AUDIO_MIME_TYPE) {			
audio_codec_complexity {			

audio_average_bitrate		**16**	**uimsbf**
audio_maximum_bitrte		**16**	**uimsbf**
audio_minimum_buffer_size		**16**	**uimsbf**
}			
}			
}			
if (mime_type == DOWNLOAD_MIME_TYPE) {			
download_capability {			
required_storage		**32**	**uimsbf**
}			
}			
}			
}			

9.3.12.3 Semantics

`table_id` – It indicates ID of DCI table.
`version` – It indicates a version of DCI table. The newer version overrides the older one as soon as it has been received.
`length` – It provides the length of DCI table counted in bytes starting from the next field to the last byte of the DCI table. The value '0' is never used for this field.
`number_of_assets` – It indicates the number of MMT Assets.
`asset_id` – It provides ID of Asset.
`mime_type_length` – indicates the length of the MIME type field.
`mime_type` – It provides the MIME type of the Asset encoded in UTF-8 character set. If the value of codec complexity in `mime_type` is different from the value of codec complexity provided by DCI table, then the value from DCI table shall take priority.
`codec_complexity_flag` – If this flag is '1', it provides the codec complexity.
`video_codec_complexity` – It provides the complexity the video decoder has to deal with.
`video_average_bitrate` – It provides the average bit-rate of the video in kilo-bit/s for the whole Asset.
`video_maximum_bitrate` – It provides the maximum bit-rate of the video in kilo-bit/s for the whole Asset.
`horizontal_resolution` – It provides the horizontal resolution of the video in pixels.
`vertical_resolution` – It provides the vertical resolution of the video in pixels.
`temporal_resolution` – It provides theaverage temporal resolution of the video in frames per second.
`video_mimimum_buffer_size` – It provides the minimum size of video decoder buffer that needs to be available in kilo-bytes.
`audio_codec_complexity` – The complexity the audio decoder has to deal with.
`audio_average_bitrate` – It provides the average bit-rate in kilo-bit/s for the whole Asset.
`audio_maximum_bitrate` – It provides the maximum bit-rate in kilo-bit/s for the whole Asset.
`audio_minimum_buffer_size` – It provides the minimum size of audio decoder buffer needs to be processed in kilo-bytes.
`download_capability` – It provides the required capability for download.
`required_storage` – It provides the size of storage in kilo-bytes required to download.

9.3.13 SIT Table

9.3.13.1 Introduction

The Security Information Table (SIT) provides the required security for the consumption of the Package.

9.3.13.2 Syntax

The syntax and semantics of the SIT table is defined in Table 28Table 28.

Table 28 — SCI Table Syntax

Syntax	Value	No. of bits	Mnemonic
SIT_table() {			
table_id		**8**	**uimsbf**
version		**8**	**uimsbf**
length		**16**	**uimsbf**
number_of_Security_System	N1		
for (i=0; i<N1; i++) {			
system_id {			
system_id_length	N2	**8**	**uimsbf**
for (j=0; j<N2; j++) {			
system_id_byte		**8**	**uimsbf**
}			
}			
systemProvider **{**			
systemProvider_URL_length	**N3**	**8**	**uimsbf**
for (i=0; i<N3; i++) {			
URL_byte		**8**	**uimsbf**
}			
}			
reserved	'1111 111'	**7**	**bslbf**
encryption_flag		**1**	**bslbf**
if (encryption_flag == 1) **{**			
encAlgorithm **{**			
encAlgorithm_length	N4	**8**	**uimsbf**
for (i=0; i<N4; i++) {			
encAlgorithm _byte		**8**	**uimsbf**
}			
}			
keySize **{**			
keySize_length	N5	**8**	**uimsbf**
for (i=0; i<N5; i++) {			
keySize_byte		**8**	**uimsbf**
}			
}			
keyUrl **{**			
key_URL_length	N6	**8**	**uimsbf**
for (i=0; i<N6; i++) {			
URL_byte		**8**	**uimsbf**
}			
}			
initializationVector **{**			
initializationVector_length	N7	**8**	**uimsbf**
for (i=0; i<N7; i++) {			
length_byte		**8**	**uimsbf**
}			
}			
ivUrl **{**			
iv_URL_length	N8	**8**	**uimsbf**

for (i=0; i<N8; i++) {			
URL_byte		**8**	**uimsbf**
}			
}			
keyUrlTemplate **{**			
keyUrlTemplate_length	N9	**8**	**uimsbf**
for (i=0; i<N9; i++) {			
length_byte		**8**	**uimsbf**
}			
}			
ivUrlTemplate **{**			
ivUrlTemplate_length	N10	**8**	**uimsbf**
for (i=0; i<N10; i++) {			
length_byte		**8**	**uimsbf**
}			
}			
reserved	'1111 111'	**7**	**bslbf**
authentication_flag		**1**	**bslbf**
if (authentication_flag == 1) **{**			
digestUrl **{**			
digest_URL_length	N11	**8**	**uimsbf**
for (i=0; i<N11; i++) {			
URL_byte		**8**	**uimsbf**
}			
}			
digestURLTemplate **{**			
digestUrlTemplate_length	N12	**8**	**uimsbf**
for (i=0; i<N12; i++) {			
length_byte		**8**	**uimsbf**
}			
}			
signatureUrl **{**			
signature_URL_length	N13	**8**	**uimsbf**
for (i=0; i<N13; i++) {			
URL_byte		**8**	**uimsbf**
}			
}			
signatureURLTemplate **{**			
signatureUrlTemplate_length	N14	**8**	**uimsbf**
for (i=0; i<N14; i++) {			
length_byte		**8**	**uimsbf**
}			
}			
signatureLength **{**			
siginature_length	N15	**8**	**uimsbf**
for (i=0; i<N15; i++) {			
Length_byte		**8**	**uimsbf**
}			
}			
signatureKeyUrl **{**			
signatureKey_URL_length	N16	**8**	**uimsbf**
for (i=0; i<N16; i++) {			
URL_byte		**8**	**uimsbf**

}			
}			
number_of_License	N17		
for (i=0; i<N17; i++) {			
license_Url {			
licenseUrl_length	N18	**8**	**uimsbf**
for (j=0; j<N18; j++) {			
URL_byte		**8**	**uimsbf**
}			
}			
licenseURLTemplate **{**			
lisenceUrlTemplate_length	N19	**8**	**uimsbf**
for (i=0; i<N19; i++) {			
length_byte		**8**	**uimsbf**
}			
}			
}			
}			
}			

NOTE How a specific DRM solution or CAS solution works with MMT is out of scope.

9.3.13.3 Semantics

`number_of_Security_System` – It provides a DRM or CAS system that can process and handle the content protection, access control and rights management.

`systemId` – It provides a UUID-formatted opaque string for the system. This can also be used to encode the type of the system.

`systemProvider` – It provides a URL location of a provider for the system. This can be used for downloading and installing the system, when needed.

`Encryption` – This optional element provides the information about encryption.

`encAlgorithm – It` provides the algorithm used for encryption.

`keySize` – it provides the size of initialization vectors in bytes.

`keyUrl` – it provides the URL for key derivation.

`IV` – it provides the initialization vector.

`ivUrl` – it provides URL for initialization vector derivation.

`keyUrlTemplate` – it provides the template for key URL generation.

`ivUrlTemplate` – it provides the template for IV URL generation.

`Authentication` – this optional element give the information required for authentication.

`digestUrl` – it provides the URL used for retrieving the digest value.

`digestUrlTemplate` – It provides the template for creating the URL used for retrieving the digest value.

`signatureUrl` – It provides the URL used for retrieving the signature value.

`signatureUrlTemplate` – it provides the template for creating the URL used for retrieving the signature value.

`signatureLength` – it provides the length of the signature. It shall appear only if the length is shorter than the normal output size of the signature algorithm.

`signaturekeyUrl` – it provides the URL for the key used for the signature.

`Lisence` – this optional element provides the information to get Lisence

`licenseUrl` – The license format can be in some standard ones or be dependent on the system that is specified with `systemId` of the System element.

`licenseUrlTemplate` – Specifies the template for license URL generation.

9.3.14 Descriptors and

There are descriptor and information related to MMT table.

9.3.14.1 CRI Descriptor

9.3.14.1.1 Introduction

CRI descriptor is used to specify the relationship between the NTP timestamp and the MPEG-2 STC. The value of clock reference related with the MMT Asset derived from the MPEG-2 TS is specified by a `clock_relation_id`.

`CRI_descriptors` are carried in a CRI table.

9.3.14.1.2 Syntax

The syntax of `CRI_descriptor()` is defined in Table 29 and the semantics of its syntax elements are provided below Table 29.

Table 29 — `CRI_descriptor()` Syntax

Syntax	Value	No. of bits	Mnemonic
CRI_descriptor() {			
descriptor_tag		**16**	**uimsbf**
descriptor_length		**16**	**uimsbf**
clock_relation_id		**8**	**uimsbf**
reserved	'11 1111'	**6**	**uimsbf**
STC_sample		**42**	**uimsbf**
NTP_timestamp_sample		**64**	**uimsbf**
}			

9.3.14.1.3 semantics

`descriptor_tag` – A tag value indicating the type of a descriptor.
`descriptor_length` – The length in bytes counted from the next byte after this field to the last byte of the descriptor.
`clock_relation_id` – The identifier of a clock relation.
`STC_sample` – The MPEG-2 STC value that corresponds to the following `NTP_timestamp_sample`. The sample value of STC is in 42-bit format.
`NTP_timestamp_sample` – The sample value of NTP timestamp that corresponds to the preceding `STC_sample`.

9.3.14.2 MMT_general_location_info

9.3.14.2.1 Syntax

An `MMT_general_location_info` syntax element group is used to provide location information for the payload of the described item. The syntax of the `MMT_general_location_info` is defined in Table 30~~Table 30~~ and the semantics of its syntax elements are provided below Table 30~~Table 30~~ .

Table 30 — `MMT_general_location_info` Syntax

Syntax	Value	No. of bits	Mnemonic

MMT_general_location_info() {			
location_type		**8**	**uimsbf**
if (location_type == 0x00) {			
packet_ id		**16**	**uimsbf**
} else if (location_type == 0x01) {			
ipv4_src_addr		**32**	**uimsbf**
ipv4_dst_addr		**32**	**uimsbf**
dst_port		**16**	**uimsbf**
packet_ id		**16**	**uimsbf**
} else if (location_type == 0x02) {			
ipv6_src_addr		**128**	**uimsbf**
ipv6_dst_addr		**128**	**uimsbf**
dst_port		**16**	**uimsbf**
packet_ id		**16**	**uimsbf**
} else if (location_type == 0x03) {			
network_id		**16**	**uimsbf**
MPEG_2_transport_stream_id		**16**	**uimsbf**
reserved	'111'	**3**	**bslbf**
MPEG_2_PID		**13**	**uimsbf**
} else if (location_type == 0x04) {			
ipv6_src_addr		**128**	**uimbsf**
ipv6_dst_addr		**128**	**uimbsf**
dst_port		**16**	**uimbsf**
reserved	'111'	**3**	**bslbf**
MPEG_2_PID		**13**	**uimbsf**
} else if (location_type == '0x05') {			
URL_length	N1	**8**	**uimsbf**
for (i=0; i<N1; i++) {			
URL_byte		**8**	**uimsbf**
}			
} else if (location_type == '0x06') {			
URL_length	N2	**8**	**uimsbf**
for (i=0; i<N2; i++) {			
URL_byte		**8**	**uimsbf**
}			
byte_offset		**16**	**uimsbf**
length		**16**	**uimsbf**
} else if (location_type == '0x07') {			
} else if (location_type == '0x08') {			
message_id		**8**	**uimsbf**
} else if (location_type == '0x09') {			
packet_ id		**16**	**uimsbf**
message_id		**8**	**uimsbf**
} else if (location_type == '0x0A') {			
ipv4_src_addr		**32**	**uimsbf**
ipv4_dst_addr		**32**	**uimsbf**
dst_port		**16**	**uimsbf**
packet_ id		**16**	**uimsbf**
message_id		**8**	**uimsbf**
} else if (location_type == '0x0B') {			
ipv6_src_addr		**128**	**uimsbf**
ipv6_dst_addr		**128**	**uimsbf**
dst_port		**16**	**uimsbf**

packet_ id		**16**	**uimsbf**
message_id		**8**	**uimsbf**
} else if(location_type == '0x0C') {			
Ipv4_src_addr		**32**	**uimbsf**
Ipv4_dst_addr		**32**	**uimbsf**
dst_port		**16**	**uimbsf**
reserved	**'111'**	**3**	**bslbf**
MPEG_2_PID		**13**	**uimbsf**
}			
}			

9.3.14.2.2 Semantics

`location_type` – This field indicates the type of the location information as defined in Table 31~~Table 30~~.

Table 31~~30~~ — Value of `location_type`

Value	Description
0x00	A data path in the same UDP/IP flow as the one that carries the data structure to which this `MMT_general_location_info()` belongs
0x01	A data path in a UDP/IP (version 4) flow
0x02	A data path in a UDP/IP (version 6) flow
0x03	A program within an MPEG-2 TS in a broadcast network. The program is indicated by a PMT PID is described in ISO/IEC 13818-1.
0x04	An elementary stream (ES) in an MPEG-2 TS over the IP broadcast network
0x05	URL
0x06	A continguous byte range in the file addressed by a URL
0x07	The same signaling message as the one that carries the data structure to which this `MMT_general_location_info()` belongs
0x08	A signaling message delivered in the same data path as the one that carries the data structure to which this `MMT_general_location_info()` belongs
0x09	A signaling message delivered in a data path in the same UDP/IP flow as the one that carries the data structure to which this `MMT_general_location_info()` belongs
0x0A	A signaling message delivered in a data path in a UDP/IP (version 4) flow
0x0B	A signaling message delivered in a data path in a UDP/IP (version 6) flow
0x0C	An elementary stream (ES) in an MPEG-2 TS over the IP v4 broadcast network
0x0D~0xFF	reserved

`packet_id` – `packet_id` in MMT packet header. (see sub-clause 7.3)
`ipv4_src_addr` – IP version 4 source address of an IP application data flow.
`ipv4_dst_addr` – IP version 4 destination address of an IP application data flow.
`dst_port` – Destination port number of an IP application data flow.

`ipv6_src_addr` – IP version 6 source address of an IP application data flow.
`ipv6_dst_addr` – IP version 6 destination address of an IP application data flow.
`network_id` – broadcast network identifier that carries the MPEG-2 TS. This field is specific to the broadcast system in use
`MPEG_2_transport_stream_id` – MPEG-2 TS identifier.
`MPEG_2_PID` – PID of MPEG-2 TS packet carrying the ES.
`URL_length` – Length in bytes of a URL. The terminating null ('0x00') shall not be counted.
`URL_byte` – A byte data in a URL. The terminating null ('0x00') shall not be included.
`byte_offset` – A byte offset from the first byte of a file.
`length` – The length in bytes.
`message_id` – MMT signaling message identifier. (see below Table 32Table 32)

9.3.15 ID and Tags values

The values of the message identifier (`message_id`) are assigned in Table 32Table 32.

Table 32 — Message Identifier (`message_id`) Values

Value	Description
0x0000 ~ 0x00FF	reserved
0x0100	PA messages
0x0500 ~ 0x44FF	MCI messages. For a package, 16 contiguous values are allocated to MCI messages. If the value % 16 equals 15, the MCI message carries complete MMT-CI. If the value %16 equals N where N = 0 ~ 14, the MCI message carries Subset-N MMT-CI.
0x2500 ~ 0x84FF	MPT messages. For a package, 16 contiguous values are allocated to MPT messages. If the value % 16 equals 15, the MPT message carries complete MPT. If the value %16 equals N where N = 0 ~ 14, the MPT message carries Subset-N MPT.
0x4500	CRI messages
0x4900	DCI messages
0x8D00 ~ 0xFFFF	reserved

The values of the table identifier (`table_id`) are assigned in Table 33Table 33.

Table 33 — Table Identifier (`table_id`) Values

Value	Description
0x00 ~ 0x0F	reserved
0x10	PA table

0x11 ~ 0x1F	Subset-0 MCI table ~ Subset-15 MCI table
0x20	Complete MCI table
0x21 ~0x30	reserved
0x31 ~ 0x3F	Subset-0 MPT ~ Subset-15 MPT
0x40	Complete MPT
0x41 ~ 0x50	Reserved
0x51	CRI table
0x52	DCI table
0x53 ~0xFF	reserved

The values of descriptor tag are assigned in Table 34~~Table 34~~.

Table 34 — Descriptor Tag Values

Value	Description
0x0000 ~ 0x000F	reserved
0x0010	CRI descriptor
0x0011	MMT_gerneral_location_info
0x0012 ~0xFFFF	reserved.

9.4 Messages for Delivery

9.4.1 Introduction

Signaling messages for delivery are MC Message, AL-FEC Message, HRBM Message, ARQ Signaling Message, Reception Quality Feedback Message and NAM Feedback Message. They are shown at Figure 25.

Figure 25 — Structure of the Signaling Messages for Delivery

9.4.2 MC Message

9.4.2.1 Introduction

An Measurement Configuration (MC) message provides the information required for an measurement. It provides the information of measurement item (e.g. a receiving entity buffer status, round trip delay, NAM parameter), measurement condition such as a measurement starting time and a period and measurement report. Syntax of MC message is shown in Table 35Table 35 and semantics are follows.

9.4.2.2 Syntax

Table 35 — MC Message Syntax

Syntax	Values	No. of bits	Mnemonic
Measurement_configuration ( ){			
message_id		16	
version		8	
length		16	
message_payload{			
measurement_item		8	
measurement_execution time{		2	
immediately			
measurement_start_time			
measurement_start_condition			
}			
measurement_period{		2	
once			
periodic_measure_time			
measure_condition			
}			
measurement_report{			
server_address		32	
reportType		1	

if (reportType == 0x00) { **reception_quality_feedback()** **} else if (reportType==0x01){** **NAM()** **}** **}** } }			

9.4.2.3 Semantics

`message_id` – It indicates MC message ID. The length of this field is 16 bits.

`version` – It indicates the version of MC messages. MMT receiving entity may check whether the received message is new or not. The length of this field is 8 bits.

`length` – It indicates the length of MC messages. The length of this field is 16 bits. It indicates the length of the MC message counted in bytes starting from the next field to the last byte of the MC message. The value '0' shall not be used.

`measurement_item` – It indicates the item for measurement such as Round Trip Time (RTT) delay, receiving entity buffer status, inter-arrival jitter, propagation delay, packet loss ratio, and available bit-rate and set of NAM parameters.

`measurement_execution_time` – It indicates when MMT receiving entity start measuring the item indicated by measurement_item. It has three modes. They are "`immediatelay`", "`measurement_start_time`", and "`measurement_start_condition`".

Table 3635 — value of `measurement_execution_time`

Value	Description
immediately	it indicates MMT receiving entity to start the measurement immediately.
measurement_start_time	it indicates MMT receiving entity to start the measurement at the time indicated by this field
measurement_start_condition	it indicates MMT receiving entity to start the measurement with a certain condition.

`measurement_period` – It indicates how frequently MMT receiving entity measures the item indicated by `measurement_item`. It has three modes. They are "`once`", "`periodic_measure_time`", and "`measure_condition`".

Table 3736 — value of `measurement_period`

Value	Description
once	it indicates MMT receiving entity to execute the measurement only once.
periodic_measure_time	it indicates MMT receiving entity to execute measurements periodically.

measure_condition	it indicates MMT receiving entity to execute the measurement with a certain condition.

`measurement_report` – it provides information on the measurement report. It has server address where MMT receiving entity sends a measurement result and a template to be used for measurement report.
`server_address` – it provides the location of server that receives an MMT measurement results.
`reportType` – This flag indicates type of measurement report request. If this flag is set to '0x00', it indicates MC message requests `reception_quality_feedback`. If this flag is set to '0x01', it indicates MC message requests set of NAM parameter.
`reception_quality_feedback` – as defined in 9.4.6.
`NAM` – as defined in 8.2.4.

9.4.3 AL-FEC Messages

9.4.3.1 Introduction

AL-FEC messages provides AL FEC configuration information for an Assets to be protected by FEC. The syntax of AL-FEC message is shown in Table 38 and the semantics are follows.

9.4.3.2 Syntax

Table 38 — AL-FEC Message

Syntax	Values	No. of bits	Mnemonic
AL_FEC ( ){			
message_id		16	
version		8	
length		16	
message_payload{			
fec_flag		1	**bslbf**
reserved	'1111111'	7	**bslbf**
if (fec_flag==1) {			
length_of_fec_flow_descriptor		16	**uimsbf**
fec_flow_descriptor() {			
number_of_fec_flows	N1	8	**uimbsf**
for (i=0; i<N1 ; i++) {			
fec_flow_id		8	**uimbsf**
source_flow_id		8	**uimbsf**
number_of_assets	N2	8	**uimbsf**
for (j=0; j<N2 ;j++) {			

packet_id		16	**uimbsf**
}			
fec_coding_structure		4	**uimbsf**
ssbg_mode		2	**uimbsf**
reserved	'11'	2	**bslbf**
length_of_repair_symbol		16	**uimbsf**
if (ssbg_mode == 10) {			
num_of_repair_symbol_per_packet		16	**uimbsf**
num_of_symbol_element_per_source_symbol		16	**uimbsf**
}			
if (fec_coding_structure == 0001) {			
repair_flow_id		8	**uimbsf**
fec_code_id_for_repair_flow		8	**uimbsf**
maximum_k_for_repair_flow		24	**uimbsf**
maximum_p_for_repair_flow		24	**uimbsf**
FEC_buffer_time		16	**uimbsf**
protection_window_time		32	**uimbsf**
protection_window_size		32	**uimbsf**
}			
if (fec_coding_structure == 0010) {			
num_of_sub_block_per_source_block		8	**uimbsf**
number_of_parity_flows	N3	8	**uimbsf**
for (k=0; k<N3 ;k++){			
repair_flow_id		8	**uimbsf**
fec_code_id_for_repair_flow		8	**uimbsf**
maximum_k_for_repair_flow		24	**uimbsf**
maximum_p_for_repair_flow		24	**uimbsf**
FEC_buffer_time		16	**uimbsf**
protection_window_time		32	**uimbsf**
protection_window_size		32	**uimbsf**
}			
}			
if (fec_coding_structure == 0011) {			

num_of_layer_for_LAFEC	N4	8	**uimbsf**
fec_code_id_for_repair_flow		8	**uimbsf**
for (l=0;l<N4 ;l++){			
repair_flow_id		8	**uimbsf**
maximum_k_for_repair_flow		24	**uimbsf**
maximum_p_for_repair_flow		24	**uimbsf**
FEC_buffer_time		16	**uimbsf**
protection_window_time		32	**uimbsf**
protection_window_size		32	**uimbsf**
}			
}			
}			
}			
}			
}			

9.4.3.3 Semantics

`message_id` – It indicates AL-FEC messages.
`version` – It indicates the version of AL-FEC messages. MMT receiving entity can check whether the received message is new or not.
`length` – It indicates the length of AL-FEC messages. The length of this field is 16 bits. It indicates the length of the AL-FEC message counted in bytes starting from the next field to the last byte of the AL-FEC message. The value '0' is never used for this field.
`fec_flag` – It indicates whether there is at least one source flow delivered with AL-FEC protection. If every source flows are delivered without AL-FEC protection, all the remaining message is not be delivered.

Table 3938 — value of `fec_flag`

Value	Description
b0	there is no FEC source flow
b1	there is at least one FEC source flow

`length_of_fec_flow_descriptor` – It indicates the length of `fec_flow_descriptor` (0 in unit of bytes.
`number_of_fec_flows` – It indicates the number of FEC encoded flows.
`fec_flow_id` – It indicates an arbitrary integer number for identifying an FEC encoded flow. Each FEC encoded flow has its associated one FEC source flow and one or more FEC repair flows.

`source_flow_id` – It indicates an arbitrary integer number for identifying an FEC source flow. Each FEC source flow has its associated one FEC encoded Flow.
`number_of_assets` - It indicates the number of Assets, which belongs to the same FEC source flow.
`packet_id` – It indicates `packet_id` in MMT packet header.
`ssbg_mode` – It indicates the applied `SSBG mode`.

Table 4039 — value of `ssbg_mode`

Value	Description
b00	ssbg_mode0 (with constant MMT packet size)
b01	ssbg_mode1 (with variable MMT packet size)
b10	ssbg_mode2 (with variable MMT packet size)
b11	reserved

`length_of_repair_symbol` – It provides the length of a repair symbol in byte unit.
`num_of_repair_symbol_per_packet` – It provides the number of repair symbols carried in an FEC repair packet when `ssbg_mode2` is used.
`num_of_symbol_element_per_source_symbol` – It provides the number of symbol elements consisting a source symbol when `ssbg_mode2` is used.
`fec_coding_structure` – It indicates the applied AL-FEC coding structure for its associated FEC source flow.

Table 4140 — value of `fec_coding_structure`

Value	Description
b0000	AL-FEC is not applied
b0001	One stage FEC coding structure
b0010	Two stage FEC coding structure
b0011	Layer-aware FEC coding structure
b0100 ~ b1111	resereved

`num_of_sub_block_per_source_block` – It provides the number of the split source packet blocks consisting a source packet block for two stage FEC coding structure.
`number_of_parity_flows` – It provides the number of FEC repair flows. In this specification, only 2 FEC repair flows are used for two stage FEC coding structure.
`num_of_layer_for_LAFEC` – It provides the number of layers of the media protected by layer-aware FEC coding structure.
`repair_flow_id` – It provides an arbitrary integer number for identifying FEC repair flow. It indicates `packet_id` in MMT packet header for FEC repair packet.
`fec_code_id_for_repair_flow` – It provides the applied FEC code for its associated FEC repair flow.
`maximum_k_for_repair_flow` – It provides the maximum allowed number of source symbols in a source symbol block for its associated FEC repair flow.
`maximum_p_for_repair_flow` – It provides the maximum allowed number of repair symbols in a repair symbol block for its associated FEC repair flow.
`FEC_buffer_time` – gives the minimum buffer time required for FEC decoding.

FEC protection window indicates in time and/or number of packets of the same parity flow the maximum window during which payloads of the same FEC packet block (source packets and parity packets) shall be sent.

NOTE Therefore, the difference between the two timp-stamps, which are set to the first FEC packet and the last FEC packet of the FEC packet block, shall be equal or less than the time which is set to this field.

`protection_window_time` – This value presents the maximum time in milliseconds between the sending of the first packet of an FEC packet block and the sending of the last packet of that FEC packet block. If the value is set to '0', '`protection_window_size`' shall be used only.

`protection_window_size` – This value presents the maximum number of source and parity payloads from the same FEC protected flow that can be sent from the first payload of an FEC block (source packets and parity packets) to the last payload of that FEC block. If the value is set to '0', '`protection_window_time`' shall be used only.

For fast zapping solution with LA-FEC UI (see D.4.4), each protectionWindow is enlarged to include the last encoded symbol of all protected flows of the same source block.

9.4.4 HRBM Message

9.4.4.1 Introduction

The Hypothetical Receiver Buffer Model (HRBM) id defined in Clause 10. An HRBM message provides the information of the fixed end-to-end transmission delay and memory requirement to MMT a receiving entity for the efficient operation in a broadcasting environment.

9.4.4.2 Syntax

Table 4241 — HRBM Message Syntax

Syntax	Values	No. of bits	Mnemonic
HRBM ( ){			
message_id		16	
version		8	
length		16	
extension {			
extension_fields_Byte			
}			
message_payload{			
required_buffer_size		**32**	
fixed_end_to_end_delay		**32**	
}			

}			

9.4.4.3 Semantics

`message_id` – It indicates HRBM messages.
`version` – It indicates the version of HRBM messages. MMT receiving can check whether the received message is new or not.
`length` – It indicates the length of HRBM messages. It indicates the length of the HRBM message counted in bytes starting from the next field to the last byte of the HRBM message. The value '0' is never used for this field.
`required_buffer_size` – It provides the information of the required buffer size of MMT Assets for MMT receing entity. The unit of buffer size is Byte.
`fixed_end_to_end_delay` – It provides the information of the `fixed_end_to_end_delay` between sending entity and receving entity. The unit of delay is ms.

9.4.5 ARQ Signaling

9.4.5.1 Introduction

Figure 26 illustrates ARQ (Automatic Repeat reQuest) signaling process. ARQ is an error control method in data communication which relies on the MMT receiving entity of the data acknowledging reception of messages correctly. There is a timeout parameter that is specified at the start of the session. Only when the timeout has elapsed will the MMT receiving entity send such an acknowledgement. There are three distinct steps to the ARQ operations.

First, ARQ configuration information, which includes the policy to be adopted by the MMT sending entity and MMT receiving entity in the event of packet loss, shall be transmitted from the transmitting MMT sending entity to the MMT receiving entity either in-band or out-of-band. At the MMT receiving entity the ARQ configuration information will be stored. With in-band transmission of ARQ configuration information this information will be transmitted in the same channel as the media. With out-of-band transmission of ARQ configuration information this information will be transmitted in a separate channel to the media. The MMT receiving entity configures the ARQ system with the ARQ configuration information in readiness for sending ARQ feedback message to the MMT sending entity.

In the event of a lost packet, this loss shall be detected at the MMT receiving entity, and a feedback message shall be generated according to the ARQ configuration information that has been stored, and then transmitted to the MMT sending entity.

The MMT sending entity receives this feedback message and then generates the retransmission message according to the ARQ configuration information and transmits to the MMT receiving entity. The MMT receiving entity receives the retransmission message that has been generated according to the ARQ configuration information. The MMT receiving entity is able to substitute the lost media packet from the retransmission message.

Details of each of these steps with the associated syntax and semantics will be described in the following three clauses.

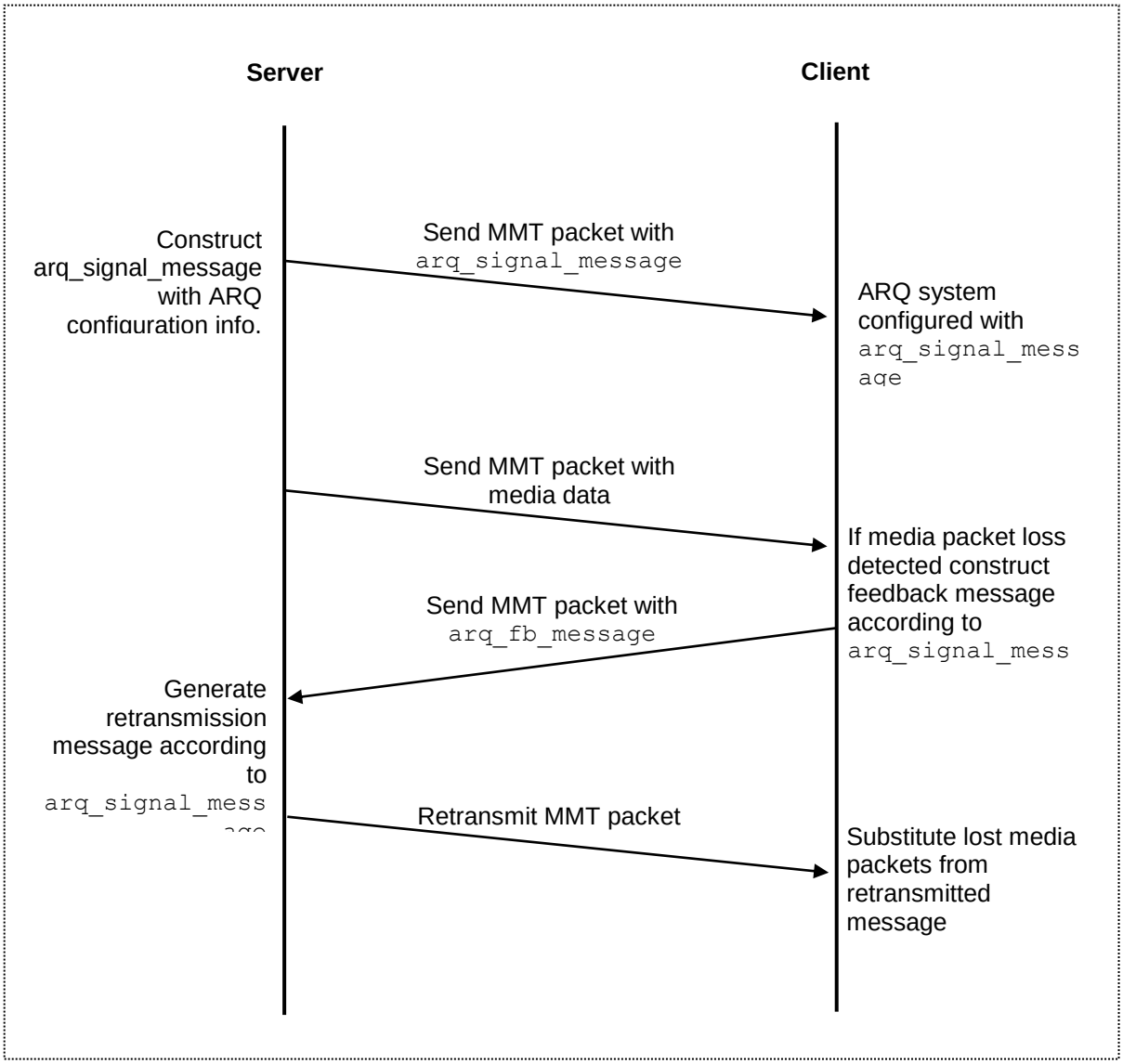


Figure 26 — overview of ARQ signaling

9.4.5.2 ARQ Configuration message

9.4.5.2.1 Introduction

ARQ configuration information, which includes the policy to be adopted by the MMT sending entity and MMT receiving entity in the event of packet loss, shall be transmitted at the beginning of a session as the ARQ Configuration (AC) message from the transmitting MMT sending entity to the MMT receiving entity either in-band or out-of-band. The AC message provides information to MMT receiving entity about:

— The profile for the retransmission mechanism

— The profile for the feedback mechanism

— The timeout window based on which MMT receiving entity could determine if a retransmission request is needed. This is the time that a MMT sending entity will keep a packet of data in buffer, and thus available for retransmission.

The syntax and the semantics in the following clauses provide one example method for retransmission and one example method for feedback in MMT. However, alternative methods for both retransmission and for feedback can be done even though they are not specified by MMT. The syntax for AC message is shown in Table 43.

9.4.5.2.2 Syntax

Syntax for ARQ signaling message is shown in Table 43.

Table 43 — Syntax for ARQ signaling

Syntax	Values	No. of bits	Mnemonic
AC_message() {			
message_id		16	
version		8	
length		16	
message_payload{			
arq_profile_identifier		32	uimsbf
fb_profile_identifier		32	uimsbf
rtx_window		32	uimsbf
}			
}			

9.4.5.2.3 Semantics

`message_id` – It indicates AC message ID. The length of this field is 16 bits.

`version` – It indicates the version of AC messages. MMT receiving entity may check whether the received message is new or not. The length of this field is 8 bits.

`length` – It indicates the length of AC messages. The length of this field is 16 bits. It indicates the length of the AC message counted in bytes starting from the next field to the last byte of the AC message. The value '0' shall not be used.

`arq_profile_identifier` - It indicates the standard for retransmit. For example when this field is set to "ZARQ", then this indicates the retransmit packet is one MMT packet, the transmit method is in-band or out-of-band. When the `packet_flow_identifier` is set to, say, 0xF0 then this indicates the retransmit flow.

`fb_profile_identifier` - It indicates the standard for feedback. For example when this field is set to "ZFBP", then this indicates the feedback pack et contains the `arq_fb_message` as defined in 9.4.5.1 is carried in-band or out-of-band. When the `packet_flow_identifier` is set to, say, 0xE0 then this indicates the feedback flow.

`rtx_window` - It indicates the retransmit window size. The unit is milliseconds

9.4.5.3 AF message

9.4.5.3.1 Introduction

In the event of packet loss, this loss is detected by MMT receiving entity. A feedback message is generated according to the information on AC message, and then transmitted to MMT sending entity. When the MMT receiving entity detects that one or more packets have been lost, it forms a mask of up to 255 bytes where each bit in a byte corresponds to a sequence number of lost MMT packets. This allows AF message can report up to 255x8 lost packets in one AF message. The syntax for AF message is shown in Table 44.

The method of packet loss detection is out of scope of this standard.

9.4.5.3.2 Syntax

Syntax for ARQ feedback message is shown in Table 44.

Table 44 — Syntax for ARQ feedback message

Syntax	Values	No. of bits	Mnemonic
AF _message() {			
message_id		16	
version		8	
length		16	
message_payload {			
packet_flow_identifier		**8**	
packet_sequence_number		**32**	
masklength		**8**	
for(i=0; i<masklength; i++)			
{			
mask_byte		**8**	
}			
}			
}			

9.4.5.3.3 Semantics

`message_id` – It indicates AF message ID. The length of this field is 16 bits.
`version` – It indicates the version of AF messages. MMT receiving entity may check whether the received message is new or not. The length of this field is 8 bits.

`length` – It indicates the length of AF messages. The length of this field is 16 bits. It indicates the length of the AF message counted in bytes starting from the next field to the last byte of the AF message. The value '0' shall not be used.

`packet_flow_identifier` – This field indicates the flow identifier.

`packet_sequence_number` – This field corresponds to the `packet_sequence_number` of the 1st packet indicated by the `mask_byte` that is identified as having been detected to be lost, and hence requiring re-transmission.

`mask_length` - It indicates the length of the data behind the mask in bytes.

`mask_byte` - mask field, each bit correspond to a MMT packet. If the packet behind the packet with `packet_id` is lost, then the corresponding bit will be set to '1'.

9.4.6 Reception Quality Feedback Message

9.4.6.1 Introduction

The MMT receiving entity sends the reception quality feedback to the MMT sending entity to inform it of the reception quality of the received MMT stream. The MMT receiving entity will need to keep track of the reception quality per MMT sending entity for the MMT stream.

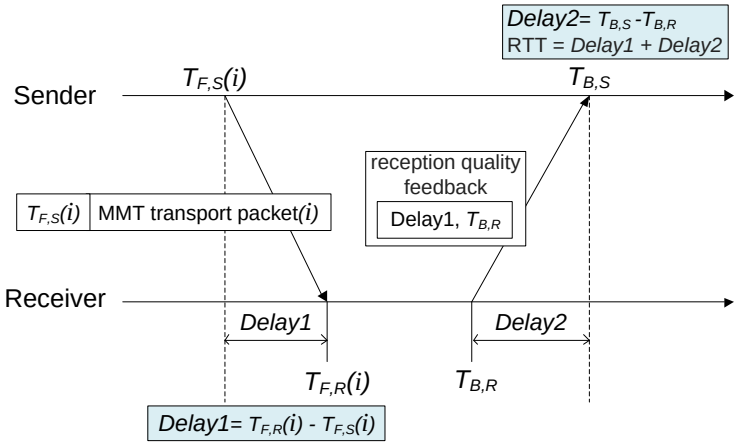


Figure 27 — Round Trip Time (RTT) calculation

The MMT receiving entity provides information in the feedback to allow the MMT sending entity to calculate the Round-Trip Time (RTT) which is shown in Figure 27. In Figure 27, Delay1 is the delivery time of the MMT packet from MMT sending entity to MMT receiving entity. Delay1 is calculated by the subtracting $T_{F,S}$(i) (NTP time at delivery instant of i-th MMT packet) from $T_{F,R}$(i) (NTP time at arrival instant of i-th MMT packet). The Delay2 is the delivery time for the MMT packet from MMT receiving entity to MMT sending entity. The Delay2 is calculated by subtracting $T_{B,R}$ (NTP time at delivery instant of the feedback report, i.e. `feedback_timestamp`) from the $T_{B,S}$ (NTP time at arrival instant of the feedback report). Thus, the MMT sending entity can calculate the RTT by adding Delay1 and Delay2. The syntax for the RQF message is shown in Table 45.

9.4.6.2 Syntax

The syntax for the reception quality feedback is shown in Table 45.

Table 45 — Syntax for `reception_quality_feedback`

Syntax	Values	No. of bits	Mnemonic
RQF_message ( ) {		16	
message_id		8	
version		16	
length			
message_payload {			
measurement_duration		16	unsigned short
packet_loss_ratio		8	unsigned char
inter_arrival_jitter		32	unsigned integer
RTT_parameter( ) {			
propagation_delay		32	unsigned integer
feedback_timestamp		32	unsigned integer
}			
}			
}			

9.4.6.3 Semantics

`message_id` – It indicates RQF message ID. The length of this field is 16 bits.

`version` – It indicates the version of RQF messages. MMT receiving entity may check whether the received message is new or not. The length of this field is 8 bits.

`length` – It indicates the length of RQF messages. The length of this field is 16 bits. It indicates the length of the RQF message counted in bytes starting from the next field to the last byte of the RQF message. The value '0' shall not be used.

`measurement_duration` - Duration in seconds measured for the parameters in the feedback.

`packet_loss_ratio` - Ratio of the number of lost MMT packets to the total number of transmitted packets. This value is equivalent to taking the integer part after multiplying the loss fraction by 256. The `packet_loss_ratio` is a result measured within the `measurement_duration`.

`inter_arrival_jitter` - Deviation of the difference in packet spacing at the MMT receiving entity compared to the MMT sending entity for a pair of packets, measured in timestamp units. It can be estimated based on the time difference between the arrivals of adjacent MMT packets. The `inter_arrival_jitter` is an average result measured within the `measurement_duration`.

`RTT_parameter` - Parameter used for calculating the round trip time (RTT). RTT is the length of time required for the MMT packet to be sent and the length of time it takes for an acknowledgement to be received. When computing the RTT, MMT sending entity records the time when the feedback is received. RTT is calculated by subtracting the `feedback_timestamp` from the recorded time and adding the `propagation_delay`.

`propagation_delay` - Propagation delay for the MMT packet to arrive at the MMT receiving entity. The MMT receiving entity calculates the `propagation_delay` by the subtracting the NTP time at the delivery instant of a MMT packet from the NTP time at the arrival instant of the MMT packet. The `propagation_delay` can be an average result of a propagation delay measured within the `measurement_duration`.

`feedback_timestamp` - NTP time in which the feedback is issued from the MMT receiving entity. This parameter is used to measure the propagation delay from the MMT receiving entity to the MMT sending entity.

9.4.7 NAM Feedback

9.4.7.1 Syntax

The syntax for the NAM feedback is shown in Table 45.

Table 46 — NAM_Feedback Message Syntax

Syntax	Values	No. of bits	Mnemonic
		16	**unsigned short**
NAM_Feedback_message ( ) {		**8**	**unsigned char**
message_id		**16**	**unsigned short**
version			
length			
payload{			
CLI_id		8	unsigned integer
available_bitrate		32	float
buffer_fullness		32	float
peak_bitrate		32	float
current_delay		32	float
SDU_size		32	unsigned integer
SDU _loss_ratio		8	unsigned integer
generation_time		32	float
BER		32	float
}			
}			
}			

8.4.7.2 Semantics

`message_id` – It indicates NAM Feedback message ID. The length of this field is 16 bits.
`version` – It indicates the version of NAM Feedback messages. MMT receiving entity may check whether the received message is new or not. The length of this field is 8 bits.
`length` – It indicates the length of NAM Feedback messages. The length of this field is 16 bits. It indicates the length of the NAM Feedback message counted in bytes starting from the next field to the last byte of the NAM Feedback message. The value '0' shall not be used.
`CLI_id` – The `CLI_id` is an arbitrary integer number to identify this NAM among the underlying network.
`available_bitrate` – the `available_bitrate` is bitrate that the scheduler of the underlying network can guarantee to the MMT stream. The `available_bitrate` is expressed in kilobits per second. Overhead for the protocols of the underlying network is not included.
`buffer_fullness` – the buffer is used to absorb excess bitrate higher than the `available_bitrate`. The `buffer_fullness` is expressed in bytes.

`peak_bitrate` – the `peak_bitrate` is maximum allowable bitrate that the underlying network can assign to the MMT stream. The `peak_bitrate` is expressed in kilobits per second. Overhead for the protocols of the underlying network is not included.

`current_delay` – the current_delay parameter indicates the last hop transport delay. The `current_delay` expressed in milliseconds.

`SDU_size` – SDU (Service Data Unit) is data unit in which the underlying network delivers the MMT data. The `SDU_size` specifies the length of the SDU and is expressed in bits. Overhead for the protocols of the underlying network is not included.

`SDU_loss_rate` – The `SDU_loss_ratio` is fraction of SDUs lost or detected as errorneous. Loss ratio of MMT packets can be calculated as a function of `SDU_loss_ratio` and `SDU_size`. The `SDU_loss_ratio` is expressed in percentile.

`generation_time` – The time when the parameters are generated. The `generation_time` is expressed in milliseconds.

`BER` – Bit Error Rate obtained from PHY or MAC layer. For `BER` from PHY layer, this value present as a positive value. For `BER` from MAC layer, this value present as a negative value which can be used as an absolute value.

10 Hypothetical Receiver Buffer Model

10.1 Introduction

The Hypothetical Receiver Buffer Model (HRBM) is used to ensure operation under a fixed end-to-end delay and limited memory requirement for buffering of incoming MMT packets. The hypothetical receiver buffer model is used by the MMT sending entity to emulate the behavior of the receiving entity. It is described in the following sub-clause in more detail.

The MMT sending entity runs the hypothetical receiver buffer model to ensure that any processing it performs on the packet stream is within the reception constraints in the MMT receiving entity. The sending entity determines the required buffering delay and the required buffer size and signals this information to the receiving entities.

At the receiving entity, several buffers are involved in reconstruction of MPU from the MMT packets received. Some delivery operations such as FEC coding, interleaving and retransmission introduce delay and jitter that requires buffering at the receiving entity to remove it. The hypothetical receiver buffer model defines operations of the buffers at the receiving entity to ensure that at any time the buffer occupancy is within the buffer size requirement. The buffers whose operation is defined are depicted in the following figure and are described in detail in the following sub-clauses.

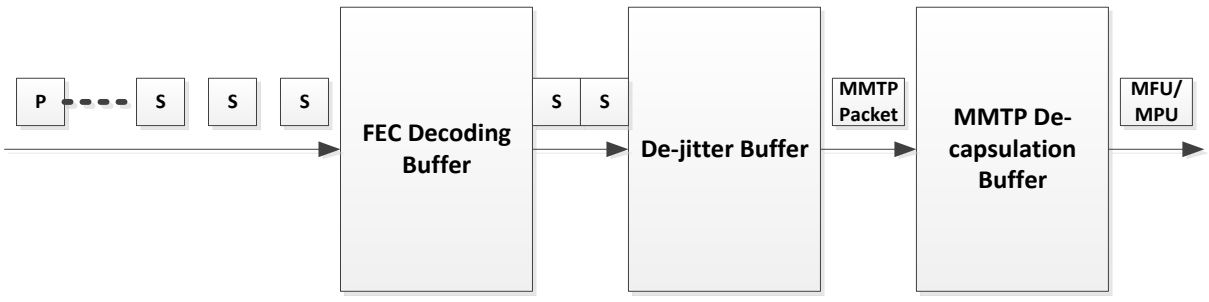


Figure 28 — MMT protocol Hypothetical Receiver Model

10.2 FEC Decoding Buffer

FEC decoding is typical for many applications, where lower layer transmission may not be sufficient to recover from channel errors or when network congestion may cause packet drops or excessive delays. To perform FEC decoding, a buffer is required where incoming packets are stored until sufficient source and repair data is available to perform FEC decoding.

In this hypothetical receiver buffer model, the FEC decoding buffer works as follows:

— The FEC decoding buffer is initially empty

— Insert incoming packet i with transmission timestamp ts into the FEC decoding buffer, if `buffer_occupancy + packet_size < max_buffer_size`, otherwise discard packet i as being non-compliant

— If FEC is applied to packet i

a. Determine source block *j* to which packet *i* belongs
b. Determine insertion time *t* of first packet of source block *j*
c. At time `t+FEC_buffer_time` move all packets (after FEC correction, if needed) of source block *j* to the de-jitter buffer
d. Discard repair packets

The `FEC_buffer_time` is the required buffer time since the reception of the first packet of a source block and until FEC decoding is attempted. This time is typically calculated based on the FEC block size.

10.3 De-jitter Buffer

The de-jitter buffer ultimately ensures that MMT packets experience a fixed transmission delay from source to the output of the MMT Protocol stack, assuming a maximum transmission delay. Data units that experience a transmission delay larger than the maximum transmission delay might be discarded by the MMT receiving entity as being very late.

The de-jitter buffer operates as follows:

— The de-jitter buffer is initially empty

— Insert the MMT packet as it arrives

— Remove MMT packet at time `ts+Δ`, where ts is the timestamp of the MMT packet and Δ is the signaled fixed end-to-end delay (as described in 8.4.4)

After the de-jittering is applied, all MMT packets that arrived correctly or were recovered through FEC/retransmissions will have experienced the same end-to-end delay.

10.4 MMT packet De-encapsulation Buffer

The MMT packet de-encapsulation buffer is used to perform MMT packet processing before output to the upper layers. The output of the MMT Protocol processor may either be the MFU payload (in low-delay operation) or a complete MPU. MPUs may be fragmented into smaller packets or aggregated into larger packets, depending on their size. The de-encapsulation (removal of the MMT packet and payload headers) and any required de-fragmentation/de-aggregation of the packets are then performed as part of the MMT Protocol processing. This procedure may require some buffering delay, called de-encapsulation delay, to perform assembly when an MPU is fragmented into multiple MMT packets. However, de-encapsulation delay is not considered as part of the transmission delay and the availability of an MPU for consumption by the coded media layer will be guaranteed by the entity fragmenting the MPU into multiple MMT packets, regardless of the de-capsulation delay.

The MMT packet de-encapsulation buffer works as follows:

— The MMT packet de-encapsulation buffer is initially empty.

— The MMT packet is inserted into the MMT packet de-capsulation buffer immediately after the de-jittering is performed.

— For MMT packets carrying aggregated payload, remove the packet and payload header and split the aggregate into separate MPUs.

— For MMT packets carrying fragmented payload, the packet is kept in the buffer until all corresponding fragments are received correctly or until a packet is received that does not belong to the same fragmented MPU.

— If all fragments of an MPU are received, remove MMT packet and payload header, reassemble and forward the re-constructed MPU to the upper layer, otherwise discard fragments of a non-complete MPU.

10.5 Usage of Hypothetical Receiver Buffer Model

Based on this hypothetical receiver buffer model, the delivery MMT sending entity is able to determine the transmission schedule, the buffer size and the buffering delay Δ, so that no packets are dropped, assuming a maximum delivery delay in the target path. The MMT sending entity guarantees that packets that experience a transmission delay below a set threshold will be output to the upper layer after a constant delay and without causing the MMT receiving entity buffer to underflow or overflow.

10.6 Estimation of end-to-end delay and buffer requirement

The MMT sending entity estimates the maximum expected and tolerable transmission delay in the transmission path down to the MMT receiving entities. If FEC is in use, the MMT sending entity adds an FEC buffering delay that covers for the time needed to assemble a source block, in the case FEC decoding is required to recover lost MMT packets. Finally, any delays that might be incurred by fragmentation of packets (and other operations) are added. The resulting estimation of the MMT packet delivery delay is then signaled to the MMT receiving entities as the fixed end-to-end transmission delay.

$$fixed\ end-to-end\ delay = maximum\ transmission\ delay + FEC\ buffering\ time$$

In order to estimate the resulting buffer requirement, the MMT sending entity uses the fixed end-to-end delay and subtracts the minimum transmission delay for the path down to the MMT receiving entity and then estimates the buffer size requirement as the product of maximum bitrate of the MMT packet stream and the calculated buffered data duration.

$$buffer\ size = (maximum\ delay - minimum\ delay) * maximum\ bitrate$$

10.7 HRBM signaling

After determining the required buffer size and the fixed end-to-end delay for the system, the MMT sending entity communicates this information to the MMT receiving entity. This is done using the signaling protocol between the MMT sending entity and the MMT receiving entity. The MMT sending entity continuously runs the hypothetical receiver buffer to verify that the selected end-to-end delay and buffer size are aligned and do not cause buffer under-runs or overruns. At the MMT receiving entity side, the MMT receiving entity is instructed to perform buffering so that each data unit experiences the signaled fixed end-to-end delay Δ before it is forwarded to upper layers. Under the assumption that clocks are synchronized, the output time is then calculated based on the transmission timestamp and the signaled fixed end-to-end delay.An MMT protocol signaling message is defined to carry the information to the MMT receiving entities. Additionally, the information is also transmitted out-of-band.

Annex A
(informative)

Example of Composition Information

A.1 Introduction

This Annex shows examples of Composition Information.

A.2 Use case: Area change

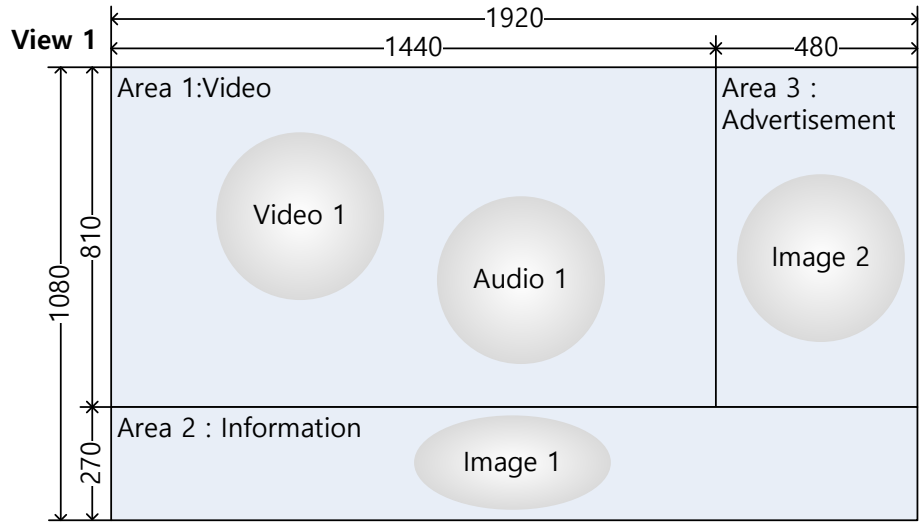


Figure A. 1 — Initial View of Areas

There are 7 different types of Asset in Package. CI is provided as **Error! Reference source not found.** first. The initial View is updated as **Error! Reference source not found.** by following conditions:

Image1 in Area 2 is updated to Image3 at 18:00 (wall-clock time).

If Image2 in Area3 is selected, Area3 is update to Area4 and Area5. Area4 represents Widget 1. And Area 5 represents Image 4

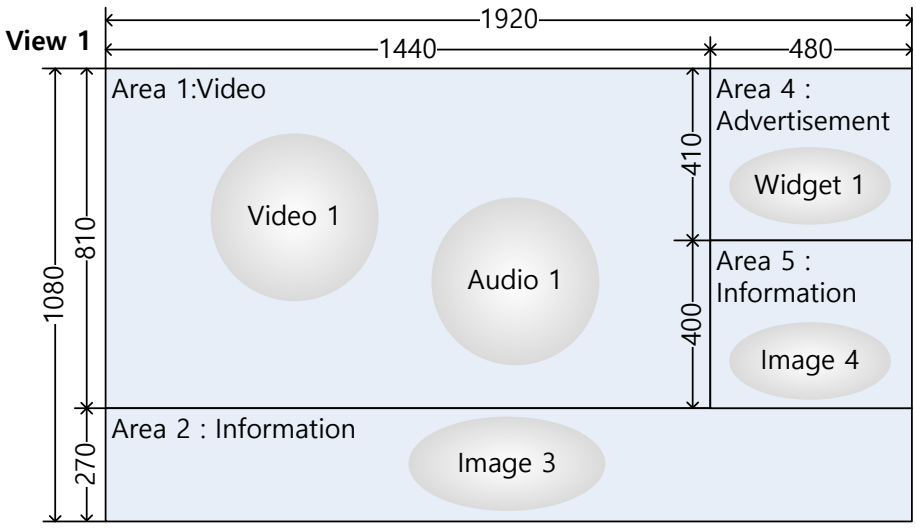


Figure A. 2 — Later View of Areas after change

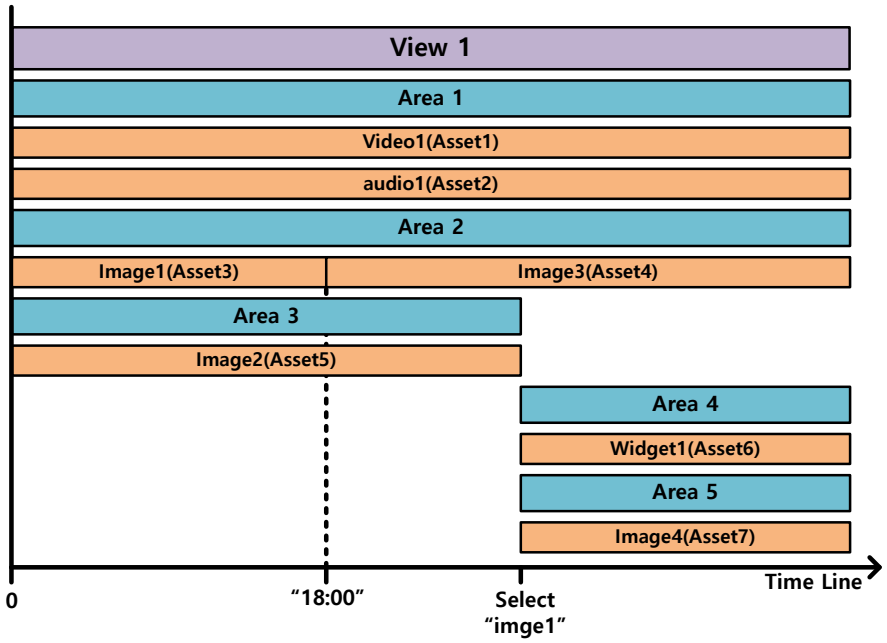


Figure A. 3 — Timeline of Area change

Table A. 1 — CI of Area change

```
<!DOCTYPE html>
<html xmlns=”http://www.w3.org/1999/xhtml”>
    <head>
        <title>Entertainment</title>
        <MMT-CI:view id="View1" style="position:absolute; width:1920px; height:1080px" MMT-
CI:begin="0s" MMT-CI:end="indefinite">
            <MMT-CI:divLocation id="divL1" style="position:absolute; left:0px; top:0px;
width:1440px; height:810px" MMT-CI:begin="0s" MMT-CI:end="indefinite" MMT-CI:refDiv="Area1"/>
            <MMT-CI:divLocation id="divL2" style="position:absolute; left:0px; top:810px;
width:1920px; height:270px" MMT-CI:begin="0s" MMT-CI:end="indefinite" MMT-CI:refDiv="Area2"/>
            <MMT-CI:divLocation id="divL3" style="position:absolute; left:1440px; top:0px;
width:480px; height:810px" MMT-CI:begin="0s" MMT-CI:end="image2.Click" MMT-CI:refDiv="Area3"/>
```

```
            <MMT-CI:divLocation id="divL4" style="position:absolute; left:1440px; top:0px;
width:480px; height:410px" MMT-CI:begin="image2.Click" MMT-CI:end="indefinite" MMT-CI:refDiv="Area4"/>
            <MMT-CI:divLocation id="divL5" style="position:absolute; left:1440px; top:410px;
width:480px; height:400px" MMT-CI:begin="image2.Click" MMT-CI:end="indefinite" MMT-CI:refDiv="Area5"/>
            </MMT-CI:view>
    </head>
    <body>
        <div id="Area1" style="position:absolute; width:1440px; height:810px">
            <video id="Video1" src="mmt://asset1.m2v" style="position:absolute; left:0px; top:0px;
width:1440px; height:810px" MMT-CI:begin="0s"/>
            <audio id="audio1" src="mmt://asset2.mp3" MMT-CI:begin="0s"/>
        </div>
        <div id="Area2" style="position:absolute; width:1920px; height:270px">
            <img id="Image1" src="mmt://asset3.jpg" style="position:absolute; left:0px; top:0px;
width:1920px; height:270px" MMT-CI:begin="0s" MMT-CI:end="18:00"/>
            <img id="Image3" src="mmt://asset4.jpg" style="position:absolute; left:0px; top:0px;
width:1920px; height:270px" MMT-CI:begin="Image1.end"/>
        </div>
        <div id="Area3" style="position:absolute; width:480px; height:810px">
            <img id="Image2" src="mmt://asset5.jpg" style="position:absolute; left:0px; top:0px;
width:480px; height:810px" MMT-CI:begin="0s"/>
        </div>
        <div id="Area4" style="position:absolute; width:480px; height:410px">
            <object id="Widget1" src="mmt://asset6.bt" style="position:absolute; left:0px; top:0px;
width:480px; height:410px" MMT-CI:begin="0s"/>
        </div>
        <div id="Area5" style="position:absolute; width:480px; height:400px">
            <img id="Image4" src="mmt://asset7.jpg" style="position:absolute; left:0px; top:0px;
width:480px; height:410px" MMT-CI:begin="0s"/>
        </div>
    </body>
</html>
```

A.3 Use case: View change

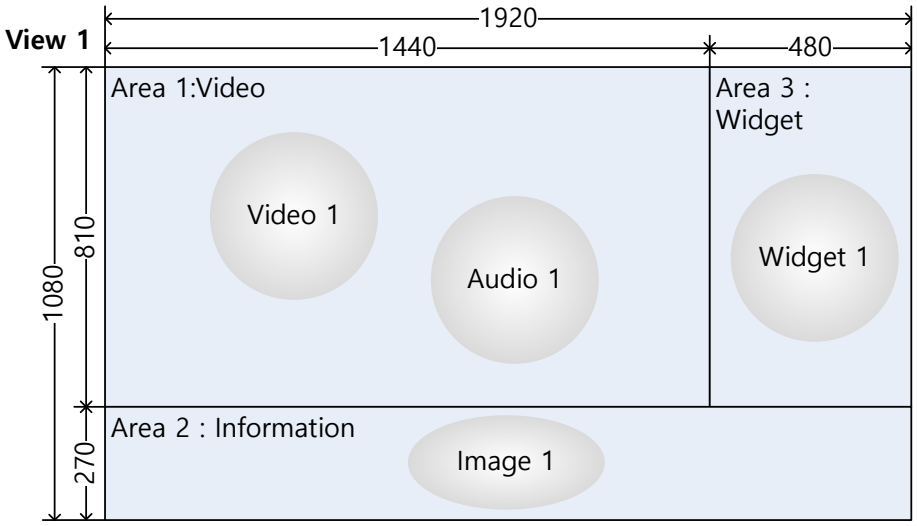


Figure A. 4 — Initial View of View change

There are 2 Views and 7 Assets in Package. CI provides View as **Error! Reference source not found.** first. The initial View is updated as **Error! Reference source not found.** by following conditions:

If Image1 in Area2 is selected, View 1 is update to View 2.

After updating, Vidio1 and Audio1 are played continuously. And Video2 in Area5 is sync with Video1. In this example, Video2 will start at the same time with Video1. However, video2 does not display. It is activated with its Area (Area4) and played on the screen.

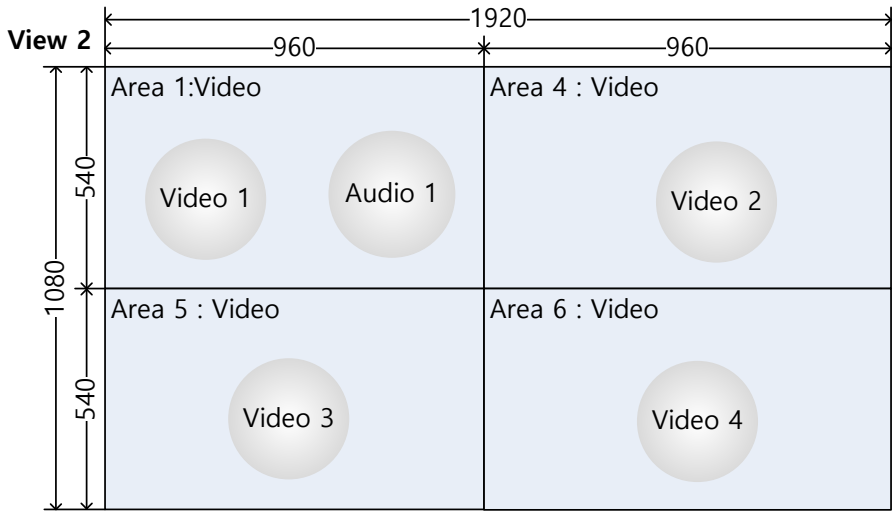


Figure A. 5 — Later View of View change

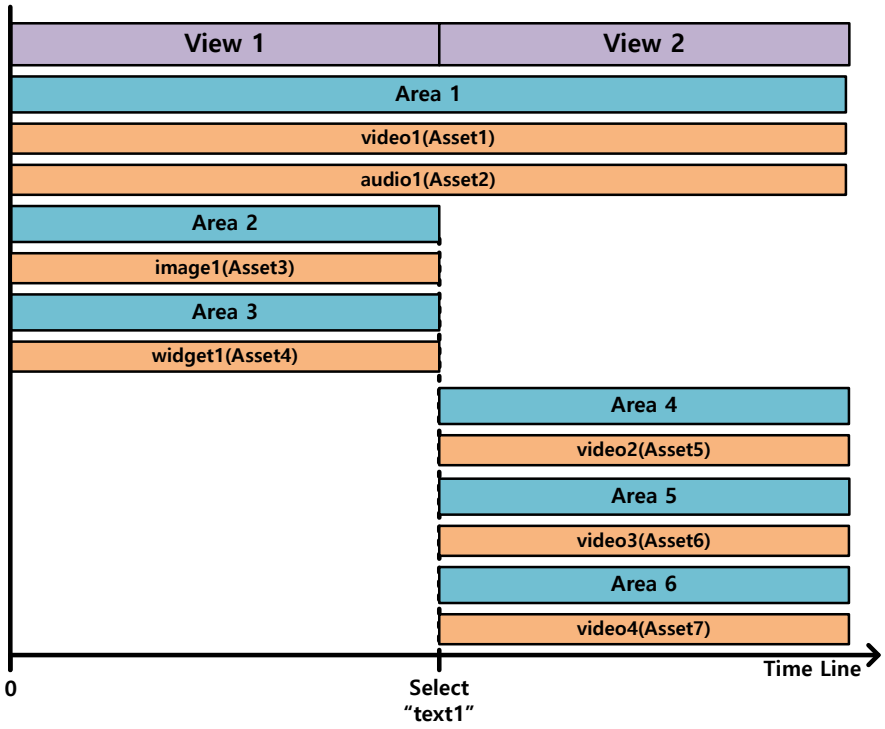


Figure A. 6 — Timeline of View change

Table A. 2 — CI of View change

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
    <head>
        <title>Sports</title>
        <MMT-CI:view id="View1" style="position:absolute; width:1920px; height:1080px" MMT-
CI:begin="0s" MMT-CI:end="text1.Click">
            <MMT-CI:divLocation id="divL1" style="position:absolute; left:0px; top:0px;
width:1440px; height:810px" MMT-CI:begin="0s" MMT-CI:end="indefinite" MMT-CI:refDiv="Area1"/>
            <MMT-CI:divLocation id="divL2" style="position:absolute; left:0px; top:810px;
```

```
width:1920px; height:270px" MMT-CI:begin="0s" MMT-CI:end="indefinite" MMT-CI:refDiv="Area2"/>
            <MMT-CI:divLocation id="divL3" style="position:absolute; left:1440px; top:0px;
width:480px; height:810px" MMT-CI:begin="0s" MMT-CI:end="indefinite" MMT-CI:refDiv="Area3"/>
        </MMT-CI:view>
        <MMT-CI:view id="View2" style="position:absolute; width:1920px; height:1080px" MMT-
CI:begin="text1.Click">
            <MMT-CI:divLocation id="divL4" style="position:absolute; left:0px; top:0px; width:960px;
height:540px" MMT-CI:begin="0s" MMT-CI:end="indefinite" MMT-CI:refDiv="Area1"/>
            <MMT-CI:divLocation id="divL5" style="position:absolute; left:960px; top:0px;
width:960px; height:540px" MMT-CI:begin="0s" MMT-CI:end="indefinite" MMT-CI:refDiv="Area4"/>
            <MMT-CI:divLocation id="divL6" style="position:absolute; left:0px; top:540px;
width:960px; height:540px" MMT-CI:begin="0s" MMT-CI:end="indefinite" MMT-CI:refDiv="Area5"/>
            <MMT-CI:divLocation id="divL7" style="position:absolute; left:960px; top:540px;
width:960px; height:540px" MMT-CI:begin="0s" MMT-CI:end="indefinite" MMT-CI:refDiv="Area6"/>
        </MMT-CI:view>
    </head>
    <body>
        <div id="Area1" style="position:absolute; width:1440px; height:810px">
            <video id="video1" src="mmt://asset1.m2v" style="position:absolute; left:0px; top:0px;
width:1440px; height:810px" MMT-CI:begin="0s"/>
            <audio id="audio1" src="mmt://asset2.mp3" MMT-CI:begin="0s"/>
        </div>
        <div id="Area2" style="position:absolute; width:1920px; height:270px">
            <img id="text1" src="mmt://asset3.jpg" style="position:absolute; left:0px; top:0px;
width:1920px; height:270px" MMT-CI:begin="0s"/>
        </div>
        <div id="Area3" style="position:absolute; width:480px; height:810px">
            <object id="widget1" src="mmt://asset4.bt" style="position:absolute; left:0px; top:0px;
width:480px; height:810px" MMT-CI:begin="0s"/>
        </div>
        <div id="Area4" style="position:absolute; width:960px; height:540px">
            <video id="video2" src="mmt://asset5.m2v" style="position:absolute; left:0px; top:0px;
width:960px; height:540px" MMT-CI:begin="video1.begin"/>
        </div>
        <div id="Area5" style="position:absolute; width:960px; height:540px">
            <video id="video3" src="mmt://asset6.m2v" style="position:absolute; left:0px; top:0px;
width:960px; height:540px" MMT-CI:begin="0s"/>
        </div>
        <div id="Area6" style="position:absolute; width:960px; height:540px">
            <video id="video4" src="mmt://asset7.m2v" style="position:absolute; left:0px; top:0px;
width:960px; height:540px" MMT-CI:begin="0s"/>
        </div>
    </body>
</html>
```

A.4 Use case: Multi-screen Presentation – Asset Sharing

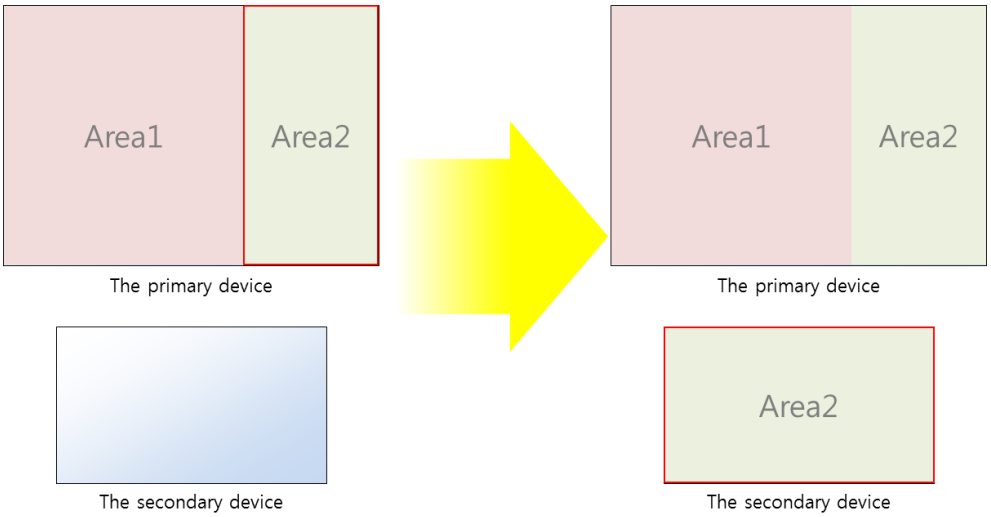


Figure A. 7 — Presentation of Asset sharing

This sub-clause provides a simple example for Asset Sharing. In the following example, an Area or Areas in a primary screen can be shared with the secondary screen after beginning of multi-screen presentation. Figure A.7**Error! Reference source not found.** shows the presentation of this example.

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <MMT-CI:view id="View1" MMT-CI:viewtype="default" style="position:absolute; width:1920px;
height:1080px">
        <MMT-CI:divLocation id="divL1" style="position:absolute; width:70%; height:100%; left:0%;
top:0%" MMT-CI:refDiv="Area1"/>
        <MMT-CI:divLocation id="divL2" style="position:absolute; width:30%; height:100%; left:70%;
top:0%" MMT-CI:refDiv="Area2"/>
    </MMT-CI:view>
    <MMT-CI:view id="View2" MMT-CI:viewtype="multiple" style="position:absolute; width:1920px;
height:1080px">
        <MMT-CI:divLocation id="divL1" style="position:absolute; width:70%; height:100%; left:0%;
top:0%" MMT-CI:refDiv="Area1"/>
        <MMT-CI:divLocation id="divL2" style="position:absolute; width:30%; height:100%; left:70%;
top:0%" MMT-CI:refDiv="Area2" MMT-CI:plungeOut="sharable"/>
    </MMT-CI:view>
</head>
<body>
    <div id="Area1" style="position:absolute; width:1000px; height:1000px">
        <video id="video1" src="mmt://package1/asset1" style="position:absolute; width:100%;
height:100%; left:0px; top:0px"/>
    </div>
    <div id="Area2" style="position:absolute; width:600px; height:1000px">
        <video id="video2" src="mmt://package1/asset2" style="position:absolute; width:100%;
height:100%; left:0px; top:0px"/>
    </div>
</body>
</html>
```

A.5 Use case: Multi-screen Presentation – Dynamic Asset Sharing

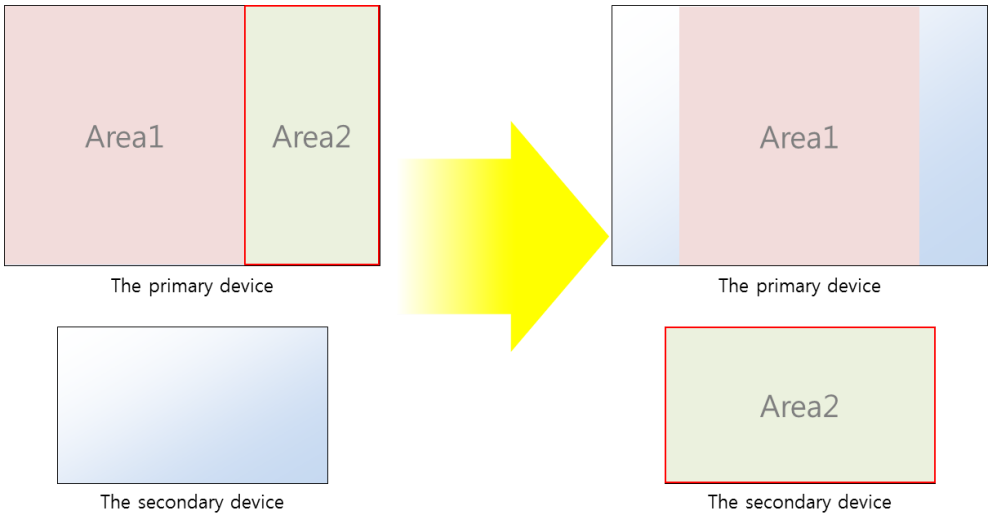


Figure A. 8 — Presentation of Dynamic Asset sharing

This sub-clause provides a simple example for Dynamic Asset Sharing. In the following example, one of the Areas in a primary screen moves to the secondary screen after beginning of multi-screen presentation, and the Area remained in the primary screen is repositioned in the center. Figure A.8 **Error! Reference source not found.** shows the presentation of this example.

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <MMT-CI:view id="View1" MMT-CI:viewtype="default" style="position:absolute; width:1920px;
height:1080px">
        <MMT-CI:divLocation id="divL1" style="position:absolute; width:70%; height:100%; left:0%;
top:0%" MMT-CI:refDiv="Area1"/>
        <MMT-CI:divLocation id="divL2" style="position:absolute; width:30%; height:100%; left:70%;
top:0%" MMT-CI:refDiv="Area2"/>
    </MMT-CI:view>
    <MMT-CI:view id="View2" MMT-CI:viewtype="multiple" style="position:absolute; width:1920px;
height:1080px">
        <MMT-CI:divLocation id="divL1" style="position:absolute; width:70%; height:100%; left:0%;
top:0%" MMT-CI:refDiv="Area1"/>
        <MMT-CI:divLocation id="divL2" style="position:absolute; width:30%; height:100%; left:70%;
top:0%" MMT-CI:refDiv="Area2" MMT-CI:plungeOut="dynamic"/>
    </MMT-CI:view>
</head>
<body>
    <div id="Area1" style="position:absolute; width:1000px; height:1000px">
        <video id="video1" src="mmt://package1/asset1" style="position:absolute; width:100%;
height:100%; left:0px; top:0px"/>
    </div>
    <div id="Area2" style="position:absolute; width:600px; height:1000px">
        <video id="video2" src="mmt://package1/asset2" style="position:absolute; width:100%;
height:100%; left:0px; top:0px"/>
    </div>
</body>
</html>
```

A.6 Use case: Multi-screen Presentation – Complementary Asset

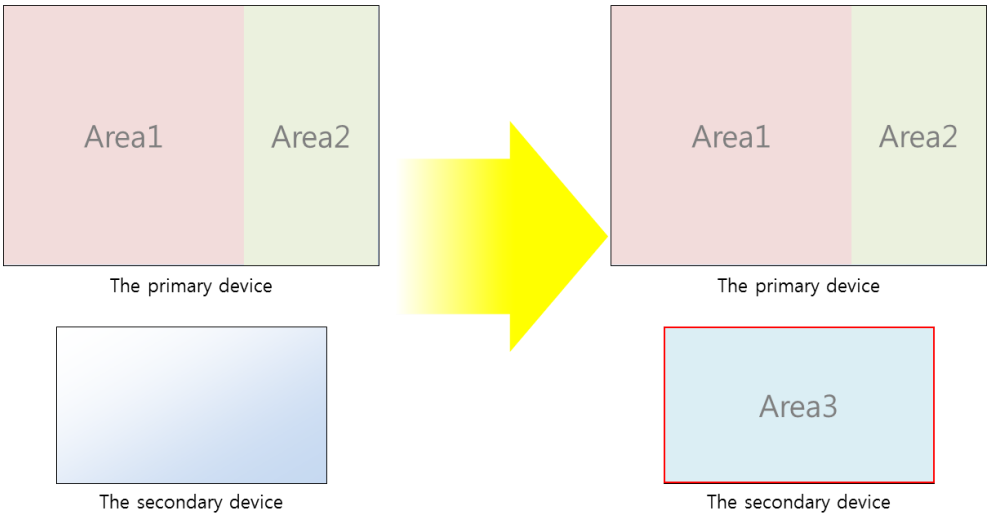


Figure A. 9 — Presentation of Complementary Asset

This sub-clause provides a simple example for Complementary Asset. In the following example, an Area invisible at the primary screen moves to a secondary screen after beginning of multi–screen presentation. Figure A.9**Error! Reference source not found.** shows the presentation of this example.

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <MMT-CI:view id="View1" MMT-CI:viewtype="default" style="position:absolute; width:1920px;
height:1080px">
        <MMT-CI:divLocation id="divL1" style="position:absolute; width:70%; height:100%; left:0%;
top:0%" MMT-CI:refDiv="Area1"/>
        <MMT-CI:divLocation id="divL2" style="position:absolute; width:30%; height:100%; left:70%;
top:0%" MMT-CI:refDiv="Area2"/>
    </MMT-CI:view>
    <MMT-CI:view id="View2" MMT-CI:viewtype="multiple" style="position:absolute; width:1920px;
height:1080px">
        <MMT-CI:divLocation id="divL1" style="position:absolute; width:70%; height:100%; left:0%;
top:0%" MMT-CI:refDiv="Area1"/>
        <MMT-CI:divLocation id="divL2" style="position:absolute; width:30%; height:100%; left:70%;
top:0%" MMT-CI:refDiv="Area2"/>
        <MMT-CI:divLocation id="divL3" MMT-CI:refDiv="Area3" MMT-CI:plungeOut="complementary"/>
    </MMT-CI:view>
</head>
<body>
    <div id="Area1" style="position:absolute; width:1000px; height:1000px">
        <video id="video1" src="mmt://package1/asset1" style="position:absolute; width:100%;
height:100%; left:0px; top:0px"/>
    </div>
    <div id="Area2" style="position:absolute; width:600px; height:1000px">
        <video id="video2" src="mmt://package1/asset2" style="position:absolute; width:100%;
height:100%; left:0px; top:0px"/>
    </div>
    <div id="Area3" style="position:absolute; width:1024px; height:768px">
        <object id="widget1" src="mmt://package1/asset3" style="position:absolute; width:100%;
height:100%; left:0px; top:0px"/>
    </div>
</body>
</html>
```

A.7 Use case: Multi-screen Presentation – Asset-level Sharing

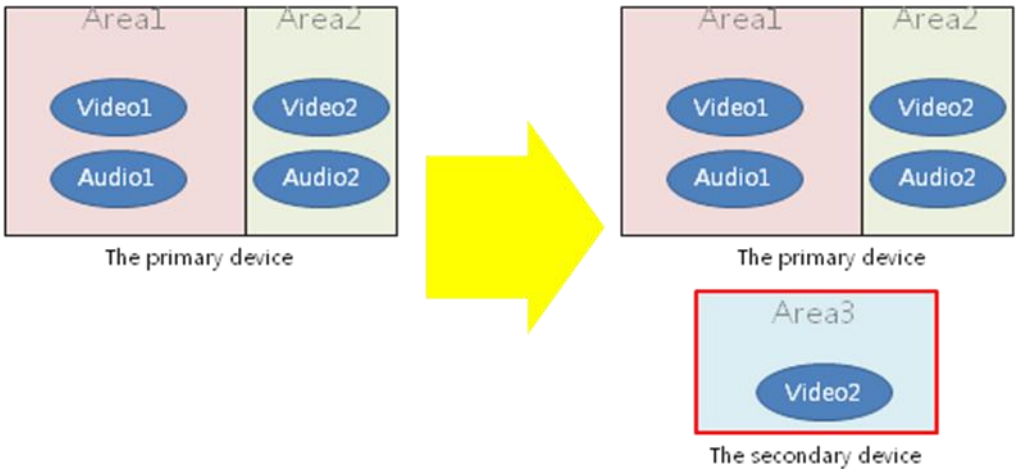


Figure A. 10 – Asset-level Sharing

This sub-clause provides a simple example for Asset-level Sharing. In the following example, Area1 has audio1 and video1, and Area2 has audio2 and video2. Area3 has video2 only. The plungeOut attribute of Area3 is set to 'complementary'. You can share the 'video2' only with the secondary screen. Because the plungeOut attribute is set to 'complementary', the primary screen does not display Area3.

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <MMT-CI:view id="View1" MMT-CI:viewtype="default" style="position:absolute; width:1920px; height:1080px">
        <MMT-CI:divLocation id="divL1" style="position:absolute; width:70%; height:100%; left:0%; top:0%"
MMT-CI:refDiv="Area1"/>
        <MMT-CI:divLocation id="divL2" style="position:absolute; width:30%; height:100%; left:70%; top:0%"
MMT-CI:refDiv="Area2"/>
    </MMT-CI:view>
    <MMT-CI:view id="View2" MMT-CI:viewtype="multiple" style="position:absolute; width:1920px; height:1080px">
        <MMT-CI:divLocation id="divL1" style="position:absolute; width:70%; height:100%; left:0%; top:0%"
MMT-CI:refDiv="Area1"/>
        <MMT-CI:divLocation id="divL2" style="position:absolute; width:30%; height:100%; left:70%; top:0%"
MMT-CI:refDiv="Area2"/>
        <MMT-CI:divLocation id="divL3" MMT-CI:refDiv="Area3" MMT-CI:plungeOut="complementary"/>
    </MMT-CI:view>
</head>
<body>
    <div id="Area1" style="position:absolute; width:1000px; height:1000px">
        <video id="video1" src="mmt://package1/asset1"/>
        <audio id="Audio1" src="mmt://package1/asset2"/>
    </div>
    <div id="Area2" style="position:absolute; width:600px; height:1000px">
        <video id="video2" src="mmt://package1/asset3"/>
        <audio id="Audio2" src="mmt://package1/asset4"/>
    </div>
    <div id="Area3">
        <video id="video2" src="mmt://package1/asset3"/>
    </div>
</body>
</html>
```

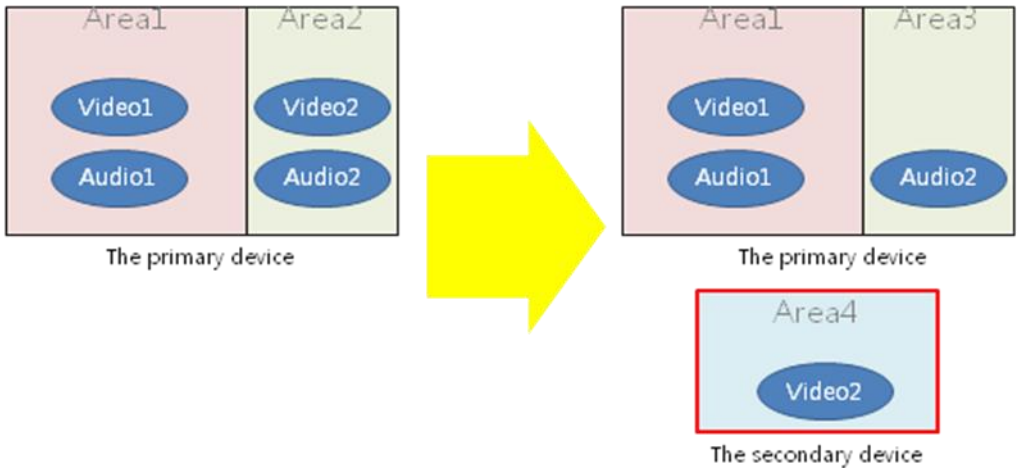


Figure A. 11 – Another example of Asset-level Sharing

Following provides another example for Asset-level Sharing. In this example, the primary device has only audio2, and video2 played in the secondary device.

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <MMT-CI:view id="View1" MMT-CI:viewtype="default" style="position:absolute; width:1920px; height:1080px">
        <MMT-CI:divLocation id="divL1" style="position:absolute; width:70%; height:100%; left:0%; top:0%"
MMT-CI:refDiv="Area1"/>
        <MMT-CI:divLocation id="divL2" style="position:absolute; width:30%; height:100%; left:70%; top:0%"
MMT-CI:refDiv="Area2"/>
    </MMT-CI:view>
    <MMT-CI:view id="View2" MMT-CI:viewtype="multiple" style="position:absolute; width:1920px; height:1080px">
        <MMT-CI:divLocation id="divL1" style="position:absolute; width:70%; height:100%; left:0%; top:0%"
MMT-CI:refDiv="Area1"/>
        <MMT-CI:divLocation id="divL3" style="position:absolute; width:30%; height:100%; left:70%; top:0%"
MMT-CI:refDiv="Area3"/>
        <MMT-CI:divLocation id="divL4" MMT-CI:refDiv="Area4" MMT-CI:plungeOut="complementary"/>
    </MMT-CI:view>
</head>
<body>
    <div id="Area1" style="position:absolute; width:1000px; height:1000px">
        <video id="video1" src="mmt://package1/asset1"/>
        <audio id="Audio1" src="mmt://package1/asset2"/>
    </div>
    <div id="Area2" style="position:absolute; width:600px; height:1000px">
        <video id="video2" src="mmt://package1/asset3"/>
        <audio id="Audio2" src="mmt://package1/asset4"/>
    </div>
    <div id="Area3" style="position:absolute; width:600px; height:1000px">
        <audio id="Audio2" src="mmt://package1/asset4"/>
    </div>
    <div id="Area4">
        <video id="video2" src="mmt://package1/asset3"/>
    </div>
</body>
</html>
```

A.8 Use case: Asset Receiving in Multi-screen Presentation

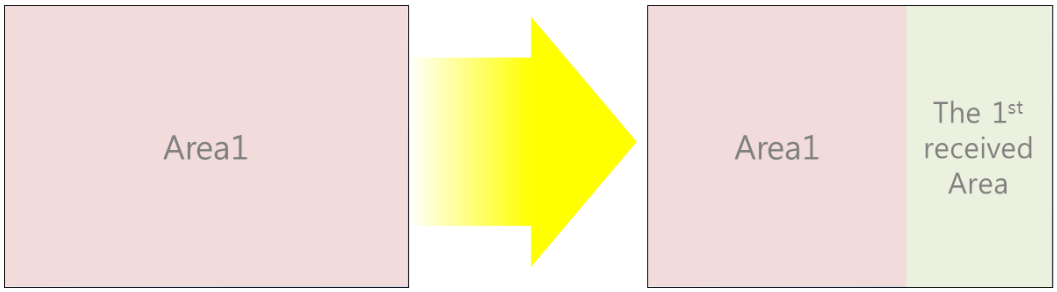


Figure A. 12 — Presentation of MMT Asset receiving in Multi-screen

This sub-clause provides a simple example for MMT Asset receiving. In the following example, the primary screen receives an Area from the secondary screen after beginning of multi-screen presentation, and the existing Area in the primary screen is resized. **Error! Reference source not found.**Figure A. 12 shows the presentation of this example.

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <MMT-CI:view id="View1" MMT-CI:viewtype="default" style="position:absolute; width:1920px;
height:1080px">
        <MMT-CI:divLocation id="divL1" style="position:absolute; width:100%; height:100%; left:0px;
top:0px" MMT-CI:refDiv="Area1"/>
    </MMT-CI:view>
    <MMT-CI:view id="View2" MMT-CI:viewtype="receptible" style="position:absolute; width:1920px;
height:1080px">
        <MMT-CI:divLocation id="divL2" style="position:absolute; width:70%; height:100%; left:0%;
top:0%" MMT-CI:refDiv="Area1"/>
        <MMT-CI:divLocation id="divL3" style="position:absolute; width:30%; height:100%; left:70%;
top:0%" MMT-CI:plungeIn="1"/>
    </MMT-CI:view>
</head>
<body>
    <div id="Area1" style="width:1000px; height:1000px">
        <video id="video1" src="mmt://package1/asset1" style="position:absolute; width:100%;
height:100%; left:0px; top:0px"/>
    </div>
</body>
</html>
```

When the primary screen receives two Areas in this example, the second received Area will be not shown although the first received Area will be shown because there is no more blank Area. On the other hand, if the **head** element in this example is changed as following, the first and second Areas will be shown as Figure A. 13**Error! Reference source not found.**.

```
<head>
    <MMT-CI:view id="View1" MMT-CI:viewtype="default" style="position:absolute; width:1920px;
height:1080px">
        <MMT-CI:divLocation id="divL1" style="position:absolute; width:100%; height:100%; left:0px;
top:0px" MMT-CI:refDiv="Area1"/>
    </MMT-CI:view>
    <MMT-CI:view id="View2" MMT-CI:viewtype="receptible" style="position:absolute; width:1920px;
height:1080px">
        <MMT-CI:divLocation id="divL2" style="position:absolute; width:70%; height:100%; left:0%;
top:0%" MMT-CI:refDiv="Area1"/>
        <MMT-CI:divLocation id="divL3" style="position:absolute; width:30%; height:100%; left:70%;
top:0%" MMT-CI:plungeIn="1"/>
        <MMT-CI:divLocation id="divL4" style="position:absolute; width:30%; height:30%; left:0%;
top:70%" MMT-CI:plungeIn="2"/>
    </MMT-CI:view>
</head>
```

```
<body>
    <div id="Area1" style="width:1000px; height:1000px">
        <video id="video1" src="mmt://package1/asset1" style="position:absolute; width:100%;
height:100%; left:0px; top:0px"/>
    </div>
</body>
</html>
```

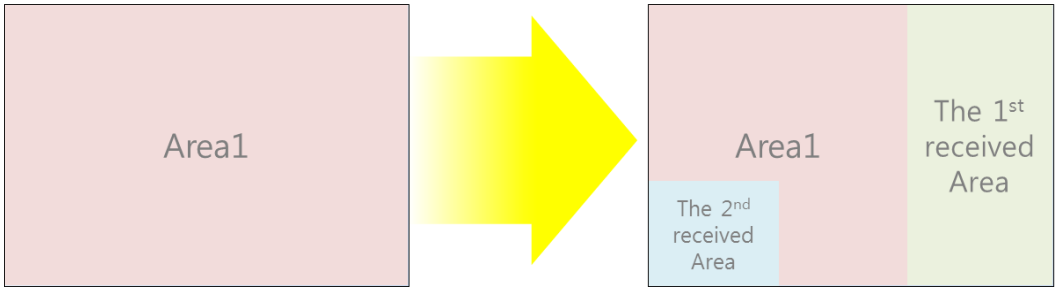


Figure A. 13 — Presentation of changed MMT Asset receiving in Multi-screen

A.9 Use case: Hierarchical CI – Subset of CI

Main CI provides the mandatory information for the Area1 and related assets; video1 and audio1. The area information for Area2 with the asset Image1 and Area3 with the asset Image2 are delivered separately in a different subset of CI. In this use case, there are 2 subset of CI; Subset-0 CI (base CI) and Subset-1 CI (for Area2 with Image1 and Area3 with Image2). After receiving Subset-0 CI, an MMT receiving entity receives the Subset -1 CI that includes area information on Area2 with Image1 and Area3 with Image2.

— Base CI (Subset-0 CI):

```
<html>
<head>
  <title>Entertainment</title>
  <MMT-CI:view id=”View1” >
    <MMT-CI:divLocation id=”divL1” MMT-CI:refDiv=”Area1” />
    <MMT-CI:divLocation id=”divL2” MMT-CI:refDiv=”Area2” />
    <MMT-CI:divLocation id=”divL3” MMT-CI:refDiv=”Area3” />
  </MMT-CI:view>
</head>
<body>
  <div id=”Area1”>
    <video id=”Video1” src=”mmt://Video1” />
    <audio id=”Audio1” src=”mmt://Audio1” />
  </div>
  <div xmlns:xlink=”http://www.w3.org/1999/xlink”
                xlink:href=”mmt://CI.subset.1” xlink:actuate=”onRequest” />
</body>
</html>
```

— Subset -1 CI:

```
<div id=”Area2”>
    <img id=”Image1” src=”mmt://Image1” />
</div>
<div id=”Area3”>
    <img id=”Image2” src=”mmt://Image2” />
</div>
```

A.10 Use case: Hierarchical CI – Fragmented CI

A service is composed of Assets provided by two different MMT sending entities and a view for the service is structured by two Areas. Main CI provides information of the areas and information for Assets consumed in Area1. Those Assets for Area1 and the main CI are delivered from the main MMT sending entity. Sub-CI provides information for Assets consumed in Area2. Those Assets for Area2 and the sub-CI are delivered from a sub-MMT sending entity.

While the layer-0 and the upper-layer CIs are delivered from single MMT sending entity in the case of above layered CI, main/sub CIs are delivered from a plurality of different MMT sending entities /networks in the case of fragmented CI. Each MMT sending entity has its own Assets, and CI is fragmented to multiple parts of CI, main CI and sub-CIs, according to the Assets owned by the MMT sending entities. The purpose of the fragmented CI mechanism is to allow separate management of CIs and their associated Assets by different MMT sending entities. MMT receiving entity receives main CI and sub-CIs composing a service and consumes the corresponding Assets.

— Main CI on main MMT sending entity: main-s:

```
<html>
<head>
   <title>Entertainment</title>
   <MMT-CI:view id=”View1”>
     <MMT-CI:divLocation id=”divL1” MMT-CI:refDiv=”Area1”/>
     <MMT-CI:divLocation id=”divL2” MMT-CI:refDiv=”Area2”/>
   </MMT-CI:view>
</head>
<body>
   <div id=”Area1”>
     <video id=”Video1” src=”mmt://main-s/Video1”/>
     <audio id=”Audio1” src=”mmt://main-s/Audio1”/>
   </div>
   <div xmlns:xlink=”http://www.w3.org/1999/xlink”
                  xlink:href=”mmt://sub-s/CI.1” xlink:actuate=”onLoad”/>
</body>
</html>
```

— Sub-CI.1 on sub MMT sending entity: sub-s:

```
<div id=”Area2”>
    <video id=”Video2” src=”mmt://sub-s/Video2”/>
    <audio id=”Audio2” src=”mmt://sub-s/Audio2”/>
```

```
</div>
```

A.11 Use case: Multi-layered media data

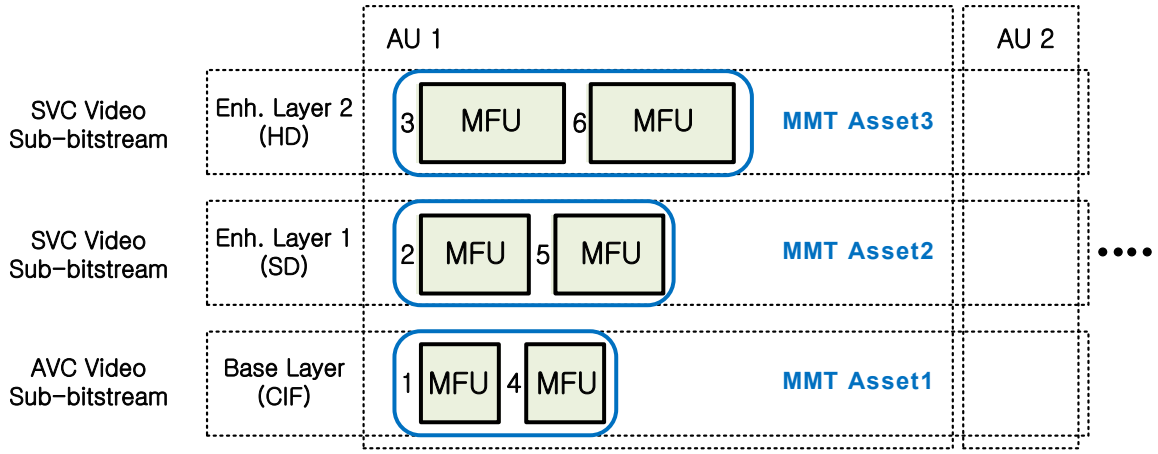


Figure A. 14 — Assets in case of SVC media data

There are 3 layers of SVC media data which consist of base layer, enhancement layer 1, and enhancement layer 2. Each layer is mapped into indivisual Asset as shown in Figure A. 14. In this example, Asset2 and Asset3 contain enhancement layer 1 and enhancement layer 2, respectively, while Asset1 cantains base layer. Asset 2 and 3 should refer to one ore more Assets in order to be properly decoded and presented.

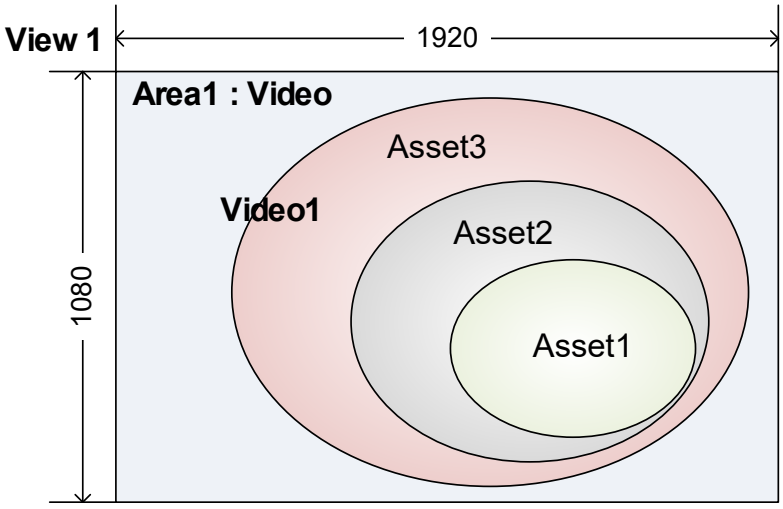


Figure A. 15 — View of SVC media data

CI provides View as Figure A. 15. The Video 1 may include one of three Assets as a *refAsset*. When *isDependent* attribute is enabled, the corresponding Asset may contain a subset of AU which corresponds to a certain layer of the layered media data. In this case, two types of the usage of the proposed attributes can be used as follows:

Type #1

```
<!DOCTYPE html>

<html xmlns=”http://www.w3.org/1999/xhtml”>

    <head>

        <title>Entertainment</title>

        <MMT-CI:view id="View1" style="position:absolute; width:1920px; height:1080px" MMT-
CI:begin="0s" MMT-CI:end="indefinite">

            <MMT-CI:divLocation id="divL1" style="position:absolute; left:0px; top:0px;
width:1920px; height:1080px" MMT-CI:begin="0s" MMT-CI:end="indefinite" MMT-CI:refDiv="Area1"/>

        </MMT-CI:view>
    </head>

    <body>

        <div id="Area1" style="position:absolute; width:1920px; height:1080px">

            <video id="Video1" src="mmt://asset3.mp4" style="position:absolute; left:0px; top:0px;
width:1920px; height:1080px" MMT-CI:begin="0s"/>

            <source src="mmt://asset3.mp4" isDependent="TRUE" depAssetID="mmt://asset2.mp4
mmt://asset1.mp4"/>

        </div>

    </body>

</html>
```

Type #2

```
<!DOCTYPE html>

<html xmlns=”http://www.w3.org/1999/xhtml”>

    <head>

        <title>Entertainment</title>

        <MMT-CI:view id="View1" style="position:absolute; width:1920px; height:1080px" MMT-
CI:begin="0s" MMT-CI:end="indefinite">

            <MMT-CI:divLocation id="divL1" style="position:absolute; left:0px; top:0px;
width:1920px; height:1080px" MMT-CI:begin="0s" MMT-CI:end="indefinite" MMT-CI:refDiv="Area1"/>

        </MMT-CI:view>

    </head>

    <body>

        <div id="Area1" style="position:absolute; width:1920px; height:1080px">

            <video id="Video1" src="mmt://asset3.mp4" style="position:absolute; left:0px; top:0px;
```

```
width:1920px; height:1080px" MMT-CI:begin="0s"/>

            <source src="mmt://asset3.mp4" isDependent="TRUE" depAssetID="mmt://asset2.mp4"/>

            <source src="mmt://asset2.mp4" isDependent="TRUE" depAssetID="mmt://asset1.mp4"/>

        </div>

    </body>

</html>
```

Annex B
(informative)

The QoS management Model for MMT

B.1 Introduction

Annex B describes the example operation in QoS control in MMT with the functionalities such as ADC (Asset Delivery Characteristics), CLI (Cross Layer Interface), and QoS-related fields in an MMT packet. ADC, CLI and QoS related information in MMT packet can be used in 3 different QoS environment, which are best effort, per-class QoS management, and per-flow QoS management. In addition, MMT ARQ process is also explained.

B.2 MMT QoS management for Best effort

In best effort network, networks do not provide any network level QoS control so end-to-end protocols such as RTCP (Realtime Transport Control Protocol) are used for QoS management. RTCP does end-to-end QoS control with exchanging SR (sender report) and RR (receiver report) packets. SR and RR provide end-to-end QoS parameters such as Round Trip Time (RTT) and Packet Loss Ratio (PLR). Those values are used for server driven QoS control. For example, if PLR becomes high, the sender may lower FEC code ratio (k/n) to transmit more repair packets for recovering more lost packets. RTCP has been used in wired network for long time but it has drawbacks when used in wireless networks. Feedback period is more or less longer comparing with dynamics of wireless channels. In addition RTT is not appropriate to calculate one-way delay in asymmetric channel. Also, important QoS parameters such as available bitrate and buffer fullness are roughly estimated from the measured RTT and PLR. Therefore the measured values may not be accurate.

Compared to RTCP, CLI provides more accurate and more prompt QoS information. Even though it does not provide end-to-end QoS information, CLI provides channel information about the time varying last mile channel which is the most dominant on QoS control in wireless networks. This information can be used for both server-driven and client-driven QoS controls. In server-driven control, channel parameters detected in the client are sent to the server. The server takes QoS action based on the received channel parameters in CLI and media parameters written in ADC. In client-driven control, media parameters written in ADC of MMT are sent to the client. The client takes QoS action based on the received media parameters and channel parameters detected through CLI. The QoS action may include decision of MMT Assets to be transmitted and AL-FEC code ratio against packet loss.

B.3 MMT QoS management per-class QoS control

Per-class QoS control is a priority-based tool, where every MMT packet has priority identifier in terms of its significance. When congestion happens in the middle of network, packets having low priority are dropped first. This operation is similar with dropping packets in DSCP (DiffServ Code Point)-aware routers.

MMT defines the priority of an MMT Asset in QoS descriptor in ADC. The entities in the middle of network such as MANE (Media Aware Network Element), intelligent CDN, and applications of cloud computing may exploit this priority information for it is packet transmission scheduling operation.

B.4 MMT QoS management per-flow QoS control

Per-flow QoS control is a reservation based tool. Certain amount of resource is guaranteed in per-flow QoS control while all resources are shared in best effort network and per-class QoS control. For each MMT Asset, network resources along the path are reserved for the entire delivery time. ADC includes information about amount of required resources. ADC parameters in MMT are compatible to those defined in RSVP protocol in IETF [1] and RSVP/IntServ in 3GPP [2].

The flow identifier in the MMT packet header is used for the identification of MMT Asset having the reserved resource. It is noted that CLI is not used in per-flow QoS control because the resource is guaranteed for that MMT Asset.

Table B. 1 — QoS management for MMT

Type	ADC	MMT packet QoS identifiers	CLI
Best -effort	Used for adaptation according to current channel condition reported through CLI	Not used	Measure time-varying channel parameters.
per-Class QoS	Used for adaptation based current channel condition reported through CLI	Priority identifier is used in the sending entity such as MANE for selective forwarding	Measure time-varying channel parameters.
per-Flow QoS	Used for resource reservation during call setup	Flow identifier is used for the sending entity such as MANE to identify packets for which reserved resources are guaranteed.	Not used because it is a guaranteed service.

B.5 MMT QoS management with ARQ in best effort network

Figure 23 illustrates the ARQ management process. The 1st step is the generation of ARQ configuration information message by MMT sending entity then it is delivered to the receiving entity. MMT receiving entity stores the ARQ configuration information. MMT receiving entity continues with receiving MMT media packets and checks whether there is a lost packet or not. A timeout parameter, profile indicator for the feedback mechanism and profile indicator for the retransmission mechanism are included in the ARQ configuration message. From the timeout parameter the MMT receiving entity will be able to determine if a packet is indeed lost. Once MMT receiving entity determines that a packet has been lost a feedback message is generated according to the defined in the ARQ configuration information. The feedback message is sent to MMT sending entity which will generate the retransmission message to be sent to MMT receiving entity. MMT receiving entity is able to substitute the lost MMT packet from the retransmission message.

One of well-known problems in ARQ happens if networks experience severe congestion. In this situation, the retransmission packet leads more congestion and causes the further network degradation. A possible solution is to re-transmit a packet having a higher priority. MMT makes this possible. With the information of current network condition and the priority of the lost packet, MMT can select and send an appropriate packet.

[1] IETF RFC 2205, “R esource ReSerVation Protocol (RSVP) Version 1 Functional Specification,” Sep. 1997.
[2] 3GPP TS 23.207, “End-to-end Quality of Service (QoS) concept and architecture,” 2004.

.

Annex C
(informative)

Examples of Hybrid delivery in MMT

C.1 Introduction

This Annex provides information on implementation of MMT in case MMT Assets are delivered on hybrid networks.

Hybrid delivery in this annex is defined as simultaneous delivery of one or more content components over more than one physically different type 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.

C.2 Classification of hybrid delivery

Basic concept of hybrid delivery is to combine media components on different channels. However, in detail, there are some types of hybrid delivery. The classification is shown in the following:

- Live and non-live
  - Combination of streaming components (Figure C. 1)
  - Combination of streaming component with pre-stored component (Figure C. 2)
- Presentation and decoding
  - Combination of components for synchronized presentation (Figure C. 1)
  - Combination of components for synchronized decoding (Figure C. 3)
- Same transport schemes and different transport schemes
  - Combination of MMT components
  - Combination of MMT component with another-format component1 such as MPEG-2 TS

[1] In this sentence, the case in which another-format component is encapsulated into MMT-format is excluded.

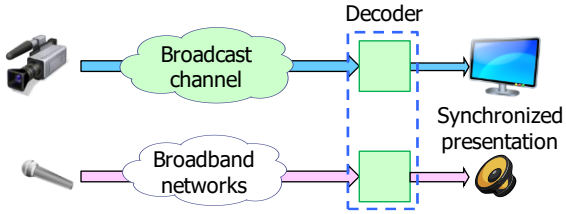


Figure C. 1 — Combination of streaming components for presentation

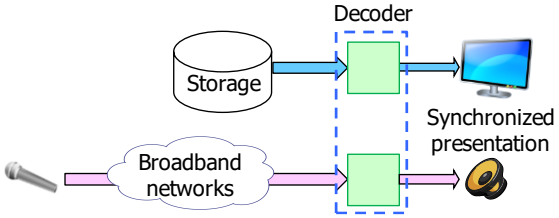


Figure C. 2 — Combination of streaming component with pre-stored component for presentation

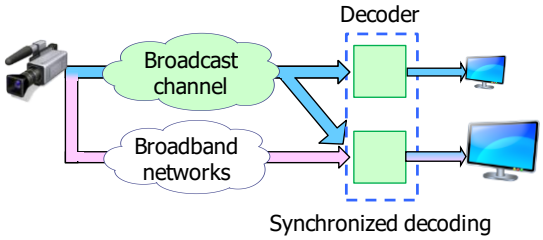


Figure C. 3 — Combination of components for decoding

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.

C.3 Technical elements for hybrid delivery

While a lot of information is specified in MMT Signaling Messages and MMT-CI, three types of information are mainly required for hybrid delivery.

— MMT Asset information[2]

— Information on spatial relationships among media components

— Signaling Message for Media Consumption

The following clauses detail the implementation of the information in individual cases categorized in clause C.2.

[2] When an MMT component is combined with another-format component, the latter component may be neither an Asset nor an MPU.

C.4 Detailed implementation

C.4.1 Use case : Combination of MMT and MPEG-2 TS for synchronized presentation

C.4.1.1 MMT Asset information

In order to identify the type and location of media components, the following signaling messages are required.

— `MMT_general_location_information` message

This message specifies the address of a media component. In the case of combination with MPEG-2 TS on broadcast channels, MEGP-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).

C.4.1.2 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_timestanp_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 and an MPEG-2 TS component are presented in synchronized manner since both components can share the same time domain as wall clock.

Annex D
(informative)

Usage of MMT AL-FEC

D.1 Usage of two stage FEC coding structure

D.1.1 Introduction

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 hands, when Asset for timed data such as AV streaming delivery service is multicasted (or broadcasted), 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.

D.1.2 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 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 which is described in sub-clause 6.5.2.1 in this specification.

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

D.1.3 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 multicasted (or broadcasted) to two different end-user groups who are under two different channel conditions each other, each Asset is packetized in MMT payloads and the packetized MMT payloads for the Assets are multiplexed on MMT packets to be 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 which is described in sub-clause 6.5.2.1 in this specification.

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 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.

Moving from User Group B to User Group A would be done without any service interruption while moving from User Group A to User Group B would require service interruption due to longer delay.

D.2 FEC Decoding Method For ssbg_mode2

D.2.1 Introduction

This Annex 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 Annex 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 Annex provides some examples of FEC decoding unit and a method to choose a proper unit.

D.2.2 source symbol block format for ssbg_mode2

In ssbg_mode2, source symbol block (SSB) usually consists of MMT packets of variable sizes. Figure A.1 presents an example of SSB for ssbg_mode2 which is built of 6 MMT packets having distinct sizes. More precisely, the 6 MMT packets and some padding data (e.g., all 00h) have been placed into the SSB. Note that any MMT packet shall be started at the first byte of an symbol element in SSB. The role of padding data may be regarded as adjusting the start point of MMT packets.

The columns of SSB in Figure D. 1 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***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, 4 in Figure D. 1. The concept of regions in an SSB will be used for recommended FEC decoding method, later..

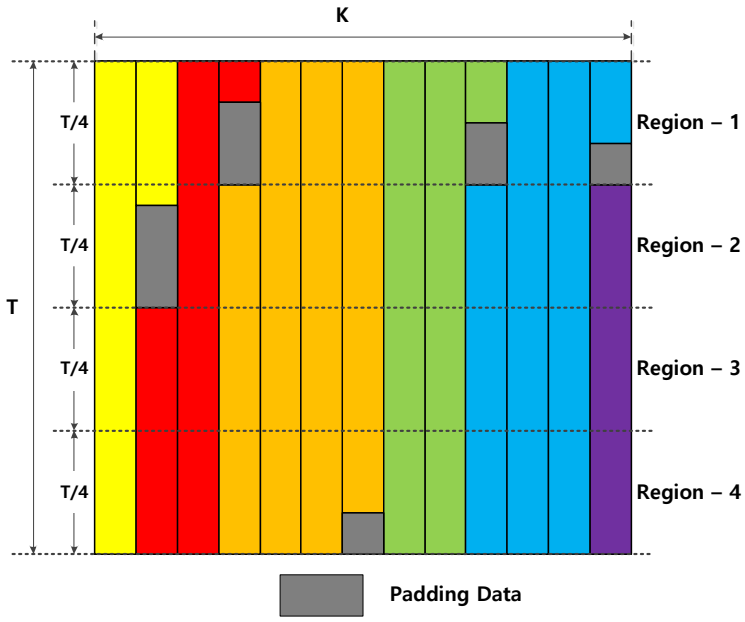


Figure D. 1 — Example of source symbol block

D.2.3 Regionalization of source symbol Block for FEC decoding

First, assume that the second and fifth MMT packets in Figure D. 1 are lost, i.e., two MMT packets are not received in the MMT receiving entity side. When an MMT 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 MMT packet and the amount of padding data, and so on. Therefore, the MMT receiving entity can rebuild SSB as depicted in **Error! Reference source not found.**. Note that source FEC payload ID provides information related to the start position of the MMT packet in SSB in terms of the symbol element for ssbg_mode2. For example, the start position of the second and fifth MMT packets in Figure D. 1 in terms of symbol elements may be 6 and 37. After rebuilding the SSB from received MMT packets, the MMT receiving entity carries out the FEC decoding process to recover the lost MMT packets.

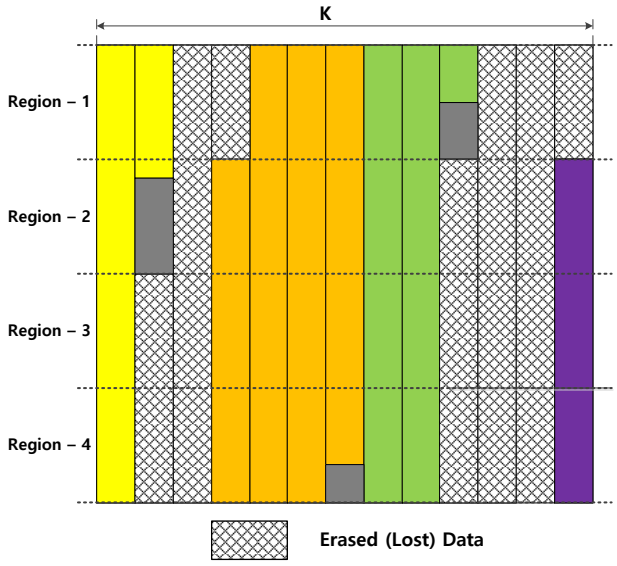


Figure D. 2 — SSB rebuilt when 2 MMT packets are lost

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, **Error! Reference source not found.** presents the FEC decoding method based on source symbol unit. To carry out the decoding process based on source symbol, SSB in **Error! Reference source not found.** should be interpreted into the SSB in **Error! Reference source not found.**. It is easily checked that any source symbol including lost MMT 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 MMT packets perfectly.

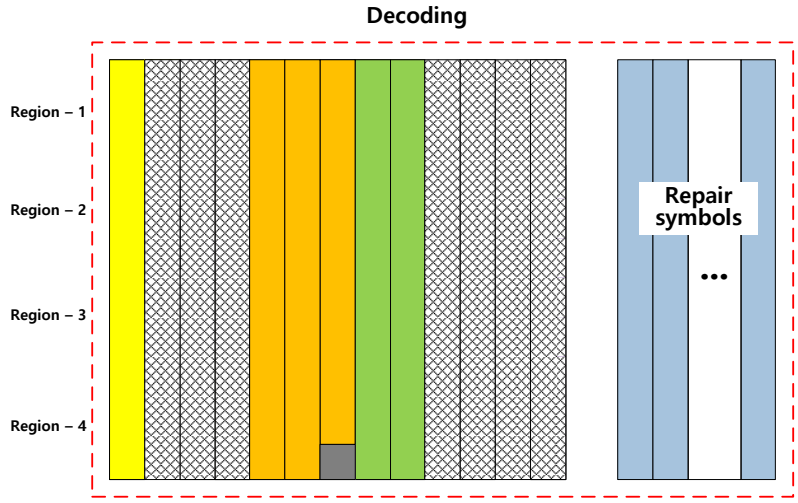


Figure D. 3 — FEC decoding based on source symbol unit

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, **Error! Reference source not found.** presents the FEC decoding method based on 2 symbol elements unit. Here 2-symbol element means a virtual unit for 2 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 MMT packets in Figure D. 1, the SSB should be divided into two regions as depicted in **Error! Reference source not found.**. One region consists of Regions-1 and -2, and the other consists of Regions-3 and -4. Next, any 2-symbol element including lost MMT packet is regarded as a lost 2-symbol element.

Consequently, one region and the other of SSB in **Error! Reference source not found.** have six and five lost 2-symbol elements, respectively. Therefore, at least six repair symbols are required to recover the lost MMT 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 2-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) shall 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.

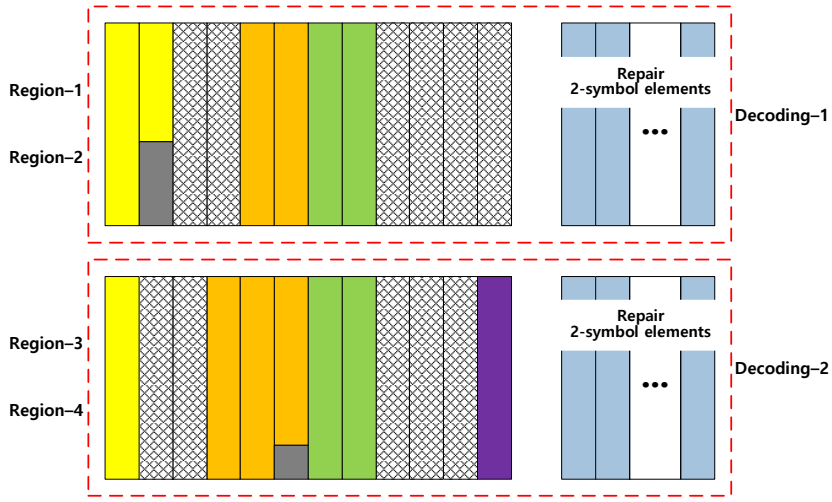


Figure D. 4 — FEC decoding based on multiple symbol elements unit

For the third example, **Error! Reference source not found.** presents the FEC decoding based on symbol element unit. In this case, the SSB should be divided into four regions as depicted in **Error! Reference source not found.**. The four regions are the same as Regions-1, -2, -3, and -4 in **Error! Reference source not found.**. 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 MMT 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) shall 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 2-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.

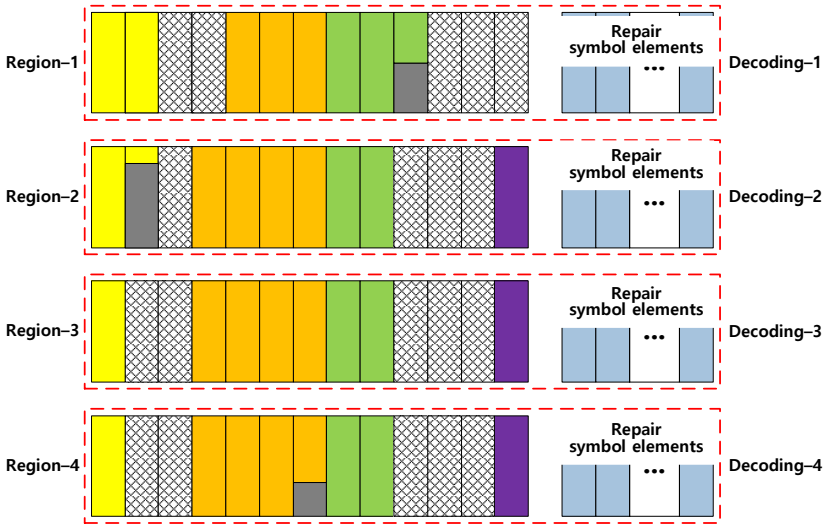


Figure D. 5 — FEC decoding based on symbol element unit

D.2.4 How to choose a proper unit of data for FEC decoding

As previous described, the number of erasures depends on the unit of data for FEC decoding, as depicted in **Error! Reference source not found.**. It is clear that the smaller unit is, the less erasures are induced. On the other hands, the larger unit is, the smaller decoding complexity is induced. Therefore, it is recommended to choose symbol element for the best FEC performance and choose 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.

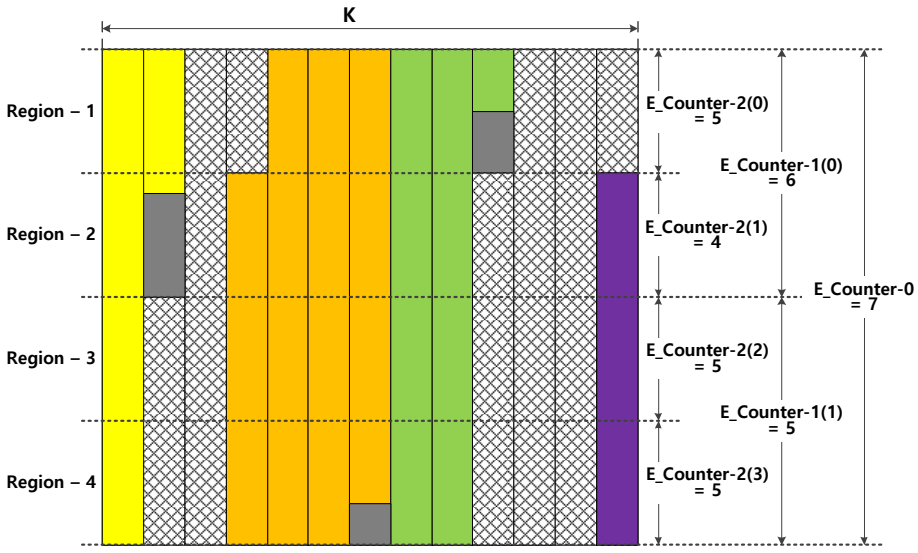


Figure D. 6 — Example of counting erasures for each FEC decoding unit

D.3 MPU mapping to source packet block

D.3.1 Introduction

An MMT FEC scheme can be applied to protect MMT assets. An FEC source flow is a flow of MMT packets delivering one or more MMT Assets. MPUs composing the MMT Assets are packetized into MMT Packets. The sequence of MMT 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 MMT 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 MMT 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.

D.3.2 Aligned MPU mapping method to source packet block

Firstly it will be assumed that an FEC source flow is a flow of MMT packets delivering a single Asset. For AL-FEC encoding of the MPUs for the Asset, the MPUs are packetized in MMT packets, and these MMT 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 MMP 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 MMT packets packetized from a single MPU of the Asset.

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

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

Figure D. 7 show examples for the three cases for “Aligned MPU mapping to source packet block (SFB)” in case of that FEC source flow consists of MMT packets (MMPT) for a single Asset. In this figure “Case 1 Ex.” is a example for Case 1, “Case 2 Ex.” for Case 2 with N = 2 and “Case 3 Ex.” for Case 3 with M = 2.

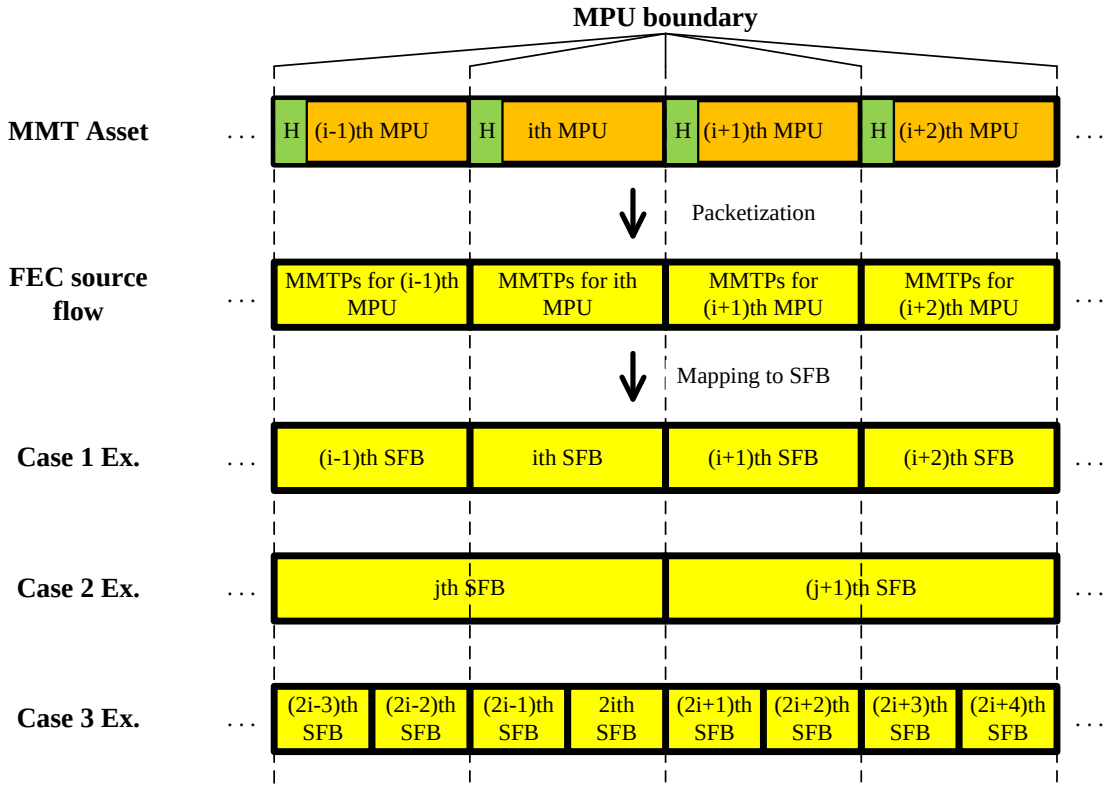


Figure D. 7 — Examples for MPU mapping to source packet block (SFB)

The aligned MPU mapping to source packet block can be easily extended to the cases where an FEC source flow is a flow of MMT 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. Let us refer to this kind of set of MPUs 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 MMP 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 MMT packets packetized from a single GMPU of the Assets.

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

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

D.4 Usage of LA-FEC

D.4.1 Introduction

This clause describes potential use cases for the LA-FEC coding structure as specified in section 6.5.3.3.

D.4.2 Use Case 1: Layered Multicast Streaming

Layered video coding such as e.g. Scalable Video Coding (SVC) or the scalable extension of HEVC (SHVC) can provide multi resolution video communication. An example is given in Figure D. 8 by a 4K/2K layered multicast streaming that supports sites that have no 4K capture/projection devices. Note that other resolutions such as e.g. VGA/QVGA can be used. Note that any video format can be used for 4K/2K layered video streaming, it allows 4K video sources to be utilized even in 2K-only environments. As shown in Figure D. 8, the 4K video from the input (4K live camera) is divided into the Enhancement Layer (EL1) consists of 4K sub-band data and the Base layer (BL) consist of 2K sub-band data. The former are passed to an IP multicast group that lies within the IP multicast group of the latter. At the decoder side, the decoder joins and receives multicast groups depending on what resolution is required by the user. The LA-FEC allows the FEC encoded flow of EL1 to be used in BL; this increases the total block length and raises the probability of recovering the base layer.

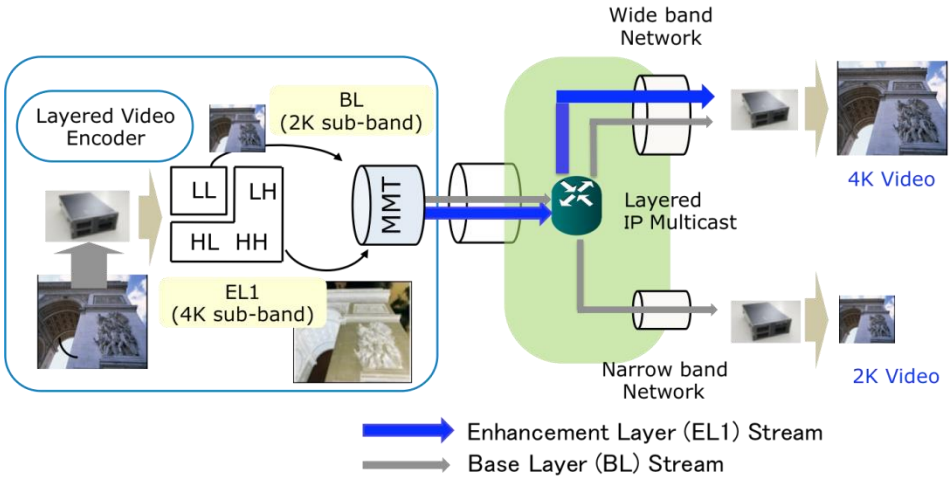


Figure D. 8 — 4K/2K Layered Multicast Streaming.

Figure D. 9 shows a multi-point 4K and 2K teleconference application using the 4K/2K layered multicast streaming. This is a three site tele-conferencing example; two sites communicate by 4K video while the remaining site uses a 2K system. Using layered multicast streaming as explained above, selection of decoding 2K or 4K is simply defined by the multicast configuration at IP routers in the networks, without any redundancy even though 4K and 2K were transmitted simultaneously. The LA-FEC can efficiently recover erasure packets in this situation.

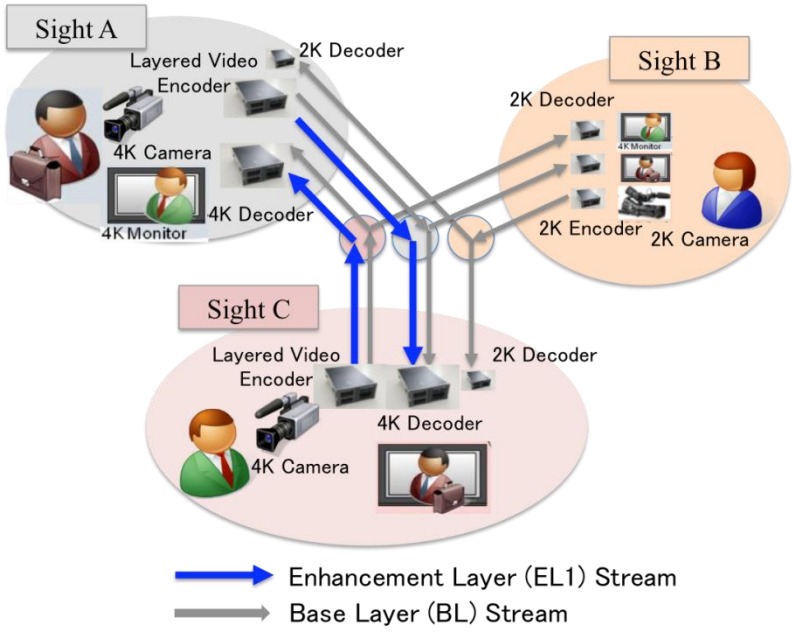


Figure D. 9 — Multi-point 4K/2K teleconference.

The LA-FEC also supports two or more layers. For example, 8K, 4K and 2K layered multicast streaming can also be realized. Furthermore, the LA-FEC cannot only apply to the multi-resolution layered multicast streaming but also to other ones. For example, by using the layered video coding which supports SNR scalability, video streams can be separated into layers by distributing qualitative (SNR) factor.

D.4.3 Use Case2: Hybrid delivery

The scenario in use case1 can also be applied to a hybrid delivery scenario, where a base layer (BL) is delivered over one network link (e.g. cellular network) and the enhancement layer (EL1) over a second network link (e.g. WiFi). Figure D. 10 illustrates such a hybrid delivery scenario with two exemplary MMT receiving entities, one (MMT receiving entity 2) receiving the BL only (e.g. 2K video) and the other (MMT receiving entity 1) receiving a higher video quality with BL and EL1 (e.g. 4k video).

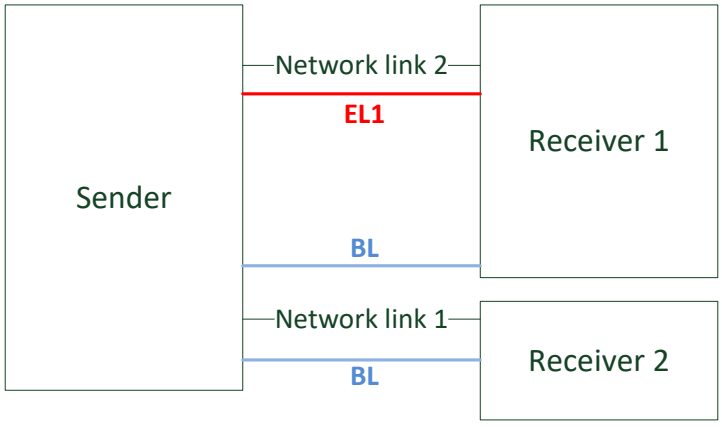


Figure D. 10 — Network scenario hybrid delivery

In such a scenario, LA-FEC allows the correction of non-correctable source packet block in network link 1 in situation where at the same time instance data of network link 2 is well received.

D.4.4 Use Case3: Fast zapping with long time interleaving (LA-FEC UI)

This clause describes a possible sending arrangement that allows MMT receiving entities robustness against long burst errors while at the same time having fast tune-in to the service. The described sending arrangement assumes that LA-FEC is activated, i.e. fec_coding_structure=0011. The scheme is illustrated in Figure D. 11 by the example of a single MMT multiplex.

The example in Figure D. 11 shows two FEC encoded flows (FEF), with FEF1 comprising the data a base layer and FEF2 comprising data related to an enhancement layer. The two colors denote packets belonging to the same source packet block, with blue blocks belonging to data succeeding the green blocks in presentation order. The empty boxes denote other data, e.g. preceding or succeeding data.

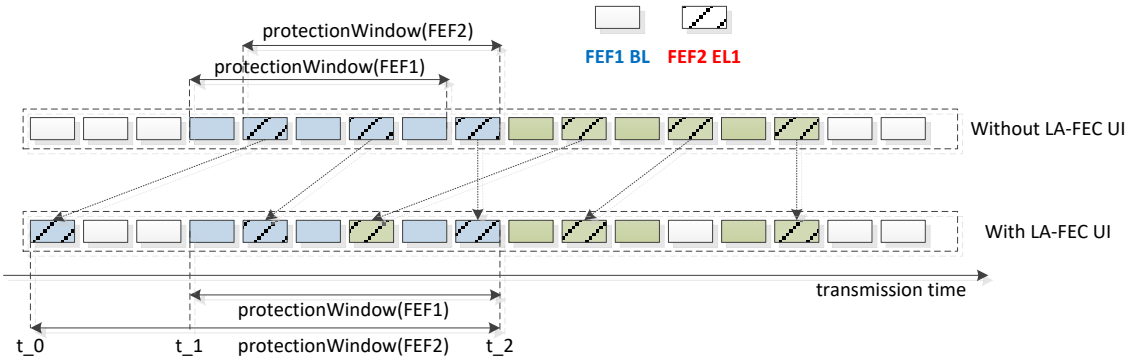


Figure D. 11 — Sending arrangement of an MMT multiplex with and without long time interleaving with fast tune-in by LA-FEC UI

Figure D. 11 shows a sending arrangement with unequal time interleaving that follows the settings defined for fast zapping solution with LA-FEC given in sub-clause 6.5.2.3. Note: point to definition of fast zapping solution with LA-FEC UI>>. With LA-FEC UI, the protectionWindow(FEF2) is increased compared to protectionWindow(FEF1) by spreading the data of FEF2 over a longer time period and interleaving it with other data. The reception of the last packet of FEF1 and FEF2 happens at a similar time instance, in the example at time instance t_2. The protectionWindows(FEF1) is increased to cover the reception of all packets of FEF1 and FEF2 succeeding the first packet of FEF1.

With the described solution, the MMT receiving entity that tunes at a time into the MMT stream can start decoding the FEF1 after a time defined by protectionWindow(FEF1). After a transition time, which depends on the difference between the protection windows, the MMT receiving entity can start decoding the FEF2 stream. After starting processing FEF2, the robustness of the FEF1 stream with protectionWindow(FEF1) is increased by the longer protectionWindow(FEF2) due to the LA-FEC protection. Since the data of FEF2 is transmitted ahead of the data of FEF1, the data of FEF1 needs to be buffered on the transmitter. This buffering period depends on the difference between both protectionWindows.

For fast zapping solution with LA-FEC UI, protectionWindowTime of protected flow of base representation is smaller than protectionWindowTime of protected flow of enhancement representation.

D.4.5 Use Case 4: Prioritized transmission

A typical deployment scenario for layered media codecs is to support different devices capabilities in a broadcast/multicast scenario. This can be e.g. the support of 2D/3D devices by Multiview Video Coding (MVC) or different resolutions such as e.g. 2k/4k or 720p/1080p or QVGA/VGA by Scalable Video Coding (SVC). A layered transmission also allows providing different robustness for each layer by AL-FEC solutions, which is especially interesting to save bandwidth and keep a stable service in difficult reception conditions (Mobile TV).

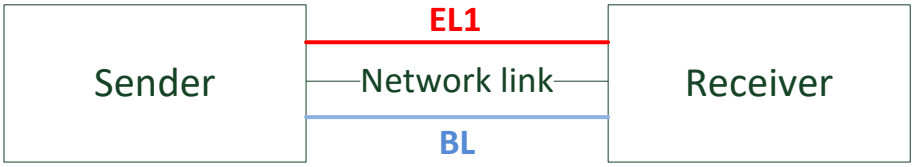


Figure D. 12 — Network scenario for prioritized transmission

With layered media coding, the enhancement representation (EL1) depends on the base representation (BL) due to inter-layer prediction. Therefore, the decoding probability of EL1 depends on the decoding probability of BL. More concretely, EL1 inherits the decoding probability of BL as its maximum decoding probability. With LA-FEC, the decoding probability of BL increases with the reception of EL1 due to the additional protection given by the protection of EL1. Therefore, the achievable maximum decoding probability of EL1 increases as well.

MMT defines an interface for exchanging Cross Layer Information between the application layer and the underlying network layer. The interface allows for top-down as well as bottom-up flow of cross layer information. The Cross Layer Information provides QoS information that may be used by the involved functions to optimize the overall delivery of the media data. MMT entities may support this interface for Cross Layer Information.

Annex E
(informative)
AL-FEC Code Specification

E.1 Introduction

This annex specifies available FEC codes for MMT AL-FEC framework. However, implementation does not require any of codes listed in this annex to be compliant to AL-FEC framework.

E.2 FEC Code Specification

E.2.1 Introduction

This annex specifies five kinds of FEC code algorithms to generate repair symbol block from source symbol block.

Table E-1 shows FEC code algorithms and its code point. Detailed specifications for FEC code algorithms are described through section E.2.2 ~ E.2.6.

Table E. 1 —FEC Code Algorithms and Its Code Point

FEC Code Algorithm	Code Point
RS code	01h
S_LDPC code	02h
RaptorQ LA code	03h
LDGM code	04h
FEC code algorithm in SMPTE 2022-1	05h

If one of FEC code algorithms, which are specified in this annex, is used for MMT AL-FEC framework, then its code point is set to `fec_code_id_for_repair_flow` field in sub-clause 8.4.3 AL-FEC message under following usages.

Table E. 2 — Usage of FEC codes

FEC Code Algorithm	Usage
RS code	K ≤ 200
S_LDPC code	K > 200
RaptorQ LA code	RaptorQ for Single Layer FEC

	RaptorQ LA for LA-FEC
LDGM code	**LDGM for Single Layer FEC**
	LA-LDGM for LA-FEC
FEC code algorithm in SMPTE 2022-1	**Contribution network**
	IPTV (Possible)

E.2.2 Specification for Reed-Solomon Codes

E.2.2.1 Introduction

In this specification, we use the following notations.

- K: number of source symbols in a source symbol block
- P: number of repair symbols in a repair symbol block
- $\boldsymbol{G} = [\boldsymbol{I}; \boldsymbol{A}]$: a systematic generator matrix for $[K+P, K]$-RS code where $\boldsymbol{I}$ is the identity matrix of order K and $\boldsymbol{A}$ is a $K \times P$ matrix.

A (N, K) Reed–Solomon code is a linear block code of length N (over Galois Field F) with dimension K and minimum Hamming distance $N - K + 1$. The Reed–Solomon code is optimal in the sense that the minimum distance has the maximum value possible for a linear code of size (N, K); this is known as the Singleton bound. Such a code is also called a maximum distance separable (MDS) code.

The section 8 of IETF RFC5510 [1] gives full specification of the RS code for the erasure channel. The generate matrix $\boldsymbol{G}$ perfectly characterizes the RS code. In this specification, we only consider the case when $m = 8$ (over GF(2^8)) with the generator matrix given in the section E.2.2.2.

E.2.2.2 Generator matrix

The generator matrix $\boldsymbol{G}$ has the form $\boldsymbol{G} = [\boldsymbol{I}; \boldsymbol{A}]$ where $\boldsymbol{I}$ is an identity matrix of size K and $\boldsymbol{A}$ is a $K \times P$ matrix, $(K + P) \leq 255$. Let α be the root of the primitive polynomial of degree 8 in Section 8.1 of IETF RFC5510 [1]. For RS code considered in this specification, the matrix $\boldsymbol{A}$ is a Cauchy matrix which has entries

$$A_{i,j} = 1/(x_i + y_j) \text{ for } 0 \leq i < K \text{ and } 0 \leq j < P$$

where x_i and y_j are elements in GF(2^8) and are defined as:

$$x_i = \alpha^{254 - i}, \text{ and } y_j = \alpha^j.$$

Therefore, the matrix $\boldsymbol{A}$ is given by

$$\boldsymbol{A} = \begin{bmatrix} \frac{1}{x_0+y_0} & \frac{1}{x_0+y_1} & \cdots & \frac{1}{x_0+y_{P-2}} & \frac{1}{x_0+y_{P-1}} \\ \frac{1}{x_1+y_0} & \frac{1}{x_1+y_1} & \cdots & \frac{1}{x_1+y_{P-2}} & \frac{1}{x_1+y_{P-1}} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \frac{1}{x_{K-2}+y_0} & \frac{1}{x_{K-2}+y_1} & \cdots & \frac{1}{x_{K-2}+y_{P-2}} & \frac{1}{x_{K-2}+y_{P-1}} \\ \frac{1}{x_{K-1}+y_0} & \frac{1}{x_{K-1}+y_1} & \cdots & \frac{1}{x_{K-1}+y_{P-2}} & \frac{1}{x_{K-1}+y_{P-1}} \end{bmatrix}.$$

Note that any sub-matrix of the Cauchy matrix is invertible [2].

E.2.2.3 References

[1] J. Lacan, V. Roca, J. Peltotalo and S. Peltotalo, “Reed-Solomon Forward Error Correction (FEC) Schemes,” IETF RFC 5510, April 2009

[2] ISO/IEC 13818-2 | ITU-T Rec.H.262, Information technology – Generic coding of moving pictures and associated audio information: Video

E.2.3 Specification for Structured Low-Density Parity-Check (S-LDPC) Codes

E.2.3.1 Introduction

A Low-Density Parity-Check (LDPC) code is a linear block code defined by its parity-check matrix. In this specification, we use a special case of LDPC codes which have an efficient encoding algorithm and adopted as an FEC code in standardizations such as IEEE 802.16e and 801.11n.

In this document, we use the following notations.

- K: number of source symbols in a source symbol block
- P: number of repair symbols in a repair symbol block
- $S(i)$: the i-th source symbol in a source symbol block ($0 \le i < K$). It can be represented as a binary column vector of length $8T$ or a column vector of length T over GF(2^8)
- $R(i)$: the i-th repair symbols in a repair symbol block ($0 \le i < P$). It can be represented as a binary column vector of length $8T$ or a column vector of length T over GF(2^8)
- $\boldsymbol{H}$: a sparse parity-check matrix of an S-LDPC code.

E.2.3.2 Structured LDPC Codes

In this specification, we consider structured LDPC (S-LDPC) codes whose parity-check matrices consist of circulant permutation matrices or the zero matrix [3].

Let $\boldsymbol{Q}$ be the $L \times L$ permutation matrix given by

$$\boldsymbol{Q} = \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \\ 1 & 0 & 0 & \cdots & 0 \end{bmatrix}$$

Note that $\boldsymbol{Q}^i$ is just the circulant permutation matrix which shifts the identity matrix $\boldsymbol{I}$ to the right by i times for any integer i, $0 \le i < L$. For simple notation, $\boldsymbol{Q}^\infty$ denotes the zero matrix.

Let $\boldsymbol{H}$ be the P' by $K' + P'$ matrix defined by

$$\boldsymbol{H} = \begin{bmatrix} Q^{a_{0,0}} & Q^{a_{0,1}} & \cdots & Q^{a_{0,k+p-2}} & Q^{a_{0,k+p-1}} \\ Q^{a_{1,0}} & Q^{a_{1,1}} & \cdots & Q^{a_{1,k+p-2}} & Q^{a_{1,k+p-1}} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ Q^{a_{p-1,0}} & Q^{a_{p-1,1}} & \cdots & Q^{a_{p-1,k+p-2}} & Q^{a_{p-1,k+p-1}} \end{bmatrix}$$

where p and k are given by $p = P' / L$ and $k = K' / L$, respectively and $a_{i,j} \in \{0, 1, \ldots, L-1, \infty\}$. If the locations of 1's in the first row of the i-th row block

$$\boldsymbol{H}_i = \begin{bmatrix} Q^{a_{i,0}} & \cdots & Q^{a_{i,k+p-1}} \end{bmatrix}$$

are fixed, then the locations of other 1's in $\boldsymbol{H}_i$ are uniquely determined.

E.2.3.3 Creating Parity-Check Matrix

For efficient encoding, we restrict the parity part of $\boldsymbol{H}$ to an almost lower triangular matrix with additional constraints as follows [4]:

$$\boldsymbol{H} = \begin{bmatrix} \boldsymbol{H}_I & \boldsymbol{H}_P \end{bmatrix} = \begin{bmatrix} & \boldsymbol{Q} & \boldsymbol{I} & \boldsymbol{0} & \cdots & \boldsymbol{0} & \boldsymbol{0} \\ & \boldsymbol{0} & \boldsymbol{I} & \boldsymbol{I} & \cdots & \boldsymbol{0} & \boldsymbol{0} \\ & \vdots & \boldsymbol{0} & \boldsymbol{I} & \cdots & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{H}_I & \boldsymbol{I} & \vdots & \vdots & \cdots & \vdots & \vdots \\ & \vdots & \vdots & \vdots & \cdots & \boldsymbol{I} & \boldsymbol{0} \\ & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \cdots & \boldsymbol{I} & \boldsymbol{I} \\ & \boldsymbol{Q} & \boldsymbol{0} & \boldsymbol{0} & \cdots & \boldsymbol{0} & \boldsymbol{I} \end{bmatrix}$$

In the first column block of the $\boldsymbol{H}_P$, $\boldsymbol{I}$ is placed only at the ceil(p/2)-st row block , $\boldsymbol{H}_{\text{ceil}(p/2)-1}$, where ceil(x) is the smallest integer not less than x.

Let $\boldsymbol{B}$ be a mother matrix having 400 column blocks and 20 row blocks with L = 16, i.e. the matrix $\boldsymbol{B}$ has 6400 columns and 320 rows. Each column block and row block of $\boldsymbol{B}$ has exactly 7 and 140 circulant permutation matrices of size 16, respectively. The remaining part of $\boldsymbol{B}$ is filled with zero matrices of size 16. The i-th row block of $\boldsymbol{B}$ can be represented as a sequence of pairs ($t_{i,j}$, $e_{i,j}$) where $t_{i,j}$ is the index of column block corresponding to the j-th circulant permutation matrix and $e_{i,j}$ is its exponent. The $\boldsymbol{B}$ matrix can support various values of K and P with techniques called scaling down and row splitting. The resulting matrix used as $\boldsymbol{H}_I$ for encoding process.

In order to support short source symbol block efficiently, the matrix $\boldsymbol{B}$ is scaled down by a scaling factor S_1. The resulting matrix is composed of circulant permutation matrices and zero matrices of size $16/S_1$, i.e. it has $6400/S_1$ columns and $320/S_1$ rows. For given source symbol block length K, the scaling factor can be obtained as $S_1 = 2^a$ where a is the largest integer satisfying $K \le (400 \times 16) / 2^a$. Note that the new matrix can be represented as the sequence of pairs ($t_{i,j}$, $e_{i,j}$ mod ($16/S_1$)) since the size of circulant permutation matrices and zero matrices are reduced from 16 to $16/S_1$.

As mentioned above, the matrix $\boldsymbol{B}$ has $320/S_1$ rows after scaled down. It means that the number of repair symbols P is $320/S_1$ at maximum. To support larger values of P, we extend the matrix $\boldsymbol{B}$ by splitting its rows. In this process, each row block is splitted into S_2 row blocks. For given repair symbol block length P and the scaling factor S_1, the row splitting factor S_2 can be obtained as S_2 = ceil(P/($320/S_1$)).

The matrix $\boldsymbol{H_I}$ is obtained from the matrix $\boldsymbol{B}$ as follows. Let $\boldsymbol{B}_i = \{(t_{i,0}, e_{i,0}), (t_{i,1}, e_{i,1}),\ldots, (t_{i,139}, e_{i,139})\}$ be the ordered sequence of pairs $(t_{i,j}, e_{i,j})$ representing the i-th row block of $\boldsymbol{B}$. Let S_1 and S_2 be the scaling factor and the row splitting factor, respectively. They are determined uniquely with K and P. Then the $(S_2 \times i + j)$-th row block of $\boldsymbol{H_I}$ can be represented as follows:

$$\boldsymbol{T}_{(S2\times i)+j} = \{(t_{i,k}, e_{i,k} \bmod (16/S_1)) \mid k \bmod S_2 = (S_2 - 1 - j),\ 0 \le k < 140\}$$

Note that the matrix $\boldsymbol{H_I}$ has 400 column blocks and S_2×20 row blocks with $L = 16/S_1$, i.e., it has $(6400/S_1)$ columns and $(S_2\times320/S_1)$ rows.

Finally, the parity-check $\boldsymbol{H}$ is obtained by augmenting the matrix $\boldsymbol{H_P}$ with appropriate size to the matrix $\boldsymbol{H_I}$. The H has 400 + S_2×20 column blocks and S_2×20 row blocks with $L = 16/S_1$, i.e., it has $(6400 + S_2\times320)/S_1$ columns and $(S_2\times320/S_1)$ rows.

E.2.3.4 Encoding Algorithm

The encoding of S-LDPC code is performed based on the following $p'\times L'$ by $(k'+p')\times L'$ parity-check matrix:

$$\boldsymbol{H} = [\boldsymbol{H_I} \quad \boldsymbol{H_P}] = \begin{bmatrix} & \boldsymbol{Q} & \boldsymbol{I} & \boldsymbol{0} & \cdots & \boldsymbol{0} & \boldsymbol{0} \\ & \boldsymbol{0} & \boldsymbol{I} & \boldsymbol{I} & \cdots & \boldsymbol{0} & \boldsymbol{0} \\ & \vdots & \boldsymbol{0} & \boldsymbol{I} & \cdots & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{H_I} & \boldsymbol{I} & \vdots & \vdots & \cdots & \vdots & \vdots \\ & \vdots & \vdots & \vdots & \cdots & \boldsymbol{I} & \boldsymbol{0} \\ & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \cdots & \boldsymbol{I} & \boldsymbol{I} \\ & \boldsymbol{Q} & \boldsymbol{0} & \boldsymbol{0} & \cdots & \boldsymbol{0} & \boldsymbol{I} \end{bmatrix}$$

where $L' = 16/S_1$, $p' = S_2\times20$ and $k' = 400$. It has $P' = S_2\times320/S_1$ rows.

The $\boldsymbol{H}$ is divided into the form

$$\boldsymbol{H} = \begin{bmatrix} \boldsymbol{A} & \boldsymbol{B} & \boldsymbol{T} \\ \boldsymbol{C} & \boldsymbol{D} & \boldsymbol{E} \end{bmatrix}$$

where $\boldsymbol{A}$ is $(p'-1)\times L$ by $k'\times L$, $\boldsymbol{B}$ is $(p'-1)\times L$ by L, $\boldsymbol{T}$ is $(p'-1)\times L$ by $(p'-1)\times L$, $\boldsymbol{C}$ is L by $k'\times L$, $\boldsymbol{D} = \boldsymbol{Q}$ is L by L and $\boldsymbol{E}$ is L by $(p'-1)\times L$. Let $\boldsymbol{c} = (\boldsymbol{s}, \boldsymbol{r_1}, \boldsymbol{r_2})$ be a codeword specified by $\boldsymbol{H}$, that is $\boldsymbol{Hc}^T = \boldsymbol{0}^T$ where $\boldsymbol{s}$ is the systematic part, $\boldsymbol{p_1}$ and $\boldsymbol{p_2}$ are the parity parts which have length L and $(p'-1)\times L$, respectively. That is $\boldsymbol{s} = [S(0),\ldots, S(K'-1)]$, $\boldsymbol{r_1} = [R(0),\ldots, R(L-1)]$ and $\boldsymbol{r_2} = [R(L),\ldots, R(P'-1)]$. Note that $S(K)$, $S(K+1)$,…, $S(K'-1)$ denote $K'-K$ padded source symbols, which are set to all zero bits and not delivered. Furthermore, $R(P),\ldots, R(P'-1)$ denote the repair symbols to be punctured, i.e., their values calculated but not contained in the corresponding repair symbol block.

Then, the detailed operations for encoding of an S-LDPC code defined by $\boldsymbol{H}$ are as follows:

Step 1) Compute $\boldsymbol{As}^T$ and $\boldsymbol{Cs}^T$.

Step 2) Compute $\boldsymbol{ET}^{-1}\boldsymbol{As}^T = [\boldsymbol{I} \ldots \boldsymbol{I}]\ \boldsymbol{As}^T$

Step 3) Compute $\boldsymbol{r_1}^T$ by $\boldsymbol{r_1}^T = \boldsymbol{ET}^{-1}\boldsymbol{As}^T + \boldsymbol{Cs}^T$

Step 4) Compute $\boldsymbol{r_2}^T$ by $\boldsymbol{Tr_2}^T = \boldsymbol{As}^T + \boldsymbol{Br_1}^T$

E.2.3.5 Decoding Algorithm

S-LDPC codes are one family of LDPC codes. Therefore any decoding algorithm for conventional LDPC codes can be applied without any modification. Consideration on the decoding algorithm of LDPC codes can be found in [5].

E.2.3.6 Base matrix

The S-LDPC codes can be fully described by the base matrix $\boldsymbol{B}$ and algorithms to calculate the scaling factor and the row splitting factor. The i-th row block of $\boldsymbol{B}$ can be represented as a sequence of pairs $T_i = \{(t_{i,0}, e_{i,0}), (t_{i,1}, e_{i,1}),\ldots, (t_{i,139}, e_{i,139})\}$ where $t_{i,j}$ is the index of column block corresponding to the j-th circulant permutation matrix and $e_{i,j}$ is its exponent for $0 \le j < 140$.

T_0 = {(1,8), (2,8), (3,10), (5,12), (6,8), (7,12), (9,8), (12,4), (14,12), (15,0), (19,0), (20,9), (26,4), (34,8), (35,1), (38,0), (45,13), (46,0), (48,13), (56,9), (57,3), (62,1), (63,8), (71,12), (75,8), (77,3), (78,2), (82,13), (83,13), (85,9), (88,1), (90,15), (92,4), (93,12), (97,0), (99,15), (104,5), (107,14), (110,13), (111,15), (116,9), (117,7), (120,9), (121,8), (125,15), (127,14), (128,15), (129,9), (131,5), (134,12), (150,12), (152,13), (156,1), (158,9), (159,13), (161,7), (163,5), (164,4), (165,13), (168,11), (171,9), (172,12), (175,12), (177,13), (180,5), (185,9), (193,1), (194,8), (195,9), (199,3), (200,9), (202,9), (204,3), (207,13), (212,13), (213,1), (214,13), (217,3), (223,13), (224,14), (226,10), (227,5), (228,7), (232,5), (236,14), (240,7), (241,7), (245,9), (250,12), (251,5), (255,5), (260,5), (267,4), (268,4), (272,4), (273,4), (275,9), (276,6), (278,5), (284,8), (288,1), (289,6), (291,2), (292,4), (297,12), (299,4), (302,4), (309,5), (310,4), (311,3), (312,5), (326,0), (330,10), (334,9), (335,4), (337,1), (338,6), (340,14), (342,10), (343,1), (347,4), (349,9), (350,1), (351,4), (357,14), (361,8), (364,0), (365,12), (367,6), (369,2), (373,4), (375,12), (376,12), (377,0), (379,0), (383,0), (384,1), (388,4), (389,0), (391,3)}

T_1 = {(2,4), (5,0), (8,9), (10,8), (12,8), (13,4), (14,8), (17,1), (23,12), (24,13), (29,9), (30,12), (33,9), (37,4), (45,0), (46,12), (47,8), (56,4), (60,0), (65,13), (73,13), (77,13), (78,12), (81,13), (89,12), (94,5), (99,5), (100,9), (102,9), (107,4), (111,5), (112,13), (117,4), (125,0), (127,12), (128,5), (133,12), (134,1), (136,1), (137,0), (138,12), (141,12), (143,4), (155,0), (157,12), (158,9), (160,0), (161,0), (163,8), (169,9), (170,12), (174,8), (176,4), (177,4), (178,8), (180,8), (182,8), (186,12), (187,8), (188,8), (189,8), (191,8), (192,1), (196,0), (198,8), (199,8), (200,0), (202,1), (204,4), (207,8), (210,1), (214,4), (217,2), (221,9), (224,0), (226,12), (228,9), (233,1), (236,8), (239,0), (241,0), (246,1), (249,1), (251,5), (256,8), (257,9), (259,9), (263,0), (264,1), (266,0), (267,1), (270,5), (271,9), (280,8), (282,0), (285,12), (286,1), (291,0), (292,8), (295,5), (302,12), (305,12), (306,9), (308,5), (309,4), (311,1), (312,13), (315,13), (316,1), (321,12), (322,5), (323,1), (324,5), (327,13), (328,5), (338,1), (342,12), (343,5), (346,13), (347,5), (349,0), (356,13), (361,1), (363,5), (367,5), (369,5), (372,5), (373,13), (374,4), (376,1), (380,4), (382,1), (387,5), (389,13), (390,5), (392,5), (393,1), (394,8), (395,9), (397,5)}

T_2 = {(1,2), (2,14), (3,3), (5,3), (6,10), (12,11), (13,3), (15,0), (19,13), (20,13), (21,6), (22,11), (33,10), (36,13), (38,6), (39,5), (40,12), (43,0), (44,3), (46,15), (47,3), (48,13), (51,7), (53,14), (57,15), (58,8), (59,6), (61,11), (70,7), (71,10), (73,10), (74,14), (79,1), (85,7), (86,14), (88,5), (89,5), (90,2), (92,10), (95,6), (99,14), (103,4), (104,6), (105,6), (111,10), (115,8), (128,3), (130,12), (133,4), (136,2), (139,6), (142,11), (145,14), (155,0), (156,14), (160,7), (168,5), (171,10), (176,12), (182,12), (183,8), (185,12), (186,9), (191,6), (193,15), (199,10), (204,0), (205,8), (207,5), (213,1), (217,4), (224,0), (226,0), (228,0), (230,4), (233,0), (234,12), (236,1), (238,8), (239,1), (240,3), (246,13), (247,12), (248,1), (249,8), (250,4), (251,1), (253,9), (254,0), (258,5), (262,3), (264,7), (268,6), (270,4), (271,1), (274,8), (277,5), (280,2), (286,11), (287,1), (293,3), (294,12), (296,7), (297,5), (299,7), (301,7), (302,11), (304,11), (306,3), (308,11), (309,15), (314,15), (315,8), (316,14), (317,10), (322,10), (325,11), (326,8), (327,15), (331,11), (333,1), (334,5), (335,9), (345,0), (348,13), (351,9), (358,9), (360,1), (362,15), (363,3), (371,3), (375,9), (376,7), (378,2), (379,1), (387,1), (389,11), (394,13), (396,15), (399,13)}

T_3 = {(2,4), (3,15), (4,7), (7,13), (9,8), (14,6), (15,5), (18,3), (22,4), (29,12), (30,7), (32,4), (36,14), (40,5), (50,5), (53,1), (54,11), (55,8), (60,8), (64,15), (68,12), (70,5), (71,9), (75,14), (77,2), (81,5), (85,10), (90,8), (91,11), (95,9), (96,11), (100,8), (101,5), (103,0), (104,4), (105,15), (107,13), (108,13), (109,13), (110,5), (111,9), (113,9), (116,0), (121,12), (123,12), (124,13), (131,9), (132,11), (133,1), (136,12), (137,11), (140,9), (144,12), (145,9), (149,0), (152,6), (155,7), (159,5), (162,0), (164,12), (166,6), (167,1), (168,9), (171,12), (174,5), (176,0), (180,5), (181,6), (182,0), (183,10), (184,11), (188,1), (189,1), (190,9), (197,6), (199,15), (203,6), (209,11), (213,15), (215,6), (223,3), (232,0), (234,6), (235,2), (240,10), (244,2), (247,6), (248,0), (253,6), (254,2), (255,2), (256,0), (257,10), (260,7), (265,2), (266,4), (274,0), (278,0), (280,4), (281,2), (286,2), (288,10), (291,2), (292,10), (294,1), (295,4), (297,0), (299,15), (301,2), (304,2), (306,3), (314,7), (321,1), (322,11), (325,10), (327,0), (334,11), (336,6), (338,8), (346,3), (348,3), (351,9), (352,8), (353,1), (355,11), (364,10), (366,10), (369,15), (370,14), (371,15), (373,13), (374,6), (376,15), (377,14), (382,5), (386,3), (387,2), (389,7), (396,8), (399,10)}

T_4 = {(0,0), (1,5), (5,9), (16,9), (17,1), (18,0), (19,0), (21,1), (26,0), (28,9), (29,8), (31,1), (34,7), (36,1), (37,5), (39,0), (40,15), (44,13), (50,9), (55,0), (56,0), (58,7), (67,15), (72,11), (73,1), (74,11), (76,5), (78,3), (80,3), (81,2), (84,0), (87,8), (89,5), (93,7), (96,2), (99,7), (107,14), (108,2), (110,0), (114,3), (117,1), (118,10), (119,10), (120,15), (123,11), (125,11), (128,10), (132,0), (138,10), (140,3), (143,2), (147,2), (152,14), (155,13), (164,13), (167,6), (171,2), (173,11), (174,8), (175,10), (180,7), (181,4), (182,14), (188,10), (191,2), (195,10), (199,9), (200,3), (204,14), (206,15), (207,0), (208,12), (211,6), (212,7), (213,7), (217,0), (218,15), (222,4), (223,2), (228,12), (230,9), (231,12), (241,6), (242,14), (249,15), (253,6), (254,1), (255,8), (256,1), (257,6), (259,11), (261,7), (262,12), (264,7), (265,15), (268,8), (269,12), (275,15), (277,11), (278,15), (280,11), (285,3), (286,13), (289,11), (293,13), (295,0), (297,3), (298,4), (300,12), (307,11), (311,1), (315,1), (316,6), (318,5), (320,4), (324,5), (326,9), (338,8), (339,13), (342,13), (346,5), (347,13), (348,7), (356,4), (357,2), (358,13), (359,2), (360,5), (361,0), (363,1), (364,8), (365,0), (368,13), (371,3), (375,8), (385,1), (388,3), (389,0), (393,5), (398,11)}

T_5 = {(1,14), (6,9), (8,11), (12,3), (18,10), (19,14), (21,9), (24,15), (25,10), (31,14), (34,11), (35,10), (37,8), (38,6), (40,7), (41,9), (43,3), (44,4), (45,14), (49,7), (53,7), (59,5), (61,13), (65,0), (72,11), (76,10), (83,15), (84,5), (86,11), (88,12), (101,5), (105,14), (108,5), (109,6), (113,6), (114,14), (115,13), (117,3), (119,0), (121,8), (122,7), (124,4), (129,1), (131,14), (132,1), (135,9), (136,8), (137,10), (139,3), (141,8), (142,15), (144,0), (148,12), (151,7), (154,11), (159,12), (166,1), (169,15), (175,3), (176,0), (183,13), (184,1), (186,3), (193,13), (194,3), (195,4), (196,3), (197,4), (205,4), (209,15), (210,0), (214,5), (217,10), (218,14), (219,4), (222,3), (223,0), (225,5), (227,9), (228,9), (229,9), (232,5), (235,5), (236,3), (241,1), (243,8), (245,0), (247,4), (248,5), (250,0), (262,8), (263,8), (266,5), (267,1), (272,1), (274,10), (275,1), (278,15), (281,0), (282,8), (287,15), (288,14), (291,14), (294,10), (298,1), (300,11), (303,10), (308,2), (309,10), (313,8), (317,0), (319,0), (325,2), (327,1), (329,2), (332,4), (333,2), (335,2), (336,0), (338,11), (350,10), (351,4), (353,3), (354,5), (355,5), (356,10),

(359,6), (360,1), (362,15), (363,2), (366,2), (368,14), (373,6), (376,10), (378,12), (379,6), (383,1), (385,7), (388,14), (398,15)}

T_6 = {(4,8), (6,3), (7,10), (11,15), (13,11), (15,3), (16,11), (22,5), (27,3), (28,12), (31,12), (33,1), (34,13), (36,13), (38,13), (42,8), (43,12), (44,1), (54,14), (55,11), (57,12), (69,9), (70,1), (71,9), (73,14), (75,9), (76,8), (78,10), (80,8), (81,1), (84,9), (86,10), (87,13), (88,13), (93,0), (95,11), (101,0), (102,1), (103,1), (104,9), (107,1), (110,4), (111,1), (112,9), (114,9), (117,2), (119,1), (120,1), (122,8), (130,1), (131,1), (134,4), (135,10), (138,9), (139,15), (147,0), (149,3), (150,11), (153,2), (155,10), (165,13), (168,7), (170,11), (171,3), (173,8), (180,11), (185,3), (188,0), (193,10), (196,3), (198,1), (201,10), (203,11), (205,7), (207,14), (208,13), (211,4), (215,2), (216,7), (217,8), (220,3), (227,14), (229,5), (231,5), (233,14), (234,0), (237,6), (241,6), (242,6), (248,0), (258,6), (261,3), (262,14), (263,5), (264,2), (266,1), (268,12), (269,6), (273,14), (279,2), (283,14), (287,15), (288,0), (290,6), (292,2), (294,1), (295,10), (296,4), (300,4), (302,11), (308,12), (313,10), (317,6), (321,14), (325,1), (326,0), (332,10), (336,4), (341,10), (342,7), (343,7), (355,13), (356,1), (358,11), (365,7), (369,9), (370,3), (371,3), (377,2), (385,2), (387,4), (388,2), (390,7), (391,8), (392,5), (395,2), (396,1), (397,5), (398,7), (399,3)}

T_7 = {(0,15), (1,1), (3,6), (4,8), (10,9), (15,15), (17,10), (18,14), (20,9), (22,8), (25,14), (27,14), (31,4), (33,10), (36,3), (38,14), (41,10), (45,11), (48,0), (52,14), (55,10), (59,3), (62,12), (63,14), (64,3), (66,14), (69,0), (74,6), (77,5), (80,2), (82,4), (84,0), (87,6), (99,8), (103,2), (107,13), (108,10), (110,1), (118,12), (121,2), (122,7), (125,3), (129,6), (131,7), (132,7), (137,10), (138,6), (139,1), (142,15), (145,7), (146,3), (147,5), (148,11), (151,2), (153,9), (154,10), (157,0), (165,3), (168,0), (173,11), (175,15), (179,9), (183,8), (184,3), (185,1), (186,2), (187,2), (189,6), (194,9), (196,4), (203,11), (204,3), (205,11), (206,5), (209,1), (215,6), (219,1), (222,1), (224,0), (225,1), (226,1), (229,3), (231,7), (232,13), (234,0), (237,2), (238,2), (242,10), (243,11), (245,5), (247,1), (248,9), (250,2), (251,3), (253,0), (255,9), (256,4), (259,1), (262,9), (263,0), (267,3), (268,1), (269,0), (270,9), (271,8), (276,9), (281,9), (285,9), (286,10), (289,11), (290,8), (292,5), (293,10), (297,1), (306,0), (308,1), (309,1), (314,0), (317,3), (318,12), (320,13), (322,12), (324,12), (328,9), (336,12), (339,13), (340,7), (341,0), (346,15), (347,15), (348,11), (360,10), (367,0), (368,5), (372,11), (375,3), (378,6), (382,4), (390,6), (396,11)}

T_8 = {(2,8), (3,13), (8,11), (11,9), (16,14), (24,3), (25,1), (26,4), (28,12), (31,4), (33,5), (34,2), (39,4), (44,0), (50,3), (51,8), (52,14), (54,8), (56,7), (57,2), (58,8), (60,1), (63,12), (64,2), (66,3), (67,1), (68,12), (69,5), (70,7), (72,7), (80,9), (83,12), (90,9), (95,13), (97,3), (98,4), (99,12), (100,8), (101,2), (103,8), (105,2), (108,8), (109,5), (118,4), (119,5), (126,11), (127,5), (130,2), (133,0), (138,9), (140,4), (145,0), (147,1), (149,6), (151,6), (152,14), (156,1), (158,14), (161,4), (162,2), (163,1), (166,1), (169,1), (170,10), (177,2), (178,8), (179,0), (181,2), (186,4), (190,0), (193,3), (196,14), (201,2), (202,1), (203,0), (205,0), (207,11), (209,2), (211,2), (214,3), (227,3), (231,9), (233,9), (239,10), (241,2), (242,6), (243,8), (244,14), (245,13), (247,9), (251,0), (254,8), (257,0), (268,7), (276,10), (277,8), (282,10), (284,10), (285,3), (288,7), (292,4), (295,0), (304,12), (305,14), (307,13), (308,10), (310,7), (312,6), (314,12), (317,14), (318,13), (323,11), (324,11), (328,6), (329,10), (330,11), (335,11), (340,13), (344,14), (345,5), (346,7), (347,6), (350,13), (352,7), (354,3), (355,4), (357,3), (361,14), (365,5), (367,3), (370,12), (372,9), (376,4), (377,7), (380,12), (382,6), (386,2), (389,9), (392,0), (394,7)}

T_9 = {(1,2), (3,11), (4,1), (5,15), (9,7), (10,0), (20,2), (21,2), (25,8), (26,1), (32,6), (33,10), (37,4), (46,2), (53,7), (54,8), (58,3), (62,5), (68,0), (70,0), (72,0), (73,9), (74,3), (80,10), (81,2), (82,10), (84,13), (86,6), (89,0), (91,4), (92,5), (96,10), (97,5), (99,2), (102,13), (111,1), (115,5), (116,5), (118,4), (123,10), (132,6), (138,8), (139,2), (141,7), (143,7), (146,14), (149,2), (152,4), (154,4), (157,3), (158,1), (159,13), (162,0), (167,0), (170,6), (171,4), (172,4), (174,1), (175,1), (177,9), (178,14), (181,4), (184,12), (190,9), (195,8), (196,0), (198,12), (199,0), (200,0), (201,0), (202,12), (204,14), (207,3), (208,5), (216,15), (220,8), (226,0), (229,15), (230,12), (233,6), (235,8), (237,8), (246,15), (247,1), (248,7), (252,10), (255,12), (257,7), (258,3), (263,2), (264,14), (265,11), (267,2), (269,0), (270,8), (271,11), (272,8), (275,10), (279,6), (286,2), (293,0), (296,10), (298,14), (302,2), (307,7), (311,3), (313,15), (316,8), (319,14), (320,2), (330,10), (331,13), (332,8), (333,2), (335,6), (339,9), (340,2), (344,6), (345,10), (346,6), (350,10), (351,10), (362,14), (363,4), (366,14), (367,4), (368,1), (369,2), (373,0), (375,1), (377,3), (378,11), (380,0), (384,11), (387,4), (390,10), (392,3), (394,1), (395,0), (397,11)}

T_{10} = {(0,10), (3,2), (4,6), (6,11), (8,0), (9,10), (10,0), (11,4), (13,12), (16,14), (17,0), (18,2), (30,10), (31,6), (34,8), (38,2), (42,7), (43,0), (45,2), (49,0), (50,8), (51,12), (52,6), (53,6), (58,10), (60,2), (61,8), (62,6), (67,0), (69,8), (74,2), (79,2), (87,4), (88,0), (91,6), (92,8), (93,2), (97,0), (98,10), (101,14), (104,2), (108,10), (111,10), (112,2), (115,2), (116,10), (122,2), (124,14), (126,8), (129,8), (130,0), (133,10), (135,2), (143,0), (144,8), (145,8), (150,8), (151,10), (156,10), (157,0), (158,10), (159,10), (161,0), (165,8), (170,2), (171,8), (173,4), (174,2), (182,14), (184,2), (191,14), (192,14), (198,4), (204,0), (206,15), (209,2), (216,12), (219,6), (220,6), (222,2), (226,6), (228,2), (232,8), (237,2), (240,14), (244,12), (245,0), (252,14), (253,10), (255,8), (257,10), (258,14), (259,4), (262,8), (267,2), (273,14), (278,6), (279,8), (290,10), (292,2), (296,6), (299,10), (301,2), (303,14), (310,6), (315,0), (319,6), (320,6), (323,8), (324,2), (326,6), (330,4), (331,4), (332,0), (333,8), (336,8), (337,10), (339,0), (340,0), (343,4), (344,4), (349,0), (352,8), (358,8), (366,8), (367,14), (372,0), (376,4), (377,14), (378,12), (379,4), (381,10), (383,12), (388,0), (389,8), (391,8), (393,6), (394,0), (397,12), (399,4), (400,0)}

T_{11} = {(4,2), (6,8), (10,10), (13,10), (15,10), (16,2), (18,10), (20,10), (23,6), (30,0), (32,10), (37,2), (39,14), (42,14), (45,0), (48,12), (49,2), (51,7), (52,6), (57,11), (59,11), (61,1), (66,0), (73,3), (75,14), (78,11), (82,3), (83,2), (87,13), (90,10), (92,11), (93,11), (95,1), (96,15), (97,3), (98,5), (106,7), (108,0), (109,3), (113,4), (115,7), (116,1), (119,1), (126,7), (130,14), (131,3), (135,5), (138,1), (142,7), (144,5), (146,9), (147,11), (148,0), (149,1), (150,3), (151,3), (153,3), (154,3), (159,1), (164,9), (166,11), (167,3), (168,3), (169,7), (170,9), (178,3), (180,1), (181,5), (183,1), (186,5), (192,3), (193,3), (197,3), (198,5), (202,3), (203,13), (210,9), (212,9), (213,3), (215,1), (217,1), (218,15), (219,5), (220,3), (221,1), (225,1), (226,3), (227,6), (229,12), (230,8), (231,2), (233,2), (235,10), (238,2), (239,1), (244,6), (246,5), (249,11), (250,9), (256,1),

(262,7), (265,2), (275,2), (276,8), (279,14), (280,0), (283,8), (286,1), (291,0), (296,2), (300,2), (303,2), (309,0), (311,0), (312,0), (317,0), (318,12), (319,0), (323,0), (327,0), (329,0), (336,0), (345,8), (348,8), (354,8), (358,8), (359,12), (360,8), (364,0), (367,10), (371,4), (373,0), (374,0), (375,8), (388,1), (390,10), (391,10), (394,2), (397,0), (398,10)}

T_{12} = {(1,3), (7,3), (8,1), (11,13), (14,2), (16,1), (22,11), (23,0), (24,15), (27,8), (28,9), (30,11), (44,9), (45,7), (46,3), (48,7), (50,3), (59,1), (62,3), (63,3), (64,3), (65,7), (66,7), (70,2), (76,1), (80,3), (82,5), (83,7), (84,9), (85,11), (86,5), (94,11), (97,5), (98,9), (100,15), (101,1), (102,3), (109,0), (112,9), (113,9), (114,7), (117,5), (120,1), (121,1), (123,1), (124,3), (127,11), (128,1), (134,3), (140,9), (141,9), (142,5), (152,1), (160,5), (162,3), (163,9), (165,1), (167,11), (172,5), (176,1), (177,0), (178,1), (179,6), (180,2), (181,1), (185,0), (187,7), (189,0), (190,2), (192,15), (198,6), (202,8), (206,2), (208,10), (218,0), (219,2), (221,4), (224,3), (225,0), (227,6), (233,6), (239,3), (240,2), (241,0), (242,0), (243,6), (249,4), (250,4), (251,8), (252,8), (254,8), (256,0), (259,1), (260,1), (263,4), (265,0), (270,1), (272,0), (273,4), (277,2), (278,2), (284,4), (287,2), (290,2), (293,6), (301,10), (307,10), (311,2), (312,6), (313,12), (316,10), (321,10), (322,12), (323,12), (324,0), (325,0), (330,8), (331,2), (334,10), (335,10), (336,8), (337,10), (341,14), (344,10), (347,10), (349,6), (350,4), (353,8), (355,0), (358,2), (359,12), (360,1), (369,2), (380,4), (383,14), (384,11), (391,3), (392,2), (393,13), (397,3)}

T_{13} = {(5,3), (7,0), (9,1), (11,11), (12,3), (13,1), (15,1), (23,3), (24,3), (25,3), (27,1), (31,9), (33,9), (35,9), (36,9), (41,11), (42,3), (43,3), (47,1), (48,7), (49,11), (51,13), (52,1), (53,3), (58,5), (65,15), (69,13), (71,1), (81,3), (85,9), (86,9), (88,7), (94,11), (96,3), (97,2), (102,7), (104,15), (106,7), (113,10), (114,2), (115,3), (120,2), (122,1), (123,3), (125,5), (127,1), (129,1), (130,4), (140,3), (142,11), (144,1), (146,3), (147,5), (148,15), (152,3), (154,0), (156,2), (157,9), (160,10), (162,6), (165,12), (166,14), (173,8), (175,8), (176,2), (179,11), (183,6), (190,9), (192,10), (193,4), (195,7), (199,0), (205,2), (206,1), (208,6), (209,0), (210,10), (212,2), (213,8), (214,0), (216,10), (218,1), (221,10), (222,0), (223,8), (224,10), (229,10), (230,0), (231,2), (236,2), (243,9), (246,0), (249,13), (261,10), (266,4), (271,0), (273,6), (276,10), (281,6), (282,0), (283,0), (284,8), (289,12), (290,0), (293,2), (298,10), (300,0), (304,8), (305,8), (307,4), (312,0), (314,4), (315,0), (316,2), (318,10), (319,12), (320,8), (325,12), (329,0), (334,1), (338,3), (342,4), (347,9), (350,1), (351,2), (352,2), (353,0), (354,11), (361,9), (364,13), (365,11), (368,9), (371,3), (372,0), (380,3), (382,4), (384,0), (385,1), (386,1), (399,4)}

T_{14} = {(0,8), (6,0), (12,0), (19,2), (20,0), (25,8), (26,10), (29,0), (30,2), (32,0), (34,0), (40,8), (41,0), (47,0), (49,0), (51,2), (52,2), (54,8), (55,0), (58,2), (60,0), (63,0), (64,0), (66,3), (69,1), (72,8), (75,9), (77,0), (78,0), (79,5), (81,9), (82,12), (87,4), (90,1), (91,15), (92,11), (94,1), (95,0), (100,11), (105,10), (106,10), (109,11), (110,2), (112,15), (115,1), (118,9), (120,3), (124,13), (125,0), (126,15), (132,3), (133,11), (135,3), (140,11), (141,2), (143,9), (145,9), (146,11), (153,7), (155,1), (161,11), (163,7), (167,8), (172,1), (175,15), (178,1), (183,0), (184,8), (186,1), (187,11), (194,3), (197,11), (201,3), (206,3), (208,13), (211,3), (216,3), (219,1), (223,3), (225,11), (235,9), (237,9), (238,3), (240,3), (246,11), (252,9), (259,1), (264,3), (273,2), (274,9), (275,9), (276,5), (277,11), (280,9), (282,9), (287,1), (289,5), (293,9), (294,0), (298,2), (300,3), (301,1), (310,1), (313,0), (315,10), (316,12), (318,0), (319,12), (320,8), (321,10), (322,1), (327,1), (328,2), (330,0), (331,0), (334,2), (337,4), (339,0), (340,2), (359,1), (362,8), (365,5), (368,8), (370,10), (371,1), (372,2), (374,2), (379,1), (381,8), (382,8), (384,4), (386,4), (387,0), (388,4), (391,3), (393,0), (395,2), (396,2), (398,10), (399,0)}

T_{15} = {(2,2), (4,10), (8,8), (11,0), (13,0), (16,0), (21,8), (26,2), (27,10), (28,2), (29,0), (32,3), (36,0), (38,2), (39,11), (42,1), (51,2), (54,10), (56,0), (59,0), (61,0), (63,1), (65,3), (67,11), (68,0), (71,3), (72,3), (74,10), (79,1), (82,0), (84,8), (87,1), (88,3), (94,8), (102,1), (106,9), (112,11), (116,8), (118,0), (121,11), (124,3), (126,9), (127,1), (131,9), (132,0), (134,1), (135,1), (136,1), (139,1), (140,1), (142,1), (149,1), (150,1), (156,1), (157,1), (158,3), (159,1), (162,11), (163,1), (165,9), (166,2), (172,0), (173,1), (174,3), (179,1), (181,1), (184,1), (185,1), (189,1), (191,3), (194,9), (195,9), (197,0), (201,1), (203,3), (208,1), (209,8), (212,8), (214,3), (215,11), (218,11), (219,8), (221,1), (227,11), (230,3), (232,3), (234,3), (243,9), (244,11), (252,2), (253,1), (260,0), (261,11), (265,9), (271,11), (272,1), (273,1), (281,10), (282,2), (283,9), (285,8), (288,10), (289,8), (291,11), (294,2), (303,3), (306,10), (310,2), (317,2), (318,11), (319,9), (323,3), (329,1), (330,1), (333,2), (337,8), (338,2), (339,2), (341,10), (342,0), (343,2), (344,0), (345,0), (349,10), (350,0), (353,8), (356,0), (359,0), (362,3), (366,8), (370,0), (374,10), (377,0), (379,10), (381,8), (383,8), (384,0), (390,2), (391,8), (395,0)}

T_{16} = {(7,4), (11,5), (17,3), (19,6), (22,5), (27,6), (29,6), (30,7), (35,14), (42,6), (47,14), (54,4), (55,4), (56,2), (63,1), (64,6), (66,0), (67,2), (71,13), (72,0), (73,0), (75,12), (76,10), (77,4), (79,0), (80,0), (85,7), (89,6), (90,4), (93,6), (94,3), (96,5), (98,4), (100,0), (106,2), (113,14), (116,4), (118,2), (122,2), (126,12), (128,6), (133,6), (136,10), (137,14), (141,2), (144,14), (147,4), (148,2), (150,0), (151,0), (153,14), (155,2), (157,2), (162,2), (163,7), (164,2), (167,1), (169,3), (176,1), (182,12), (185,4), (187,3), (189,0), (190,1), (191,11), (197,1), (203,5), (210,12), (211,0), (212,0), (216,0), (218,3), (220,11), (224,4), (225,7), (228,10), (232,3), (235,8), (236,0), (237,1), (238,9), (244,7), (245,1), (252,0), (256,1), (258,3), (260,7), (261,1), (265,7), (266,1), (269,5), (270,1), (275,3), (276,2), (277,5), (279,0), (280,7), (284,7), (285,9), (288,1), (290,1), (295,7), (297,9), (298,1), (299,3), (300,3), (301,0), (302,1), (303,1), (304,3), (305,1), (308,1), (311,0), (313,1), (315,1), (321,5), (325,2), (326,5), (328,5), (331,10), (332,13), (334,1), (335,7), (337,5), (341,5), (344,4), (355,4), (356,5), (357,5), (358,4), (362,6), (366,5), (373,4), (374,4), (381,1), (385,1), (386,13), (395,4), (397,5), (398,6)}

T_{17} = {(0,1), (2,5), (7,2), (12,1), (14,5), (19,1), (21,5), (22,5), (23,7), (24,3), (27,2), (28,9), (32,1), (35,13), (37,1), (39,1), (40,1), (41,0), (43,1), (44,10), (46,1), (47,0), (53,5), (57,1), (62,11), (64,5), (65,5), (68,5), (77,7), (79,1), (83,2), (85,1), (86,4), (91,1), (92,3), (94,4), (95,4), (103,3), (104,0), (105,4), (106,8), (107,5), (109,7), (112,7), (113,2), (119,5), (120,7), (126,1), (135,1), (143,2), (144,4), (145,12), (146,13), (153,5), (154,5), (160,4), (164,6), (168,0), (169,2), (172,0), (177,13), (178,4), (179,7), (188,0), (190,4), (192,8), (194,8), (195,4), (196,0), (200,0), (201,5), (213,11), (215,4), (229,1), (230,4), (231,0), (235,0), (242,4), (243,0), (244,4), (246,6), (247,4), (249,4), (252,4), (253,4), (254,4), (258,0), (264,0), (266,4), (269,7), (271,0), (272,0), (274,4), (281,2), (283,0), (284,0), (287,6), (289,2), (298,7), (299,10), (304,1), (305,3), (306,4), (307,7), (309,6), (320,14), (321,1), (322,7), (327,7), (328,6), (329,2), (331,10), (333,6), (343,0), (345,3), (346,0), (351,1),

(352,11), (353,5), (354,2), (357,2), (359,1), (362,0), (363,3), (364,7), (365,6), (368,5), (369,5), (374,0), (378,5), (381,2), (382,7), (383,3), (384,12), (385,6), (386,0), (390,3), (393,3), (396,1), (399,1)}

T_{18} = {(0,2), (9,3), (10,6), (14,2), (17,6), (23,7), (24,7), (26,2), (28,5), (29,12), (35,5), (39,4), (41,5), (43,3), (47,7), (48,3), (49,4), (50,0), (60,2), (61,2), (62,4), (65,0), (66,5), (67,14), (68,2), (69,4), (70,4), (74,4), (75,6), (76,2), (79,2), (89,6), (91,4), (93,6), (98,7), (100,4), (103,4), (106,1), (114,0), (122,4), (123,6), (124,14), (125,14), (128,0), (129,4), (130,4), (134,6), (136,4), (137,4), (143,5), (146,5), (148,4), (150,5), (154,0), (160,4), (161,4), (164,0), (170,4), (173,0), (174,1), (177,4), (182,5), (187,1), (188,6), (191,0), (192,1), (200,7), (202,0), (210,3), (211,9), (215,1), (216,1), (220,0), (221,1), (222,5), (223,3), (225,0), (234,5), (238,0), (239,5), (254,0), (255,5), (257,1), (260,13), (261,7), (267,7), (268,1), (272,5), (274,3), (277,3), (278,5), (279,1), (281,1), (283,6), (285,3), (287,0), (290,1), (294,1), (296,1), (297,1), (303,1), (304,0), (305,5), (307,5), (310,0), (313,2), (314,1), (323,3), (326,1), (328,2), (329,0), (332,1), (337,0), (341,0), (342,3), (343,3), (344,1), (348,1), (349,1), (352,1), (353,2), (354,1), (356,9), (357,0), (360,1), (361,13), (363,1), (366,0), (370,5), (375,0), (378,5), (380,1), (381,4), (385,0), (386,5), (387,4), (392,6), (393,7), (396,5), (398,5)}

T_{19} = {(0,4), (5,12), (8,4), (9,8), (10,12), (14,8), (17,0), (18,10), (20,4), (21,9), (23,13), (25,5), (32,9), (35,12), (37,0), (40,12), (41,8), (42,0), (46,1), (49,1), (50,13), (52,12), (55,9), (56,0), (57,6), (59,14), (60,4), (61,10), (67,0), (68,0), (76,4), (78,6), (83,0), (89,6), (91,8), (96,0), (98,0), (101,8), (102,6), (105,1), (110,2), (114,5), (117,15), (119,1), (121,10), (123,5), (127,15), (129,0), (134,7), (137,9), (139,10), (141,1), (148,13), (149,4), (151,9), (153,4), (156,10), (158,11), (160,1), (161,11), (166,1), (169,12), (172,2), (179,11), (187,6), (188,9), (189,2), (194,5), (197,0), (198,1), (200,7), (201,13), (205,2), (206,13), (210,9), (211,1), (212,3), (214,11), (220,1), (221,2), (222,1), (234,1), (236,7), (237,8), (238,1), (239,3), (240,1), (242,7), (245,13), (248,13), (250,5), (251,7), (258,5), (259,3), (260,7), (261,14), (263,13), (269,5), (270,9), (274,0), (279,5), (282,6), (283,4), (284,0), (291,5), (295,3), (296,11), (299,5), (301,6), (302,2), (303,5), (305,4), (306,5), (310,9), (312,4), (314,13), (324,5), (332,12), (333,6), (339,7), (340,13), (341,14), (345,13), (348,12), (349,0), (352,12), (354,0), (355,1), (357,8), (361,0), (364,13), (370,13), (372,14), (379,12), (380,4), (381,11), (383,5), (392,10), (394,10), (395,8)}

E.2.3.7 References

[3] M. P. C. Fossorier, “Quasi-Cyclic Low-Density Parity-Check Codes”, in Proc. ISIT 2003, Yokohama, Japan, June/July 2003, p. 150.

[4] S. Myung, K. Yang, and J. Kim, "Quasi-cyclic LDPC codes for fast encoding," IEEE Transactions on Information Theory, vol. 51, pp. 2894-2901, Aug. 2005.

[5] V. Roca, C. Neumann and D. Furodet, “Low Density Parity Check (LDPC) Staircase and Triangle Forward Error Correction (FEC) Schemes,” IETF RFC 5170, June 2008

E.2.4 Specification of RaptorQ LA

E.2.4.1 Introduction

. If RaptorQ LA is used for `fec_coding_structure!=0011` or `(fec_coding_structure==0011` and source block is generated from a base layer`)` in sub-clause 8.4.3 AL-FEC message, then the RaptorQ encoding process defined in IETF RFC 6330 section 5 [6] is used to generate repair symbols from source symbol block.

If RaptorQ LA is used for (`fec_coding_structure==0011` and the source block is generated from an enhancement layer`)`, then E.2.4.1.1, E.2.4.1.2, E.2.4.1.3 and E.2.4.1.4 for the Layer-Aware RaptorQ encoding process is used to generate repair symbols from source symbol block

E.2.4.1.1 Definitions

This section is based on the RaptorQ specification in IETF RFC6330 **Error! Reference source not found.**[6] and uses the terminology and algorithms specified therein. It describes the extensions for LA-FEC for section “5.3.3 First Encoding Step: Intermediate Symbol Generation” and section “5.3.4 Second Encoding Step: Encoding” of IETF RFC6330 **Error! Reference source not found.**[6].

K_x is the number of source symbols in the source block of media layer x (cf. K in section 5.1.2 in IETF RFC 6330 **Error! Reference source not found.**[6]).

K'_x is the number of source symbols in the extended source block of media layer x (cf. K' in section 5.1.2 in IETF RFC 6330 **Error! Reference source not found.**[6]).

p_x is the number of parity symbols of media layer x.

L_x denotes the number of intermediate symbols for a single extended source block of media layer x (cf. L in section 5.1.2 in IETF RFC 6330 **Error! Reference source not found.**[6]).

C_x denotes an array of intermediate symbols, $C_x[0], ... C_x[L_x-1]$, of media layer x

Enc[K', C ,(d, a, b, d1, a1, b1)] denotes an encoding symbol generator (cf. section 5.3.5.3 in IETF RFC 6330 **Error! Reference source not found.**[6]).

ISI is the internal symbol ID (cf. section 5.3.1 in IETF RFC 6330 **Error! Reference source not found.**[6]).

ESI is the encoded symbol ID (cf. section 5.3.1 in IETF RFC 6330 **Error! Reference source not found.**[6]).

$G\_ENC_{m\_LA\_z}$ is the LA-FEC z^{th} extension matrix of the G_ENC matrix of media layer m.

Matrix A is the constraint matrix in calculation process of intermediate symbols in the first encoding step of the RaptorQ process (cf. Figure 5 in section 5.3.3 in IETF RFC 6330 **Error! Reference source not found.**[6]). Matrix A is illustrated in Figure X1.

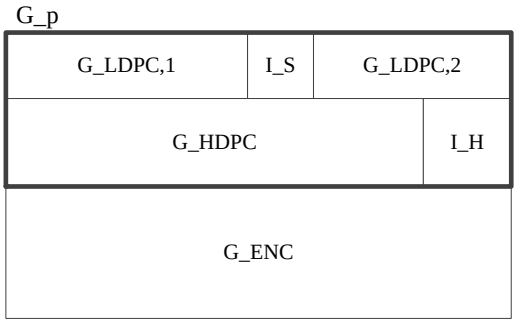


Figure X1: Constraint Matrix A (cf. Figure 5 in section 5.3.3 in IETF RFC 6330 Error! Reference source not found.**[6])**

It consists of six submatrices, 'G_LDPC,1', 'I_S', 'G_LDPC,2', 'G_HDPC', 'I_H' and 'G_ENC'. The construction of those submatrices is described in section 5.3.3 in IETF RFC 6330 **Error! Reference source not found.**[6].

We define the submatrix G_p as the submatrix which contains all submatrices for precoding.

$$G\_p = \begin{bmatrix} G\_LDPC,1 & I\_S & G\_LDPC,2 \\ & G\_HDPC & I\_H \end{bmatrix}$$

The matrix A consists of G_p and G_ENC:

$$A = \begin{bmatrix} G\_p \\ G\_ENC \end{bmatrix}$$

The intermediate symbols C can then be calculated from the source symbols D as:

C=(A^^-1)*D

E.2.4.1.2 Mapping of parameters from MMT

This section defines the mapping from the MMT payload ID to the definitions in 6.5.8.1.2 and in **Error! Reference source not found.**[6].

K_x is equal to SB_length[x] of layer x

In case of source symbols of layer x:

ESI_x = SS_ID_x - SS_Start

In case of a repair symbol of source block of layer x:

ESI_x = RP_ID_x+ K_x – SS_Start -1

D1 = First K_1 source symbols of the source block

Dx = K_x source symbols of the source block succeeding the source symbols of layer x with x!=1

E.2.4.1.3 Layer-Aware RaptorQ First Encoding Step – Intermediate symbol generation

This clause specifies the intermediate symbol generation for Layer-Aware RaptorQ for a given media layer l=2 that depends on media layer l=1. Media layer l=1 represents the base layer with dependency_id = 0 and media layers l=2 represents the enhancement layer with dependency_id = 1 as shown in Figure X5.

<<edt. Note: Figure X5 points to the figure in the LA-FEC section>>

With Layer-Aware RaptorQ, matrix A is extended for integration of all media layers in the constraint matrix A in way to preserve the systematic behavior of the code. Matrix A_{LA_2} for the Layer-Aware RaptorQ process for media layer l=2 is shown in Figure X2.

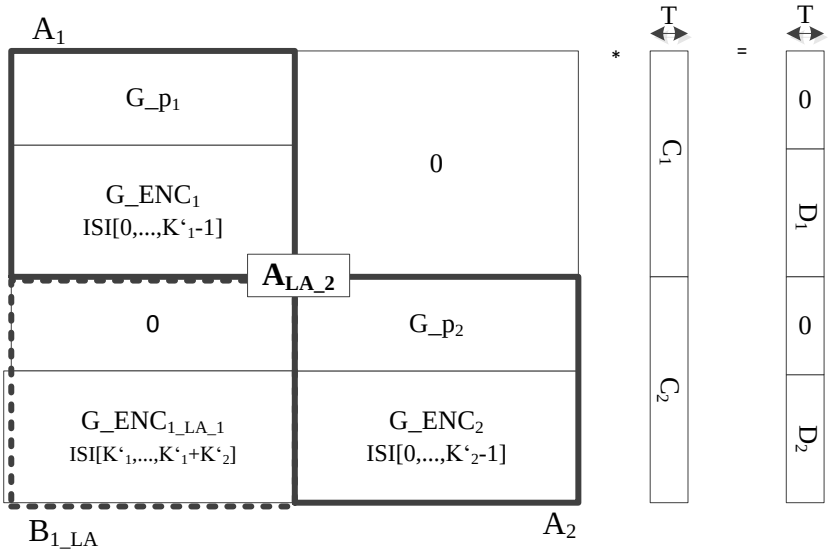


Figure X2: Encoding process with layer-Aware RaptorQ constraint matrix A_{LA_2} for l=2 media layers

In the given example in Figure X2, matrix A consists of the matrices A_1, A_2, the LA-FEC extension matrix B_{1_LA} and a zero matrix.

A_1 and A_2 are generated as described in section 5.3.3 in IETF RFC 6330 **Error! Reference source not found.**[6] with the constraint that the symbol size of all source and intermediate symbols T is equal in all cases which may result in different number of source symbols k_1, k_2. The matrix with '0' denote a matrix with only zeros in its matrix.

The LA-FEC extension matrix B_{1_LA} comprises the continuation matrix $G\_ENC_{1\_LA\_1}$ and a zero matrix. $G\_ENC_{1\_LA\_1}$ is the continuation of the G_ENC_1 matrix of media layer l=1, which is part of the A_1 matrix. The continuation matrix $G\_ENC_{1\_LA\_1}$ is generated in a way that each row is unique in G_ENC_1 and $G\_ENC_{1\_LA\_1}$. It is generated in the same way as G_ENC_1 by function Enc, which is described in section 5.3.3.4.2 in IETF RFC 6330 **Error! Reference source not found.**[6]. While G_ENC_1 is generated from ISIs[0: K'_1-1], $G\_ENC_{1\_LA\_1}$ is generated from ISIs[K'_1,…,K'_1+K'_2-1].

We describe the LA-FEC continuation matrix as $B_{1\_LA\_1}$=[0; $G\_ENC_{1\_LA\_1}$], the matrix A_{LA_2} of Layer-Aware RaptorQ is described for media layer l=2 by:

$$A_{LA\_2}=\begin{bmatrix} G\_p_1 & 0 \\ G\_ENC_1 & 0 \\ 0 & G\_p_2 \\ G\_ENC_{1\ LA\ 1} & G\_ENC_2 \end{bmatrix}=\begin{bmatrix} A_1 & 0 \\ B_{1\_LA\_1} & A_2 \end{bmatrix}$$

The intermediate symbols C_1 and C_2 can then be calculated from source symbols D_1 and D_2 as:

$$[C_1C_2] = (A_{LA\_2}\text{^^-1})*[D_1D_2]$$

This concept can further be extended for the n^{th} media layer as shown in matrix A_{LA_n}:

$$A_{LA\_n}=\begin{bmatrix} A_1 & 0 & \cdots & 0 \\ B_{1\_LA\_1} & A_2 & & 0 \\ \vdots & & \ddots & \vdots \\ B_{1\_LA\_n-1} & B_{2\_LA\_n-2} & \cdots & A_n \end{bmatrix}$$

with $B_{1\_LA\_n-1}$ being the n^{th}-1 continuation matrix of G_ENC_1 and $B_{2\_LA\_n-2}$ being the n^{th}-2 continuation matrix of G_ENC_2. With $G\_ENC_{1\_LA\_n-1}$ matrix in $B_{1\_LA\_n-1}$ being generated from ISIs$[\sum_{x=1}^{n} K'_x : \sum_{x=1}^{n} K'_x + K'_n - 1]$ and $G\_ENC_{2\_LA\_n-2}$ matrix in $B_{2\_LA\_n-2}$ being generated from ISIs$[\sum_{x=2}^{n} K'_x : \sum_{x=2}^{n} K'_x + K'_n - 1]$.

The intermediate symbols C_1,C_2,…,C_n can then be calculated from source symbols D_1 and D_2 as:

$$[C_1C_2...C_n] = (A_{LA\_n}\text{^^-1})*[D_1D_2…D_n]$$

For solving the equation an efficient decoding process as described by the example FEC Decoder in section 5.4 in IETF RFC 6330 **Error! Reference source not found.**[6] can be applied.

E.2.4.1.4 Layer-Aware RaptorQ Second Encoding Step

The second encoding steps generates the repair symbols E_x of media layer x from the intermediate symbols C.

In the following we consider the case with *n* media layers with the intermediate symbols C=[C_1C_2... C_n] and describe the encoding process for generation of repair symbols E_{l_target} of media layer l_target with l_target being the target layer for which repair symbols should be encoded.

The generation of the repair symbols *E* from intermediate symbols *C* is described in section 5.3.4 in IETF RFC 6330 **Error! Reference source not found.**[6]. With Layer-Aware RaptorQ, the repair symbols E_{l_target} are generated from all intermediate symbols C=[C_1C_2... C_{l_target}]. The process for generation of the non-systematic encoding symbols is described in the following pseudo code:

For ISI (i= $\sum_{n=1}^{l\_target} K'_n + \sum_{n=1}^{l\_target-1} p_n$; i< $\sum_{n=1}^{l\_target} K'_n + \sum_{n=1}^{l\_target} p_n$; i++)

E_{l_target} [i] = 0;

for layer 1:l_{target}

E_{l_target} [i]= E_{l_target} [i] XOR Enc(K'$_{layer}$, C_{layer}[0], C_{layer}[1],…,C_{layer}[L_{layer}-1], (d,a,b,d1,a1,b1))

end

end

This process is illustrated for generation of encoding symbols E_1 of layer l=1 and E_2 layers l=2, in Figure X3.

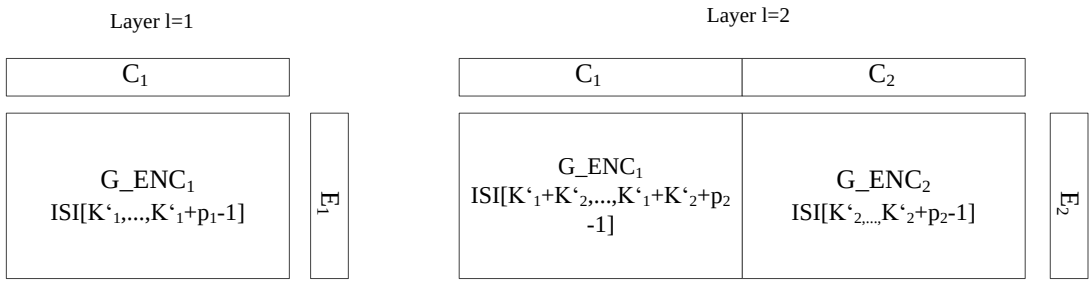


Figure X3: Generation of non-systematic encoding symbols E_1 of layer l=1 and E_2 of layer l=2

E.2.4.1.5 Layer-Aware RaptorQ Decoding

The decoding process is performed analog to the encoding process from a set of received encoding symbols represented by the tuples of R_x with the related ISI[R_x] of all involved media layers x=1,…,L. The decoding process of decoding all layers together is similar to the encoding process (section 6.5.8.1.2). Note that layers used for prediction, e.g. base layer, can also be decoded independently of predicting layers, e.g. enhancement layer. The difference is that for the first encoding process the matrix G_ENC$_1$ and G_ENC$_2$ are generated from ISI[R_1] and ISI[R_2] and G_ENC$_{1_LA_1}$ is generated from ISI[R_2(+K'$_1$)] of all received encoded symbols R_2, where (+K'$_1$) denotes that each element in R_2 is incremented by (+K'$_1$).

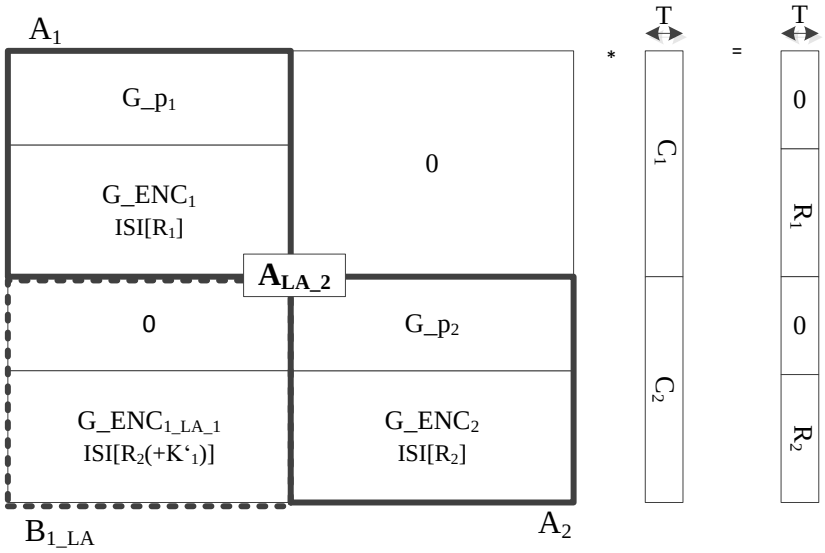


Figure X4: Decoding process with layer-Aware RaptorQ constraint matrix A_{LA_2} for I=2 media layers

An exemplary decoder for solving the decoding process is specified in clause 5.8 of IETF RFC 6330 [6].

The second encoding process is analog to the process described in 6.5.8.1.3 with C=[C_1C_2] as input from the first encoding process.

E.2.4.2 References

[6] M. Luby, A. Shokrollahi, M. Watson, T. Stockhammer, L. Minder, "RaptorQ Forward Error Correction Scheme for Object Delivery," IETF RFC 6330, August 2011.

The second encoding process is analog to the process described in 6.5.8.1.3 with C=[C_1C_2] as input from the first encoding process.

E.2.5 Specification for Low Density Generate Matrix (LDGM) codes

E.2.5.1 Introduction

The LDGM code is a special case of LDPC staircase codes in [RFC 5170]. The specified LDGM codes use packet-division scheme and structured interleaving. Also the difference of the specified LDGM code and LDPC staircase in [RFC 5170], the specified LDGM codes can use optimized parity check matrixes for low complexity message passing decoding algorithm. Furthermore, Layer-Aware LDGM codes that consist of LDGM codes are suitable to support Layer-Aware FEC coding structure. This Annex provides full specification of LDGM/Layer-Aware LDGM codes, which works on the same code point.

E.2.5.2 Low Density Generator Matrix (LDGM) Codes

E.2.5.2.1 Definition

- $\mathbf{H}_{P',N'}$: denotes a sparse parity check matrix with P' rows and N' columns.

- $\mathbf{G}_i$: denotes a mother matrix written in a sparse matrix format described in section E.2.5.2.4.
- $\mathbf{G'}_{P',K'}$: denotes a punctured sparse matrix with P' rows and K' columns.
- $\mathbf{T}_{P'}$: denotes a P'-by-P' staircase matrix.
- $\mathbf{W}_{N'}$: denotes N' Encoding multiple symbols.
- $\mathbf{S}_{K'}$: denotes K' multiple source symbols.
- $\mathbf{C}_{P'}$: denotes P' multiple repair symbols
- K : denotes a number of source symbols
- P : denotes a number of repair symbols
- d : denotes division number of one source symbol
- K' : denotes a number of multiple source symbols K*d
- P' : denotes a number of multiple repair symbols P*d
- $\mathbf{V}_{P'}$: denotes a P' intermediate multiple repair symbols

E.2.5.2.2 Specification of the LDGM Scheme

LDGM codes are one type of XOR based linear code; the parity check matrix contains mostly 0's and only a small number of 1's. The parity check matrixes of LDGM codes consist of 2 parts: one is punctured sparse matrix **G**' and the other is staircase matrix **T** and it given by

$$\mathbf{H}_{P',N'} = [\mathbf{G}_{P',K'}; \mathbf{T}_{P'}]$$

$$= \left[\begin{array}{ccc|ccccc} & 1 & & 1 & & & \\ 1 & & & 1 & 1 & & \\ \cdots & \cdots & \cdots & & \ddots & \ddots & \\ & & 1 & & & 1 & 1 \end{array}\right] \quad (1)$$

where **G**' is a P'-by-K' sparse matrix generated from mother matrix **G** and $\mathbf{T}_{P'}$ is a P'-by-P' square staircase matrix.

E.2.5.2.3 Creating a Punctured Sparse Matrix

The systematic part of parity check matrix is a punctured sparse matrix, which is created by mother matrices. The mother matrices are provided by called an alist format and the format is described in E.2.5.2.4. The maximum number of source symbols and repair symbols is 2^24 in MMT AL-FEC, then, the mother matrices are connected 6 files which is depend on K'. The mapping rule between alist file and K' is shown in Table 1.

Table 1 — Mapping rule between alist file and K'

Alist file name	Corresponding to K' value
LDMG_MMT_0.alist	K' <= 2^8
LDMG_MMT_1.alist	2^8 < K' <= 2^10
LDMG_MMT_2.alist	2^10 < K' <= 2^12
LDMG_MMT_3.alist	2^12 < K' <= 2^14
LDMG_MMT_4.alist	2^14 < K' <= 2^16
LDMG_MMT_5.alist	2^16 < K' <= 2^24

The mother matrix **G** corresponding to the K' value is a square matrix with M rows and M columns. The relationship between M and K' is given by Table 2.

Table 2 — Relationship between K' value and M value

K' value	M value of corresponding to K' value
K' <= 2^8	M = 8
2^8 < K' <= 2^16	M = ceil[ceil[log2(x)]/2]*2
2^16 < K' <= 2^24	M = 24

The punctured matrix is created as follows:

Step1) Removing (2^M-K') columns of 2^M-by-2^M mother matrix from right side. (Fig. 1 (step 1))

Step2) Punctureing (2^M-P') colums of 2-M-by-K' imidiate sparse matrix from top side. This puncturing operation is a simple sum operation of two rows over GF(2). (Fig. 1 (step 2))

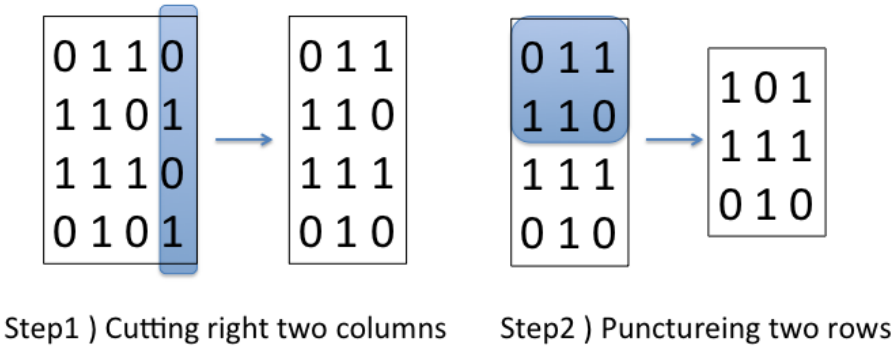


Figure 1 — Puncture operation example to generate punctured matrix

(M=2, K'=3, P'=3 case)

After puncturing process, the P' by K' punctured space matrix **G**' is generated.

The detail of a puncture algorithm to generate a puncture sparse matrix is provided below:

```
/*
 * Return an exponent value of a mother matrix
 * determined by given K' value
 */

int
get_mothermatrix_exp(int k)
{
    if (k <= 256) {
        return 8;
    }
    else if (k > 65536) {
        return 24;
    }
    else {
```

```
        return ceil(ceil(log2(k))/2)*2;
    }
}

/*
 * Return an index of a mother matrix
 * determined by given m value
 */

int
get_mothermatrix_index(int m)
{
    if (m <= 8) {
        return 0;
    }
    else if (m > 16) {
        return 5;
    }
    else {
        return (m+1)/2-4;
    }
}

/*
 * Determine a mother matrix and create a punctured
 * matrix by given K' and P' value
 */
#define EXP2(x) (1<<(int)(x))
#define ISEXP2(x) (((x) & (-x)) == (x))
```

```
void puncture_matrix(int k, int p)
{
    int i;
    int j;
    int l;
    int m;
    int id;

    if(k > p){
       m = get_mothermatrix_exp(k);
    }
    else{
       m = get_mothermatrix_exp(p);
    }
    id = get_mothermatrix_index(m);
    set_mothermatrix(id);
    delete_columns(EXP2(m)-k);

    if(EXP2(m) != p){
       for (i = 0; i < m - floor(log2(p)) - 1; i++) {
           for (j = 0; j < EXP2(m-i-1); j++){
               puncture_rows(EXP2(i+1)*j, EXP2(i+1)*j+EXP2(i));
           }
       }
       if(ISEXP2(p)){
           l = p;
       }
       else{
```

```
            l = EXP2(ceil(log2(p))) - p;
        }
        for (j = 0; j < l; j++){
            puncture_rows(EXP2(i+1)*j, EXP2(i)*(2*j+1));
        }

        for (j = 0; j < l; j++){
            pickout_row(EXP2(i+1)*j);
        }
        for (j = 0; j < p - l; j++){
            pickout_row(EXP2(i)*j + EXP2(i+1)*l);
        }
    }
}
```

E.2.5.2.4 An Alist Format

The mother matrix file is written by alist format. The structure of alist file format is described bellow:

```
/* An Alist Format */

typedef struct {
  int 2^M , 2^M ;      /* size of the matrix */
  int **mlist;     /* list of integer coordinates in the m direction where the non-zero entries are */
  int **nlist;     /* list of integer coordinates in the n direction where the non-zero entries are */
  int *num_mlist;  /* weight of each row */
  int *num_nlist;  /* weight of each column */
  int *l_up_to ;
  int *u_up_to ;
  int *norder ;
  int biggest_num_m ;       /* actual biggest sizes */
  int biggest_num_n ;
  int biggest_num_m_alloc ; /* sizes used for memory allocation */
  int biggest_num_n_alloc ;
  int tot ;
  int same_length ;  /* whether all vectors in mlist and nlist have same length */
} alist_matrix ;
```

E.2.5.2.5 Source symbol division scheme

In the case of ibg_mode 0 and 1, each source symbol has S or S+2 bytes, respectively. The source symbol splitter divides one source symbol to d multiple source symbols (Fig. 2). The division number of d is given number on user side but S%d=0 or (S+2)%d = 0 is recommended.

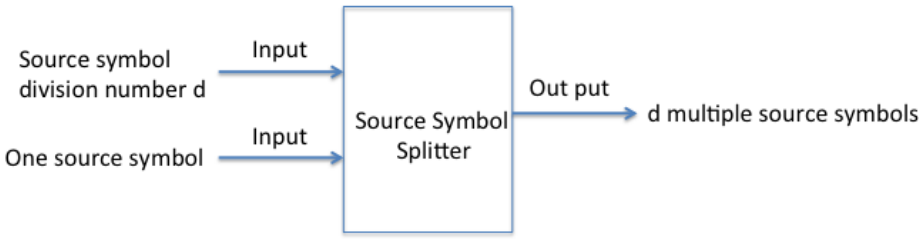


Figure 2—Source symbol splitter process

In the ibg_mode 2 case, single source symbol size is set to ceil [T/K'], where T is FEC information payload size. Then, the K' multiple source symbols are generated.

E.2.5.2.6 Structured interleaving and de-interleaving scheme

Before encapsulation, the generated multiple repair symbols are interleaved. The interleaving just changes the index of its symbols. Let Sj be the j-th multiple repair symbol and always j starts 1 for each block and is incremented by 1. The interleaved new index is given by

$$i' = [(i-1)\%g] * d + \left\lfloor \frac{i-1}{g} \right\rfloor + 1 \quad (2)$$

where [i%g] returns the value of modulo arithmetic, g is number of multiple repair symbols and floor[*] returns the value of cutoff arithmetic. After structured interleaving process (Fig. 3 step 1), i-th multiple repair symbol is mapped to i'-th symbol. (Fig. 3 step2) After this process, repair symbols are regenerated.

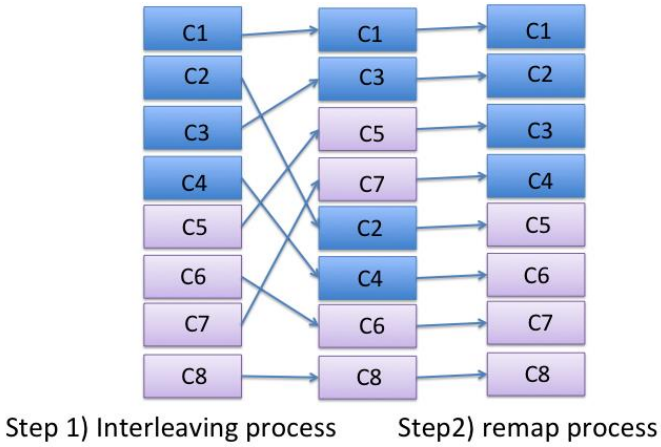


Figure 3— Example of structured interleaving process in the case of g = 8 and d = 4.

In the opposite of interleaving, the de-interleaving process is remap operation from i'-th index to i-th index.

E.2.5.2.7 LDGM code Encoding Algorithm

Briefly, at the encoder side, the LDGM coder calculates each parity packet as the XOR operation of sparse matrix **G'** and the K' multiple source symbols. From definition of linear code, the K' multiple source symbols and the P' multiple repair symbols are related as follows:

$$\begin{aligned} 0 &= [\mathbf{H}_{P',N'}]\mathbf{W}^{t}_{N'} \pmod 2 \\ &= [\mathbf{G'}_{P',K'};\mathbf{T}_{P'}] \begin{bmatrix} \mathbf{S}^{t}_{K'} \\ \mathbf{C}^{t}_{P'} \end{bmatrix} \pmod 2 \qquad (3) \\ &= [\mathbf{G}_{P',K'}]\mathbf{S}^{t}_{K'} + [\mathbf{T}_{P'}]\mathbf{C}^{t}_{P'} \pmod 2 \end{aligned}$$

The multiple repair symbols, $\mathbf{C}_{P'}$, can be written as

$$\mathbf{C}^{t}_{P'} = [\mathbf{T}^{-1}_{P'}][\mathbf{G}_{P',K'}]\mathbf{S}^{t}_{K'} \pmod 2 \qquad . (4)$$

From above equation, the encoding operation can divide to two steps:

Step 1) compute intermediate multiple repair symbols $\mathbf{V}_m = [\mathbf{G}_{P',K'}]\mathbf{S}^{t}_{K'}$,

Step 2) pass intermediate multiple repair symbols $\mathbf{V}_m$ through an accumulator to create $\mathbf{C}^{t}_{m}$.

These process are written as

$$\begin{aligned} \mathbf{C}_1 &= \sum_{j=1}^{K'} \mathbf{G}_{1,j} S_j \pmod 2 \\ \mathbf{C}_2 &= \mathbf{C}_1 \sum_{j=1}^{K'} \mathbf{G}_{2,j} S_j \pmod 2 \\ &\vdots = \vdots \\ \mathbf{C}_{P'} &= \mathbf{C}_{P'-1} \sum_{j=1}^{K'} \mathbf{G}_{P',j} S_j \pmod 2 \end{aligned} \quad (5)$$

Where $\mathbf{G}_{P',j}$ are the matrix elements of each column. From above equation, the P' multiple repair symbols $\mathbf{C}_{P'}$ can be computed in linear time.

E.2.5.2.8 LDGM code Decoding Algorithm

On the decoder side, if packet losses are frequent, source symbols and repair symbols are divided into multiple source and repair symbols, which are then de-interleaved in multi repair symbols part. This process avoids burst multiple repair symbol errors and multiple repair symbols losses do not neighbor each other. After above processing, the decoding of the proposed LDGM codes recover erasure multiple symbols by using MPA algorithm. Since the LDGM codes is linear code, it is possible to write as follow:

$$\begin{aligned} 0 &= [\mathbf{H}_{P',N'}]\mathbf{W}^t_{N'} \pmod 2 \\ \mathbf{H}_L \mathbf{W}^t_L &= \mathbf{H}_C \mathbf{W}^t_C \pmod 2 \end{aligned} \quad (6)$$

where $\mathbf{H}_L$ is parity check matrix corresponding to erasure multiple symbols $\mathbf{W}_L$, and **Hc** is parity check matrix corresponding to correct multiple symbols $\mathbf{W}_C$. In the above left equation, if the decoder fined one erasure symbol in one row, the erasure multiple symbol is recovered by sum of all corrected symbols in same row equation.

Finally, the decoded source symbols are reconstructed again and the decoding process is finished.

E.2.5.3 Layer-Aware LDGM (LA-LDGM) Codes

E.2.5.3.1 Specification of the LA-LDGM Scheme

Layer-Aware LDGM Codes allow the source symbols for the enhancement layers to be used in the base (or lower) layer; this increases the total block length and raises the probability of recovering the base layer. The layer-Aware LDGM codes are made by combining several sparse matrixes. The layer-Aware LDGM codes are defined by

$$\mathbf{0} = \left[\mathbf{H}_{L\ m,n}\right] \mathbf{W}^{t}_{L\ n} \tag{7}$$

where the $\mathbf{H}_L$ is layered sparse matrix, which uses Layer-Aware LDGM codes, and the $\mathbf{W}_L$ is received multiple symbols, which include each layer's data. $\mathbf{H}_L$ and $\mathbf{W}_{Ln}$, are given by

$$\mathbf{H}_{L,P',K'} = \left[\begin{array}{c|ccc} \mathbf{G}_{P'1,K'1} & 0 & 0 & 0 \\ \hline \mathbf{G}_{P'2,K'1+K'2} & 0 & 0 & \\ \hline \vdots & & 0 & \\ \hline \mathbf{G}_{P'L,K'1+K'2+\ldots+K'L} & & & \end{array}\right] \mathbf{T}_{P'} \tag{9}$$

$$\mathbf{W}_{L\ n} = \left[\mathbf{S}_{L\ k} | \mathbf{C}_{L\ m}\right] = \left[\mathbf{S}_{k1}, \ldots, \mathbf{S}_{kL} | \mathbf{C}_{m1}, \ldots, \mathbf{C}_{mL}\right]. \tag{10}$$

Table 1 shows multiple source and repair symbols assigned at base layer and enhancement layers.

Table 1 Multiple souce symbols and corresponding repair symbols at each layer.

	Source symbol	Correspondng repair symbol
Base Layer	**S***k1*	**C***m1*
Enhancement Layer1	**S***k2*	**C***m2*
...	...	...
Enhancement Layer L-1	S*kL*	**C***mL*

E.2.5.3.2 Encoding Algorithm

The encoding algorithm is same process as the basic LDGM codes described in E.2.5.2.7

E.2.5.3.3 Decoding Algorithm

The decoding of the Layer-Aware LDGM codes is carried out by MPA algorithm of the basic LDGM codes described in E.2.5.8. according to the eq. (9). The tanner graph of the Layer-Aware LDGM codes is illustrated in Fig. 4. From Fig. 4, we can see that received symbols for the base layer are connected to those of enhancement layers. From Fig. 4, it is clear that the proposed structure yields the valuable characteristic that the base layer is protected by more parity symbols, and so is very suitable for layered video data. Furthermore, the Layer-Aware LDGM code supports partial decoding.

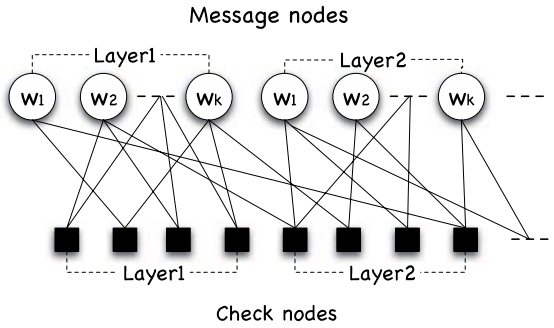


Figure 4 Corresponding to tanner graph of Layer-Aware LDGM codes.

E.2.6 Specification for FEC code algorithms in SMPTE 2022-1

Refer to FEC code algorithms which are specified in SMPTE 2022-1.

Annex F

(informative)

Operation of Downloadable DRM and CAS

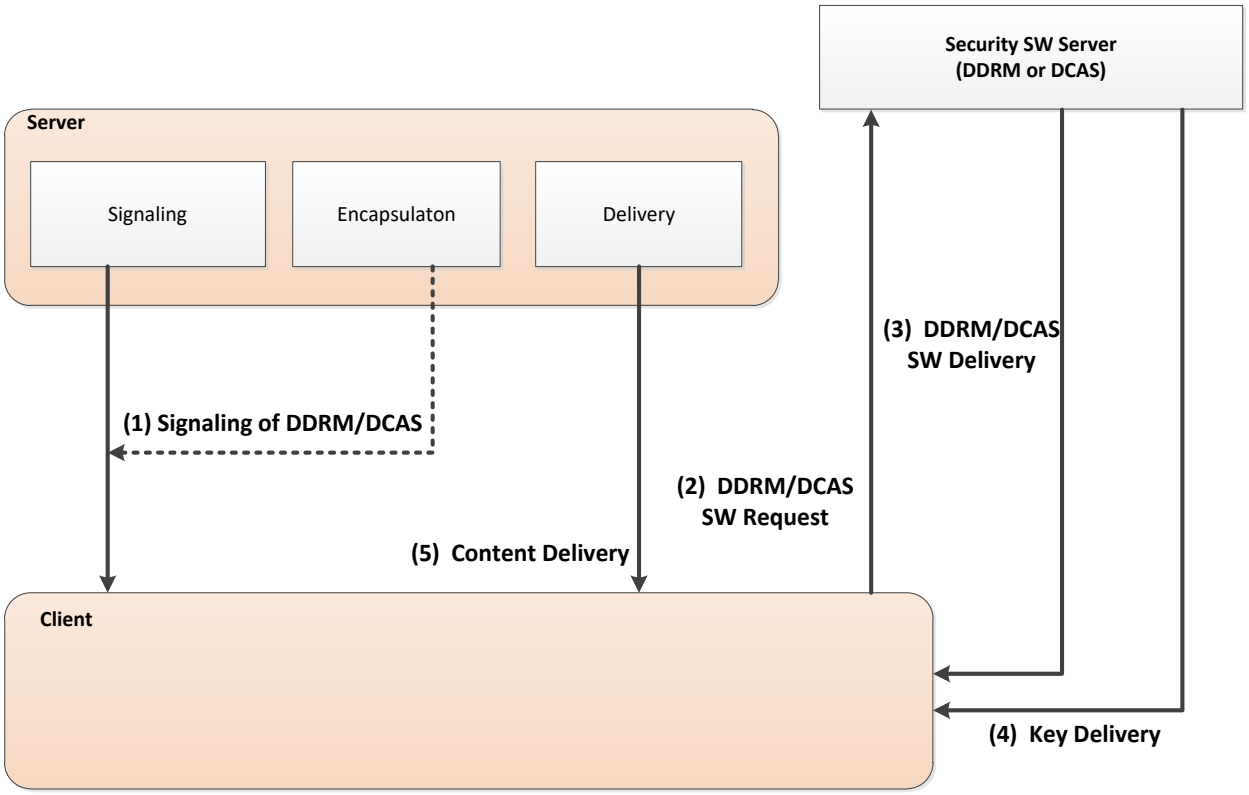


Figure E. 1 - Architecture and Flow for DDRM/DCAS

Figure E. 1 shows a possible architecture of DDRM/DCAS in MMT and also shows flows. Step (1), “Signaling of DDRM/DCAS”, provides the information of a DDRM/DCAS solution and also provides the indication for package and/or asset encryption (the dotted line in step (1)). Step (2) indicates the procedure that Client asks Security SW Server to send the DDRM/DCAS SW. Client needs to send the information required for client authentication in step (2). When Client cannot be authenticated, the secure delivery of the requested DDRM/DCAS SW is not guaranteed. The Security SW Server having the DDRM/DCAS SW sends the SW to Client in step (3). Keys, if any, for Package/Asset are transmitted in step (4) from Security SW Server back to Client. Finally, Client decrypts Package/Asset delivered by step (5).

Security SW Server is a logical entity. It can be located inside of a server having MMT or could be the different physical entity. Among the 5 steps, step (4) is out of our scope. In addition, all procedures related to security key material exchange shall be out of the scope.

Bibliography

[1] ISO/IEC 13818-1 | ITU-T Rec. H.222.0, Information technology – Generic coding of moving pictures and associated audio information: Systems

[2] ISO/IEC 13818-2 | ITU-T Rec.H.262, Information technology – Generic coding of moving pictures and associated audio information: Video

[3] ISO/IEC 14496-10, Information technology – Coding of audio-visual objects – Part 10: Advanced Video Coding

[4] ISO/IEC 23007-1, Information technology – Rich media user interfaces – Part 1: Widgets

[5] IETF RFC 4281, The Codecs Parameter for 'Bucket' Media Types, November 2005.

[6] IETF RFC 5646, *Tags for Identifying Languages*, September 2009.